

Arnold for Softimage User Guide

SltoA Version 3.0

13th Jan 2014

1. Arnold for Softimage User Guide	5
1.1 Release Notes	7
1.1.1 4.1	8
1.1.2 4.0	9
1.1.3 3.16	13
1.1.4 3.15	14
1.1.5 3.14	15
1.1.6 3.13	16
1.1.7 3.12	17
1.1.7.1 3.12.1	18
1.1.8 3.11	19
1.1.9 3.10	20
1.1.10 3.9	21
1.1.11 3.8	22
1.1.12 3.7	23
1.1.13 3.6	24
1.1.14 3.5	25
1.1.15 3.4	26
1.1.16 3.3	27
1.1.17 3.2	28
1.1.18 3.1	29
1.1.18.1 3.1.1	30
1.1.18.2 3.1.2	31
1.1.19 3.0	32
1.1.19.1 3.0.1	33
1.1.19.2 3.0.2	34
1.1.20 2.10	35
1.1.21 2.9	36
1.1.21.1 2.9.1	37
1.1.21.2 2.9.2	38
1.1.22 2.8	39
1.1.23 2.7	40
1.1.23.1 2.7.1	41
1.1.24 2.6	42
1.1.25 2.5	43
1.1.26 2.4	46
1.1.27 2.3	47
1.1.28 2.2	48
1.1.29 2.1	49
1.1.30 2.0	50
1.2 Getting started	51
1.2.1 Arnold	52
1.2.2 Installation	53
1.2.3 Updating SitoA	54
1.2.4 Rendering your first scene	55
1.2.5 Licensing Arnold	64
1.3 Tutorials	65
1.3.1 Understanding Physically Based Rendering in Arnold	66
1.3.2 AOV Tutorials	74
1.3.2.1 Cryptomatte	75
1.3.3 Kick	78
1.3.3.1 Getting Started With Kick	79
1.3.3.2 How to Read a Render Log	83
1.3.4 Lighting	93
1.3.4.1 Lighting a Room	94
1.3.4.1.1 Area Lights	95
1.3.4.1.2 Emissive Geometry	99
1.3.4.1.3 Lighting with the Physical Sky	103
1.3.4.1.4 Lighting with the Sky	106
1.3.4.1.5 Lighting with the Sky Dome Light	109
1.3.4.1.6 Rendering with Ambient Occlusion	112
1.3.4.2 Studio Automotive Rendering	116
1.3.4.3 Studio Lighting	127
1.3.5 Shading	131
1.3.5.1 Displacement Mapping Using the <udim> Token	132
1.3.5.2 Flakes Example	135
1.3.5.3 Hair Tutorial	138
1.3.5.4 Rendering Glass Surfaces with Arnold	152
1.3.5.4.1 Modeling Glass and Liquid	153
1.3.5.4.2 Shading Glass and Liquid	158
1.3.5.4.3 Render Settings	160
1.3.5.5 Shading a Rose	162
1.3.5.6 Specular BRDF (Bidirectional Reflectance Distribution Function)	168
1.3.5.7 Water Bottle	173
1.3.6 Rendering	178
1.3.6.1 Depth of Field	179
1.3.6.2 How to Render a Mandelbulb with SitoA	181
1.3.6.3 ICE Time Shifted Instances	188
1.3.6.4 Rendering Maya Fluids with Softimage	190
1.3.6.5 Rendering an Ocean with Displacement	192
1.3.6.6 Subdivision Tutorial	200
1.3.7 Removing Noise	202
1.3.7.1 What is Sampling?	213
1.3.7.2 Removing Noise Workflow	221
1.3.7.3 Getting rid of noise	225

1.3.7.3.1 Other ways to reduce noise	229
1.4 Arnold Render Options	231
1.4.1 System	232
1.4.2 Output	235
1.4.2.1 TIFF	237
1.4.2.2 EXR	238
1.4.2.3 Deep EXR	240
1.4.2.4 Multi-Layer EXR	244
1.4.3 Sampling	246
1.4.3.1 Samples	248
1.4.3.2 Clamping	255
1.4.3.3 Sample Filtering	257
1.4.4 Motion Blur	259
1.4.5 Subdivision	264
1.4.6 Ray Depth	266
1.4.7 Textures - settings	
1.4.8 Diagnostics	275
1.4.9 ASS Archives	281
1.5 Shapes	283
1.5.1 Arnold Properties	284
1.5.1.1 Arnold Parameters	286
1.5.1.2 Arnold Parameters (Hair and ICE strands)	296
1.5.1.3 Camera Options	299
1.5.1.4 Matte	300
1.5.1.5 Sidedness	302
1.5.1.6 Procedurals	303
1.5.1.7 Texture Options	307
1.5.1.8 User Data Blobs	309
1.5.1.9 User Options	313
1.5.1.10 Visibility	316
1.5.1.11 Volume	320
1.5.2 ICE	325
1.5.2.1 Points	326
1.5.2.2 Strands	327
1.5.2.3 Instance Shapes	329
1.5.2.4 Instancing Lights	330
1.5.2.5 Instancing Stand-ins	332
1.5.2.6 ICE Motion Blur	333
1.5.2.6.1 Troubleshooting sub-frame motion blur	335
1.5.3 Volumes	339
1.5.3.1 Writing volume shaders	344
1.6 Lights	349
1.6.1 Cylinder Light	359
1.6.2 Disk Light	361
1.6.3 Distant Light	364
1.6.4 Mesh Light	366
1.6.5 Photometric Light	371
1.6.6 Point Light	373
1.6.7 Quad Light	375
1.6.8 Skydome Light	381
1.6.9 Spot Light	385
1.6.10 Light Filters	388
1.6.10.1 Light Decay	392
1.6.10.2 Gobo	396
1.6.10.3 Barndoor	401
1.6.10.4 Light Blocker	406
1.6.10.5 passthrough_filter	412
1.7 Camera	413
1.7.1 Perspective	418
1.7.2 Spherical Camera	421
1.7.3 Cylindrical Camera	422
1.7.4 Fisheye Camera	424
1.7.5 VR Camera	426
1.7.6 Camera Motion Blur	433
1.8 Shaders	437
1.8.1 Closure Connector	438
1.8.2 Displacement	439
1.8.2.1 Vector Displacement	440
1.8.3 Arnold shaders	444
1.8.3.1 AOV Shaders	445
1.8.3.2 Atmosphere	446
1.8.3.2.1 Fog	447
1.8.3.2.2 Atmosphere Volume	451
1.8.3.3 Bump	460
1.8.3.3.1 Bump2d	461
1.8.3.3.2 Bump3d	463
1.8.3.3.3 Normal Map	465
1.8.3.4 Closure	467
1.8.3.5 Color Shaders	468
1.8.3.5.1 Color Convert	469
1.8.3.5.2 Color Correct	470
1.8.3.5.3 Color Jitter	476
1.8.3.5.4 Composite	483
1.8.3.6 Conversion	487
1.8.3.7 Environment Shaders	490

1.8.3.7.1 Physical Sky	491
1.8.3.8 Math Shaders	499
1.8.3.8.1 Mix RGBA	502
1.8.3.8.2 Mix Shader	503
1.8.3.8.3 Range	507
1.8.3.8.4 Space Transform	508
1.8.3.9 Shading State	509
1.8.3.10 Surface	511
1.8.3.10.1 Ambient Occlusion	512
1.8.3.10.2 Car Paint	520
1.8.3.10.3 Complex IOR	537
1.8.3.10.4 Flakes	540
1.8.3.10.5 Lambert	543
1.8.3.10.6 Matte Shader	545
1.8.3.10.7 Shadow Matte	546
1.8.3.10.8 Standard Hair	556
1.8.3.10.9 Standard Surface	569
1.8.3.10.10 Thin Film Shader	627
1.8.3.10.11 Two Sided	630
1.8.3.11 Texture	631
1.8.3.11.1 Camera Projection	632
1.8.3.11.2 Checkerboard	633
1.8.3.11.3 Image	636
1.8.3.11.4 Noise	645
1.8.3.12 Utility Shaders	651
1.8.3.12.1 Blackbody	652
1.8.3.12.2 Cache	653
1.8.3.12.3 Clamp	654
1.8.3.12.4 Curvature	655
1.8.3.12.5 Facing Ratio	660
1.8.3.12.6 Flat	663
1.8.3.12.7 Motion Vector	664
1.8.3.12.8 Passthrough	666
1.8.3.12.9 Ray Switch	667
1.8.3.12.10 Ray Switch RGBA	671
1.8.3.12.11 Switch	673
1.8.3.12.12 Trace Set	675
1.8.3.12.13 Utility	677
1.8.3.12.14 UV Transform	691
1.8.3.12.15 Wireframe	698
1.8.3.13 Volume Shaders	701
1.8.3.13.1 Standard Volume	702
1.8.3.13.2 Volume Sample Float	713
1.8.3.13.3 Volume Sample RGB	721
1.8.3.14 User Data Shaders	727
1.8.4 Softimage shaders	729
1.8.5 Dll-so shaders	733
1.8.6 Third Party Shaders	735
1.8.7 Legacy Shaders	736
1.8.7.1 Hair	737
1.8.7.1.1 Hair - Diffuse	740
1.8.7.1.2 Hair - Specular	742
1.8.7.1.3 Hair - Transmission	746
1.8.7.2 Skin	747
1.8.7.2.1 SSS Layers	750
1.8.7.2.2 Specular Layer	755
1.8.7.2.3 Opacity	759
1.8.7.2.4 Advanced.	760
1.8.7.2.5 Skin AOV Names	761
1.8.7.3 Sky	763
1.8.7.4 Standard	767
1.8.7.4.1 Diffuse	771
1.8.7.4.2 Specular.	775
1.8.7.4.3 Bounce Factor	786
1.8.7.4.4 Reflection	787
1.8.7.4.5 Refraction	792
1.8.7.4.6 Sub-Surface Scattering	809
1.8.7.4.7 Emission.	816
1.8.7.4.8 Standard AOV Names	819
1.8.7.5 Volume Collector	820
1.9 Textures	829
1.9.1 Maketx	831
1.9.2 Tokens	834
1.10 Exporting to file	837
1.11 SlttoA Paths	838
1.12 The SlttoA Viewer	841
1.13 Commands	846
1.14 Events	852
1.15 Naming Convention	854
1.16 FAQ	857
1.16.1 Arnold 5.0	860
2. PDF Archive / Updates	862

Arnold for Softimage User Guide

! SltA is no longer maintained by Autodesk. The official SltA plugin source code can be found [here](#).



for Arnold 5

This manual introduces SltA, a plug-in for Autodesk Softimage which allows you to use the Arnold renderer directly in Softimage. Before beginning, you should read the sections on [installation](#) and [licensing](#). Also, don't forget to look at the [Tutorials](#) section. We are adding new tutorials all the time.

i Make sure to check the [release notes](#) for the latest Arnold 5 features. There is also an [FAQ](#) page that endeavors to answer any questions related to transitioning to Arnold 5.

! [Arnold Answers](#) is a reputation-based forum where Arnold users of all kinds can ask their colleagues and peers questions related to Arnold.

✓ Use the / key to perform a quick search.

The Arnold for Softimage User Guide consists of the following sections:

- [Release Notes](#)
- [Getting started](#)
- [Tutorials](#)
- [Arnold Render Options](#)
- [Shapes](#)
- [Lights](#)
- [Camera](#)
- [Shaders](#)
- [Textures](#)
- [Exporting to file](#)
- [SltA Paths](#)
- [The SltA Viewer](#)
- [Commands](#)
- [Events](#)
- [Naming Convention](#)
- [FAQ](#)



.



Release Notes

- [4.1](#)
- [4.0](#)
- [3.16](#)
- [3.15](#)
- [3.14](#)
- [3.13](#)
- [3.12](#)
- [3.11](#)
- [3.10](#)
- [3.9](#)
- [3.8](#)
- [3.7](#)
- [3.6](#)
- [3.5](#)
- [3.4](#)
- [3.3](#)
- [3.2](#)
- [3.1](#)
- [3.0](#)
- [2.10](#)
- [2.9](#)
- [2.8](#)
- [2.7](#)
- [2.6](#)
- [2.5](#)
- [2.4](#)
- [2.3](#)
- [2.2](#)
- [2.1](#)
- [2.0](#)

4.1

August 9, 2017

This version uses [Arnold 5.0.1.1](#)

See also the [Arnold 5.0.1.0](#) release notes for the full list of core enhancements and fixes.

Download link

- [Solid Angle Downloads](#)

Enhancements:

- Polymeshes can now be volume containers.
- Indirect AOV checkbox added to the skydome light.
- Indirect Sample Clamp and Indirect Specular Blur exposed in the rendering options.
- Iridescence (thin film) section added to the Standard Surface shader.
- Normal Map shader now exposing a strength parameter.
- Coat built-in AOVs added.
- Radial distortion parameter for the perspective camera
- New CLM license.

Fixes:

- Standard shaders representation in the viewport.



4.0

June 12, 2017

This version uses [Arnold 5.0.0.3](#)

See the [Arnold 5.0.0.0](#), [Arnold 5.0.0.1](#), [Arnold 5.0.0.2](#) release notes for the full list of core enhancements and fixes.

Download link

- [Solid Angle Downloads](#)

Enhancements

Shaders

- **Standard Surface** : a new `standard_surface` shader was added, with intuitive and energy conserving parameters, a secondary specular coat with separate normal, metallic Fresnel, thin surface support and more. The standard shader is still available but considered deprecated.
- **Standard Hair** : a new physically based `standard_hair` shader was added, with much more accurate specular and transmissive scattering, better diffuse scattering for dirty hair, melanin randomization, and simple and intuitive parameters. The older hair shader is still available but considered deprecated.
- **Standard Volume** : a new `standard_volume` shader was added, usable for rendering a wide variety of volumes. The shader provides independent control of density, scattering color and transparency, in a way that is energy conserving by default. Fire can be rendered using blackbody emission driven by temperature. Displacement can be used to add more detail to volumes. This supersedes the density volume shader, which has been removed.
- **Shader mixing** : surface, hair and volume shaders now output closures rather than colors. This makes it possible to mix shaders efficiently, using the new `mix_shader` node to blend or add two shaders including all their light AOVs. Adopted from the common shaders, there is also a vanilla color mixing shader `mix_rgba`. The ray switch shader has been split up into two `ray_switch_shader` and `ray_switch_rgba` nodes, for switching between shaders and one for switching between texture colors respectively.
- **Bump shaders** : `bump2d` and `bump3d` now output a normal vector that can be linked to new normal parameters in `ambocc`, `lambert`, `standard_surface` and utility shaders. These bump shaders no longer function as passthrough shaders. The bump port of the Softimage material is not used anymore.
- **Noise shader** : The `noise.coord_space` parameter now has a `uv` enum value for texturing using the object's local UV coordinates. Note that this calls the faster 2D noise API, not the 3D noise like all other coordinate spaces. In addition, an arbitrary coordinate space can be specified manually by linking another shader into the new `P` parameter. The shader can now output colors, in scalar mode by blending between `color1` and `color2`, and in vector mode with a separate noise signal per color channel. A new time parameter can be used to smoothly vary noise over time.
- **Utility shader** : the utility shader has been enhanced with a new metal shading mode, and a roughness control has been added that affects plastic and metal modes.
- **Env vars in image filename** : the image node filename field will now expand environment variables of the form `[var]`, in addition to it already expanding them when they are in the texture searchpath.
- **New tags in image filename** : the image node filename field now accepts `<utile>` and `<vtile>` tags. They take an optional integer offset `<utile: 2>` that adds to the tile number; e.g. an offset of 2 from a U coordinate of 1.4 will output tile number 3.
- **Renamed image missing tile parameters** : on the image texturing node, `ignore_missing_tiles` has been renamed to `ignore_missing_textures`, and `missing_tile_color` has been renamed to `missing_texture_color`. The handling of missing textures has been extended to non-tagged textures. It is still recommended to not ignore missing textures and fix them (the render will emit an error to the log and abort if `options.abort_on_error` is left on). However in some cases missing UDIM or other tiled textures may be a deliberate choice. These options allow control, per image node, what happens in those cases.
- **Built-in utility common shaders** : the common utility shaders that shipped at least in part with the plugins have been integrated into the core.
- **Color Jittering shader** : a new `color_jitter` shader can be used to generate random colors within a specified gain, hue and saturation range. Colors may be randomized per face, object, procedural instance or user data, or a combination.
- **Triplanar Shader** : a new triplanar shader can be used to quickly map image textures without needing UV coordinates. The texture is projected onto the object from the 6 sides, and smoothly blended together at the seams.
- **Dedicated normal and vector map shaders** : The common shader `space_transform` had extra functionality for dealing with tangent-space normal or vector maps, and that has been removed. Instead, there are now two new shaders, `normal_map` and `vector_map` that take typical normal map and vector displacement map data, respectively, and prep them for their typical use later in a shading network, similar to how the bump shaders would be used.
- **Skydome light camera visibility** : New camera and transmission parameters set the amount of light contributed to camera and specular transmission rays, so that it is no longer required to use a separate background shader for such purposes. A new shader parameter may be used to link closure shaders to control color and transparency. Skydome lights are now preferred over background shaders, as they provide the same functionality with better sampling.
- **volume_scattering** renamed to `atmosphere_volume`.

Sampling

- **Dithered sampling** : most samplers (e.g. soft shadows, indirect illumination, depth of field) will now take advantage of dithered sampling, which improves the visual distribution of noise especially at low sample rates.
- **Quad lights** : sampling has been improved, reducing noise for surface lighting. For comparisons with the previous sampler, `options.enable_new_quad_light_sampler` can be disabled.



- Cylinder lights : sampling has been improved, significantly reducing noise for cylinder lights in volumes or where cylinder lights are located near other surfaces.
- Disk lights : a novel sampling algorithm has been implemented for disk lights which can greatly improve their rendering quality when shading points near the disk's surface, particularly in volumes.
- Russian roulette : on average better performance for hair, transmission and volume scattering.
- Motion blur time samples : rendering motion blur is now faster when using non-default `deform_time_samples` or `transform_time_samples`.
- Caustic noise reduction : a new method was added to reduce noise from caustics. Noise from GGX microfacet speculars in particular is reduced. `options.indirect_specular_blur` controls the trade-off between more accurate noisy renders at 0.0, and more blurry biased renders with reduced noise at higher values.
- Faster opacity mapping : opacity-mapped transparent surfaces, such as tree leaves, are now sampled more efficiently and can render up to 20% faster, specially in machines with many threads.

Cameras

- New VR camera : a new VR camera is provided.

OSL, Closures and AOVs

- Open Shading Language : shaders can now be written in Open Shading Language, an advanced shading language for production GI renderers. OSL shaders placed in the shader search path are automatically registered as Arnold shader nodes, with their parameters converted to Arnold parameters. Once loaded, they can be inspected, instantiated and linked in exactly the same way as C++ shaders.
- Closures : a new closure parameter type has been added, which shaders can output instead of final colors. There are BSDF, BSSRDF, emission, matte, transparency and volume closures. See the API documentation and examples for more details on how to use these. Linking a color to a closure parameter will automatically create an emission closure with that color. A closure parameter however can't be linked or converted to a color, as the integrator only computes lighting after shader evaluation. In Softimage, the closure type shows in purple. To connect closures to the material's surface slot, a passthrough closure node is provided.
- Light group AOVs : surface shaders now natively support light group AOVs, previously this feature was only available for volume shading. (#4305)
- Light path expressions : light path expressions are used to write lighting components into separate AOVs. No longer should individual shaders define AOVs for direct/indirect light and various layers, rather a regular expression syntax is used to define the subset of all scattering and emission events in the scene that should be written to each AOV. Built-in AOVs are available for the common cases. •This is not exposed by SltOA at the moment.

Other Enhancements

- OpenVDB : rendering of .vdb files is now supported in the Arnold core, using OpenVDB 4.0. The volume node now has a filename parameter that accepts .vdb files. The parameters for both nodes are the same as in the previous OpenVDB plugin, except for the velocity shutter start and end which has been replaced by `motion_start` and `motion_end` parameters shared with other shapes. Velocity grids named `v`, `vel` or `velocity` are automatically detected and used if motion blur is enabled in the scene. `options.texture_searchpath` is used to resolve relative filepaths.
- Improved OpenVDB startup time : On Linux when rendering with many threads, large OpenVDB datasets could cause significant slowdowns at the start of rendering. Startup times are noticeably improved, resulting in buckets rendering faster especially for low AA samples such as during IPR.
- Faster curves : The curves geometric primitive now renders about 5 to 15% faster. In addition, when rendering dense hair clumps, `min_pixel_width` is now up to 2x faster and results in more accurate shadowing, at the expense of a slight increase in sampling noise.
- Quad and spot light roundness : Quad and spot lights now have a roundness parameter, going from a square shape at 0, to rounded corners, to a disk shape at 1. Quad lights also have a new `soft_edge` parameter to soften the edges, similar to the `penumbra_angle` for spot lights.
- RGB camera filter maps : The data type of camera filter maps has changed from float to `AtRGB` so that colored effects are possible.
- Hair UVs : `curves.uvs` is a new parameter to set `sg->u` and `sg->v` default texture coordinates, similar to polymeshes. UVs may be specified per curve or per point, with an array that has the same length as uniform and varying user data respectively. `sg->bu` and `sg->bv` still contain the UV coordinates along the width and length of the curve respectively. The hair parameters `uparam` and `vparam` have been removed, as they can now be set through `curves.uvs`.
- `ARNOLD_PLUGIN_PATH` : This environment variable is now used automatically to load plugins, and to find procedural and volume DSOs.
- Updated to RLM 12.2BL2: We have upgraded the license server and the external library controlling the licensing subsystem from version 12.0BL2 to 12.2BL2, a more stable release fixing various crashes, bugs, hangs and memory leaks. (#5754)
- Updated to OIIO 1.7.12: We have upgraded from OIIO 1.7.7 to OIIO 1.7.12. (#5672, #5826)

Incompatible changes

Options

- Renamed `options.shader_searchpath` to `options.plugin_searchpath`, all types of node plugins are loaded from this path.
- Removed `options.light_gamma` and `options.shader_gamma`. All shader and light attributes are now assumed linear.
- Removed rarely used bottom, right and woven modes from the `options.bucket_scanning` enum.

Lights



- Lights with normalize off and radius zero now no longer emit any light, for consistency with lights with very small radii.
- Light sampling now uses options.GI_diffuse_samples and options.GI_specular_samples for multiple importance sampling.
- Removed constant light decay, and the decay_type parameter, so that all lights now have quadratic decay.
- Removed support for negative lights (negative color, intensity, or shadow_color). Negative values will now be clamped to zero.
- Removed affect_diffuse, affect_specular and affect_volumetrics parameters. Instead diffuse, specular, sss and volume multipliers can be set to zero.
- Renamed gobo parameters scale_s, scale_t, wrap_s, wrap_t to sscale, tscale, swrap and twrap.

Shaders

- Several old Softimage shaders (Phong, Lambert, etc.) have been discontinued.

Ray Types

- The ray types used for ray visibility, the ray switch shader, GI depth and samples have changed. There are now 5 scattering ray types: diffuse reflection, diffuse transmission (includes SSS), specular reflection, specular transmission and volume scattering. In the options node there is now a GI depth and number of samples for diffuse, specular and transmission rays. kick now has arguments to set the diffuse, specular and transmission depth and samples.

Motion Blur

- Arnold and its DCC plugins used to encourage the use of "canonical" 0..1 time for all data in .ass files and in Arnold. For example, if motion blur samples went from frame-relative time of -0.25 to 0.25, in Arnold-land the -0.25 time sample would become 0.0 and the 0.25 time sample would become 1.0. This simplified the data in .ass files, etc but made it difficult or impossible to share .ass standins among vendors or even shots that differed in their time ranges, and to know why the motion blur was mismatched. Now, the transform_time_samples and deform_time_samples arrays, along with time_samples arrays on lights and cameras have all been removed in favor of the simpler motion_start and motion_end float parameters on each object. These should always be frame-relative values, e.g. -0.25 and 0.25 respectively in the example above. NOTE that this means non-uniform motion time samples are no longer possible; if an object has N motion keys, the time interval will always be divided up into N-1 equally-space time intervals. Also, this means that transform motion blur and deformation motion blur will always use the same time range.
- Volume motion blur is also controlled by motion_start and motion_end parameters. The velocity_fps should be set to the framerate, so that the velocity can be scaled to the length of the frame. With the frame relative convention, motion start and end should typically default to -0.5 and 0.5 respectively, indicating the velocity was sampled at the center of the frame.

Other Changes

- Visual Studio 2015 redistributable : In Windows, we now require the VS 2015 redistributable to be installed in order to run Arnold. We expect most of our users to have it already, but if not the installer can be downloaded from the following link: [#48145](https://www.microsoft.com/en-us/download/details.aspx?id=48145)
- Custom cameras : Custom cameras can now implement camera_reverse_ray to convert from camera space to raster space. This makes it possible for any custom camera to work as a dicing camera, support min_pixel_width, support accurate raster-space wireframe width, etc. (#4477)
- procedural node : The procedural node now only supports loading .ass, .obj, .ply files. The dso parameter has been renamed to filename. options.procedural_searchpath is now only used for this node.
- Deferred procedurals : Removed support for deferred procedurals. All procedurals are now loaded during scene initialization (right before rendering). Also, load_at_init, min and max parameters have been removed.
- Gamma options removed :
 - Output drivers now take a color_space string attribute to specify which color space to use for rendered images. Built-in values are linear, sRGB and Rec709. The default value auto will use sRGB for 8 bit formats and linear otherwise. All driver_*.gamma have been removed.
 - image nodes now have an color_space attribute to specify which color space the texture is assumed to be in. Built-in values are linear, sRGB and Rec709. The default value auto will use sRGB for integer (8 or 16 bit) formats and linear otherwise. The attribute options.texture_gamma has been removed.
 - Legacy gamma related options in kick (-tg/sg/lg) have been removed. You can now set the output color space for a render with kick -ocs <string>.
- Removed legacy pixel filters : The following pixel filters, which were rarely used, have been removed: catrom2d_filter, cook_filter, cone_filter, cubic_filter, disk_filter, video_filter.
- Always use auto-bounds for volumes and implicits : Volume and implicit plugins already notify Arnold of the bounds around the data they provide. Users could previously override this with min and max parameters, but those have been removed for volume nodes and are unused for implicit plugins. All volume plugins will automatically have a bounds_slack parameter available to expand the natural bounding box around the data for purposes of adding positional noise in shaders, etc. Volume plugins are responsible for increasing their reported bounds by the bounds_slack amount. Also, the load_at_init parameter has been removed from both volume and implicit type nodes, as it had no effect in practice.

SltoA tickets

.



- #1773 Per Light AOV's
 - #1876 Upgrade to AtString
 - #1935 Change 'Volume Scattering' to 'Atmosphere Volume'
 - #1940 Change naming of 'Stand-ins' to 'Procedurals'
 - #1950 Migrate repo to git
 - #1952 Update to Arnold 5.0
 - #1959 Remove the texture cache enabling flag
 - #1960 Remove obsolete or duplicated shaders
 - #1961 Closure parameter and output for shaders
 - #1962 Expose all the core shaders
 - #1963 Update lights to Arnold 5
 - #1964 Update procedural to Arnold 5
 - #1965 Update volume to Arnold 5
 - #1966 Update the visibility and sidedness properties to Arnold 5
 - #1967 New Arnold 5 motion blur bounds
 - #1968 Update the rendering options to Arnold 5
 - #1969 Update AOVs to Arnold 5
 - #1970 Discontinue the Mask mode of the Matte property
 - #1971 Export the main hair UV set to the new curves uvs array
 - #1972 Fix the testsuite for SltoA 4
 - #1973 Match the image clip color profiles
 - #1974 Vector displacement handle
 - #1975 Limit the verbosity level for the rendering pre-passes
 - #1976 VR camera
 - #1977 Rename sta_passthrough_filter to passthrough_filter
- .

3.16

December 19, 2016

This version uses [Arnold 4.2.16.1](#) (see also [Arnold 4.2.16.0](#))

Download link

- [Solid Angle Downloads](#)

Enhancements

- Updated to [Arnold 4.2.16.1](#).
- Light portals mode for skydome, now having a `portal_mode` parameter with options `off`, `interior_only` and `interior_exterior` to respectively turn off portals, block any light outside portals for interior only scenes, and let light outside portals through for mixed interior and exterior scenes.
- Complex IOR shader, to render materials with complex index of refractions. It also comes with a number of presets for common metals. This shader should be used as a specular/reflection color map in the Standard shader, with the Fresnel option disabled, as it does its own Fresnel calculations.
- Thin film shader, reproducing the interference pattern effects of a thin film on a surface. It can be linked to the Specular or Reflection color of a Standard shader with the Fresnel option disabled, as the shader does its own Fresnel calculations.
- Flakes shader: creates a procedural flake normal map that can be used for materials such as car paint. Note that the map does not perform any filtering, so small flakes require more AA samples to render cleanly.
- `mib_texture_remap`'s transform parameter is now texturable.
- `shadow_matte` shader enhanced and not conflicting anymore with the `shadow_matte` built-in AOV.

3.15

November 18, 2016

This version uses [Arnold 4.2.15.1](#) (see also [Arnold 4.2.15.0](#))

Download link

- [Solid Angle Downloads](#)

Enhancements

- Updated to [Arnold 4.2.15.1](#).
- Light portals: skydome lights can now use portals to reduce noise for interior scenes, where light comes in through relatively small openings. Light portals are regular quad_light nodes where the new parameter "portal" is enabled.
- OpenVDB motion blur optimization: faster rendering of motion blur from physics simulations, between 25% and 80% in test scenes with strong motion blur. The new `velocity threshold` parameter controls filtering of noisy/extreme velocities.
- Volume shader support for SDF (Signed Distance Function) grids.
- New core AOVs available: diffuse albedo, shadow matte.
- Added a detailed texture statistics diagnostic flag.



3.14

July 14, 2016

This version uses [Arnold 4.2.14.1](#) (see also [Arnold 4.2.14.0](#))

Download link

- [Solid Angle Downloads](#)

Enhancements

- Updated to [Arnold 4.2.14.1](#).
- New [curvature](#) and [shadow_matte](#) shaders.
- Adding back the "Info/progress" verbosity level, and cleaning up the messages when logged to file.
- Warning about incomplete clusters.
- Updated About and EULA boxes.

3.13

March 7, 2016
This version uses [Arnold 4.2.13.0](#)

Download link

- [Solid Angle Downloads](#)

Enhancements

- Update to [Arnold 4.2.13.0](#).
- New spread parameter for quad and disk lights.
- New radius parameter for the photometric light.
- Deep EXR now supporting append mode.

Linux installation notes

- Starting from SItOA 3.6.0 (which uses Arnold 4.2.6.0), the minimum requirements for Linux have been raised to the equivalent of RHEL /CentOS 6 for glibc and libstdc++. However, Softimage ships with a legacy libstdc++ that is put in the LD_LIBRARY_PATH and that hijacks the system one, making it impossible to load the Arnold DSO. To fix this, one solution is to preload the system libstdc++ before launching Softimage:

```
export LD_PRELOAD=/usr/lib64/libstdc++.so.6
```

- Another solution is to remove the libstdc++ from the Softimage install and let the system one take over, for example:

```
cd my_softimage_install/Application/bin
mv libstdc++.so.6 libstdc++.so.6.bak
mv libstdc++.so.6.0.10 libstdc++.so.6.0.10.bak
```

3.12

January 12, 2016
This version uses [Arnold 4.2.12.1](#)

Download link

- [Solid Angle Downloads](#)

Enhancements

- Update to [Arnold 4.2.12](#).
- New specular distribution (Beckmann or GGX) and SSS profile parameters in the standard shader.
- Support for per-light AOVs for volumetrics.
- Allow for negative thread count, to indicate how many cores not to use.
- A licensing dialog is now available from the Arnold menu, a simple dialog to troubleshoot and install license files. You can query the license status, save diagnostic information to a file to send to support or obtain your host ID to generate a node-locked license.

Linux installation notes

- Starting from SltOA 3.6.0 (which uses Arnold 4.2.6.0), the minimum requirements for Linux have been raised to the equivalent of RHEL /CentOS 6 for glibc and libstdc++. However, Softimage ships with a legacy libstdc++ that is put in the LD_LIBRARY_PATH and that hijacks the system one, making it impossible to load the Arnold DSO. To fix this, one solution is to preload the system libstdc++ before launching Softimage:

```
export LD_PRELOAD=/usr/lib64/libstdc++.so.6
```

- Another solution is to remove the libstdc++ from the Softimage install and let the system one take over, for example:

```
cd my_softimage_install/Application/bin
mv libstdc++.so.6 libstdc++.so.6.bak
mv libstdc++.so.6.0.10 libstdc++.so.6.0.10.bak
```



3.12.1

January 18, 2016

This is a bugfix release, using [Arnold 4.2.12.2](#)

Download link

- [Solid Angle Downloads](#)

3.11

November 4, 2015
This version uses [Arnold 4.2.11.0](#)

Download link

- [Solid Angle Downloads](#)

Enhancements

- Added support for per-node ICE attributes on polymeshes (SltOA #1917).
- Custom metadata for deep EXR outputs (SltOA #1922).
- The standard and lambert shaders now use Russian Roulette termination to more efficiently render with high GI depth. For AA samples 5 or higher the increase in noise is typically very small. Indoor scenes with high GI depth will benefit the most, but also scenes with lots of glass and high refraction/reflection depth. In such scenes we have measured between 1.5x and 5x faster renders. (core #4901).
- Render options and stats in EXR metadata: Several global render options, such as sample settings and ray depths, are now stored in the image file as EXR metadata (core #4849, #4860).

Fixes

- Fixing broken CAVs and UVs on objects with geometry operators such as local subdivisions (all versions) and boolean (2013 only) (SltOA #1623, #1652, #1841, #1919).
- Incorrect render with matte surfaces behind volumes (core #4918)

Linux installation notes

- Starting from SltOA 3.6.0 (which uses Arnold 4.2.6.0), the minimum requirements for Linux have been raised to the equivalent of RHEL / CentOS 6 for glibc and libstdc++. However, Softimage ships with a legacy libstdc++ that is put in the LD_LIBRARY_PATH and that hijacks the system one, making it impossible to load the Arnold DSO. To fix this, one solution is to preload the system libstdc++ before launching Softimage:
 -

```
export LD_PRELOAD=/usr/lib64/libstdc++.so.6
```

- Another solution is to remove the libstdc++ from the Softimage install and let the system one take over, for example:

```
cd my_softimage_install/Application/bin
mv libstdc++.so.6 libstdc++.so.6.bak
mv libstdc++.so.6.0.10 libstdc++.so.6.0.10.bak
```

3.10

October 2, 2015
This version uses [Arnold 4.2.10.0](#)

Download link

- [Solid Angle Downloads](#)

Enhancements

- Up to 40% faster export of polygon meshes.
- EXR append mode now works properly.
- Improved OpenVDB plugin and shaders.

Linux installation notes

- Starting from SItOA 3.6.0 (which uses Arnold 4.2.6.0), the minimum requirements for Linux have been raised to the equivalent of RHEL /CentOS 6 for glibc and libstdc++. However, Softimage ships with a legacy libstdc++ that is put in the LD_LIBRARY_PATH and that hijacks the system one, making it impossible to load the Arnold DSO. To fix this, one solution is to preload the system libstdc++ before launching Softimage:

```
export LD_PRELOAD=/usr/lib64/libstdc++.so.6
```

- Another solution is to remove the libstdc++ from the Softimage install and let the system one take over, for example:

```
cd my_softimage_install/Application/bin
mv libstdc++.so.6 libstdc++.so.6.bak
mv libstdc++.so.6.0.10 libstdc++.so.6.0.10.bak
```

3.9

September 11, 2015
This version uses [Arnold 4.2.9.0](#)

Download link

- [Solid Angle Downloads](#)

Enhancements

- More robust IPR.
- Direct and indirect AOVs for volume and SSS.
- Improved OpenVDB plugin, standard and image shaders.
- Adaptive tessellation also works in object space.
- Faster volumes, curves, deep EXR writing, subdivision, and rendering of opacity-mapped objects.

Linux installation notes

- Starting from SltOA 3.6.0 (which uses Arnold 4.2.6.0), the minimum requirements for Linux have been raised to the equivalent of RHEL /CentOS 6 for glibc and libstdc++. However, Softimage ships with a legacy libstdc++ that is put in the LD_LIBRARY_PATH and that hijacks the system one, making it impossible to load the Arnold DSO. To fix this, one solution is to preload the system libstdc++ before launching Softimage:

```
export LD_PRELOAD=/usr/lib64/libstdc++.so.6
```

- Another solution is to remove the libstdc++ from the Softimage install and let the system one take over, for example:

```
cd my_softimage_install/Application/bin
mv libstdc++.so.6 libstdc++.so.6.bak
mv libstdc++.so.6.0.10 libstdc++.so.6.0.10.bak
```

3.8

July 13, 2015
This version uses [Arnold 4.2.7.5](#)

Download link

- [Solid Angle Downloads](#)

Enhancements

- Faster export times overall on Windows (up to 20% faster).
- Faster export of polygon meshes (up to 35% faster).
- Much faster export of ICE instances (up to 5x faster) and ICE cylinders, discs, cones, boxes (up to 3x faster).
- Faster rendering when using SltOA shaders on Windows (up to 15% faster).

Linux installation notes

- Starting from SltOA 3.6.0 (which uses Arnold 4.2.6.0), the minimum requirements for Linux have been raised to the equivalent of RHEL /CentOS 6 for glibc and libstdc++.
- However, Softimage ships with a legacy libstdc++ that is put in the LD_LIBRARY_PATH and that hijacks the system one, making it impossible to load the Arnold DSO.
- To fix this, one solution is to preload the system libstdc++ before launching Softimage:

```
export LD_PRELOAD=/usr/lib64/libstdc++.so.6
```

- Another solution is to remove the libstdc++ from the Softimage install and let the system one take over, for example:

```
cd my_softimage_install/Application/bin
mv libstdc++.so.6 libstdc++.so.6.bak
mv libstdc++.so.6.0.10 libstdc++.so.6.0.10.bak
```

3.7

Release Date

June 10 2015

This version uses Arnold 4.2.7.3

Download links

- [Solid Angle Downloads](#)

HIGHLIGHTED NEW FEATURES

- Faster cutout texture mapped opacity, with more accurate renders (in previous versions, an object was rendered more transparent when it was further away from the camera). This optimization can introduce noise or flicker on far away geometry, similar to when rendering far away high-resolution geometry. If this happens, it is best to use a high amount of AA samples to reduce flickering in animation. The fast opacity flag is enabled for anything that is connected to an RGB parameter called "opacity", so third-party shaders that have an RGB "opacity" parameter will automatically benefit from this.
- Multiple scattering for volumes: Indirect light in volumes now supports an arbitrary number of bounces instead of being fixed to one bounce. It is now possible to render volumes such as clouds for which multiple scattering has a large influence on their appearance. The new Volume parameter in the Ray Depth settings sets the number of bounces.
- Per-light volume contribution: A volume contribution scaling parameter was added to lights, similar to the existing diffuse and specular parameters.
- Deep volume output support: Volumes are now visible in deep renders (note that older atmosphere shaders and volumetric mattes are not supported yet).
- The volume property now shows the names of available grids in VDB files.
- Faster UDIMS: UDIMS accessed through the built-in image shader node now internally use texture handles, which helps improve multi-threading performance.

NOTES

- The default value of Volume indirect samples has been changed from 0 to 2, which could change the look of old scenes.

•

3.6

Release Date

April 27 2015

This version uses Arnold 4.2.6.0

HIGHLIGHTED NEW FEATURES

- Optimized volume rendering
- Fixed long-standing artifacts with large ray marching steps
- Faster volume caches (such as OpenVDB) in IPR

NOTES

- We restored the old matte settings. The Matte property now allows you to choose between Matte (the built-in Arnold matte) and Mask (the old SltA matte)
 - Starting from SltA 3.6.0 (which uses Arnold 4.2.6.0), the minimum requirements have been raised to the equivalent of RHEL/CentOS 6 for glibc and libstdc++.
- However, Softimage ships with a legacy libstdc++ that is put in the LD_LIBRARY_PATH and that hijacks the system one, making it impossible to load the Arnold DSO.
- To fix this, one solution is to preload the system libstdc++ before launching Softimage:

```
export LD_PRELOAD=/usr/lib64/libstdc++.so.6
```

- Another solution is to stow away the libstdc++ from the Softimage install and let the system one take over, for example:

```
cd my_softimage_install/Application/bin
mv libstdc++.so.6 libstdc++.so.6.bak
mv libstdc++.so.6.0.10 libstdc++.so.6.0.10.bak
```

•

3.5

Release Date

March 09 2015

This version uses Arnold 4.2.4.0

HIGHLIGHTED NEW FEATURES

- Transmission component in the hair shader.
- Support for the Volume primitive as a property and volume_sample shaders added. Mattes now work with volumes as well.

NOTES

- Arnold now supports node-locked licences. A node-locked license (.lic) is automatically used if you put the license file in the same folder as the Arnold binaries (libai.so or ai.dll). If you put the license file somewhere else, you must set the `solidangle_LICENSE` environment variable to point to the location of the license file. The `solidangle_LICENSE` environment variable can include both RLM servers (in the `port@host` form) and paths (to folders containing license files, or full paths to specific license files). On Linux, use the colon (`:`) to separate multiple entries, and on Windows, use the semi-colon (`;`).

INCOMPATIBLE CHANGES

- Prior to SltoA 3.5.0, we applied the Output Gamma value to the display driver as well as to 8-bit output file types (such as JPEG). The Output Gamma value no longer affects the display (Softimage already has proper color management) and the option is now on the Output tab, where it affects the output of JPEG, PNG, and TIFF files only.

3.4

Release Date

January 16 2015

This version uses Arnold 4.2.3.1

HIGHLIGHTED NEW FEATURES

- Faster Curves and Volumes.
- Opacity controls added to the Skin Shader.
- The autobump detail enhancement now works for procedurally displaced polymeshes even when there are no UV texture coordinates.

NOTES

- Arnold now supports up to 128 threads. Shaders that make use of `AI_MAX_THREADS` will need to be recompiled otherwise crashes may occur in upcoming machines with more than 64 threads. In addition, like in Linux, Windows can now make use of the existing options. `pin_threads`.
- User-specified polymesh vertex normals are now preserved and respected during displacement, and also when using linear subdivision. For cases where Catmull-Clark subdivision is not needed, this allows sharp edges to remain sharp when increasing mesh density for detailed displacement.

INCOMPATIBLE CHANGES

- Raised minimum Windows version to Windows 7: We no longer support Windows versions before Windows 7 and Windows Server 2008 R2.
- Spherical and cylindrical cameras now maintain their FOV vertically regardless of frame and pixel aspect ratio. For spherical cameras this means that the render will always reach all the way to the poles vertically. For cylindrical cameras `vertical_fov` will now be correctly respected.

3.3

Release Date

November 12 2014

This version uses Arnold 4.2.2.0

HIGHLIGHTED NEW FEATURES

- The ambient occlusion shader has a new check to only gather occlusion against the same object.
- Subdivision soft creases and ICE-driven edge and vertex creases are supported.
- The importance sampling tables needed by textured quad area lights can now be computed in parallel. This can result in important speedups in scenes with many lights that use high-resolution tables.
- Faster thread creation/destruction.
- Reduced texture IO.

NOTES

- Now we expose the new Arnold 4.2.2 matte parameter from shapes in the same Arnold matte PPG. We removed the old matte functionality that was implemented through the STA_SHADINGROOT shader.

INCOMPATIBLE CHANGES

- SSS with zero diffusion radius: previously such shaders would render black, now the result is the same as a diffuse BRDF. This affects the standard shader.



3.2

Release Date

September 19 2014

This version uses Arnold 4.2.1.2

HIGHLIGHTED NEW FEATURES

- EXR overscan support.
- Improved sampling (textured quad lights, Cook-Torrance specular and refraction.
- Improved performance for shadow ray and rough glossy texture lookups.

INCOMPATIBLE CHANGES

- Difference in microfacet BTDFs: Rough refraction viewed at grazing angles or using Phong-smoothed normals renders slightly darker now, especially for high roughness values. This affects the standard shader and `AiMicrofacetBTDFIntegrate()`, and fixes a bug in the previous BTDF sampling.



3.1

Release Date

HIGHLIGHTED NEW FEATURES

- Many rendering optimizations in Arnold 4.2 including faster ray accel build, subdivision, displacement, polymesh processing, rendering of many-light scenes, backlighting, SSS, image shader and deep output.
- Improved quality in motion blur sampling and indirect specular sampling.
- Smoother bump near shadow terminator and reduced edge darkening in Cook-Torrance BRDF.
- The Standard shader can now output refraction opacity as an AOV. It is advised to write the complete RGBA set of channels, since the alpha component will constitute a mask that can be used to mix it in with the regular opacity, multiplying in refraction opacity to regular opacity based on the mask.
- Support for Custom shutter shapes.
- Support for Rolling shutter.
- `trace_set` shader implemented.
- ICE improvements: Hierarchies can be shape-instanced and texturable rectangular ICE shapes.
- `sib_texture2d_grid`, `txt2d-grid` shaders and Factory Modulus Shader implemented.

NOTES

- This will be the last version supporting Softimage 2012.
- We are statically linking OIIO in Windows, the same way we were doing in Linux and OSX.
- Photometric IES files are now searched for in `texture_searchpath`.
- Removed the AutoTransparency Mode.
- The `low_light_threshold` and camera exposure parameters from the Arnold core have been exposed in the SltA UI.
- "Use Autobump in SSS" checkbox added in the sampling tab to control whether displacement autobump is taken into account when computing sub-surface scattering.
- `skin_sss` has been removed.

INCOMPATIBLE CHANGES

- Removed cubic format from the sky and skydome light.
- Removed `specular_brdf` from standard shader: Since the Cook-Torrance is similar in performance to the Ward-Duer BRDF yet superior in quality due to its reduced edge darkening and higher energy conservation, the `specular_brdf` parameter has been removed from standard, leaving Cook-Torrance as the default (and only!) BRDF of the standard shader. Therefore, the look of existing standard shader-based materials that use the anisotropic Ward-Duer BRDF might change a little bit for the better.
- Removed the lens shaders defining the camera type; this is now done in the camera options property. We have also moved the DOF controls here as well, so DOF can be enabled per camera.



3.1.1

Release Date

July 10 2014

This version uses Arnold 4.2.0.5

#1807	Implement sib_texture2d_grid and txt2d-grid
-------	---



3.1.2

Release Date

August 8 2014

This version uses Arnold 4.2.0.6

#1811	Restore the support for custom cameras
#1813	Make light and camera exposure's soft range -5.0 5.0
#1814	mib_twosided should have RGBA inputs and output, not RGB



3.0

Release Date

January 08 2014

This version uses Arnold 4.1.3.3

HIGHLIGHTED NEW FEATURES

- Faster: physical sky, multi-layered SSS, .ass loading, Perlin noise, light sampling and many-light scenes, bump3d and gobo shader.
- Reduced memory overhead in several fronts (polymesh data, per-object, per-thread and per-pixel overhead).
- Automatic UI generation for shaders in a .so/.dll. You can now use your favorite third party shaders without having to mess with .sdpl files.
- Ability to add user-data to geometric shapes, which can later be queried by shaders at render time.
- Texture support in mesh_light.
- Smoother, curved rotational motion blur.
- Improved performance in SITOA_Viewer.
- New, faster skin shader with a simplified workflow.
- Deep EXR support.
- Custom EXR metadata.
- Improved the photometric light UI.

NOTES

- The UI of the shaders of existing scenes will render without a proper layout due to the new automatic UI generation for shaders.
- The skin_sss shader is considered legacy and will be removed in the future.
- Enable AOV composition is now always on. RGB and RGBA AOVs support opacity composition.
- The ability to export native hair as a .bin file has been discontinued since the .ass binary data mode is good enough.

INCOMPATIBLE CHANGES

- The Motion Blur controls have been simplified and unified across the different plugins. This might change the look of older scenes.
- Reversed U axis of latlong and angular environment mappings. This change affects the sky, skydome_light and the sta_environment shader.
- Pointcloud SSS has been removed, including all the old parameters. SSS is now computed exclusively with the ray traced SSS engine, which has a single quality control (BSSRDF samples). This leads to warnings when loading old scenes, and possibly a different look and/or noise level.
- The stretched_phong BRDF has been removed.
- sta_volume_collector has been renamed to "volume_collector", and adopted the common shader's code. Therefore old existing scenes that use this shader will break.
- The name of the SITOA_DestroyScene command has changed from "Arnold Scene Destroyer" to "SITOA_DestroyScene". If you have a keyboard mapping for "Arnold Scene Destroyer", you'll have to clear that mapping and then assign the key to "SITOA_DestroyScene".



3.0.1

Release Date

January 14 2014

This version uses Arnold 4.1.3.3.

#1744	Better message on missing or invalid SITOA_LINKTAB_LOCATION variable
#1745	Restore the shaders' UI mapping for the ogl view
#1746	Always create the Arnold render channels on the first instance of the shaders



3.0.2

Release Date

January 14 2014

This version uses Arnold 4.1.3.3.

#1783	Support time shifted ICE shapes on strands
-------	--

2.10

Release Date

July 25 2013

This version uses Arnold 4.0.16.0.

HIGHLIGHTED NEW FEATURES

- Support for Multiple UV projections (~~#278~~). Color vertices at maps and normal maps will work as expected in all the cases.
- New scene rebuild mode to control how scene changes are handled inside an IPR session (Auto, Always, Manual and Fly-through). See [details](#).
- Fog Shader exposed (~~#1630~~). See [details](#).
- Added a photometric light (~~#1088~~). See [details](#).

NOTES

- Softimage 2011 is not supported anymore.
- SlttoA for Softimage 2014 will only work with SP2 or later due to an API compatibility break from Autodesk.
- Due to the support for multiple UV projections, the look of old scenes using multiple uv projections can change.
- The complex_fresnel shader has been removed as it was buggy and there are better solutions around it ([#1632](#)).
- Improvements to the physical sky could break old scenes that used it.



2.9

Release Date

June 5 2013

This version uses Arnold 4.0.15.0.

HIGHLIGHTED NEW FEATURES

- Optimized polymesh storage and faster ray accel.
- Added a physically-based sky shader. See [details](#).
- Holdout Matter Property. See [implementation details](#).
- Support for creating and destroying objects during an IPR session.

NOTES

- This is the last version supporting Softimage 2011.
- A regression has been confirmed from Autodesk in Softimage 2014 when using ICE strands. This will make the exportation time slower than in previous versions.



2.9.1

Release Date

June 07 2013

This version uses the Arnold 4.0.15.1 core and fixes some important bugs with the physical sky implementation



2.9.2

Release Date

July 09 2013

This version uses the Arnold 4.0.15.1 core and reverts back the workaround introduced in Softimage 2014 for the slowdown accessing the ICE attributes that has been fixed in Softimage 2014 SP2. SltA 2.9.2 2014 addon won't work in a version previous to Softimage 2014 SP2.



2.8

Release Date

April 15 2013

This version uses Arnold 4.0.14.0.

HIGHLIGHTED NEW FEATURES

- Faster export times.
- Improved behavior on isolate selection.

NOTES

- Beware : We now follow the official Softimage cross-platform deployment workflow. This means that the SltoA plugin is installed in the Application\Plugins\bin\nt-x86-64 folder on Windows and Application/Plugins/bin/linux on Linux. • This can make old scenes stop working unless the shaders_path and procedurals_path are changed to reflect the new path.
- Beware : We have removed the legacy light filters and the sta_displacement shader, which can make old scenes stop working.
- This is the first release with Linux binaries.
- This is the first version supporting Softimage 2014.
- Support for the Softimage motion blur property has been removed. The Arnold Parameters PPG now has two checks to independently disable transformation and deformation motion blur.

-
-



2.7

Release Date

February 22 2013

This version uses the Arnold 4.0.12.1 core.

Highlighted New Features

•

- Volume Rendering support. See [implementation details](#).
- Per-light scaling for Diffuse / Specular / SSS.
- Exported .ass files are now smaller and faster to read and write thanks to enhanced binary encoding.
- Secondary specular highlight in hair shader.
- The Standard Shader can now compute Fresnel reflectance based on the IOR.
- Multiple objects can be tagged as belonging to the same SSS set so that illumination will blur across object boundaries. This is done via the "SSS Set Name" string found in polymeshes. This feature is only available in raytraced SSS, and not in pointcloud-based SSS.
- The Image node now supports ICE texture coordinates.
- Sharper textures.

Notes

•

- SltOa now defaults to raytraced SSS instead of pointcloud-based SSS (which is deprecated and will disappear in a future version). Scenes saved with older versions will render differently, potentially noisier, depending on the AA and BSSRDF sample settings.
- Textures Glossy Blur default value has been changed to 0 to reflect the improvements done in Glossy reflections using the Cook-Torrance BRDF, which will now pull the correct texture mipmap level across all distances while maintaining an optimal amount of texture sharpness and reducing the texture I/O.
- Renamed the per-light "Bounces" and "Bounce Factor" parameters to "Max Bounces" and "Indirect" to better reflect their purpose.
- For the new `texturetime` AOV to work, the global option `shader_timing_stats` must be enabled first. This is not exposed in the UI, so it must be added through the User Options field in the renderer settings. Note that `shader_timing_stats` should only be used for debugging and not for final renders, as the more accurate timing will cause renders to be a little slower.
- Sharper textures involve a bit of expense of increased texture file I/O and will improve the look of previous scenes. If you want to go back to older look you must set to 1.0 the `texture_max_sharpen` parameter in the User Options field in the renderer settings.



2.7.1

Release Date

March 05 2013

This version uses the Arnold 4.0.12.1 core and fixes some important bugs.

2.6



Release Date

18 December 2012

New Features

- Custom Light Filter support. See [details](#).
- Custom Property to set images filter type, mipmap bias, wrapping to the image nodes.
- Centralized Arnold menu, where you can get access to all the Arnold properties and nodes.
- Flags support to resolve tokens for the standin's data and user parameters.

Notes

- We have changed the light filter workflow but support the legacy filters for a couple of releases, for scenes saved with a previous SItOA version. These scenes, when loaded, will default to the old method, and the new filter shaders, if applied, will be ignored. For new scenes, or for old ones with re-generated rendering options, you'll see a new flag, which allows you to use the legacy properties in the renderer settings.
- To apply the new Arnold_Texture_Options property to the image nodes, select an image and run Arnold->Properties->Texture Options. The property is placed under the Custom folder of the image.
- We have improved various aspects of the SITO A commands. Now, the SITO A_Add*Properties commands will return the collection of added properties, so to easy custom scripting, as well as exposed commands for creating lights. See [Sito A Commands](#).

2.5

What's new in Version 2.5

SltoA version 2.5 is based on version 4.0.10.1 of the Arnold core.

New features include:

- User Options now has a flag called Resolve Tokens. If enabled, the Softimage tokens such as [Frame] are resolved before the string is passed to Arnold. See the [User Options](#) topic for details.
- Help links have been updated and now point to the most specific topic in the on-line help.
- Improved the exporting time with light associations.
- Added <udim> and <tile> token tag support to sib_image_clip.
- sta_vector_displacement shader added. This shader has support for tangent-space vector displacement. This is meant to supercede and replace the old sta_displacement shader, which is now deprecated.
- 25% faster bump mapping (less shader evaluations per call).
- 25% faster ray acceleration structure build (30-80%, depending on platform and compiler).
- Several other speed and memory optimizations.
- Auto-instancing of standins whose Path parameter points to an .ass file; subsequent standins trying to load the same .ass file will instance the previously loaded geometry, potentially saving lots of both memory and disk I/O. By default this feature is turned off, it can be enabled by setting the global option "enable_procedural_cache true" in the [User Options](#) string field in the renderer settings.

Notes:

- After implementing vector displacement (#1312), sta_displacement is no longer needed to perform scalar displacement mapping. The displacement branch can be plugged directly into the displacement slot of the material. Usage of sta_displacement will still be allowed for both scalar and vector displacement, but is deprecated and will be removed in a future version.
- Unified convention names. For more details see the note 'On Exported Names' below.
- Ticket #280 can affect the look of assets shaded with the standard and / or skin shaders. Previously, color values at the edges in the standard shader could go negative when using both mirror and glossy reflection at the same time. A darkening effect could also appear when the Fresnel reflection at the edge was set to less than 100%.
- SltoA for Softimage 2013 is now compiled in Visual Studio 2010.

On Exported Names

In SltoA 2.5 we tried to standardize the way we name the exported Arnold nodes after the corresponding Softimage object (see ticket #1211). The purpose of the ticket was to:

1. Have a unique syntax for all the nodes (before, we had several different ways to compose the names, and different characters to separate the name fields).
2. Give a scripter/developer the ability to retrieve a given node out of the native Softimage object, and viceversa. For instance, after an .ass file has been exported, a user may want to edit one of its nodes having as a reference the Softimage scene.

•

As a general rule, the names now have the following format:

Softimage Object Full Name + ".SltoA." + something where "something" is a string depending on the node type and described in more details below.

So, if you need to retrieve the Softimage object out of the Arnold node name, you can simply trim the name string up to the last occurrence of the ".SltoA." substring. Vice-versa, if you need to retrieve an exported node name out of a Softimage object, here are the details, depending on the node type:

•

Softimage Object	Arnold Node(s) ••••	Arnold Node(s) Name
Mesh	polymesh	Mesh FullName + ".SItoA." + time
Light	point_light,•distant_light, etc.	Light FullName + ".SItoA." + time
Light filter	gobo,•barndoor, etc.	Filter property FullName + ".SItoA." + time
•	image•node for the•gobo•slidemap	Filter property FullName + ".SItoA.GoboSlidemap." + time
Camera	camera	Camera plain Name, because of the camera projection shader
Shader	standard,•utility, etc.	Shader FullName + ".SItoA." + time + "." + UniqueId
Standin	procedural	Standin FullName + ".SItoA." + time
Instance	ginstance	Instance Model FullName + " " + master node name
Hair	one•curves•per chunk	Hair FullName + ".SItoA." + time + "." + chunkId
Hair by procedural	procedural	Hair FullName + ".SItoA." + time, .bin file names unchanged

ICE:

PointCloud Shape	Arnold Node(s) ••••	Arnold Node(s) Name
Points	points•(mode=disk)	PointCloud FullName + ".SItoA.Disk." + time
Sphere	points•(mode=sphere)	PointCloud FullName + ".SItoA.Sphere." + time
Disc	one•disk•per point	PointCloud FullName + ".SItoA.Disc." + time + "." + discId
Box Rectangle	one•box•per point	PointCloud FullName + ".SItoA.Box." + time + "." + boxId
Cylinder	one•cylinder•per point	PointCloud FullName + ".SItoA.Cylinder." + time + "." + cylinderId
Cone	one•cone•per point	PointCloud FullName + ".SItoA.Cone." + time + "." + coneId
Instance	one•ginstance•per point	PointCloud FullName + ".SItoA.Instance." + time + "." + instanceId + " " + master node name
Strands ••••curves	curves	PointCloud FullName + ".SItoA.Strands." + time
Shape on strands	polymesh	PointCloud FullName + ".SItoA.InstanceStrands." + time + "." + strandId

where time is the Softimage frame multiplied by 1000 and cast to integer. In cpp that's:

```
(LONG)floor(in_frame * 1000.0f + 0.5f);
```

We use spaces in just one case, that is instances, for IPR reasons. For example, if you have a cube under a Model at frame 35, the cube is exported as:

```
polymesh
{
  name Model.cube.SItoA.35000
  ...
  If this model is instanced by Model_Instance, then the instanced
  cube is exported as
  ginstance
  {
    name "Model_Instance Model.cube.SItoA.35000"
    node "Model.cube.SItoA.35000"
    ...
  }
```



So, also for ginstance nodes, it is easy to retrieve the master node and its Softimage object.

The various ID's (chunkId, disclId, etc.) are unique integers, needed when several Arnold nodes derive from a unique Softimage object. For instance, say you have a pointcloud setting the particle shape to cone. A unique Softimage object (the point cloud) is exported as a collection of Arnold cone nodes, and each must have a unique name, as required by Arnold. This is what the .ass file looks like at frame 35:

```
cone
{
  name pointcloud.SItoA.Cone.35000.0
  ...
  shader "Sources.Materials.DefaultLib.Scene_Material.Phong.SItoA.
35000.1"
  id 949
}
cone
{
  name pointcloud.SItoA.Cone.35000.1
  ...
  shader "Sources.Materials.DefaultLib.Scene_Material.Phong.SItoA.
35000.1"
  id 949
}
cone
{
  name pointcloud.SItoA.Cone.35000.2
  ...
  shader "Sources.Materials.DefaultLib.Scene_Material.Phong.SItoA.
35000.1"
  id 949
}
...
```

.

.



2.4

What's new in Version 2.4

•

SltoA version 2.4 is based on version 4.0.7.0 of the Arnold core.

New features include:

- Ray traced BSSRDFs: A new alternative to point-cloud subsurface scattering has been added. This method is brute-force ray-tracing based, but produces visually identical results to the pointcloud technique. This approach removes a number of shortcomings of the pointcloud method, trading them for unbiased noise instead. The new method consumes zero extra memory, supports motion blur, supports interactive relighting, is fully multi-threaded, starts up immediately and does not degrade in performance as the scattering radius shrinks. For backwards compatibility this is provided as an option, with point clouds being the default setting. To use ray traced subsurface scattering, check the Raytraced option from the [Sampling controls](#) dialog.

•

- Mesh light: There is a new type of light available called mesh_light which takes a polymesh as a parameter and emits light using the polymesh's surface geometry. Read the [mesh light topic](#) for more details.

•

- A number of other improvements in the Arnold core are of interest:

•

- Improved sampling at secondary bounces: The 2D sampling patterns generated by Arnold will now be stratified even for secondary bounces instead of reverting to fully random sampling as before. This can greatly reduce the amount of noise, particularly when rendering things that are seen through near-specular effects like glossy reflections or refractions

•

- Improved sampling for SSS Point cloud: When baking diffuse lighting calculations into the SSS pointcloud, Arnold now uses MIS (multiple importance sampling) to reduce noise from large nearby area lights and the skydome. This may increase the number of rays slightly, but give higher quality sampling in those cases

•

- Improved smoothed normals: The quality of the smooth shading normals stored per-vertex in triangular tessellations has been improved by adjusting the weights used to average the triangle normals around a vertex. In addition to triangle area, the weights now factor the angle between the two triangle edges containing the shared vertex. This will be most evident for coarse meshes where the face normal and the smoothed vertex normals substantially differ.

•

Note:

We have finally removed support for Softimage 2010 in this release.

•

•

2.3

What's new Version 2.3

SlttoA version 2.3 is based on version 4.0.6.0 of the Arnold core.

Highlighted new features:

- A new property, [Arnold User Options](#), has been added. This is a general purpose property, consisting of a string, and flag to 'mute' (ignore) the property. If enabled, the string can be set to override any attribute of a core Arnold node. This can be useful for accessing new features in newer Arnold core releases even before they are fully supported natively in the SlttoA UI. The property can be applied poly-meshes, hair, lights, cameras and point clouds.
- New spherical camera which helps to create lat-long environment maps more easily.
- Support for oriented curves for ICE strands. If no instance shape is in place, you can tell the curves primitive to orient the ribbons along the strand orientation directions. To do so, apply an Arnold Property to the pointcloud, and set the Curves Mode to Oriented Ribbon (ICE Strands).
- Export standins with all their materials and displacement for Exocortex Alembic. The Exocortex website has more details about [Alembic for Arnold](#).
- The Stanford Polygonal format (.ply) has been added to the supported file extensions of standins (.ply support was added to Arnold core in 4.0.6).
- Added an ID mode to the utility shader to visualize a different color for each object which, unlike the obj mode which is based on the object name string, is guaranteed to not change through an animation.

•

New commands:

The following new commands have been added:

- SlttoA_ExportObjects
- SlttoA_AddUserOptionsProperties

•

See the [SlttoA Commands](#) section for details.

•

Migration Issues:

Note that the `SlttoA_ExportScene` command now no longer has a parameter `toRender`, this has been removed.

•

•

2.2

What's new in Version 2.2

Based on version 4.0.4.0 of the Arnold core.

Highlighted new features:

- Softimage 2013 support.
- Preview standin in the viewport. See the [Slt to A Viewer](#) section for details.
- Per-face displacement support (previously we only supported one displacement shader tree per object).
- Custom procedurals can be driven by data from ICE. See details [here](#).
- Support for ICE materials system in Softimage 2012 and later.
- Custom user parameters on standins/procedural. See details [here](#).
- Several new shaders ported (txt3d-marble, txt3d_checkerboard).
- Gamma Workflow in the sib_image_clip shader. Check the notes at ticket#506 comment:28.

•

Incompatible changes:

Because displacement is now handled per-face, some of the parameters that previously were assigned in the material level (in `sta_displacement`) have been moved to the Arnold Parameters PPG, per object. This has the effect that displacement parameters from older scenes will break and have to be reapplied individually.

•

•



2.1

What's new in Version 2.1

Based on version 4.0.2.0 of the Arnold core.

Highlighted new features:

- **SSS Skin Shader** added. •
- ***_Passthrough** shaders added. •
- **Color Lookup map** shader with texture map support added. •
- **ICE attributes** on instanced lights. See Implementation details. •
- **Ability to embed a DataWindow** when saving EXR files. It stores a tight bounding box around non-empty pixels in the image, which can greatly accelerate compositing operations in Nuke as it will be interpreted in as a ROI (Region Of Interest). •
- **Subframe deformation blur** on ICE point clouds. •

•

Incompatible changes:

- **Displacement Height** no longer affects the displaced mesh's bounding box (i.e. the displacement "padding"). Now it only sets the scale for displacement. **Displacement Padding** must be set separately. If **Displacement Padding** is not set, you may get clipping artifacts in meshes with big displacements.

•

•



2.0

What's new in Version 2.0

Version 2.0 of SltoA is based on version 4.0.1 of the Arnold core, and as such includes a number of important changes.

•

License:

As the first version digit of the Arnold core changed, your license will need to be upgraded to remove the watermark. Contact us to obtain a new license file using the "licensing@..." email address.

•

Significant new features include:

••Transparency support for AOV. It is now possible to automatically compose through semi-opaque surfaces rendered using auto-transparency. You can enable this option (turned off by default because there is an associated performance hit in the Arnold core which will be improved for future releases) in the output tab of the renderer settings.

••The Hair shader now exposes `direct_diffuse`, `direct_specular` and `indirect_diffuse` AOVs.

••ICE attributes on polymesh objects. Constant, per polygon and varying data are supported. It works only with Softimage 2012 and newer. The supported attribute types are: `boolean`, `integer`, `float`, `vector3` and `rgba`. Instance, 2d vectors etc. are not exported.

••Implemented the new handle based texture lookups in the image shader node. This can give around 1.2 times speedups in texture/autobump heavy scenes and in simple scenes with very long filenames the speedup was closer to 2 times.

••Smooth tangents for subdivision surfaces option. This feature only works with subdivided objects and allows to have Ward anisotropy with a non-faceted look (the object needs UV).

•

Incompatible changes:

There are some incompatible changes introduced with Arnold 4.0.1.0. This will make older scenes to render different than in previous versions.

••Now lights have a decay control, which has been removed from the Light Decay filter. By default lights use a physically correct quadratic decay and linear and cubic light decay types have been removed.

••Decay clamp and Decay radius have therefore been removed from the Light Decay filter.

••The default value of the `specular_roughness` parameter in the Standard shader has changed from 0.467138 to 0.466905

•

•Notes:

••Remember that now the shaders and the procedurals search path obey rules specified on the wiki [here](#).

••In this version, only AOVs of type RGB support transparency.



Getting started

This section describes how to install SltA and begin using Arnold in Softimage.

-

Topics in this section:

- [Arnold](#)
- [Installation](#)
- [Updating SltA](#)
- [Rendering your first scene](#)
- [Licensing Arnold](#)

-

-



Arnold

Arnold is an advanced cross-platform rendering library, or API, used by a number of prominent organizations in film, television, and animation, including Sony Pictures Imageworks. It was developed as a photo-realistic, physically-based ray tracing alternative to traditional scanline based rendering software for CG animation.

Arnold uses cutting-edge algorithms that make the most effective use of your computer's hardware resources: memory, disk space, multiple processor cores, and SIMD/SSE units.

The Arnold architecture was designed to easily adapt to existing pipelines. It is built on top of a pluggable node system; users can extend and customize the system by writing new shaders, cameras, filters, and output driver nodes, as well as procedural geometry, custom ray types and user-defined geometric data. The primary goal of the Arnold architecture is to provide a complete solution as a primary renderer for animation and visual effects. However, Arnold can also be used as:

- A ray server for traditional scanline renderers.
- A tool for baking/procedural generation of lighting data (lightmaps for videogames).
- An interactive rendering and relighting tool.

Why is Arnold different?

Arnold is a highly optimized, unbiased, physically-based 'Monte Carlo' ray/path tracing engine. It doesn't use caching algorithms that introduce artifacts like photon mapping and final gather. It is designed to efficiently render the increasingly complex images demanded by animation and visual effects facilities while simplifying the pipeline, infrastructure requirements and user experience.

Arnold provides interactive feedback, often avoiding the need for many render passes and allowing you to match on-set lighting more efficiently. By removing many of the frustrating elements of other renderers, Arnold fits better with your work-flow, produces beautiful, predictable and bias-free results, and puts the fun back into rendering!

What is wrong with algorithms like photon mapping or final gather?

Such algorithms attempt to cache data that can be re-sampled later, to speed up rendering. However, in doing so, they use up large amounts of memory, introduce intermediate steps that break interactivity, and introduce bias into the sampling that causes visual artifacts. They also require artists to understand the details of how these algorithms work to correctly choose various control settings to get any speed up at all without ruining the render. Worse than that, these settings are almost always affected by other things in the scene, so it's often possible to accidentally use settings for the cache creation/use that make things worse, not better, or that work fine in one situation but are terrible in another, seemingly similar, situation.

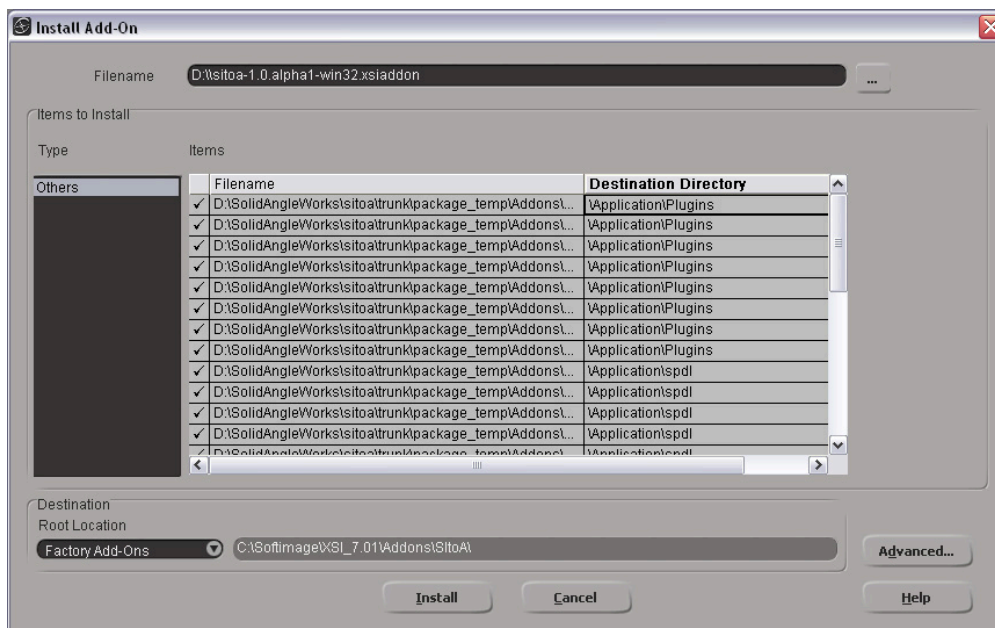
In short, they are not predictable, other than for very experienced users, and require artists to learn way too much about the algorithms to gain any benefit. We believe that your time is more valuable than your computer's time; why spend an extra 30 minutes working with photon mapping or final gather settings, even if it saves 30 minutes render time (and more often than not it doesn't). That's still 30 minutes not spent modeling, animating or lighting.

What are the advantages of being physically-based? Does this have implications for modeling and lighting scenes?

The advantage of being physically-based is that artists can work in a physically accurate, high dynamic range work-flow, and by using plausible values (being consistent about how you set lighting intensities and modeling scale) the end results will be more predictable. The process becomes much more akin to how a real scene is lit and photographed. It also ensures that other aspects of rendering are not broken.

Installation

The SitoA plug-in is distributed as a Softimage add-on. To install it, simply download the correct SitoA pack and unzip the file. Then in Softimage, select the File->Add-on->Install menu option and point it at the SitoA add-on (.xsiaddon) file. Then follow the steps in the wizard:



So here you install all of the data (Factory, User, Workgroup). It will then install all of the files that are required to execute the Arnold Renderer.

Please refer to the Softimage documentation for other ways to install Softimage add-ons.

RUNNING SITO A on LINUX

Starting from SitoA 3.6.0 (which uses Arnold 4.2.6.0), the minimum requirements have been raised to the equivalent of RHEL/CentOS 6 for glibc and libstdc++.

However, Softimage ships with a legacy libstdc++ that is put in the LD_LIBRARY_PATH and that hijacks the system one, making it impossible to load the Arnold DSO.

To fix this, one solution is to preload the system libstdc++ before launching Softimage:

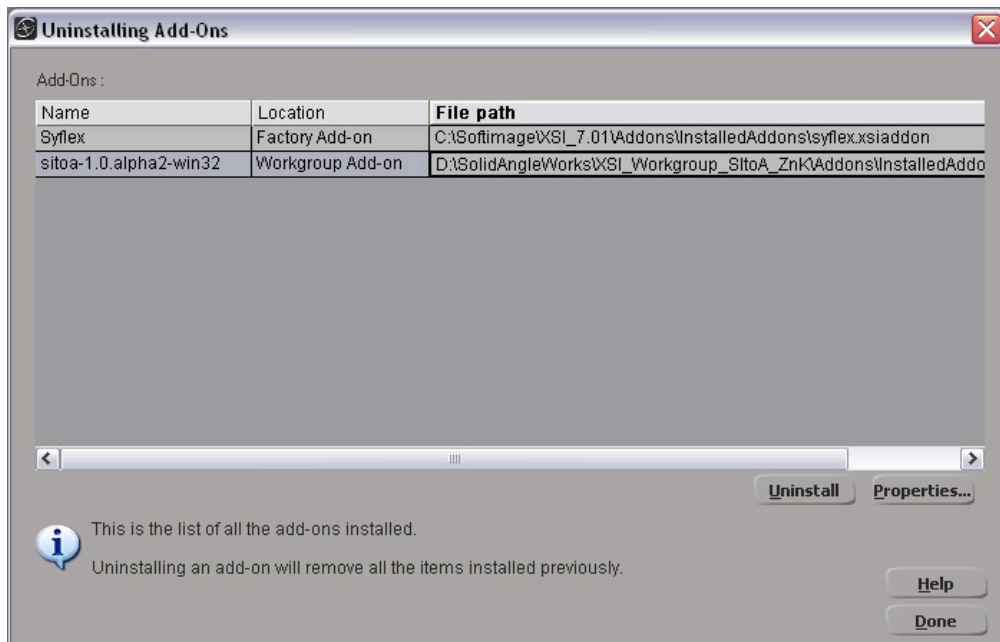
```
export LD_PRELOAD=/usr/lib64/libstdc++.so.6
```

Another solution is to stow away the libstdc++ from the Softimage install and let the system one take over, for example:

```
cd my_softimage_install/Application/bin
mv libstdc++.so.6 libstdc++.so.6.bak
mv libstdc++.so.6.0.10 libstdc++.so.6.0.10.bak
```

Updating SItOA

Uninstall the previous version of SItOA from the File->Add-on->Uninstall menu option in Softimage:



After the uninstall process is completed, and before installing the new version, make sure there is no SItOA folder in the Add-ons location. If a user is running SItOA while another user is uninstalling it, the Add-on may not be completely removed from the workgroup, leaving things in a corrupted state.

There is also an Arnold Renderer Custom preferences preset file stored in the folder;

\\Users\YourUsername\Autodesk\Softimage_Version\Data\Preferences

which is not deleted by uninstalling the add-on. Make sure you delete the **Arnold Render.Preset** file here. Also, the changes done in the Custom preferences are stored in the **default.xsipref** from the same folder. Open it in a text editor and make sure there are no Arnold entries (if there are, delete those lines).

Finally, you can install the new SItOA version (see the [Installing SItOA](#) section).

If loading existing scenes, be sure to run the **SITOA_RegenerateRenderOptions()** command from the script editor, or from the Arnold top menu after opening a scene. This will ensure the update of the PPGs, adding new parameters or removal of old parameters etc. in a scene created with a previous version of SItOA (also see the [SItOA Commands](#) section).

To summarize:

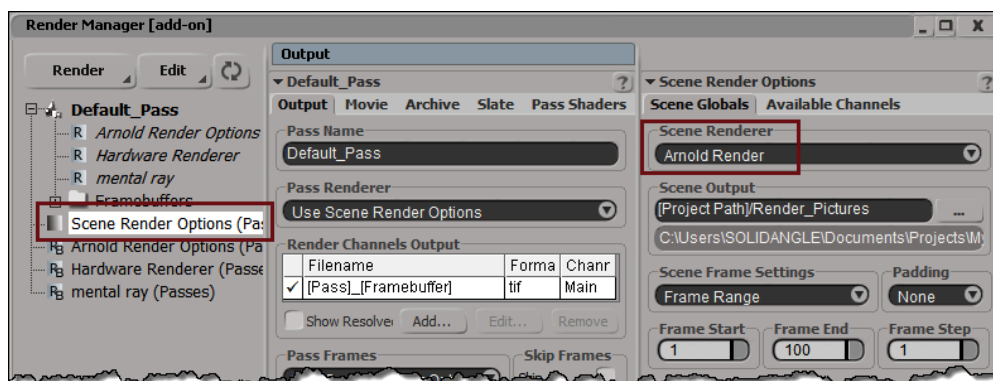
1. Uninstall the old SItOA (File->Add-on->Uninstall).
2. Check there is no SItOA folder in the Add-ons location.
3. Install the new SItOA (File->Add-on->Install).
4. Run **SITOA_RegenerateRenderOptions()**

Rendering your first scene

In this tutorial, we'll cover some of the basics of working with Arnold in Softimage. It is assumed that you have already [installed the SlttoA addon in a workgroup](#), and (optionally) set up your Arnold license.

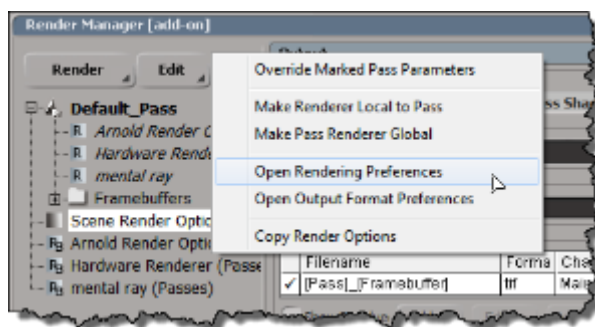
Changing the Scene Renderer

- The first thing to do is to make Arnold the scene render. Open the Render Manager and in the Scene Render Options, select Arnold Renderer as the Scene Renderer.



You will have to do this for any scene you open that was using a different renderer.

If you want Arnold to be the default scene renderer for new scenes, then in the Render Manager explorer, click Edit > Open Rendering Preferences and click Arnold Renderer in the Rendering Engine list.

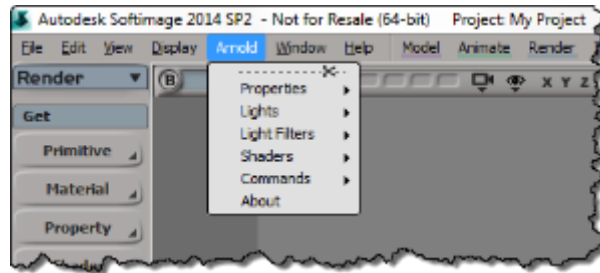


When Arnold is the default renderer, the SlttoA addon automatically sets up new scenes to work with Arnold:

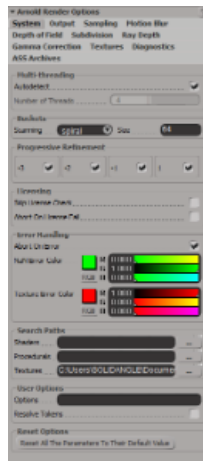
- Create a default Arnold light.
- Add the Arnold render channels (also known as AOVs in Arnold).
- Make sure the render regions use the Arnold Renderer.

Working with Arnold in Softimage

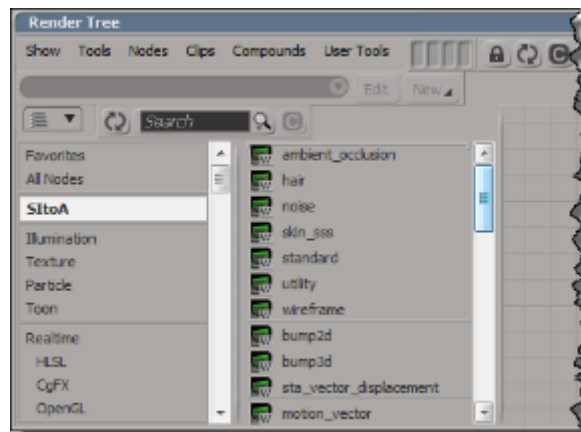
The Arnold menu gives you quick access to Arnold [lights](#), [shaders](#), [properties](#), and [commands](#).



The [Arnold Render Options](#) control render settings at the scene or pass level. You can access the Arnold render options in the Render Manager , or by clicking Render > Renderer Options in the Render toolbar.



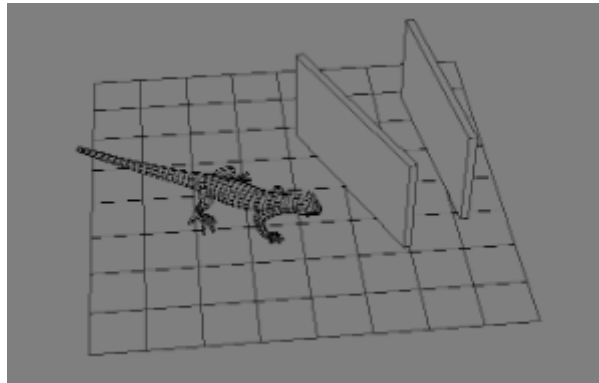
In the Render Tree preset manager , the [Arnold shaders](#) appear in the [SitoA](#) category. SitoA also [supports](#) most of the built-in Softimage shaders (see [here for the complete list of supported Softimage shaders](#)).



Setting up Arnold Shading and Lighting

Start a New Scene

Let's create a simple scene and set up some Arnold lights and shaders.



Open the script editor (press Alt+4), copy and paste this VBScript, and click  (Run) to create a new scene with a floor, a lizard, and a couple of cubes.

NewScene

```
'SetValue "Passes.RenderOptions.Renderer", "Arnold Render"
CreatePrim "Grid", "MeshSurface"
Scale , 2, 2, 2, siRelative, siLocal, siObj, siXYZ, , , , , , , 0

GetPresetModel "Lizard", "Lizard"
Translate , -2, 0, 1, siRelative, siLocal, siObj, siXYZ, True, , , , , , 0
Rotate , 0, 60, 0, siRelative, siAdd, siObj, siXYZ, , , , , , , 0
Scale , 2, 2, 2, siRelative, siLocal, siObj, siXYZ, , , , , , , 0

CreatePrim "Cube", "MeshSurface"
Scale , .04, .5, 1, siRelative, siLocal, siObj, siXYZ, , , , , , , 0
Rotate , 0, 25, 0, siRelative, siAdd, siObj, siXYZ, , , , , , , 0
Translate , 2, 2, 1, siRelative, siLocal, siObj, siXYZ, True, , , , , , 0

Duplicate "cube", , 2, 1, 1, 0, 0, 1, 0, 1, , , , , , , 0
Translate , 4, 0, 0, siRelative, siLocal, siObj, siX, True, , , , , , 0
Rotate , 0, -17.5655573983635, 0, siRelative, siAdd, siObj, siXYZ, , , , , , , 0
```

If you made Arnold the default scene renderer, then your new scene will have an Arnold distant light. And since SltOA includes an implementation of the Phong shader, which is the default shader for new objects in Softimage, you could now draw a render region and see some results.

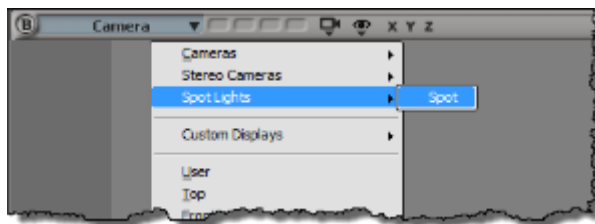


Before you continue, make sure that Arnold is the scene renderer.

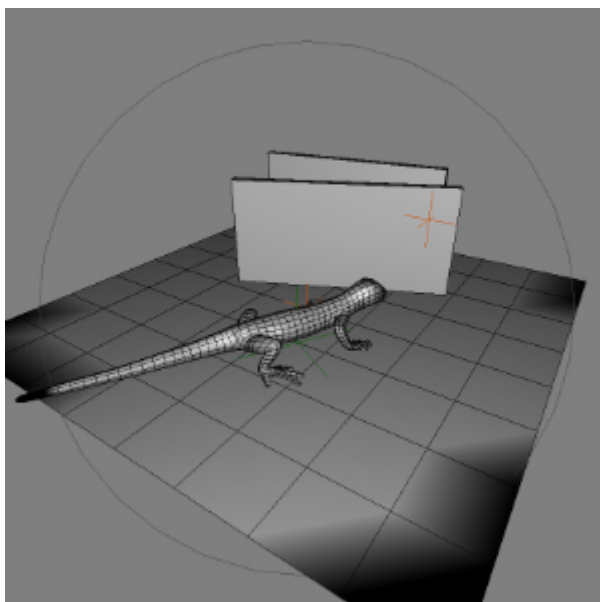
Set up a Spot Light

By default, a new scene has a distant light (if Arnold is the default scene renderer) or an infinite light (if mental ray is the default). We'll replace that default light with an [Arnold spot light](#).

1. Open the Material Manager, and change the default Scene_Material to use an Arnold [Standard Surface](#) shader instead of a Phong. The Standard Surface shader is a multi-purpose shader capable of producing all types of materials, from simple plastic to car paint or skin, with effects like sub-surface scattering and transmission.
2. Delete the default light.
3. On the render toolbar, click Modify > Ambience , and turn the global ambience down to 0,0,0.
4. In the main menu bar, click Arnold > Lights > Spot .
5. Switch to a spotlight view



and then navigate to a view like this:

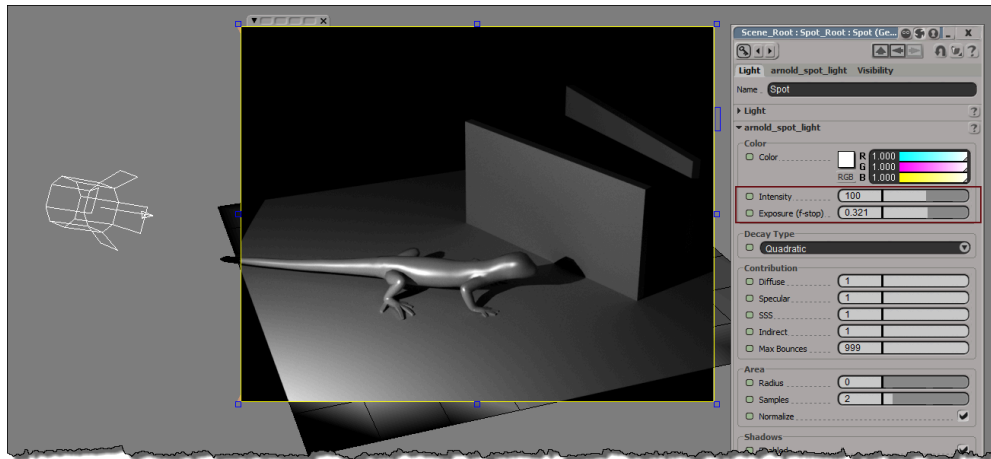


The Spot light should be positioned at about roughly -15,5,5. You can always use the Transform panel in the MPC to enter a specific XYZ position for the light.

1. Middle-click the [Views menu](#) to switch back to the Camera view.
2. Draw a render region.

The render is probably totally black, or at least very dark unless you positioned the Spot quite close to the objects. That's because, by default, Arnold uses physically-correct quadratic decay for lights. Quadratic decay follows the inverse-square law, so an object (of the same size) twice as far away, receives only one-quarter the light.

3. Open the property page for the spot light, and adjust the Intensity and [Exposure](#) values to get more light.



In Arnold, the total intensity of the light is computed with the following formula: $\text{color} * \text{intensity} * 2\text{exposure}$. You can get exactly the same output by changing either the intensity or the exposure. For example, intensity=1, exposure=4 is the same as intensity=16, exposure=0. Increasing the exposure by 1 results in double the amount of light.

1. To get softer shadows, increase the [Radius](#) (under the Area group).

With a non-zero value for radius, the light behaves more like a spherical source. The larger the radius, the softer the lighting from this light source (zero radius gives hard shadows).

Set up object shading

Now, we'll apply [Standard shaders](#) to each object in the scene.

Applying the Arnold Standard Surface Shader

First, let's do the floor (the grid):

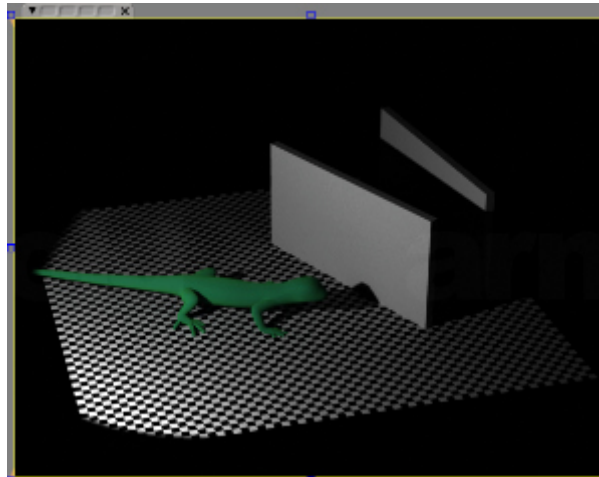
1. Select the grid, and then click Arnold > Shaders > Standard Surface. This creates a new material that uses the Standard shader.
2. Depending on your Softimage settings, the Standard Surface shader property page (PPG) may (or may not) pop up automatically. If the PPG doesn't pop up automatically, click Render > Modify > Shader.
3. Connect a Checker Board shader to the Base Color. The Check Board shader is one of the [Softimage shaders supported by Arnold](#).

If you use an unsupported shader, such as Flagstone or Fractal, then when you render you'll see an "unable to load XXX from the Arnold plugins" error like this:

[sitoa]: Unable to load bd3d_flagstone_v3 from the Arnold plugins (first occurrence: Sources.Materials.DefaultLib.Material1.Flagstone)

Now, let's do the lizard:

1. Select the lizard and middle-click the Arnold menu (to repeat the last command and apply a Standard shader).
2. Adjust the [Base Color](#) to give the lizard a greenish hue.



•

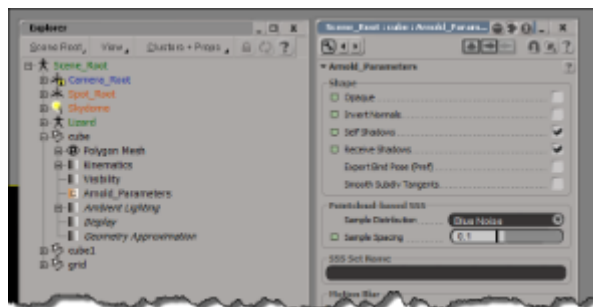
Finally, let's assign [Standard Surface](#) shaders to the two cubes:

1. Select cube and middle-click the Arnold menu (to repeat the last command and apply a Standard Surface shader).
2. Select cube1 and middle-click the Arnold menu.

Opacity

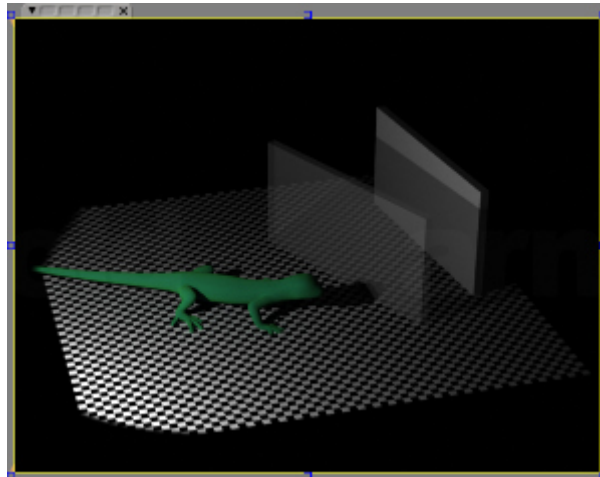
Make one cube transparent:

1. Select the cube and in the Arnold menu, click Properties > Parameters to add an [Arnold Parameters](#) property to the cube.
2. In the Arnold Parameters PPG, clear the [Opaque](#) check box. By default, all objects are opaque (non-transparent), so we need to turn this off if we want to make an object transparent using opacity.



•

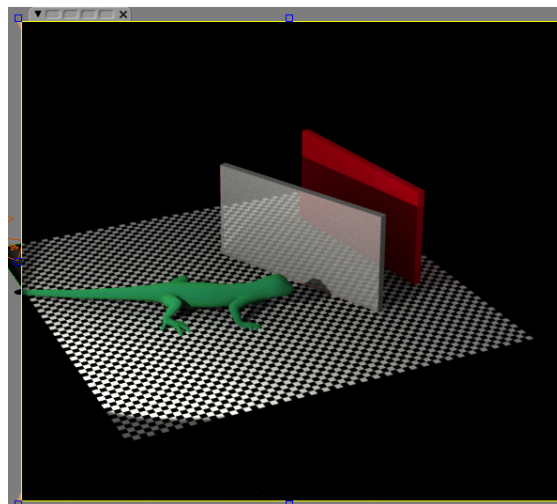
1. Open the Standard Surface shader PPG for cube (either by clicking Render > Modify > Shader or by pressing 7 to open the Render Tree).
2. In the [Geometry](#) tab of the Standard Surface shader PPG, change the Opacity Color to something like 0.2, 0.2, 0.2.



•

Transmission

You can use [Transmission](#) instead, and leave the opacity at 1,1,1 and the Opaque check box selected (which will give you faster renders). You can read more about Transmission and Opacity [here](#).



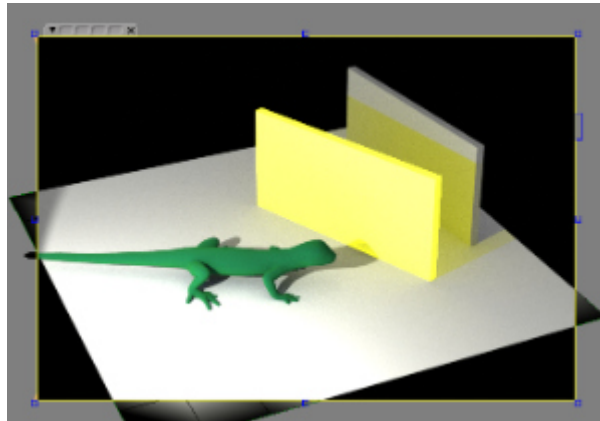
For the above example, I set Opaque = on in the Arnold Parameters of the cube. Then, in the Standard Surface shader applied to cube, I set the following:

- Base color = 0.7, 0.7, 0.7
- Transmission Color = 0.7, 0.7, 0.7
- Transmission Depth = 1
- IOR = 1.5
- Opacity = 1,1,1

I also applied a Standard Surface shader to the second cube, and made it red so it stood out a bit more.

If I wanted colored shadows, I would set Opaque = off in the Arnold Parameters of the cube. Then the shadows would render with the refraction color.

•



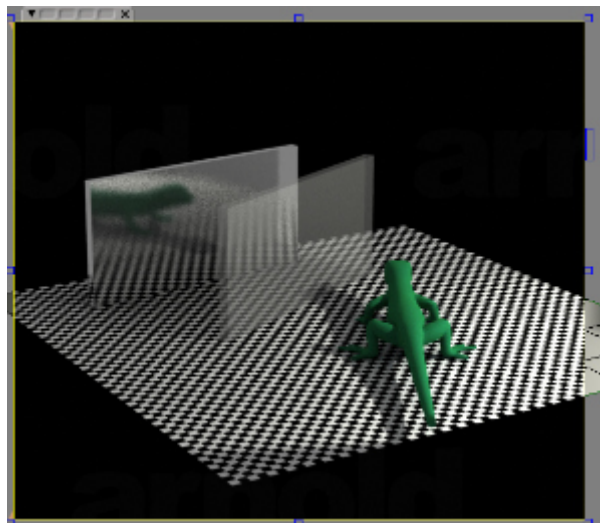
•

Specular Reflection

Let's make the second cube more specular, to get some reflections of the lizard. Specular is direct and indirect reflections which can be made glossy (blurred).

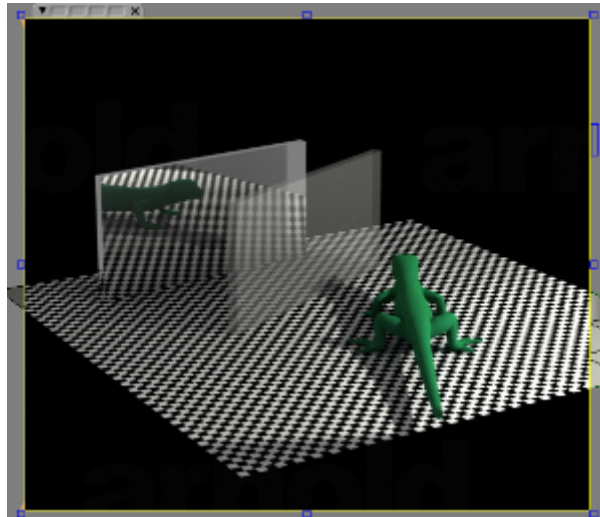
1. Select cube1 and open the Standard Surface shader PPG (either by clicking Render > Modify > Shader or by pressing 7 to open the Render Tree).
2. In the Specular group, increase the Weight to 1, and in the Base group, decrease the Weight to 0.

Because the Specular Roughness is nonzero, you will get a glossy reflection:



•

1. To get a mirror-like reflection, set the Roughness to 0:





Licensing Arnold

- If you would like to use Arnold in production you can subscribe or [buy a license](#).
- With a valid license key installed, the Arnold watermark will disappear from your rendered images.
- Solid Angle issues floating licenses that require an [RLM license server](#).

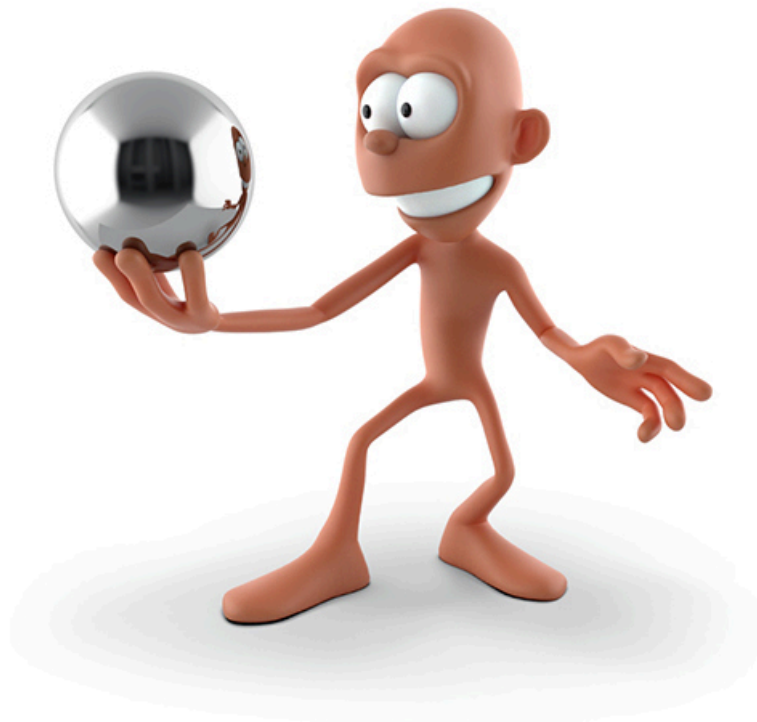
•

Tutorials

Here are the Arnold for Softimage tutorials that cover the following topics:

- [Understanding Physically Based Rendering in Arnold](#)
- [AOV Tutorials](#)
- [Kick](#)
- [Lighting](#)
- [Shading](#)
- [Rendering](#)
- [Removing Noise](#)

Understanding Physically Based Rendering in Arnold

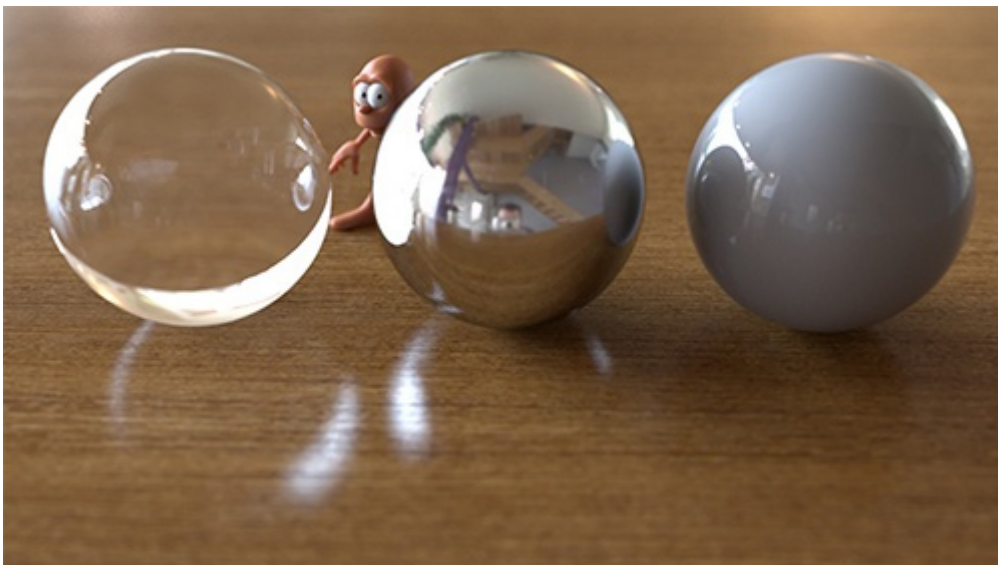


Designing materials based on physical laws can tremendously simplify shading and lighting, even when we do not necessarily strive for realism or physical accuracy. By understanding and applying a few principles, we can make images that are more believable, and create materials that behave more predictably in different lighting setups.

•

In modern renderers, physically based rendering refers to concepts like energy conservation, physically plausible scattering and layering in materials and linear color spaces. Arnold is a physically based renderer, but it also lets you break the rules and create materials and lights that do not obey the laws of physics if you wish. In this document, we'll explain the underlying theory and how to set up your shaders to follow these principles.

•



Arnold supports various third party applications such as [Substance Painter](#).

Photons and Scattering

In rendering we simulate photons emitted from lights, traveling through the air and bouncing off surfaces and through volumes, eventually ending up on a camera sensor. The combination of millions of photons on the camera sensor then forms the rendered image.

This means that from a physics point of view, surface shaders describe how the surface interacts with photons. Photons hitting an object can be absorbed, reflect off the surface, refract through the surface, or scatter around inside the object. The combination of these components results in a wide variety of materials.

Energy Conservation

Unless an object is a light source that emits photons, it can't return more energy than is being contributed by the incoming light. For a material to be energy conserving the number of photons leaving the surface should be smaller or equal to the number of incoming photons. If a material is not energy conserving, materials will appear overly bright and render with increased noise, especially when using global illumination.

To keep materials energy conserving, the weight and color of material components should never exceed 1. Further, we must be careful to ensure that the combination of all components is energy conserving, which we'll explain in detail later.



Materials

At the microscopic level, object surfaces are intricately detailed. For rendering, we do not use geometry to represent all of this detail, but rather use statistical models that have easy to understand parameters.

Arnold's `Standard` `Surface` shader models objects with one or two specular layers, and a diffuse or transparent interior. This model can represent a wide variety of materials. Let's look at the individual components.

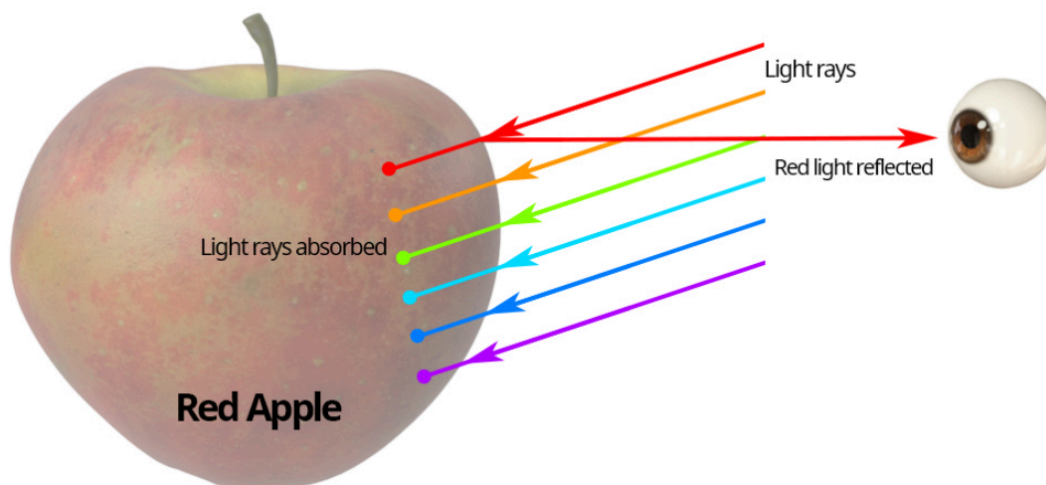
Diffuse and Subsurface Scattering

First, consider the diffuse interior. Incoming photons will enter the object, scatter around inside and either get absorbed or leave the object at another location.

If photons scatter many times, we get a diffuse appearance, due to photons leaving the surface in many different locations and directions. For materials like skin, photons can scatter relatively far under the surface giving a very soft appearance, which we render with `subsurface scattering`. For materials like unfinished wood, photons do not scatter very far which gives a harder appearance, and we render these as `diffuse`. For thin objects like leaves, the photons can scatter all the way to the other side of the object, which we render as `diffuseSSS` with `thin_wall` enabled.

Note that fundamentally all of these types of materials have the same underlying physical mechanism, even though we provide separate controls for them in the shader.

The diffuse interior also typically has the biggest influence on the overall color of the material. Each photon has an associated wavelength, and depending on the properties of the material some photons with some wavelengths are more likely to be absorbed than others. This, in turn, means that photons with some wavelengths are more likely to leave the surface, which will give it a colored appearance.



The skin of a red apple mostly reflects red light. Only the red wavelengths are scattered back outside the apple skin, and the others are absorbed by it.

•

Energy Conservation

A single photon can only participate in one of the diffuse, subsurface scattering and backlighting components, for physical correctness we do not want more photons leaving the surface than entering. For Standard Surface, it is automatically ensured that the sum of these components is not higher than 1.

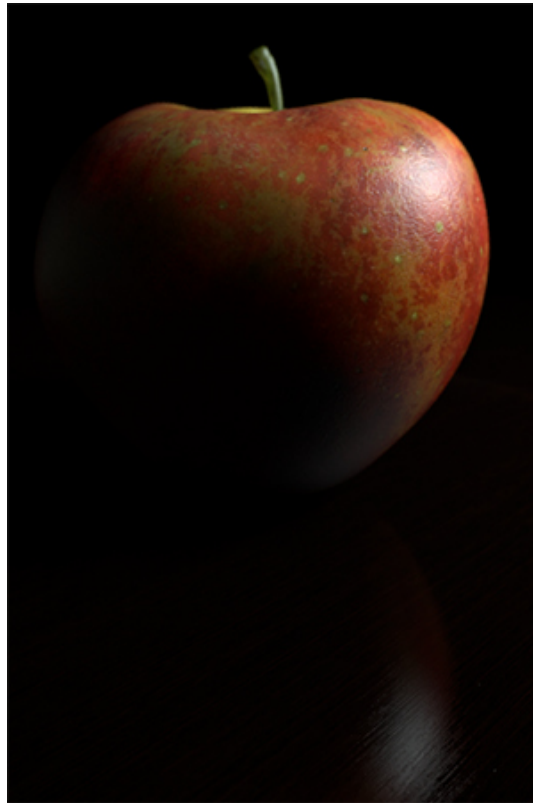
Specular Scattering



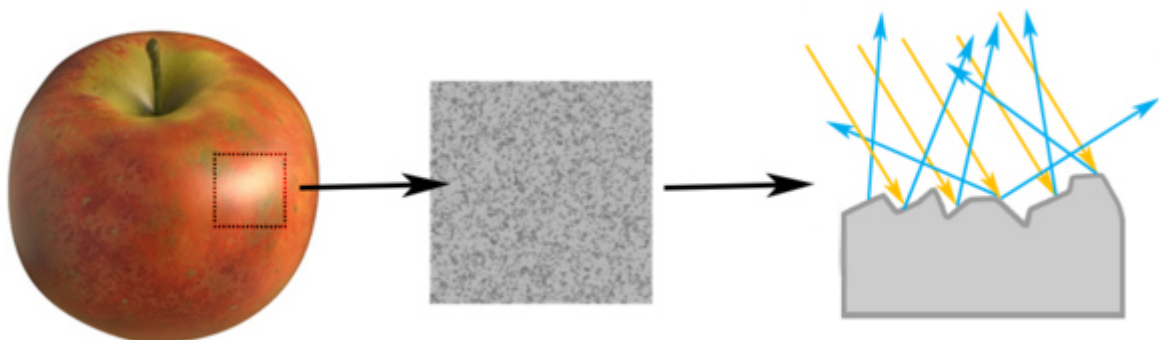
Specular Roughness 0 to 1

Roughness

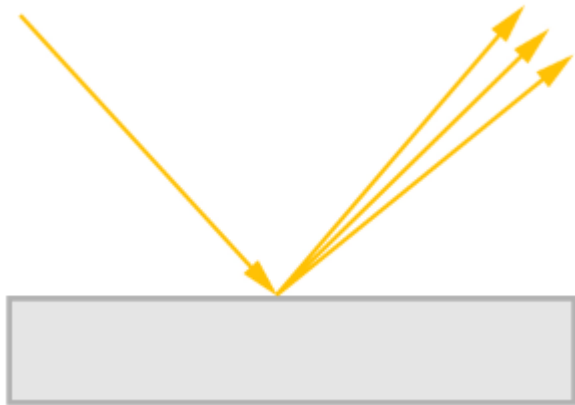
The specular layer is modeled using a microfacet distribution. We assume that the surface consists of microscopic faces oriented in random directions. A surface with low roughness such as a mirror will have little variation between the faces, resulting in sharp reflections. With high roughness there will be a lot of variation resulting in softer, glossy reflections.



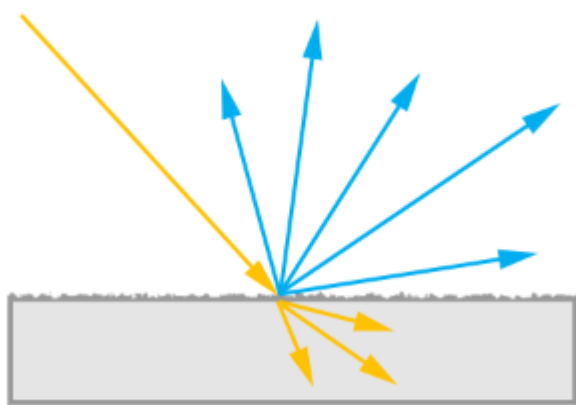
A strong Specular highlight is visible on the apple. Note the table's specular reflection which is broad and dull (high Specular Roughness value).



Rough reflections caused by scattered light rays



Glossy surface. Angle of incidence and reflection are equal.



Diffuse surface. Ray direction varies randomly.

Roughness Map

To get variation in the highlights of the surface, a map should be connected to the Specular Roughness. This will influence not only the brightness of the highlight but also its size and the sharpness of the environmental reflection.



Low Specular Roughness



High Specular Roughness

'Scratches' texture connected to Specular Roughness (via Range shader)



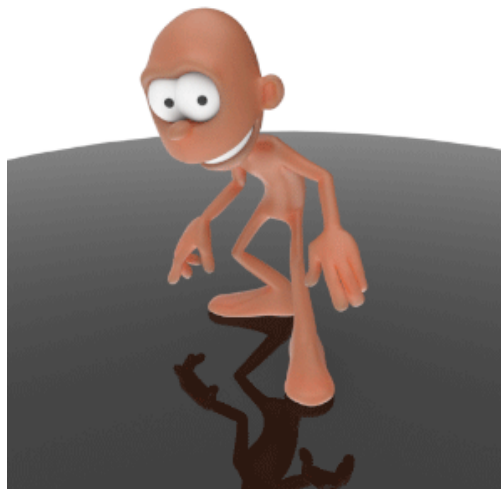
Transmission

Photons can not only be reflected off the surface, but can refract through it as well. Photons will pass through the specular layer, typically changing direction when exiting on the other side of the layer, controlled by the index of refraction (IOR).

If the interior of the surface is transparent, such as for clear glass, then photons can pass through the object and exit on the other side. If there is a diffuse interior, the photon can scatter inside the object and get absorbed or exit the object again. The more refractive the specular layer, the more the underlying diffuse interior will be visible. For materials like metals, photons refracting through the specular are often immediately absorbed, and so the diffuse interior is not visible.

Fresnel

The percentage of photons reflected or refracted by the specular layer is view dependent. When looking at surfaces head on, most light is refracted, while at grazing angles most light is reflected. This is called the Fresnel effect. The index of refraction controls exactly how this effect varies with the viewing angle.



Variation of a Specular BRDF with respect to the view direction

Opacity and Transmission

Opacity is best understood as a way to model surface geometry using textures. It does not affect how photons interact with the surface, but rather indicates where the surface's geometry is absent and the photons can pass straight through.



Ramp texture connected to the opacity

-

A typical use for opacity would be a sprite type of effect, such as cutting out the shape of a leaf from a polygon card or making the tips of hair strands transparent. Be warned however that scenes containing many opacity sprites (for example tree leaves) can slow down rendering considerably.*



Leaf Opacity: Enabled



Leaf Opacity: Disabled

-

Transmission depth is similar, but rather than the surface it controls the density of the object interior. Denser volumes will absorb more photons as they pass through the interior, making the object darker where it is thicker.



Transmission Color: White



Transmission Color: light blue

-
-
-



-
-

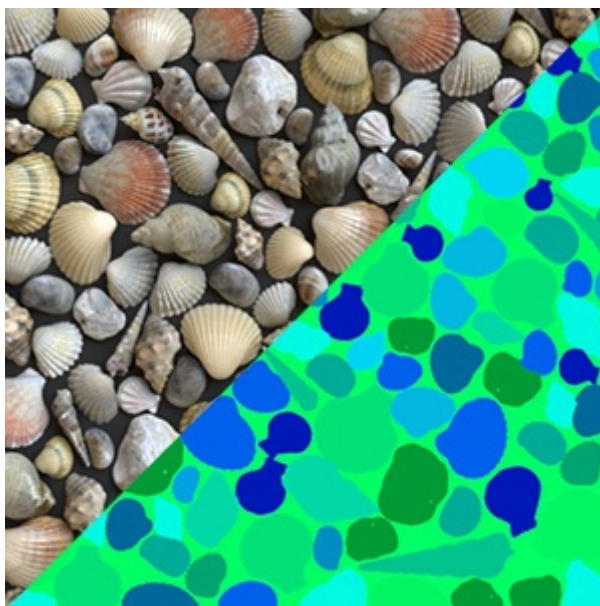
Pepe model by Daniel M. Lara([Pepeland](#)).

AOV Tutorials

The following [AOV](#) tutorials are available for SlttoA:

- [Cryptomatte](#)

Cryptomatte



Beauty AOV / Cryptomatte material AOV

Arnold supports Cryptomatte, an ID matte creation tool created by Jonah Friedman. It creates ID mattes automatically with support for motion blur, transparency, and depth of field. Names, object namespaces, and material names can be used to organize ID mattes in the scene.

The following Cryptomatte AOVs are available under AOVs.

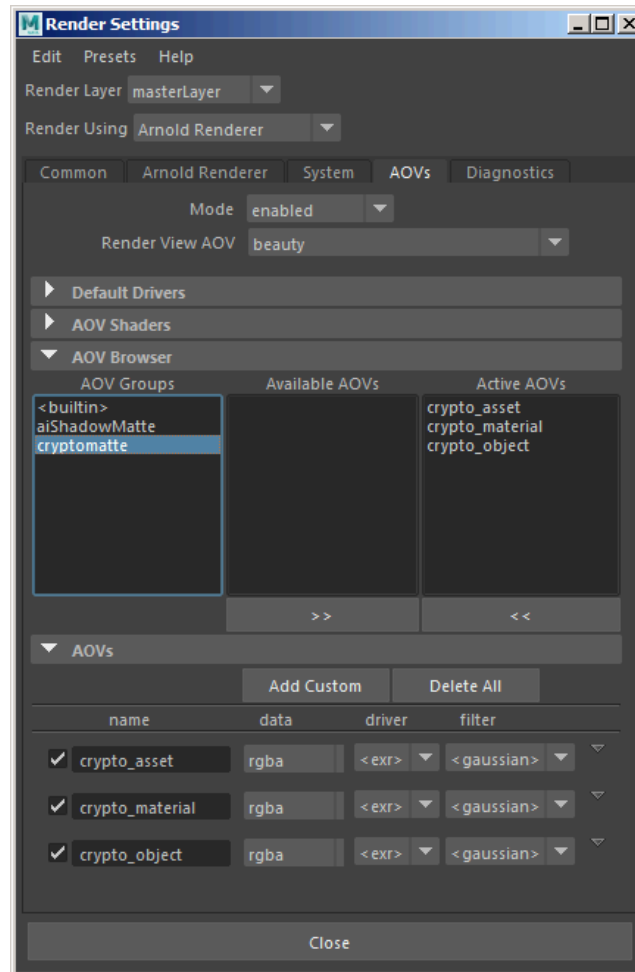
- `crypto_asset` creates the same matte for all assets.
 - `crypto_material` creates a matte based on the assigned shader.
 - `crypto_object` creates a matte based on the object name.
- Copy them to the shaders folder of the MAXtoA plugin. This would typically be somewhere like this:

```
C:\solidangle\mtoadeploy\2017\shaders\cryptomatte.dll
```

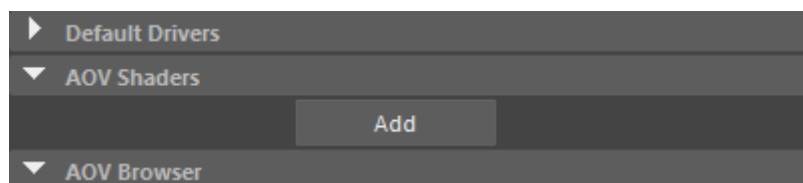
```
C:\solidangle\mtoadeploy\2017\shaders\cryptomatte.mtd
```

The following cryptomatte AOVs should then be available in the AOVs Settings in the Render Setup window.

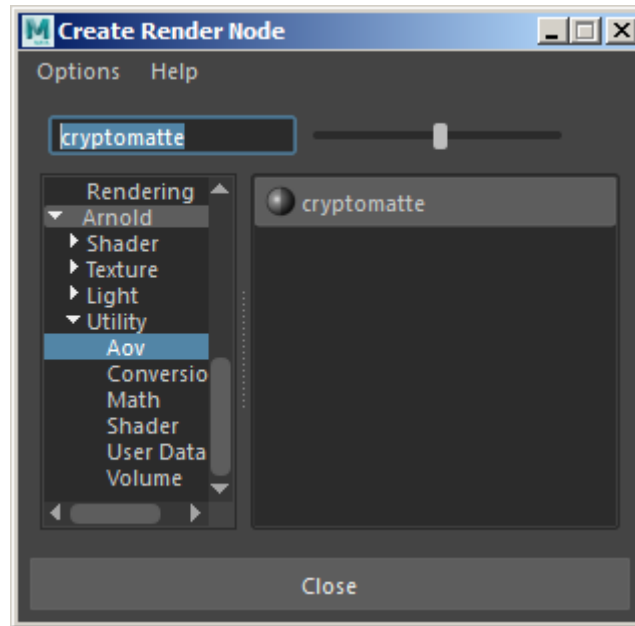
- `crypto_asset` creates the same matte for all assets.
 - `crypto_material` creates a matte based on the assigned shader.
 - `crypto_object` creates a matte based on the object name.
- Create the cryptomatte AOVs that you require in the AOV Browser in the AOVs tab of the Render Settings window.



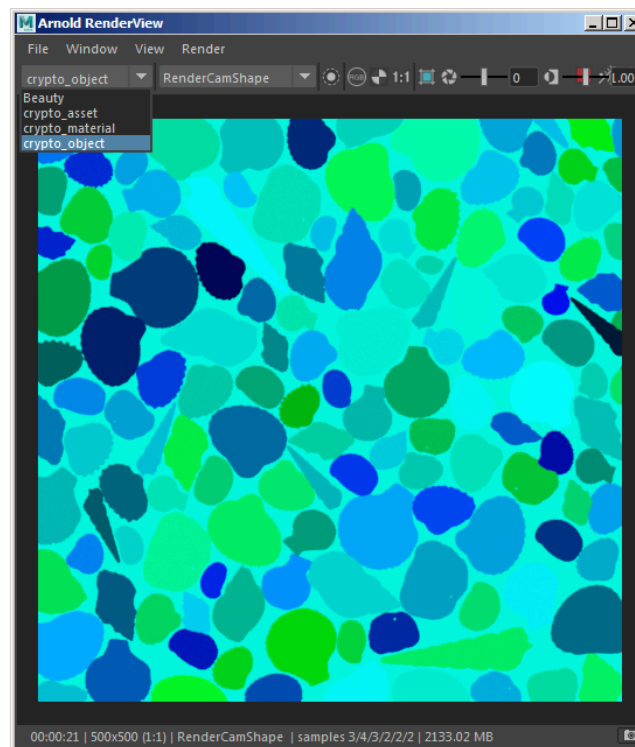
- In the AOV Shaders section (above the AOV Browser), select Add .•



- In the Create Render Node window, go to the section Arnold->Utility->AOV and you should see the cryptomatte shader.



- Render the scene. The cryptomatte AOVs should now be available in the [Arnold RenderView](#) window.



Kick

The following tutorials show you how to render an `.ass` file using Kick.

-

- [Getting Started With Kick](#)
- [How to Read a Render Log](#)

Getting Started With Kick

Introduction

Arnold's command-line renderer is called kick. This reads an [.ass](#) and [.usd](#) file, renders the scene using Arnold, and outputs an image file. Kick can also be used to query Arnold nodes for their parameters and default values. It can also be used for scene debugging.

Kick is included in the plug-in downloads. After installation, you will find it in the bin subdirectory of the Arnold distribution.

Running Kick

To run kick, first, open a shell (terminal). Here's an example on Windows OS.



Kick always loads shaders and procedurals from the current directory, so don't run kick in a folder with lots of other DLLs/SOs/DYLIBs. Kick will try to load each DLL/SO/DYLIB, to check whether it contains shaders or procedurals.

```
$ set ARNOLD_BIN_PATH=C:\solidangle\Arnold-5.0.0.0\bin
$ %ARNOLD_BIN_PATH%\kick
Arnold 5.0.0.0 [2cfbe09c] linux clang-3.9.1 oiio-1.7.12 osl-1.8.2
vdb-4.0.0 rlm-12.2.2 2017/04/10 16:48:44
No arguments. Try kick --help for a command summary
```

Useful Commands

To try out the following commands download one of the files like [cornell.ass](#) from the [example ass files](#) page.

One of the most useful commands is "-h" or "--help". This will display a list of all the available options in kick:

```
kick -h
```

Use the "-i" option to read an .ass file and render it:

```
kick -i path/to/cornell.ass
```

Note that the "-i" option is not strictly necessary, kick automatically recognizes arguments ending in .ass, so this works too:

```
kick path/to/cornell.ass
```

By default, a window will pop up displaying the image as it's being rendered. You can turn off the display window with the '-dw' option:

```
kick cornell.ass -dw
```

If the geometry is colored pink, it could be because the path to the shaders may be required. This can be added using the "-l" flag:

```
kick cornell.ass -l /path/to/plugin/shaders/
```

Log information is sent to stdout. You can increase or decrease the log verbosity with the "-v <n>" option (default verbosity is 1). The most verbose option is "-v 6":



```
kick cornell.ass -v 2
```

Switch off the log output with "-v 0":

```
kick cornell.ass -v 0
```

To save the rendered image in an output file use the "-o" flag:

```
kick cornell.ass -o cornell.exr
```

Change the render display size with the "-r <width> <height>" option:

```
kick cornell.ass -r 1024 720
```

Print the Arnold version number, or the entire version string:

```
kick -av  
kick --version
```

Print diagnostic information about the license servers and list installed licenses available and in use:

```
kick -licensecheck
```

To override the antialiasing samples:

```
kick cornell.ass -as 3
```

To override the diffuse GI samples:

```
kick cornell.ass -ds 3
```

To disable the progressive refinement mode:

```
kick cornell.ass -dp
```

For debugging purposes you can globally disable several features like textures, lights, and shaders, motion blur, subdivs, displacement or SSS:



```
kick cornell.ass -it
kick cornell.ass -il
kick cornell.ass -is
kick cornell.ass -imb -isd -idisp -isss
```

You can install custom Arnold nodes by loading them from a dynamic library (.dll or .so) with:

```
kick cornell.ass -l path\to\plugin -l path\to\more\plugins
```

You can also get a list of all the installed nodes (both built-in and dynamically-loaded) with:

```
kick -nodes
kick -l path\to\plugins -nodes
```

You can inspect nodes with "-info":

```
kick -info polymesh
kick -info options
kick -l path\to\plugins -info custom_plugin_node
```

Or get more information about a given parameter:

```
kick -info polymesh.sidedness
kick -info options.bucket_scanning
```

•Override any parameter of any node using the "-set" command:•

```
kick cornell.ass -set options.AA_samples 3
```

•

Override any parameter of all nodes of a particular type:

```
kick cornell.ass -set curves.mode thick
```

Get kick to abort the render if no valid license is found, rather than rendering with a watermark:

```
kick cornell.ass -set options.abort_on_license_fail true
```

Override several parameters:



```
kick first.ass -set options.AA_samples 3 -set options.bucket_size  
16
```

Interactive Mode

To render in interactive mode use the "-ipr q" option. This will allow you to (very crudely) navigate the scene and switch to various debug shading modes such as flat/smooth normals, UVs, wireframe, etc.:

```
kick cornell.ass -ipr q
```

- Navigate (pan/zoom) with the chosen interaction mode: "q" for Quake controls (WASD), and "m" for Maya controls (Alt + Mouse).
- Increase or decrease the image exposure with the "[" and "]" keys.
- Click in the render window and press lowercase "i" to ignore any existing shaders. You can restore the scene shaders by pressing uppercase "I"
- macOS and Linux : use the number keys (0,1,2,3,4,5,6,7,8,9) to switch between the various debug shading modes.

How to Read a Render Log

The Render Log prints the progress messages, warnings and errors to the console or a file as kick renders an image. As well as showing the percentage completed, these detailed statistics can be used to optimize and debug scenes.

 The Arnold log provides detailed statistics that are useful for debugging, optimizing and benchmarking renders. It is the first thing to examine should you encounter errors and usually the first information to send to support.

 When encountering problems with renders, the first thing you should do with a log file is search for the word "ERROR". Most of the time you will find the source of the problem in that ERROR message.

The verbosity levels vary from kick to the plugins.

Kick	Plugins
0 - No Output	Errors
1 - Progress	Warnings + Info
2	
3	
4	Debug
5 - Detailed	
6 - Debug	

The log below contains the most common output. The first column of the log, `00:00:00`, shows the time elapsed in `hh:mm:ss`. The second column, `773MB`, shows the memory usage at that stage.

Initialization (Info)

This section shows the version of Arnold being used and its build details. It also lists the hardware specifications and operating system of the machine it was rendered on.

```
00:00:00    773MB    | log started Tue Jul 21 15:26:25 2015
00:00:00    773MB    | Arnold 4.2.7.4 windows icc-14.0.2 oiio-
1.5.15 rlm-11.2.2 2015/06/15 09:39:31
00:00:00    773MB    | host application: MtoA 1.2.3.1
03a85380bec8 (Master) MtoA-1.2.3.1 Maya 2016
00:00:00    773MB    | running on PC, pid=12188
00:00:00    773MB    | 1 x Intel(R) Xeon(R) CPU E5-1650 v2 @
3.50GHz (6 cores, 12 logical) with 32712MB
00:00:00    773MB    | Windows 7 Professional Service Pack 1
(version 6.1, build 7601)
```

There is a single initialization process for the whole render session (only on the first render), with an updated process for every following render.

```
00:00:00    773MB      | [mtoa.session]      Initializing at
frame 1.000000
00:00:00    773MB      | [mtoa] Exporting Arnold options
'defaultArnoldRenderOptions'
00:00:00    773MB      | [mtoa.extensions]  aiOptions Using
translator <built-in>, provided by <built-in>(<built-in>).
00:00:00    773MB      | [mtoa.session]
defaultArnoldRenderOptions      | Exporting plug
defaultArnoldRenderOptions.message for type aiOptions
```

Loading / Plugins

This section displays the results of importing any shaders or procedurals.

Just rendering through kick on its own shows no plugin and the ass location.

```
00:00:00    773MB      | loading plugins from . ...
00:00:00    773MB      | no plugins loaded
00:00:00    773MB      | [ass] loading project/scenes
/simplescene.ass ...
00:00:00    773MB      | [ass] read 11386 bytes, 11 nodes in 0:
00.00
```

If a plugin is being loaded (and verbosity is high enough) the shaders will be listed:

```
00:00:00    773MB      | loading plugin: C:/solidangle/mtoadeploy
/2016/shaders/mtoa_shaders.dll ...
00:00:00    773MB      | mtoa_shaders.dll: MayaMultiplyDivide
uses Arnold 4.2.7.4
00:00:00    773MB      | mtoa_shaders.dll: MayaClamp uses
Arnold 4.2.7.4
00:00:00    773MB      | mtoa_shaders.dll: MayaGammaCorrect
uses Arnold 4.2.7.4
00:00:00    773MB      | mtoa_shaders.dll: MayaCondition uses
Arnold 4.2.7.4
....
00:00:00    774MB      | mtoa_shaders.dll: volume_sample_float
uses Arnold 4.2.7.4
00:00:00    774MB      | mtoa_shaders.dll: volume_sample_rgb
uses Arnold 4.2.7.4
00:00:00    774MB      | mtoa_shaders.dll: driver_mplay uses
Arnold 4.2.7.4
00:00:00    774MB      | loaded 90 plugins from 1 lib(s) in 0:
00.00
```

License Check

If a license is available, the log will display the following.

```
00:00:00    810MB      | [rlm] checkout of "arnold 20150615" in
progress ...
00:00:00    810MB      | [rlm] checkout of "arnold 20150615"
from server PC in 0:00.01
00:00:00    810MB      | [rlm] expiration date: 31-dec-2015 (164
days left)
```

.

However, if the RLM server is not running, it will display a warning.

```
00:00:00    810MB WARNING | [rlm] could not connect to license
server on 5053@localhost
```

Color Management

```
00:00:00    56MB      |
00:00:00    63MB      | [color_manager_syncolor] Using
syncolor_color_manager Version 2018.0.80
00:00:00    63MB      |                               with the
native catalog directory from /path/to/synColor
00:00:00    63MB      |                               and the
optional custom catalog directory from /other/path/to/Shared/
```

When using the SYNCOLOR environment variable, kick will override the ass color management paths so the output would be:

```
00:00:00    63MB      | [color_manager_syncolor] Using
syncolor_color_manager Version 2018.0.80
00:00:00    63MB      |                               from the
preference file /path/to/synColorConfig.xml
00:00:00    63MB      |                               and the
optional custom catalog directory from /other/path/to/Shared/
```

Scene Contents ••

Here it will list the numbers of lights and objects (and their types) in the scene. •

```
00:00:00 818MB | there are 1 light and 2 objects:
00:00:00 818MB |     1 persp_camera
00:00:00 818MB |     1 distant_light
00:00:00 818MB |     1 utility
00:00:00 818MB |     1 lambert
00:00:00 818MB |     1 driver_exr
00:00:00 818MB |     1 gaussian_filter
00:00:00 818MB |     1 polymesh
00:00:00 818MB |     1 list_aggregate
00:00:00 818MB |     1 MayaShadingEngine
00:00:00 818MB |     1 renderview_display
00:00:00 818MB |     1 color_manager_syncolor
```

Resolution / Samples

This section displays the output resolution and details about the number of samples.

```
00:00:00 818MB | rendering image at 640 x 480, 3 AA
samples
00:00:00 818MB | AA sample clamp <disabled>
00:00:00 818MB | diffuse          samples 3 / depth
1
00:00:00 818MB | specular         samples 3 / depth
2
00:00:00 818MB | transmission     samples 1 / depth
2
00:00:00 818MB | volume indirect  <disabled by depth>
00:00:00 818MB | total            depth
10
00:00:00 818MB | bssrdf           <disabled>
00:00:00 818MB | transparency     depth 10 / fast
opacity off
```

Nodes

Before rendering starts, nodes are initialized. The stats here give insight into the objects and lights in the scene and can show warnings if anything is configured incorrectly. Note that some data is not immediately initialized, some computations are delayed until the first ray hits the object so that further information can be shown during the render.

```
00:00:00    818MB    |   initializing 11 nodes ...
00:00:00    818MB    |   creating root object list ...
00:00:00    818MB    |   scene bounds: (-0.995465517
-1.01209259 -0.995465755) -> (1.00453472 0.98790741 1.00453484)
00:00:00    818MB    |   node initialization done in 0:00.00
(single-threaded)
00:00:00    818MB    |   updating 12 nodes ...
00:00:00    818MB    |   directionalLightShape1: distant_light
using 1 sample
00:00:00    818MB    |   node update done in 0:00.00 (single-
threaded)
```

Drivers / AOVs

Here are the drivers and AOVs used in the scene, and any warnings if they are set up incorrectly.

```
00:00:00    818MB    |   [aov] parsing output statement: "RGBA
RGBA defaultArnoldFilter@gaussian_filter
defaultArnoldDisplayDriver@renderview_display"
00:00:00    818MB    |   [aov] parsing output statement: "RGBA
RGBA defaultArnoldFilter@gaussian_filter
defaultArnoldDriver@driver_exr.RGBA"
00:00:00    818MB    |   [aov] registered driver:
"defaultArnoldDisplayDriver@renderview_display"
(renderview_display)
00:00:00    818MB    |   [aov] * "RGBA" of type RGBA filtered
by "defaultArnoldFilter@gaussian_filter" (gaussian_filter)
00:00:00    818MB    |   [aov] registered driver:
"defaultArnoldDriver@driver_exr.RGBA" (driver_exr)
00:00:00    818MB    |   [aov] * "RGBA" of type RGBA filtered
by "defaultArnoldFilter@gaussian_filter" (gaussian_filter)
00:00:00    818MB    |   [aov] done preparing 1 AOV for 2
outputs to 2 drivers (0 deep AOVs)
```

Progress

Now that the render is starting, the log will show how many buckets are being used and at what size. If any importance tables need to be generated, in this case for the skydome, they will be done so now, showing the average energy value. Percentage completed is shown in 5% increments along with the average rays per pixel. This value will show at a glance if your sampling settings are too high.

```
00:00:00    825MB      | starting 12 bucket workers of size
64x64 ...
00:00:00    835MB      |      0% done - 9 rays/pixel
00:00:00    835MB      | [accel] bvh4 done - 0:00.00 - 400
prims, 1 key
00:00:00    836MB      |      5% done - 9 rays/pixel
00:00:00    836MB      |     10% done - 9 rays/pixel
00:00:00    836MB      |     15% done - 9 rays/pixel
00:00:00    837MB      |     20% done - 9 rays/pixel
00:00:00    837MB      |     25% done - 9 rays/pixel
00:00:00    837MB      |     30% done - 9 rays/pixel
00:00:00    837MB      |     35% done - 10 rays/pixel
00:00:00    837MB      |     40% done - 10 rays/pixel
00:00:00    838MB      |     45% done - 12 rays/pixel
00:00:00    838MB      |     50% done - 12 rays/pixel
00:00:00    838MB      |     55% done - 13 rays/pixel
00:00:00    838MB      |     60% done - 13 rays/pixel
00:00:00    838MB      |     65% done - 13 rays/pixel
00:00:00    838MB      |     70% done - 13 rays/pixel
00:00:00    838MB      |     75% done - 13 rays/pixel
00:00:00    838MB      |     80% done - 13 rays/pixel
00:00:00    838MB      |     85% done - 13 rays/pixel
00:00:00    838MB      |     90% done - 13 rays/pixel
00:00:00    838MB      |     95% done - 13 rays/pixel
00:00:00    838MB      |    100% done - 13 rays/pixel
00:00:00    824MB      | bucket workers done in 0:00.38
```

Output

Once the render is complete, it will be written to file (or screen) using the selected output driver. The log will show the relative path.

```
00:00:00    824MB      | [driver_exr] writing file `C:/Users
/Documents/example.exr'
00:00:00    821MB      | render done
```

Scene Creation Stats

These timings show the time taken to load plugins and .ass files.

```
00:00:00    821MB      | scene creation time:
00:00:00    821MB      | plugin loading           0:00.07
00:00:00    821MB      | system/unaccounted      0:00.12
00:00:00    821MB      | total                   0:00.12
( 1.05% machine utilization)
```

Render Time Stats

These timings show the time spent performing various tasks. Typically pixel rendering will take up most of the time if that's not the case then looking at the node initialization and geometry stats might reveal inefficiencies. Low machine utilization may indicate that other processes on the same machine are slowing down the render. But it can also indicate that a lot of time is spent performing single threaded tasks, for example in node initialization, procedurals or mesh processing. Another possibility is that there is a lot of (slow) file access, typically for textures or volumes. Looking at the texture stats might reveal problems.

```
00:00:00    821MB    | render time:
00:00:00    821MB    |   node init                      0:00.00
00:00:00    821MB    |   sanity checks                  0:00.00
00:00:00    821MB    |   bucket rendering               0:00.38
00:00:00    821MB    |   mesh processing                0:00.00
00:00:00    821MB    |   accel. building                0:00.00
00:00:00    821MB    |   pixel rendering                0:00.38
(multi-threaded render, this value may not be reliable)
00:00:00    821MB    | system/unaccounted              0:00.17
00:00:00    821MB    | total                            0:00.55
(69.23% machine utilization)
```

Memory Stats

Here memory used at startup and peak memory for various types of data are listed. When using a plugin, the startup memory shows how much memory the host application is using, when rendering from kick the startup memory is small which typically means larger scenes can be rendered. If memory usage is too high, these stats indicate which data would be most helpful to reduce.

```
00:00:00    821MB    | memory consumed in MB:
00:00:00    821MB    |   at startup                      807.25
00:00:00    821MB    |   plugins                        2.52
00:00:00    821MB    |   AOV samples                    11.67
00:00:00    821MB    |   output buffers                  5.32
00:00:00    821MB    |   node overhead                  0.00
00:00:00    821MB    |   message passing                 0.02
00:00:00    821MB    |   memory pools                   13.54
00:00:00    821MB    |   geometry                       0.01
00:00:00    821MB    |   polymesh                       0.01
00:00:00    821MB    |   accel. structs                 0.02
00:00:00    821MB    |   strings                        0.30
00:00:00    821MB    |   texture cache                  0.00
00:00:00    821MB    |   unaccounted                    1.64
00:00:00    821MB    |   total peak                     842.29
```

Ray Stats

These are the number of rays of each type, per pixel and per AA sample. If the render time is high, these numbers give insight into which types of samples are most expensive to render and would be most helpful trying to reduce.

```
00:00:00    821MB      | ray counts:                                (
/pixel , /sample) (% total) (avg. hits) (max hits)
00:00:00    821MB      | camera                                2318256
(   8.71,    1.00) ( 64.57%) (    0.11) (    1)
00:00:00    821MB      | shadow                                228076
(   0.86,    0.10) (  6.35%) (    0.00) (    0)
00:00:00    821MB      | diffuse                               1043914
(   3.92,    0.45) ( 29.08%) (    0.00) (    0)
00:00:00    821MB      | total                                3590246
(  13.48,    1.55) (100.00%) (    0.07) (    1)
00:00:00    821MB      | max depth                             1
```

Shader Stats

The number of shader calls in various contexts is shown here, which can be useful to figure out which shader calls are most helpful to speed up.

- Primary: surface shading of non-shadow rays (this is traditional shading).
- Transparent_shadow: surface shading of shadow rays when it goes through a transparent surface (needed to shade to know if the ray is blocked or can continue, and if it continues, has its color changed).
- Autobump: each time autobump is computed, it has to compute displacement.
- Background: the background shader.
- Importance: calls made while computing importance tables.
- Volume: volume shader calls.

```
00:00:00    821MB      | shader calls:                                (
/pixel , /sample) (% total)
00:00:00    821MB      | primary                                523006
(   1.96,    0.23) (100.00%)
00:00:00    821MB      | total                                523006
(   1.96,    0.23) (100.00%)
```



You can use these shader stats to improve render times. If for instance, you see lots of transparent_shadow, that is usually an indication that your light samples are very high or you have lots of transparent objects. Another example would be if autobump is very high, you could try disabling autobump on meshes that don't seem to benefit from it as much.*

Geometry Stats

These stats give insight into the amount of geometry in the scene. If memory usage is high or scene setup takes a long time, these can be used to find the objects that contribute to it most, and would benefit from being simplified or having their subdivision iterations reduced.

```
00:00:00    821MB      | geometry:
(% hit ) (instances) (  init mem,  final mem)
00:00:00    821MB      | lists                                1
(100.0%) (          0) (   0.00,    0.00)
00:00:00    821MB      | polymeshes                           1
(100.0%) (          0) (   0.02,    0.01)
00:00:00    821MB      |
-----
```


Texture Stats

Detailed statistics for image textures. These give insight into which textures use most memory, and which textures are untiled and would benefit from being converted to .tx files. The percentage of the main cache misses should be very small for good performance, if it is high render time can be significantly increased. If the main cache misses are too high, it helps to ensure only .tx files are used, reduce the number of textures, or tweak the texture settings.

00:00:00	841MB		OpenImageIO Texture statistics			
00:00:00	841MB		Queries/batches :			
00:00:00	841MB		texture	:	261503 queries in	
261503 batches						
00:00:00	841MB		texture 3d	:	0 queries in 0	
batches						
00:00:00	841MB		shadow	:	0 queries in 0	
batches						
00:00:00	841MB		environment	:	0 queries in 0	
batches						
00:00:00	841MB		Interpolations :			
00:00:00	841MB		closest	:	0	
00:00:00	841MB		bilinear	:	398256	
00:00:00	841MB		bicubic	:	701789	
00:00:00	841MB		Average anisotropic probes : 2.8			
00:00:00	841MB		Max anisotropy in the wild : 671			
00:00:00	841MB		OpenImageIO ImageCache statistics			
00:00:00	841MB		(000000003D574040) ver 1.5.15			
00:00:00	841MB		Images : 1 unique			
00:00:00	841MB		ImageInputs : 1 created, 1 current,			
1 peak						
00:00:00	841MB		Total size of all images referenced			
: 1.7 MB						
00:00:00	841MB		Read from disk : 1.7 MB			
00:00:00	841MB		File I/O time : 0.2s (0.0s average			
per thread)						
00:00:00	841MB		File open time only : 0.0s			
00:00:00	841MB		Tiles: 25 created, 24 current, 24 peak			
00:00:00	841MB		total tile requests : 1454328			
00:00:00	841MB		micro-cache misses : 35383			
(2.43294%)						
00:00:00	841MB		main cache misses : 25 (0.00171901%)			
00:00:00	841MB		Peak cache memory : 2.3 MB			
00:00:00	841MB		Image file statistics:			
00:00:00	841MB		opens	tiles	MB read	I/O
time res						
00:00:00	841MB		1	1	12	1.7
s	768x 768x3.u8					0.2
			C:/Users/Documents/example.tif UNTILED			



```
UNMIPPED MIP-COUNT [12,7,3,2,1,0,0,0,0,0]
00:00:00    841MB      |
00:00:00    841MB      | Tot:      1      12      1.7      0.2s
00:00:00    841MB      | 1 not tiled, 1 not MIP-mapped
```

Shutdown

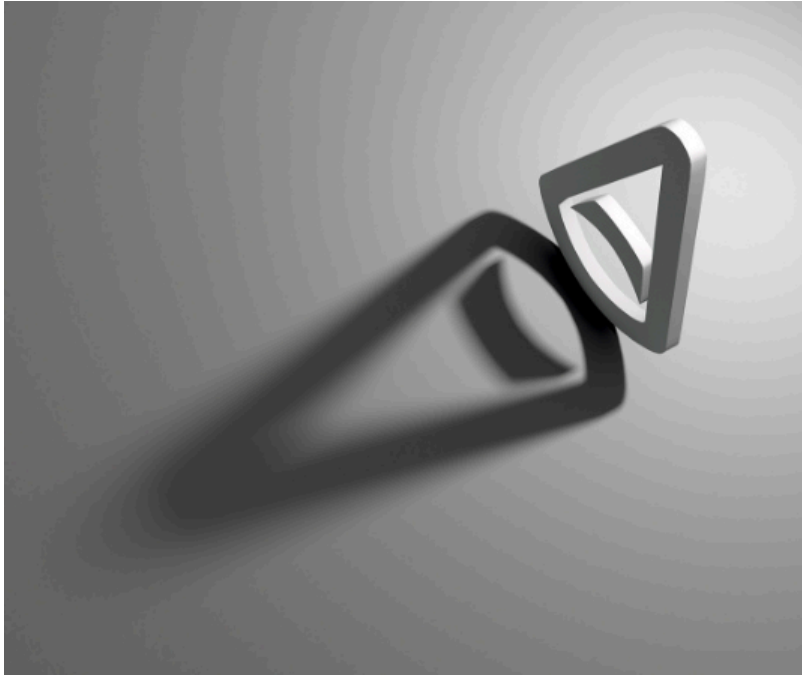
When Arnold is finished, it will release any resources, memory/threads, and shutdown.

```
00:00:00    841MB      | releasing resources
00:00:00    831MB      | unloading 27 plugins
00:00:00    829MB      | unloading plugins done
00:00:00    828MB      | Arnold shutdown
```



More information about getting started with Kick can be found [here](#).

Lighting



- [Lighting a Room](#)
- [Studio Automotive Rendering](#)
- [Studio Lighting](#)

Lighting a Room



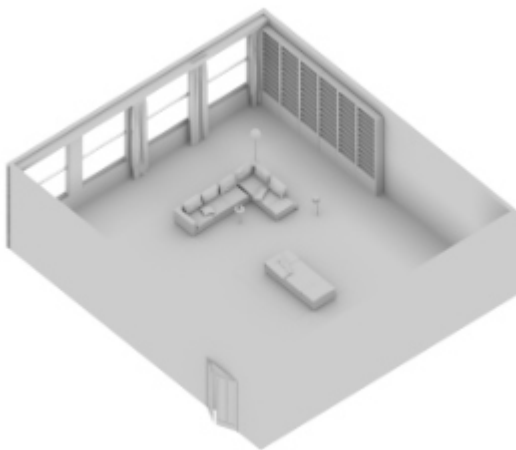
Modeling credit Giorgio Luciano

Arnold is capable of producing awesome interior renders. This tutorial shows a series of alternate ways to light an interior room from outside. We will also look at alternative ways to light the scene to reduce render times and noise.*

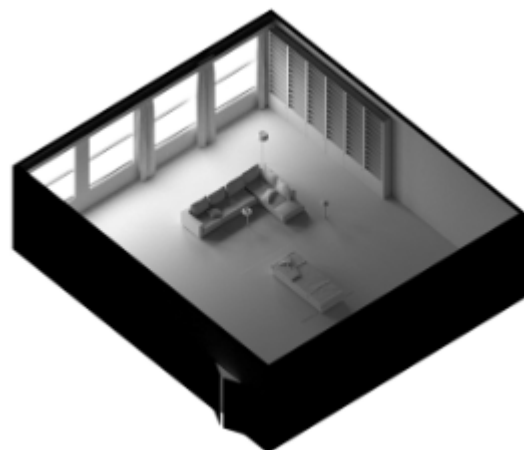
The scene file can be downloaded• [here](#) (modeling credit: Giorgio Luciano).



All of the HDR maps used in this tutorial can be downloaded•[here](#).



Ambient Occlusion



Area Lights

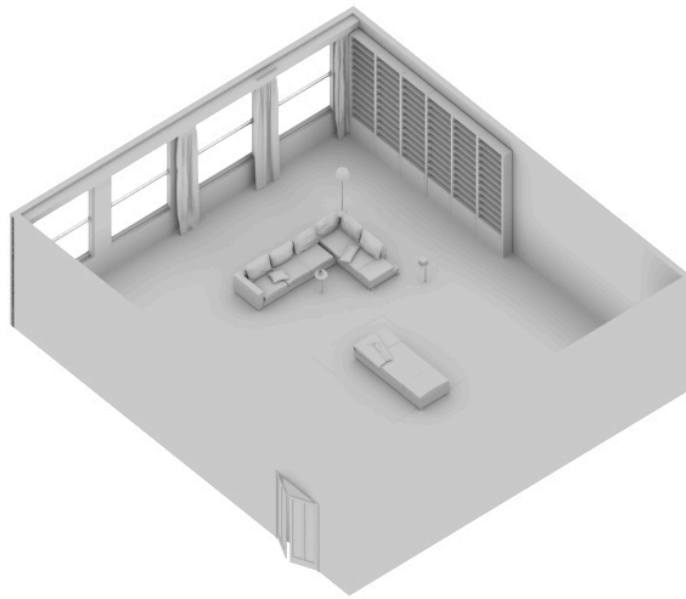
Area Lights



Ambient occlusion and direct light render combined using post-processing software

•

An alternative technique to lighting a room is to use Ambient Occlusion. A typical workflow is to composite the Ambient Occlusion render onto a Diffuse rendered image that does not have any GI Diffuse rays. The advantage is that it is typically faster to render and can give more creative control when compositing. You will also not need to add any GI Diffuse or GI Specular rays to the scene. However, it is not a physically accurate method, and some experience with compositing is required to fine-tune the final look.



Ambient Occlusion shader

- Start by creating an Ambient Occlusion shader and assign it to the room geometry. Increasing the Samples of the Ambient Occlusion shader to around 6 will reduce any noise associated with this shader.
- Render the scene using the Ambient Occlusion shader. It should look like the image below:



Rendered using only the Ambient Occlusion shader

•

Direct lighting render

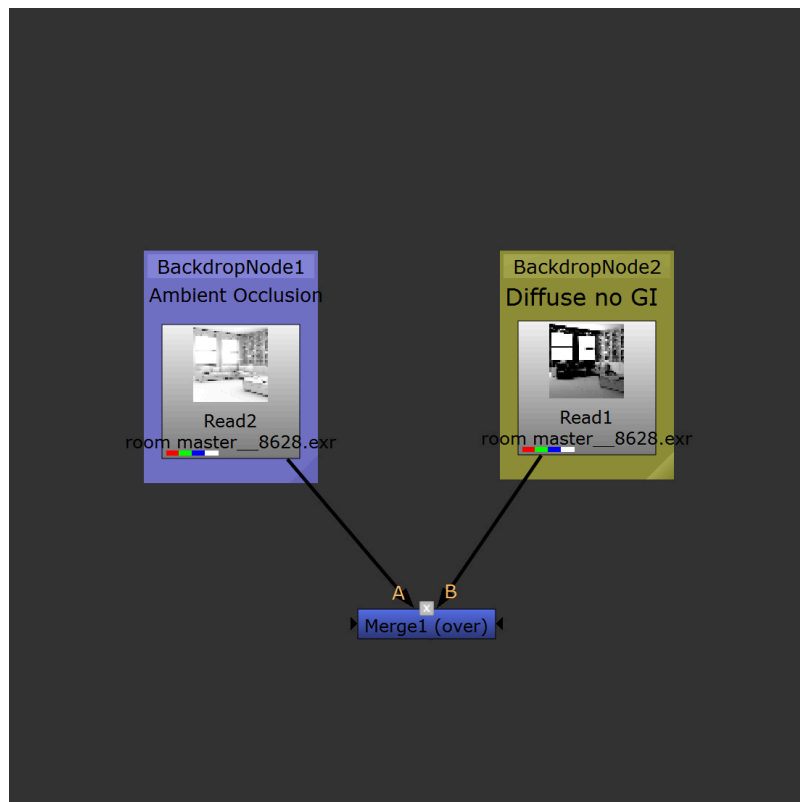
- Now render the scene using direct lighting only (in this case area lights have been positioned outside the windows, but you could try other lights such as the SkyDome light). However, in this case, ensure that GI Diffuse samples are set to 0. You should notice when you render the scene that there is no bounced light and the back wall will be black. We will use the ambient occlusion in Nuke to 'fill in' the areas that have no bounced GI Diffuse light.



Direct light (Indirect 0)

Composite

- Open both renders in Nuke. Create a **merge** node and connect the **Ambient Occlusion** render to the **'A'** input and the **Direct light** to the **'B'** of the **'merge'** node. Make sure the **'Operation'** is set to **'Over'**. Adjust the **'mix'** to around **0.5**. You should notice the Ambient Occlusion appear in the black areas of the 'Direct light' image. It should be obvious that we are in effect 'cheating' the use of bounced light using Ambient Occlusion.



Ambient occlusion render merged with diffuse render (no GI)

The images below show the results of merging (over) the 'Ambient Occlusion' render onto the diffuse lighting (without GI) render. The benefits are that it is reasonably fast to render and does not contain any GI noise.



Direct light (Indirect 0)



Ambient Occlusion render



Ambient

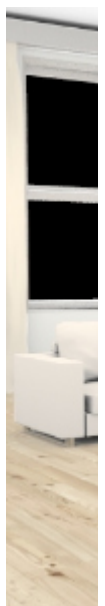
The images below show the result of rendering using a Utility shader set to flat color, merged with the ambient occlusion AOV. This renders very quickly. However the results are far from photo-realistic.



Ambient occlusion



Utility shader set to flat color



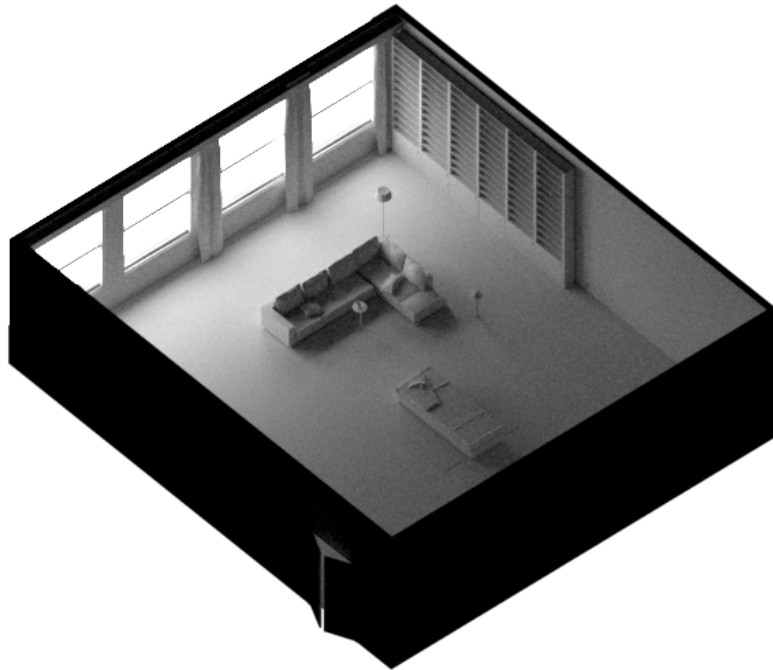
Ambie

Emissive Geometry



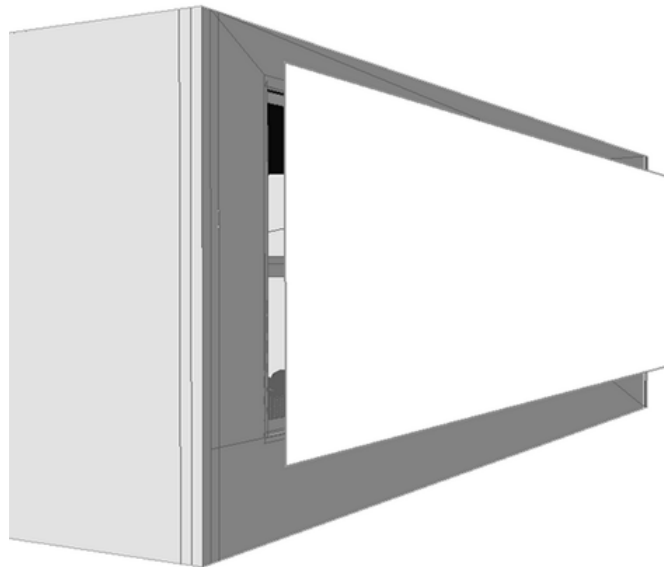
HDRI connected to emission attribute of a Standard Surface shader

It is possible for geometry to 'emit light' by adding **Emission** to a Standard Surface shader. It is, therefore, possible to light the room by placing some emissive geometry outside of the windows. However, this technique is discouraged. Using the emission from a Standard Surface shader applied to a plane is quite inefficient as it will only catch the lighting from the diffuse rays. Therefore it will be quite noisy. Textured area lights (set to quad) will always be superior in this respect.



Plane outside window with Emission Scale set to 1

- Start off by creating a polygon plane and position it so that it covers the area of the windows outside the room.

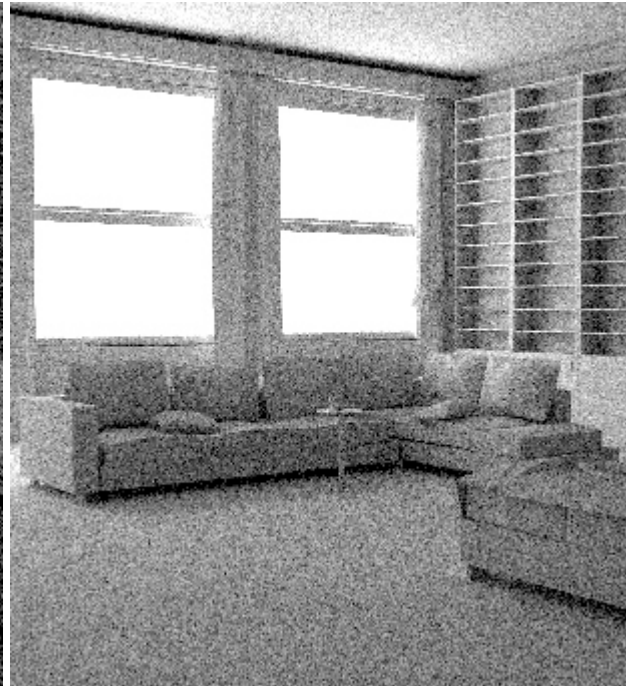


- Assign a Standard Surface shader to the plane. Increase the Emission Scale to 8. We want it to be purely emissive, so it is good practice to reduce the Base Weight value to 0.

As you can see in the images below, using this method can introduce a lot of noise. The number of Gi Diffuse Samples has to be increased a lot to reduce noise. However, this will increase render times.



Plane with Emission (Ray Depth 1).



Plane with Emission (Ray Depth 3).



Diffuse Samples 1 (3 mins 50 secs).



Diffuse Samples 6 (68 mins).

Emissive Plane with HDRI

Adding an HDRI map to the Emission Color of the Standard Surface shader is another option to light the scene. In the images below an HDRI of a view through a window has been used.



HDRI assigned to Emission Color of Standard assigned to plane outside window



HDRI in window appears blown out

The window HDRI appears blown out when assigned to the emissive plane. A way around this is to use the **Rayswitch** shader. A **Standard Surface** shader with the HDRI image is used to control **Diffuse GI** rays and another **Standard Surface** that contains the same image as a JPEG is used for the camera rays.



Rayswitch shader assigned to emissive plane geometry. The window JPEG texture is visible to camera rays, while the HDRI is visible in diffuse rays

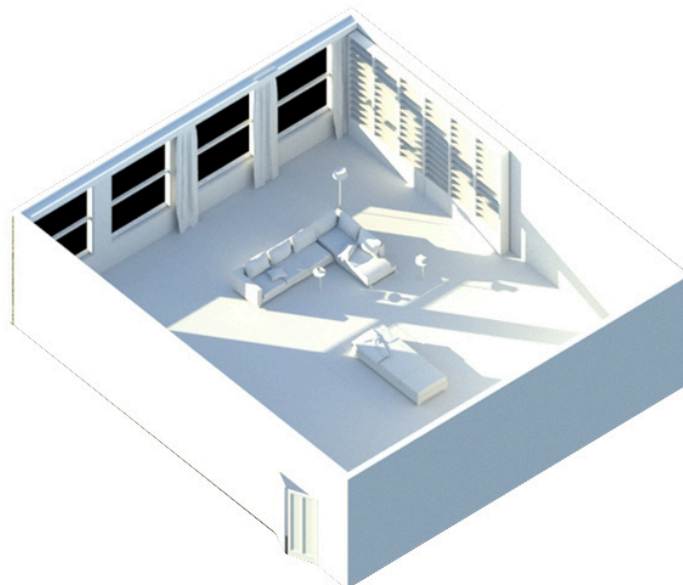
Lighting with the Physical Sky



The Physical Sky is a very easy way to get accurate lighting in your scene. Simply by adjusting the 'Elevation' and 'Azimuth' settings it is possible to create an interesting 'golden hour' lighting effect. This can be achieved very simply by going through the following steps.



It is not recommended to use a skydome_light for interior scenes. This light is designed for outdoor scenes and is represented by a spherical dome in the background. Multiple importance light sampling will trace rays to specific directions of this dome. However, in an interior scene, most of these rays will hit an object, getting no contribution from the light at all and thus creating noise. In this situation, [quad lights](#) or [directional lighting](#) would be a better option.



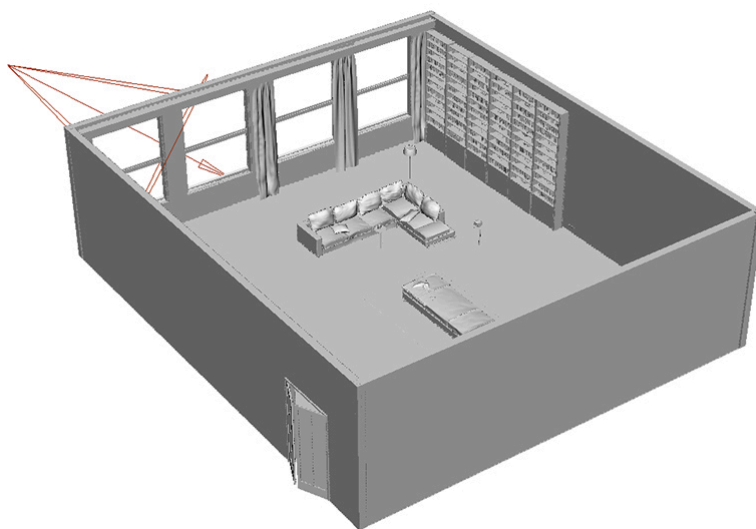
Scene lit with a Physical Sky node connected to a Sky Dome light

- Start off by creating a **Physical Sky** shader.
- Create a **SkyDomeLight** and connect the **Physical Sky** to the color attribute of the **SkyDomeLight**.
- IPR render the scene and adjust the Elevation and Azimuth settings so that the sun is shining through the window and hitting the far right wall and floor. In this case, the **Elevation** is quite low (15), and the **Azimuth** has been angled so that the sun light hits the far wall (143).

Atmosphere Volume

The SkyDome light will not contribute any volume/atmosphere scattering effect. However, we can cheat this by adding a spotlight to the scene

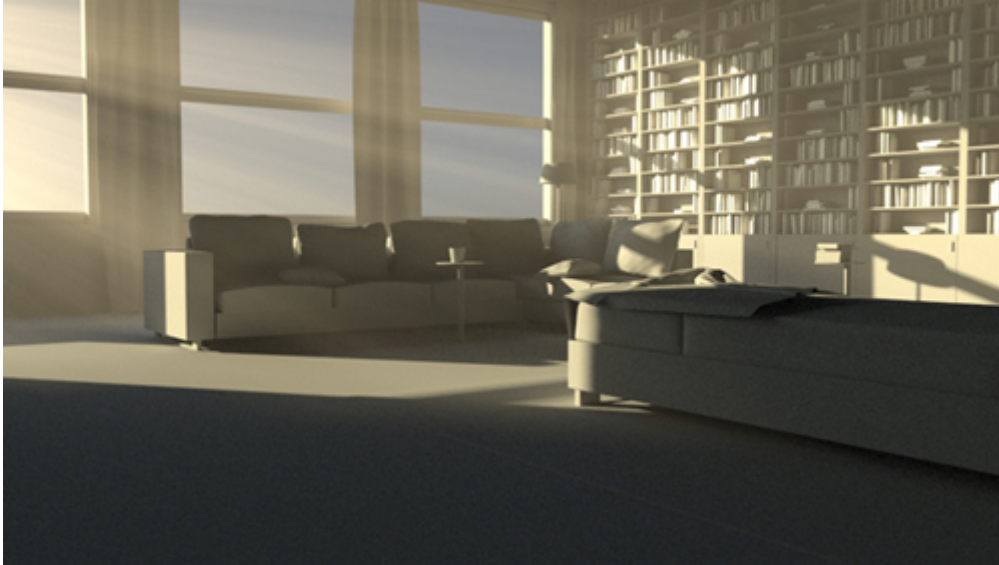
- Position a **spotlight** so that it points through the windows. Rotate it so that it matches the angle of sunlight from the Physical Sky. Be aware that we now have two lights representing sunlight in our scene. This will create two types of shadows which we do not want (the spotlight is only being used to create a volume scattering effect). If this effect is apparent, you could try to disable 'enable_sun' in the physical_sky shader and use the spotlight to simulate the direct sunlight.



Spotlight pointed through window

To achieve a 'god ray' effect, we can add a **Gobo** filter to the spot light.

- Select the spotlight and connect a **Gobo** filter to it.
- Select the gobo and connect a **noise map** to the **Slide Map**.
- Enable **Atmosphere Volume** and increase the **Density** to 1.
- Finally, change the **Color Temperature** of the spotlight to around 4500 to get a nice warm sun light effect through the windows.



Spotlight with Atmosphere Volume added to scene

-
-



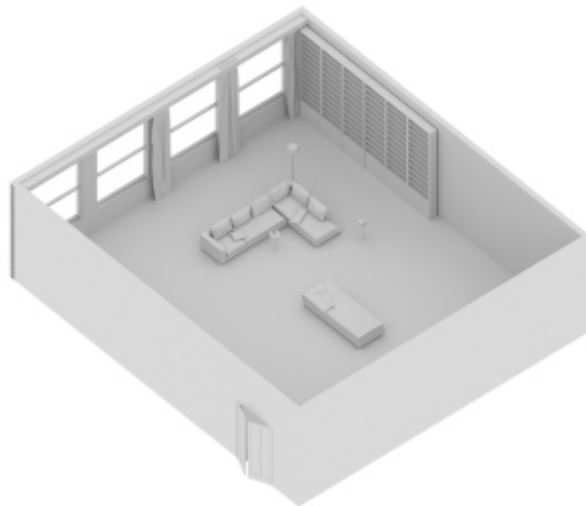
Further example of an interior lit with a Physical Sky connected to a Sky Dome light

Lighting with the Sky

! The sky shader is deprecated. You should use the skydome light instead.



One of the simplest ways to light the interior would be to add a **Sky** to the scene. Hiding the ceiling geometry allows light from the sky to fill the room. Whilst this is a quick and easy way to light the interior, it does not look particularly realistic.



Room lit with a Sky (ceiling removed)

The images below show the effect of adding a white Sky to the scene. The room is being lit solely by the Sky.

There is a lot of noise in the indirectly lit areas of the room. Increasing the (GI) Diffuse Samples resolves the noise, but also adds more to the render times.



Diffuse Samples 1 (42 secs)



Diffuse Samples 2 (2 mins 06 secs)

Increasing the GI Diffuse Ray depth allows more light to bounce around the interior. Care must be taken when increasing this value as render times will increase.



Diffuse Depth 1



Diffuse Depth 2

- The scene lacks any directional sun light. Create a distant light and rotate it so that it points downwards towards the floor and the bookcase as in the image below.



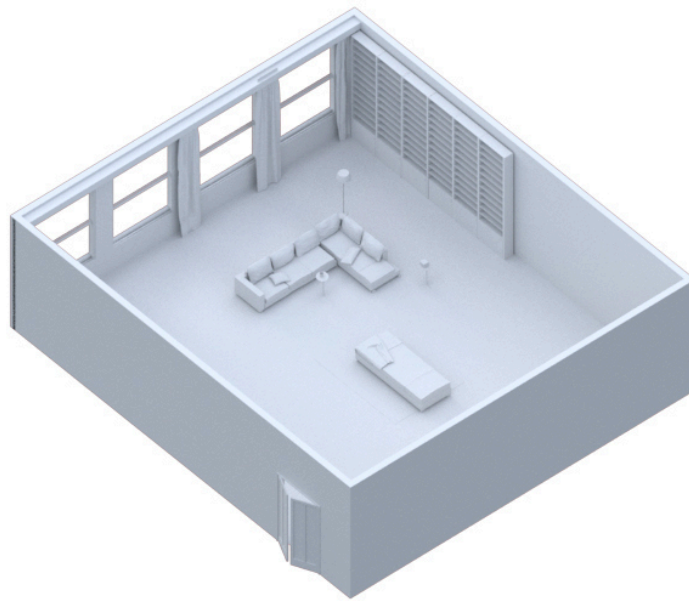
Lighting with the Sky Dome Light



We can also use the sky_dome_light to light the interior. However, this light is designed for outdoor scenes that require illumination from multiple directions. It assumes a spherical dome in the horizon of the scene and the objects will trace rays to sample the light. In an interior most of these rays will hit an object, getting no contribution at all.



This light is designed for outdoor scenes and is represented by a spherical dome in the background. Multiple importance light sampling will trace rays to specific directions of this dome. However, in an interior scene, most of these rays will hit an object, getting no contribution from the light at all and thus creating noise. In this situation, adding light portals to the windows will help to reduce noise in an interior scene when using Skydome lighting.



- Create a Sky Dome Light.

The following images show the Sky Dome Light with the Color set to white. With the Sky Dome light samples set to 3 and the GI Diffuse samples set to 8, we get a cleaner image.



Diffuse Depth 1



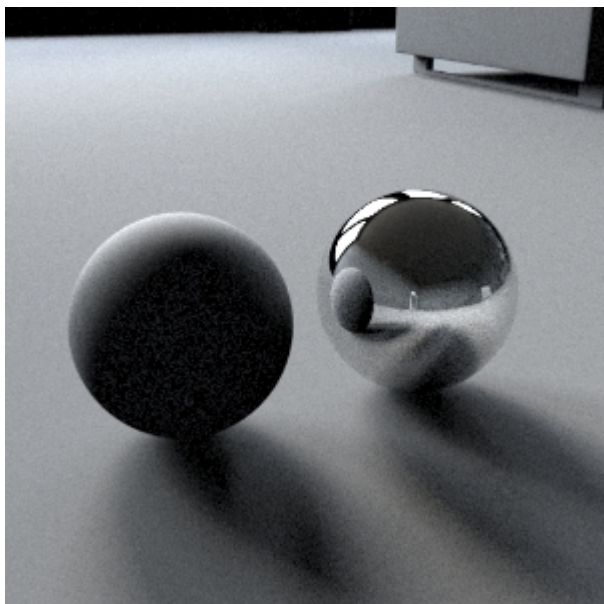
Diffuse Depth 3

HDRI

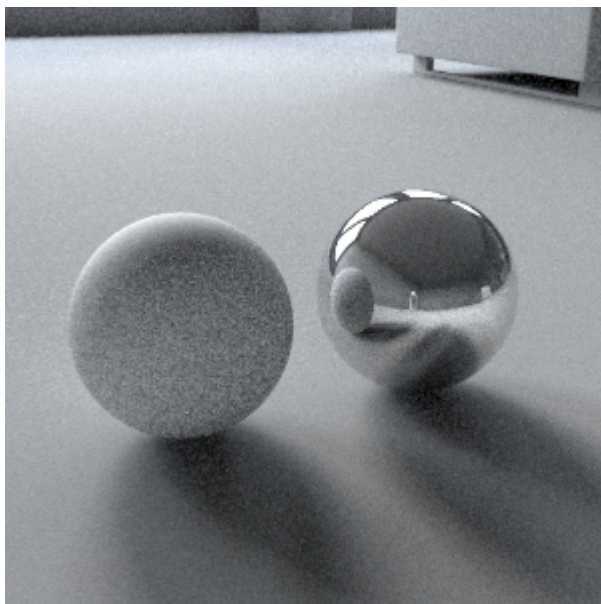
We can also light the room using an HDRI. This can give a feeling of natural lighting in the room. Some HDRI's work better than others in this type of interior lighting situation. You are better off using an HDRI that has strong sun light to give you better directional light coming through the window.

- Create a file texture and open the HDRI. In this case, a HDRI with a strong directional sun light has been used. This will give us stronger, sharper directional shadows through the window. Connect the HDRI to the Color attribute of the Sky Dome light.

Increasing the Diffuse samples reduces noise in the image. However, the render times become more costly, and even with 8 Diffuse samples, the image is still noisy.



Diffuse Samples 0



Diffuse Samples 3



Diffuse Samples 3: 3 mins



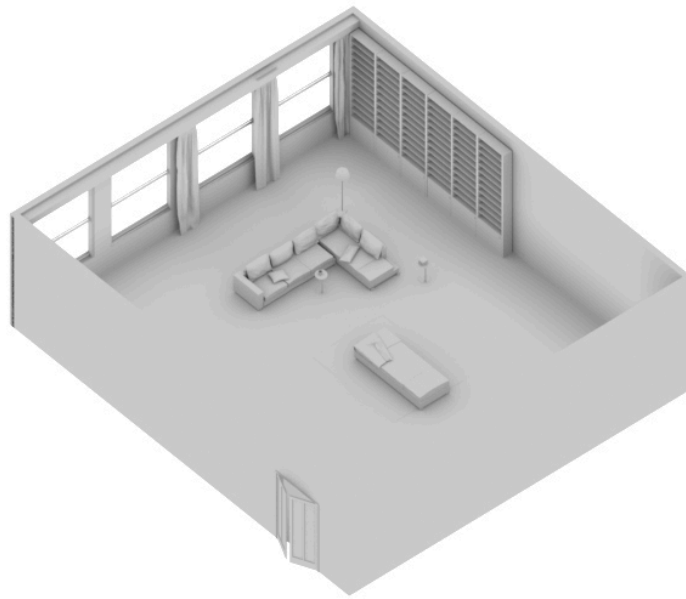
Diffuse Samples 8: 14 mins

Rendering with Ambient Occlusion



Ambient occlusion and direct light render combined using post-processing software

•
An alternative technique to lighting a room is to use Ambient Occlusion. A typical workflow is to composite the Ambient Occlusion render onto a Diffuse rendered image that does not have any GI Diffuse rays. The advantage is that it is typically faster to render and can give more creative control when compositing. You will also not need to add any GI Diffuse or GI Specular rays to the scene. However, it is not a physically accurate method, and some experience with compositing is required to fine-tune the final look.



Ambient Occlusion shader

- Start by creating an Ambient Occlusion shader and assign it to the room geometry. Increasing the Samples of the Ambient Occlusion shader to around 6 will reduce any noise associated with this shader.
- Render the scene using the Ambient Occlusion shader. It should look like the image below:



Rendered using only the Ambient Occlusion shader

•

Direct lighting render

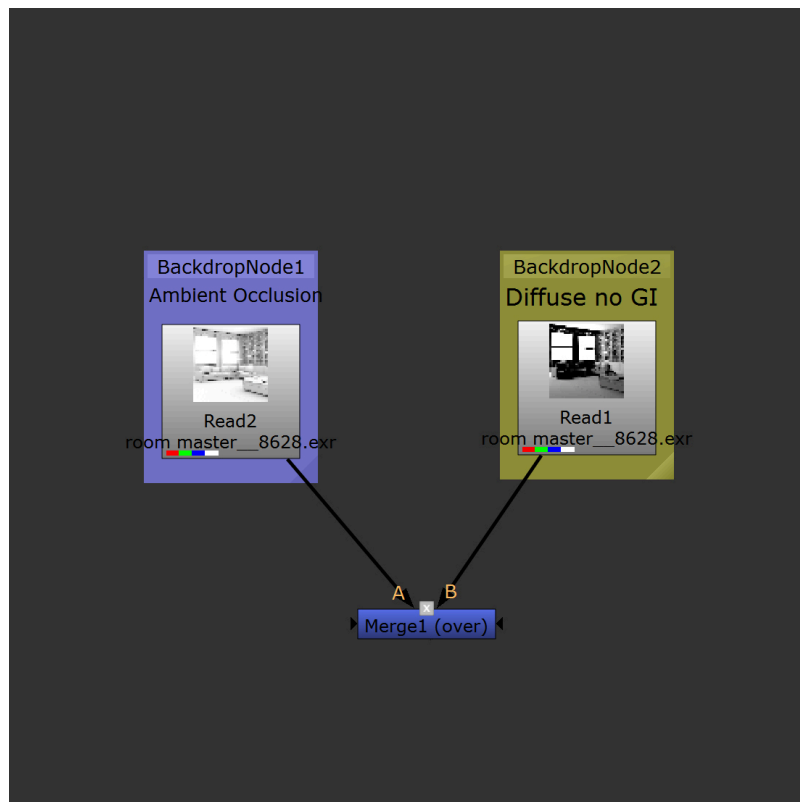
- Now render the scene using direct lighting only (in this case area lights have been positioned outside the windows, but you could try other lights such as the SkyDome light). However, in this case, ensure that GI Diffuse samples are set to 0. You should notice when you render the scene that there is no bounced light and the back wall will be black. We will use the ambient occlusion in Nuke to 'fill in' the areas that have no bounced GI Diffuse light.



Direct light (Indirect 0)

Composite

- Open both renders in Nuke. Create a **merge** node and connect the **Ambient Occlusion** render to the **'A'** input and the **Direct light** to the **'B'** of the **'merge'** node. Make sure the **'Operation'** is set to **'Over'**. Adjust the **'mix'** to around **0.5**. You should notice the Ambient Occlusion appear in the black areas of the 'Direct light' image. It should be obvious that we are in effect 'cheating' the use of bounced light using Ambient Occlusion.



Ambient occlusion render merged with diffuse render (no GI)

The images below show the results of merging (over) the 'Ambient Occlusion' render onto the diffuse lighting (without GI) render. The benefits are that it is reasonably fast to render and does not contain any GI noise.



Direct light (Indirect 0)



Ambient Occlusion render



Ambient

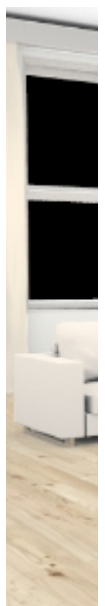
The images below show the result of rendering using a Utility shader set to flat color, merged with the ambient occlusion AOV. This renders very quickly. However the results are far from photo-realistic.



Ambient occlusion



Utility shader set to flat color



Ambie

Studio Automotive Rendering



This 'making of' tutorial covers some lighting, shading, and rendering tips for replicating an automobile studio-style shot. The key to successful automotive rendering is with realistic 'real world' lighting. A lot of work and 'fine-tuning' goes into lighting an automobile correctly in a studio. Achieving the right lighting is key to creating interesting specular reflections that will help accentuate the aesthetics of the automobile. The tips provided here aim to make this process easier.

 An example of lighting an automobile in the real world can be found [here](#).

This tutorial has been broken up into the following chapters:

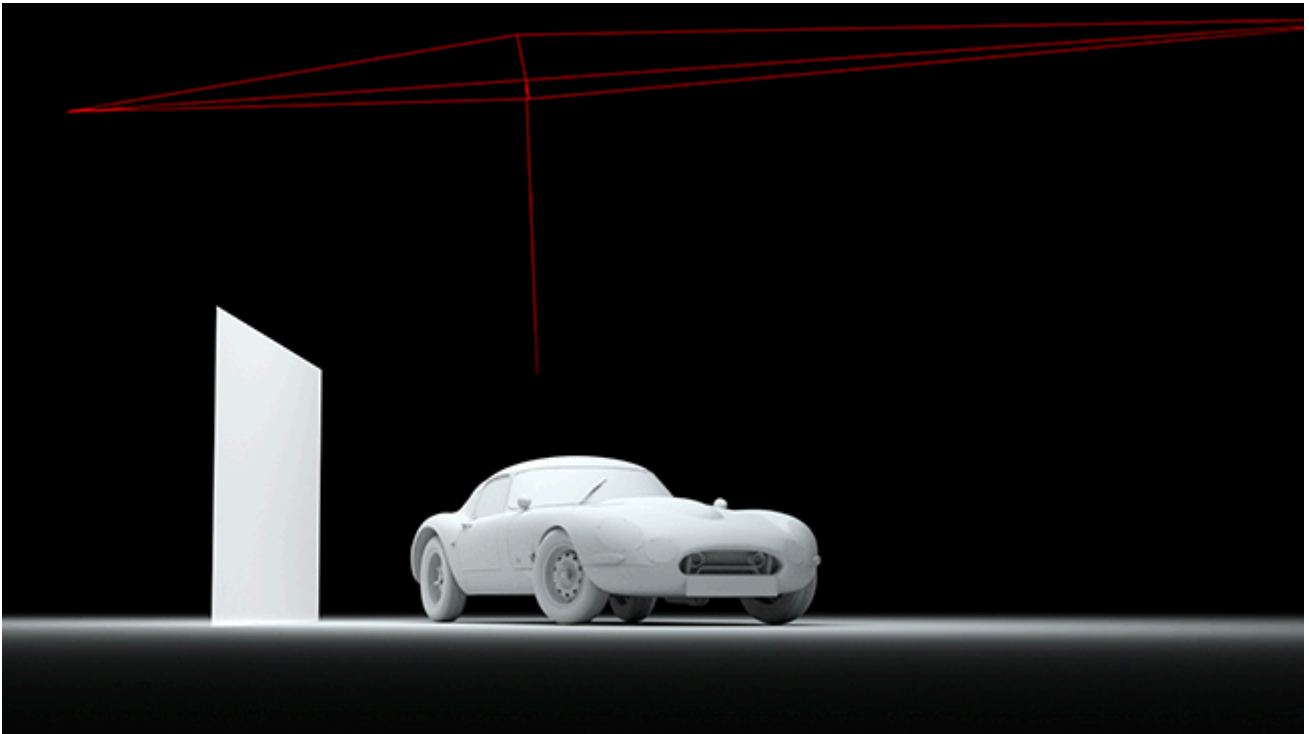
[Lighting](#)

[Reflector](#)

[Shading](#)

Quad Area Lighting

The scene is simply lit with a **Quad** area light representing large studio softbox lighting. A reflector has also been added to show reflections in the side of the car body.



Color Texture

To get realistic, real-world color and specular reflections in the car paint and windscreen glass, it is best to add an HDR map to the color of the light. You can download the sample HDR map used in this scene [here](#) (Flat_2.hdr).



With HDR map

Roll over image to view without the HDR map. Note that the reflection (and color) of the light lacks interest without the HDR map.

Soft Edge

We can specify a smooth falloff for the edge of the light by using `soft_edge` in the `quad_light`. This creates a softer look to the specular reflections of the light in the car paint and windscreen.



Roll over image

Here we can see the effect when we increase `soft_edge` to 0.5 using the render camera (rollover image):



Roll over image

Specular Contribution

We are getting some blown highlights on the hood of the car. Reducing the exposure of the light would help with this. However, it would also reduce the overall illumination in the scene, which is not what we want. A simpler and more effective method is to reduce the specular contribution of the quad_light.



quad_light.s specular: 1. Roll over image to view quad_light.s specular: 0.6



Adding a `light_blocker` to the `quad_light` can give you more artistic control over where and how the light is reflected in the car. Notice the effect it has on the lighting and reflections on the right side of the car (rollover image).



Roll over image to view with `light_blocker` (plane).

Emissive Plane

A reflective white plane has been placed to add reflection and illumination to the side of the car. This is because, typically car photographers do not normally shine lights directly onto the car, but rather shine lights onto surfaces which are reflected in the car paint. In this case, `emission_weight` is 0.5. You can, of course, also map a texture to the `emission_color`. However, in this case, a white `emission_color` works fine.



Roll over image to view emissive plane (notice the effect on lighting in body and wheels).

• We can see exactly the effect that the emissive plane is having in the reflection of the car if we enable 'Ignore Lights', found under Diagnostics->Feature Overrides.



Effect of emissive plane is more clearly visible when 'Ignore Lights' is enabled (under Feature Overrides)

Shading



This section is mostly redundant as these shading effects can be achieved using the [car_paint](#) shader.

Facing Ratio Shader

The `facing_ratio` can be useful for creating a pearlescent color coating to the car paint. In the example below a `coat_color` has been added to a `standard_surface` shader. In this case, the `facing_ratio` shader has been connected to the `coat_weight` and used to exaggerate the `coat_color` effect.



base_color



metalness



With Facing Ratio

Roll over image to see `coat` without `facing_ratio` shader. The effect is more visible in areas that are facing the camera such as the front light.

Thin Film

`thin_film` in the `standard_surface` shader can also be used to create a multi-tone car paint material (Thickness: 400, IOR: 1.43).



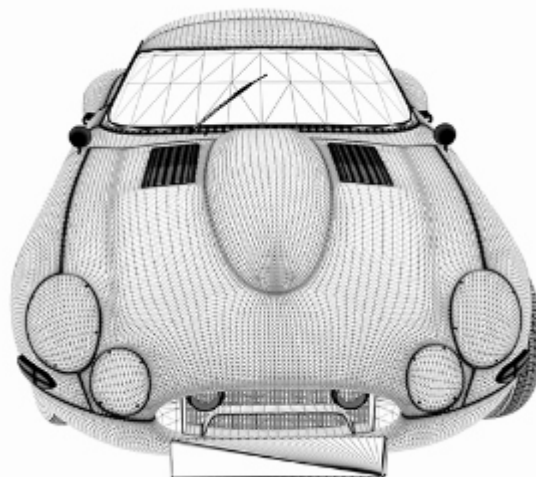
Roll over image to view without `thin_film`

Utility Shaders

There are some shaders that can be useful for rendering car models. The **Utility** and **Wireframe** shaders can be used to diagnose contour lines and spot continuity problems when evaluating models. The **Reflection Lines** mode in the Utility shader can also give you an idea of how real-life reflections would behave. This sort of shading is especially useful when **modeling cars** where surface evaluation is critical.



Reflection Lines (Utility shader)



Wireframe shader

The utility shader also has a 'Metal' `shade_mode`. This can be used as a simple shading method for testing and checking reflection highlights in the car.



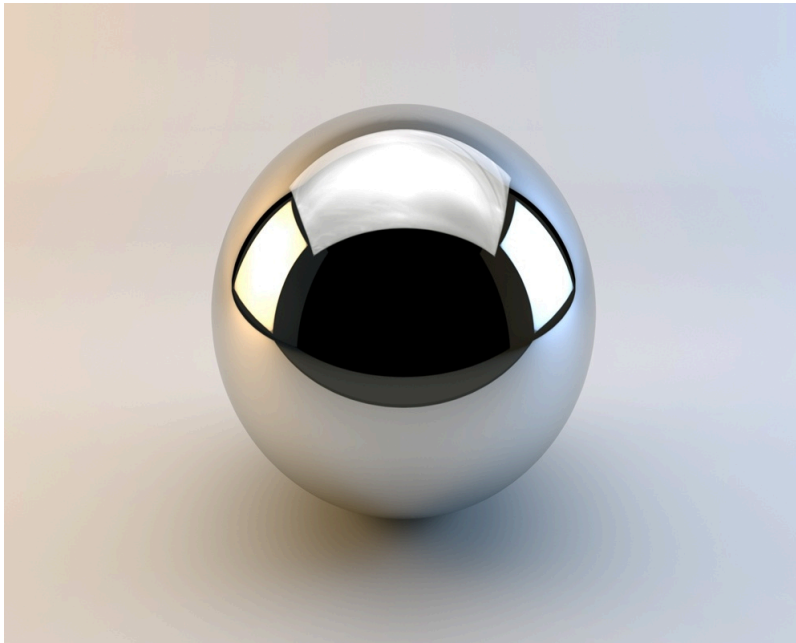
The light's specular reflections are more easily viewed with the utility shader (`shade_mode` set to 'Metal'). Roll over image to view low roughness.

That concludes this tutorial on automotive studio rendering. Why not try replicating a studio photo of a car using some of the techniques presented here.

-
-



Studio Lighting

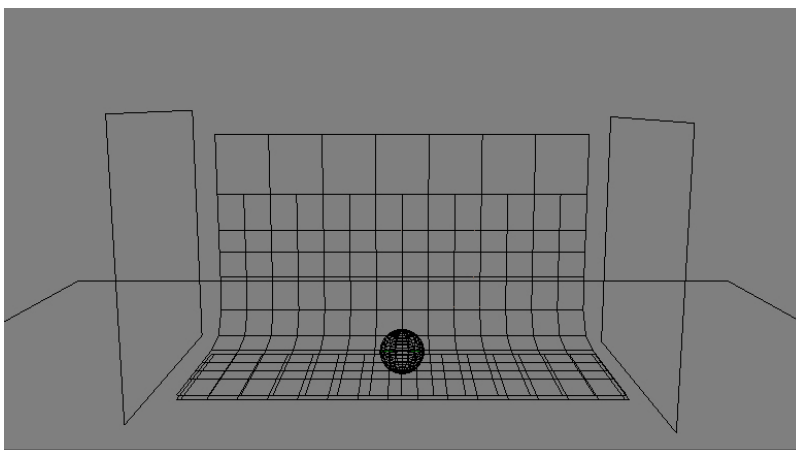


In this tutorial, we will cover how to recreate a simple [photographic lighting studio setup](#) that can be used for lighting and render all manner of objects. We will use a combination of Arnold area quad lights and HDR maps for creating realistic specular reflections in the scene. The HDR maps will provide rich specular reflections in the surface of any model that sits in front of it and will add to the photo-realism of the final render.

•

Set Model

The scene consists of a curved backdrop with a floor plane and one plane on either side with a dark gray [Standard Surface](#) material assigned to them both. •Click [here](#)•to download the Softimage scene (light texture files not provided).

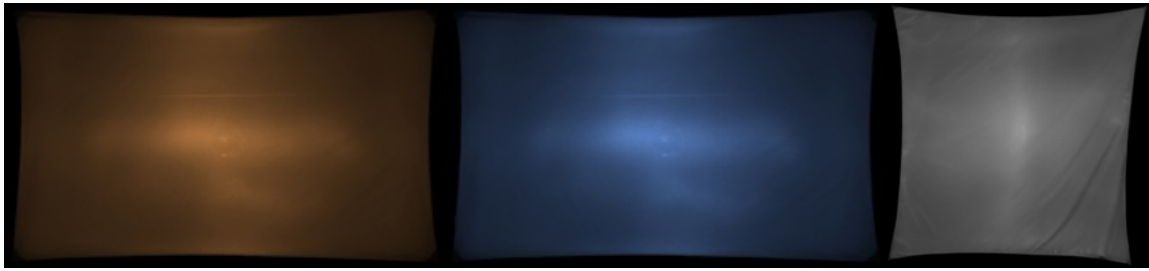


•

Lights

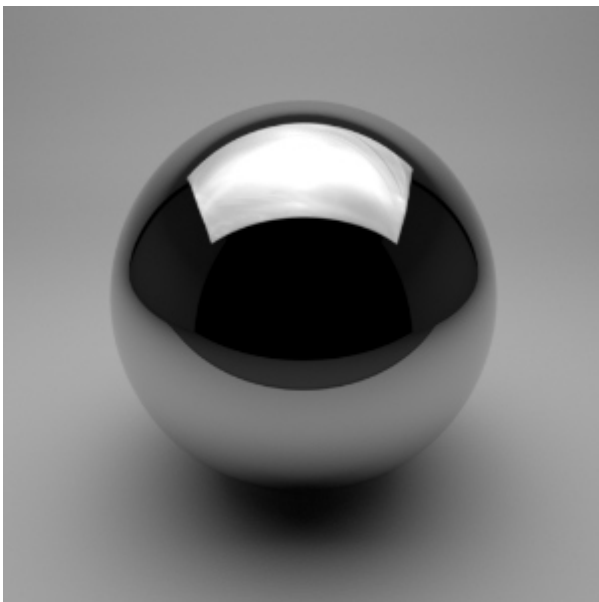
- Create an Arnold area quad light for each of the three lights - top, left and right. These will represent large soft boxes as used in a photographic studio.

i HDR maps have been used in this tutorial. If you do not want to use file textures, simple box ramp texture maps could be used. However, the reflections in your model will not appear as realistic.

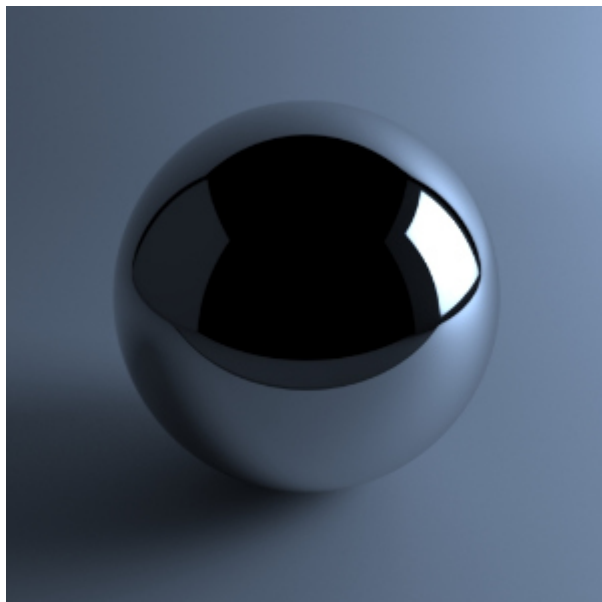


Left to right: softbox_middle_wrm.HDR, • softbox_middle_cold.HDR, • softbox_square_gs.HDR.

- Connect each HDR map to the Color attribute of each area light as indicated in the images below:



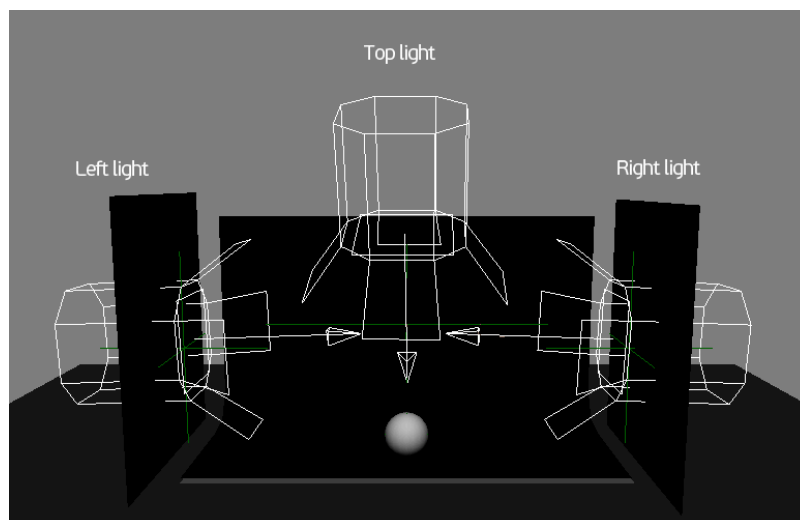
softbox_square_gs.hdr connected to Color attribute of Arnold area light 'Top'



softbox_middle_cold.hdr connected to Color attribute of Arnold area light 'Right'



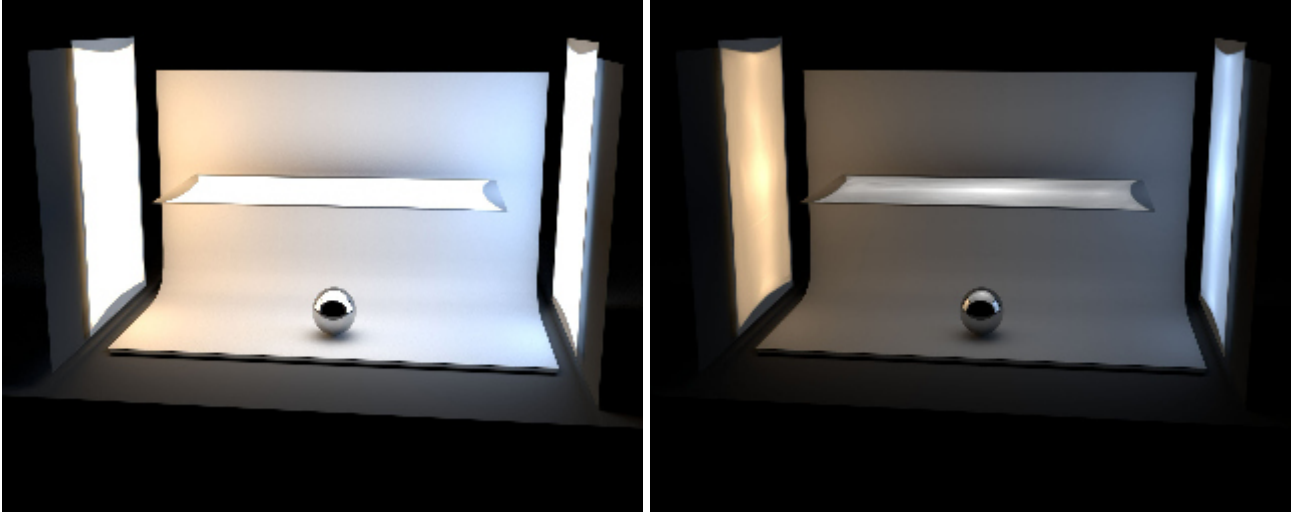
softbox_m



Create Quad lights in the positions above

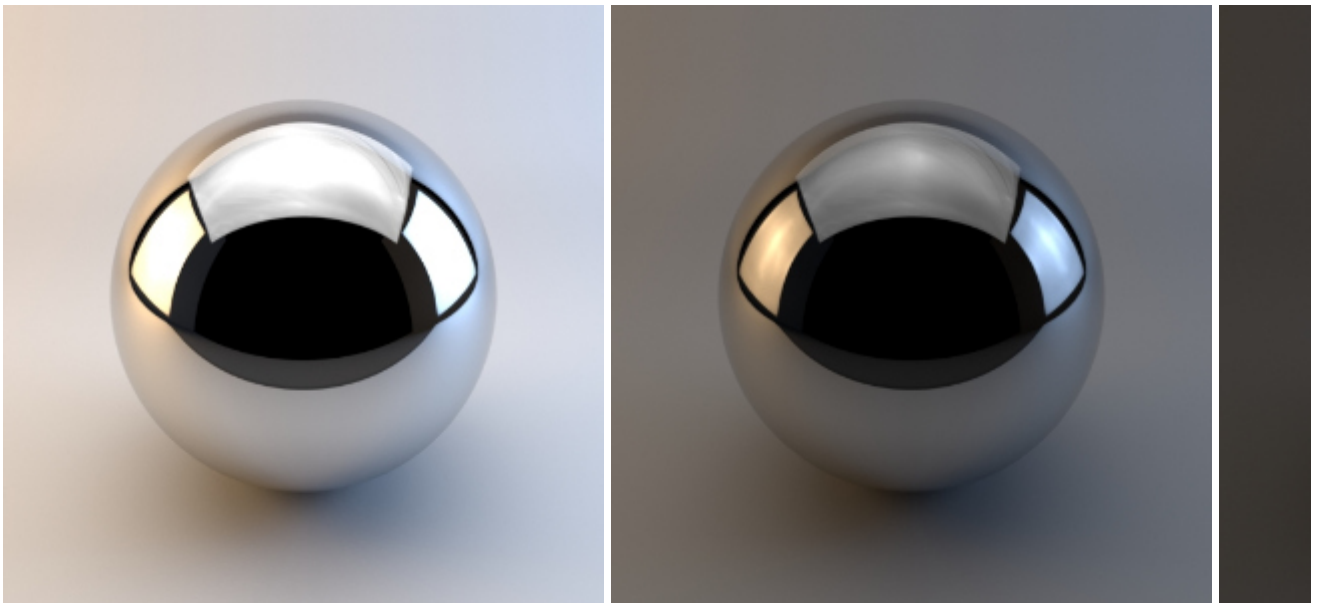
Exposure

- Test render the scene. You should get something like the image below. If the reflections appear too bright, remember that you can lower the exposure settings in the Arnold attributes for the render camera.



The high dynamic range of the HDR maps are visible when the lowering Arnold's camera exposure attribute

The light HDR map details are visible when lowering the camera exposure. You can see the full high dynamic range visible within the specular reflections in the scene:



Exposure 0

Exposure -2

Alternatively, you can experiment with changing the lights intensity or changing the HSV attributes of the HDR maps to get the effect you want.



If you find that the lighting appears a little dark at the front of your object in the studio you could also try adding a reflector plane in front of the camera (turn off 'Primary Visibility' for the plane).

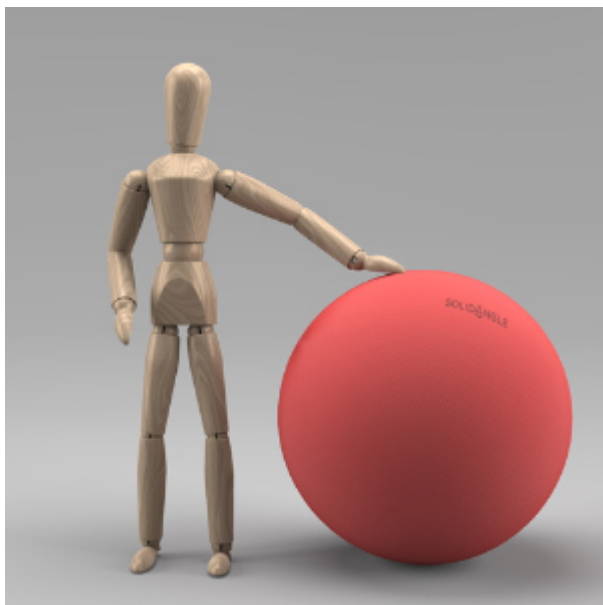
Spherical camera lens

Once you have completed this tutorial, why not have a go at converting the scene into an HDR map using the Spherical camera lens.



Scene rendered with Spherical camera lens

Example renders



Further examples using this studio lighting setup

Shading



- [Displacement Mapping Using the <udim> Token](#)
- [Flakes Example](#)
- [Hair Tutorial](#)
- [Rendering Glass Surfaces with Arnold](#)
- [Shading a Rose](#)
- [Specular BRDF \(Bidirectional Reflectance Distribution Function\)](#)
- [Water Bottle](#)

Displacement Mapping Using the <udim> Token



Rollover image

The aim of this simple tutorial is to explain how to render texture maps that have been exported from Mari using the correct displacement workflow with Arnold. This example uses a creature model and displacement maps (kindly provided by [Alex Huguet](#)).

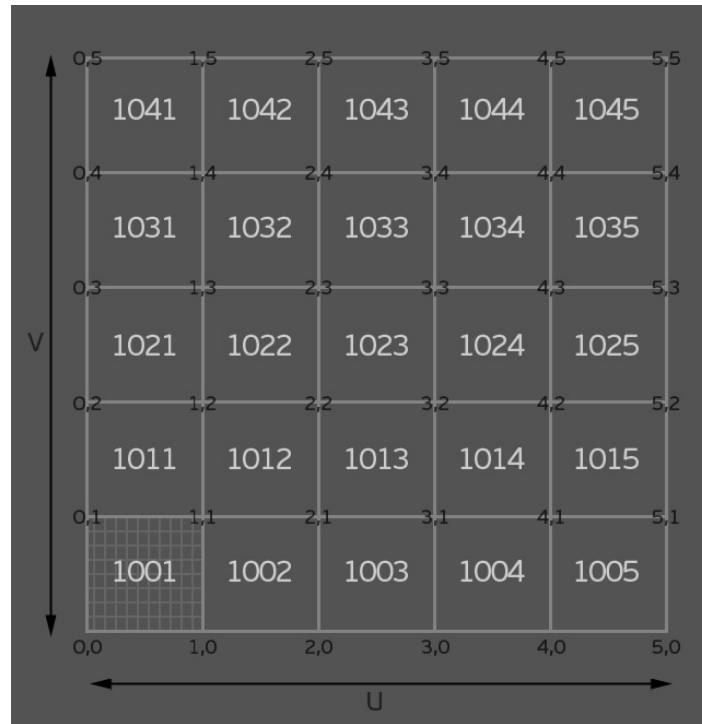
UV TILE - UDIM

UDIM values are a way of representing the integer coordinates of a texture, from the coordinates of its bottom-left corner in UV space. This way, multiple textures can be used to cover a model, rather than a single, much larger, texture.

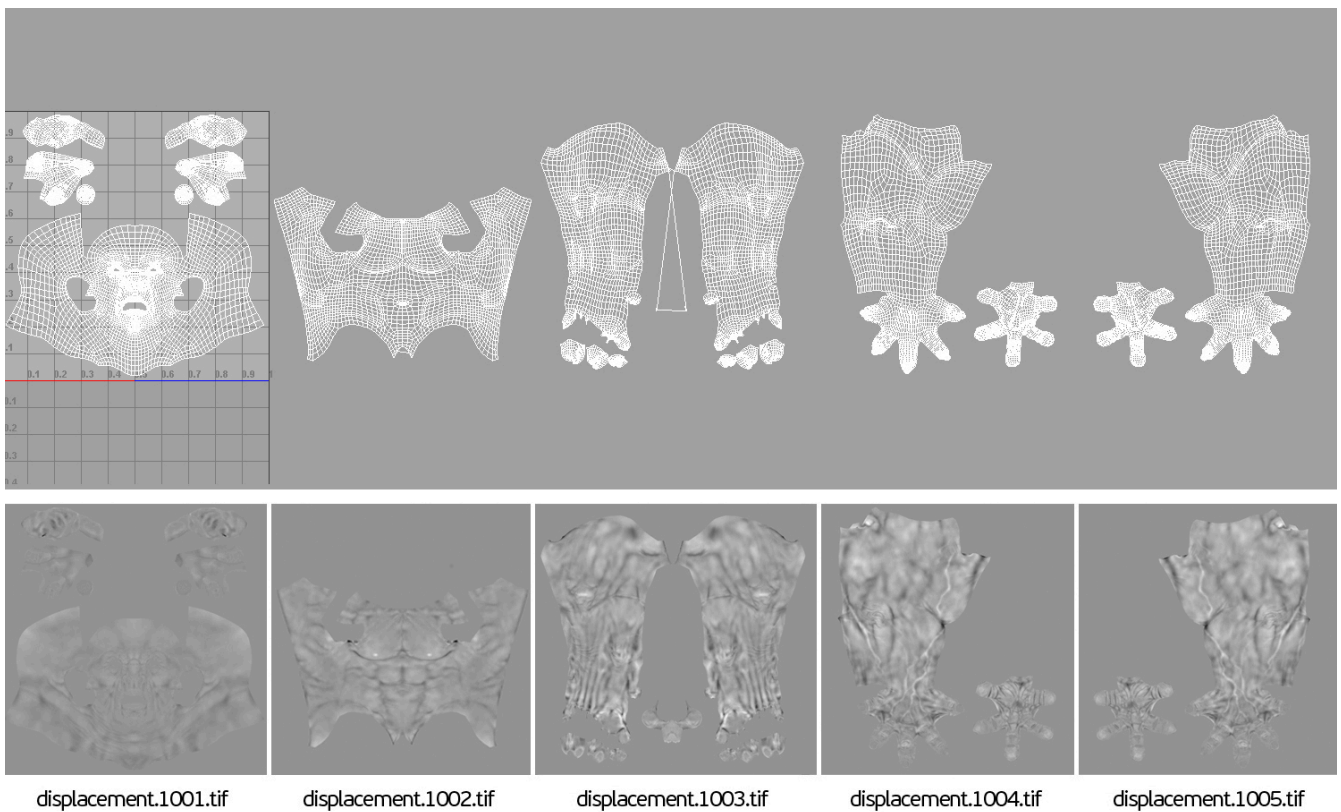
This is the convention used by Mari when exporting its UV texture space. It starts at 1001 (0,0) and continues in U for another 10 offsets. It then progresses onto the next row of V, meaning for every 10 in U it moves 1 in V.

1021	1022	1023	1024	1025	1026	1027	1028	1029	1030
1011	1012	1013	1014	1015	1016	1017	1018	1019	1020
1001	1002	1003	1004	1005	1006	1007	1008	1009	1010

The UV offset grid below better shows this relationship (this example only shows a range of five as the creature model uses five displacement maps):



You can see in the UV Texture Editor, that the UV's have been laid out in UV space as 1001 (0,0) to 1005 (5,0) and that the displacement texture files have been named accordingly:



Displacement attributes

A file texture (using an absolute path name of `C:\Users\Documents\projects\Displacement-troll\sourceimages\displacement.<udim>.tif`) is connected to a displacement node which is connected to the shader assigned to the creature.

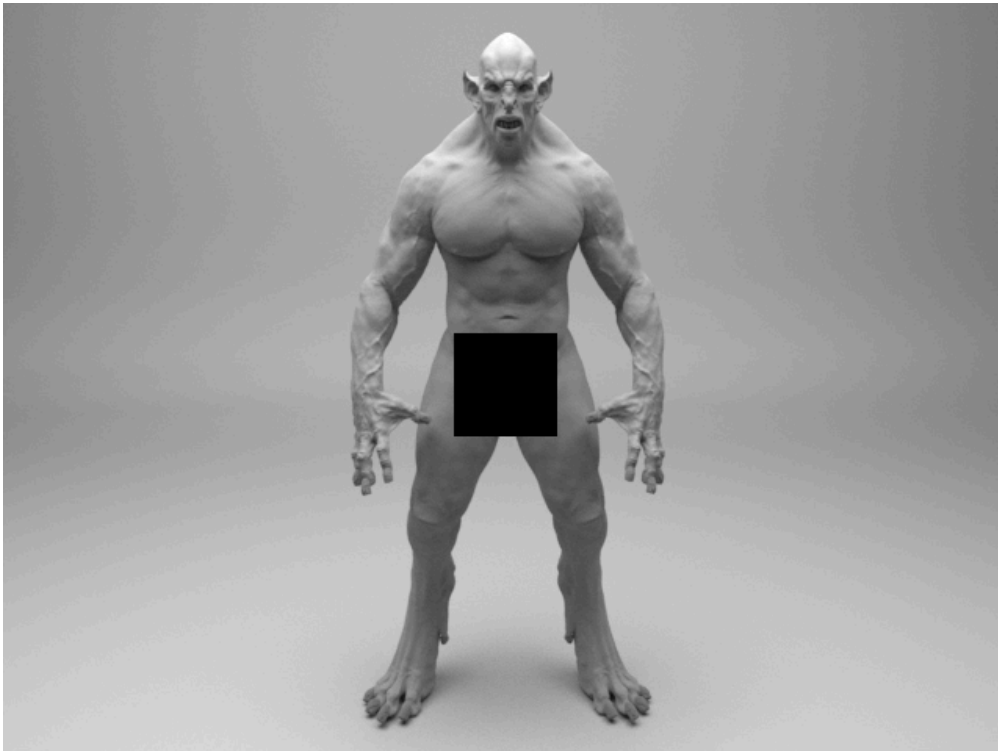
The Vector Space has been set to 'World' space, and the scale has been set to 0.75. The 'Bounds Padding' has been set to 1, and the 'Scalar Zero Value' has been set to 0.25 (these values are meant as a rough guide and will need to change according to how and where the displacement map was generated).

Subdivision

To see the detail from the displacement maps, the model's subdivision settings need to be increased. In this case, the subdivision type has been set to Catclark, and the iterations increased to 2.

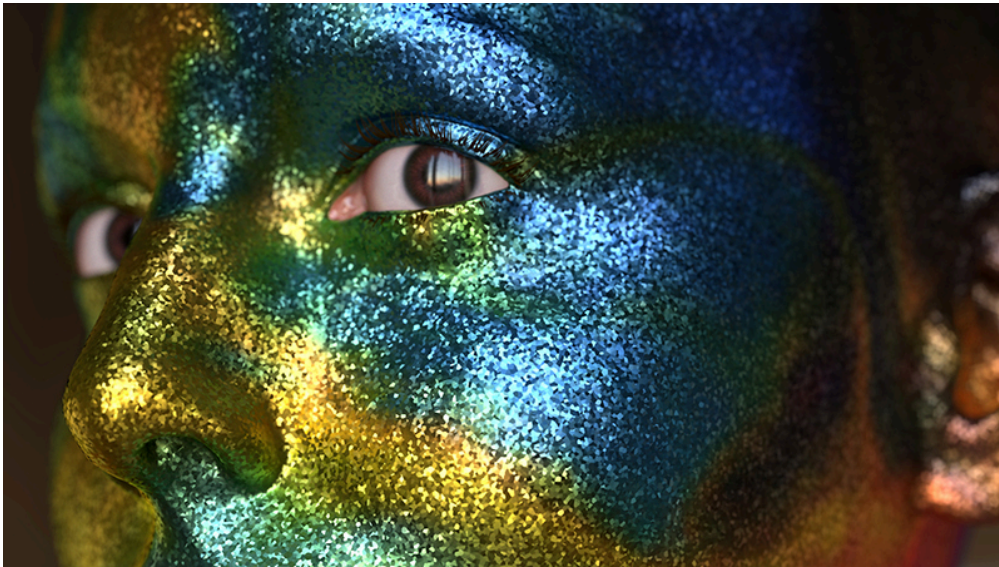
•

That concludes this tutorial for rendering displacement maps saved from Mari using `<udim>` tokens.



The final render (minus one bucket)

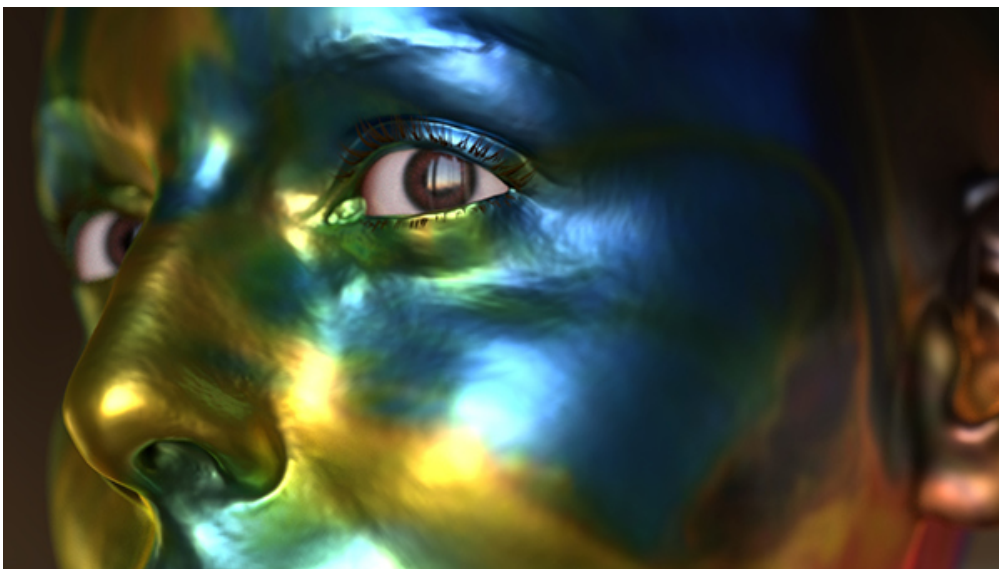
Flakes Example



There are several ways to use the `flakes` shader, and this tutorial aims to demystify some of its uses. In this short, making-of tutorial we demonstrate how to use the `flakes` shader (in combination with the `standard_surface`) to create some glitter make up.

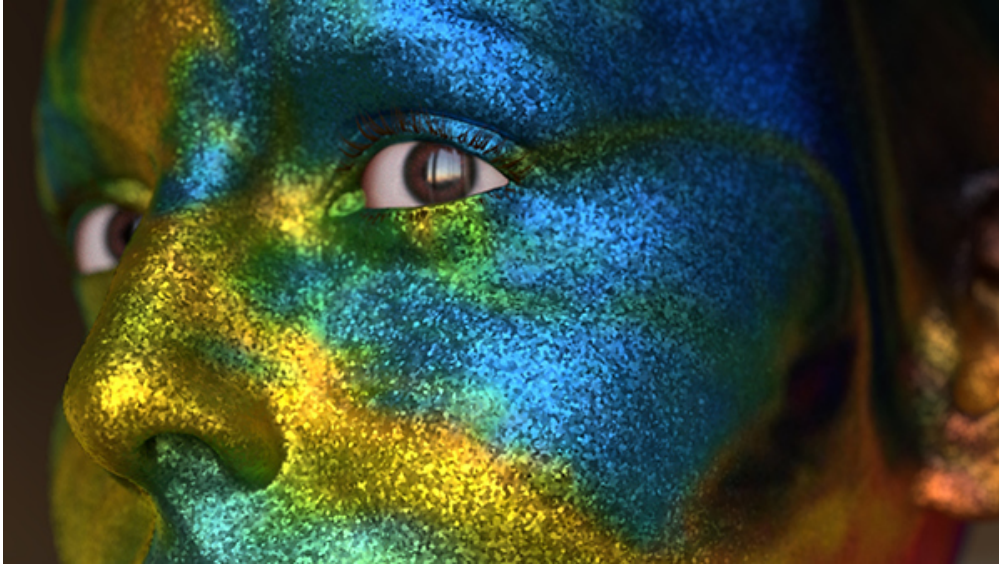
Specular Color

- To start off with, here we have a default `standard_surface` shader with a color texture map connected to the `specular_color`. `base_weight` is not necessary for this example.



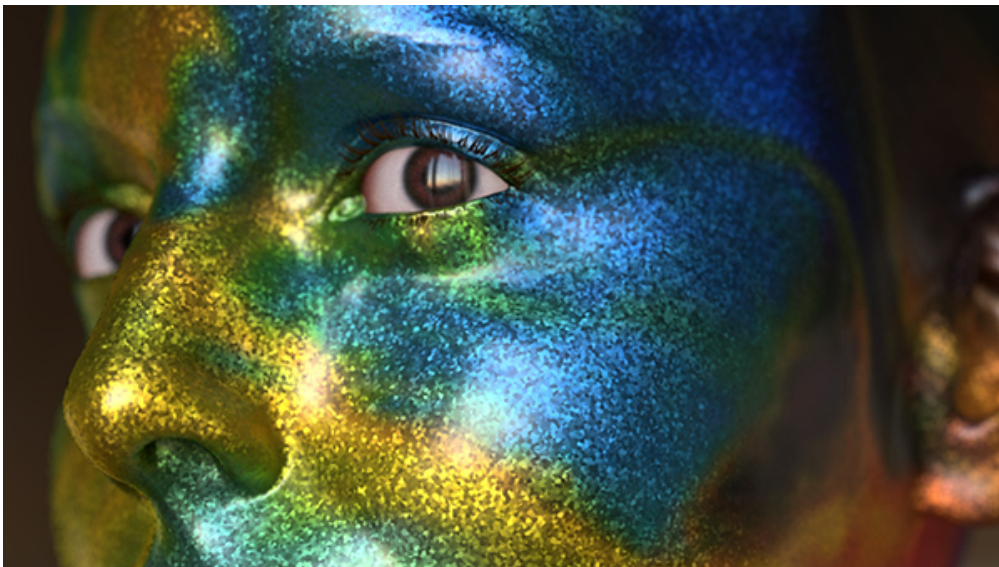
Normal

- If we want variations in the color of the flaked reflection, we need to have the `flakes` in a lower layer (Normal -> Bump Mapping -> under Geometry). This effect is evident in materials such as metallic car paint.



Normal and Coating (Default White Color)

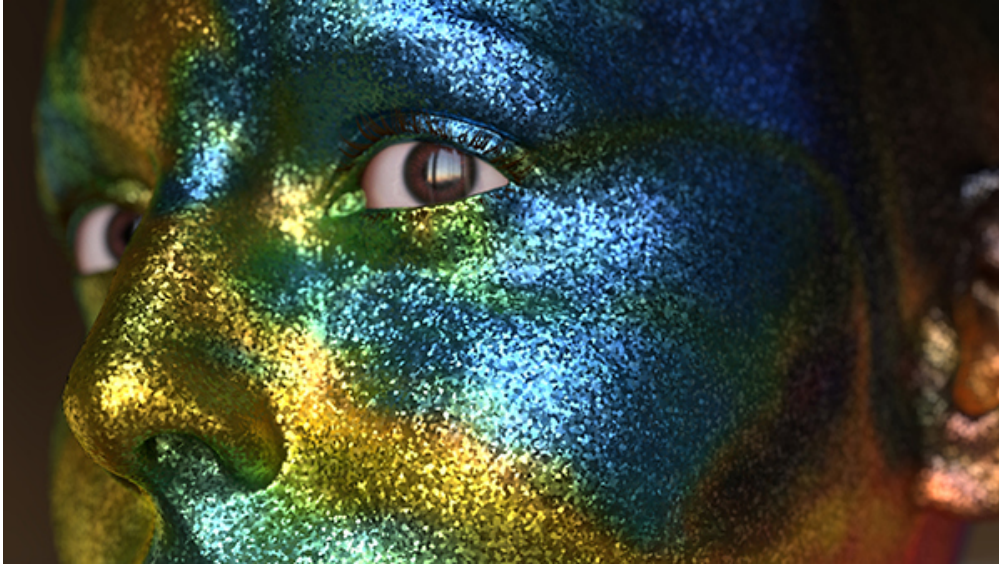
- If we increase the `coating_weight`, we just get a white `coating_color` (default). This effect would be useful for car paints or plastics but does not look suitable, as it is, for glitter makeup. We must also add the `flakes` to the `coat_normal` as well.



The color you assign to the `coating` will just tint the lower `base`, `metal`, and `specular` layers. The `coat` layer is a semi-opaque dielectric type layer that will always reflect white, no matter the color you assign it. The color of the `coat` layer will tint the underlying `base` and `specular` layers. For example, and somewhat counter-intuitively, a green-tinted glass does not have green reflections. It is the refraction of what is underneath the glass that is tinted green. The same thing happens with colored plastics.

Normal and Coat Normal

- Connecting the `Flakes` into both the `normal` attribute and the `coat_normal` gives us the final result that we want. In the end, we have used the `normal` on both the specular and coat layers. This method makes sense. However, the appearance would also depend on how much clear enamel/oil/whatever the glitter in the makeup is mixed with.



Ensure that the `output_space` of the flakes shader is set to world.

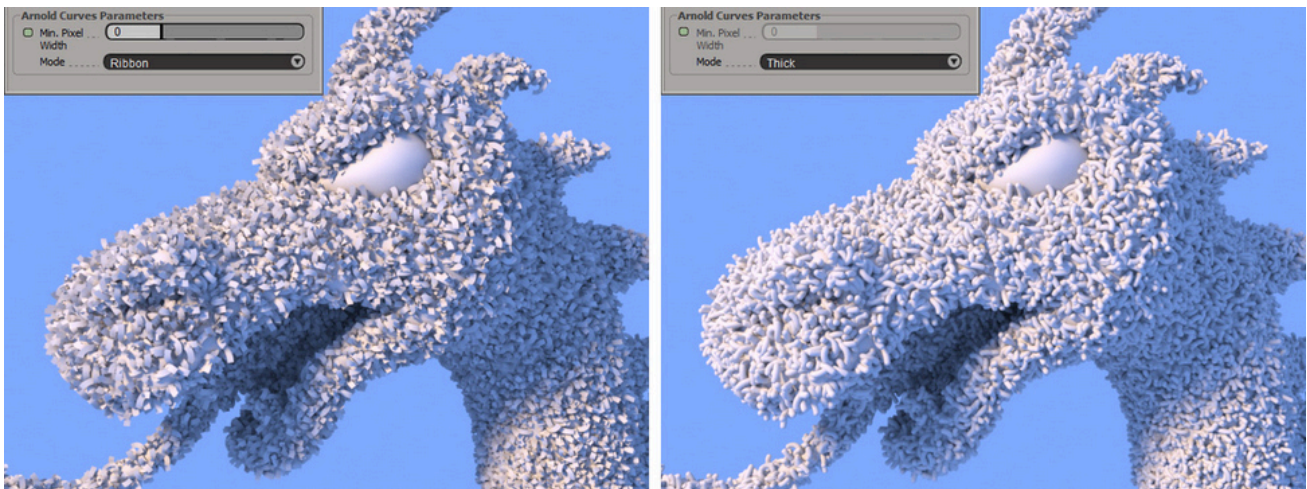
With car paint, the `coat` layer would be quite thick, giving it a smooth polished appearance that is easy to reproduce. However, with this makeup example, the `coat` layer would be quite thin and less polished. The `normal` of the `coat` layer would be a blend of the bumpiness of the underlying skin, the orientation of the flakes of glitter, the distribution of the makeup itself with thicker and thinner patches, and the smooth surface that the surface tension of a thin layer of liquid gives you over pretty much any rough surface. As the makeup dries and the face expresses, the surface of the `coat` layer could also become cracked and dulled.

Hair Tutorial

The purpose of this section is to give users a better understanding of how to render hair efficiently in Arnold. The key parameters that affect hair rendering speed will be explained and illustrated with rendering examples. General optimization tips and tricks will also be covered. The following tutorial covers aspects relating to both the [Arnold curves shape node](#), and the [Arnold hair shader node](#).

Hair rendering mode

Arnold can render hairs in two different modes: Ribbon and Thick mode, the following image illustrates the difference. In this document, only the Ribbon mode will be used as it is the best mode for rendering thin hair/fur (which is what this tutorial will focus on).



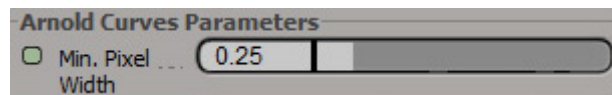
Hairs in ribbon mode (left) and thick mode (right)

i These modes, as well as other settings related to hair, are described in more detail [here](#).

Min Pixel Width (MPW)

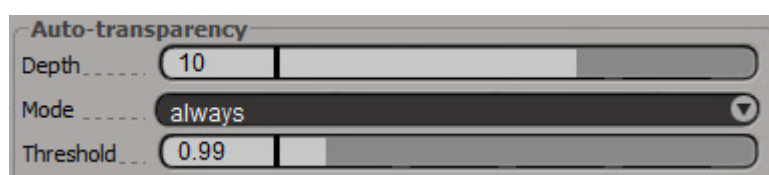
The nature of hair or fur is that it is made up of a lot of very fine/thin strands. Typically the smaller something is in screen space the more Anti-Aliasing (AA) samples is required to render it smooth and without flickering in animation, which makes rendering hair challenging. To make fur rendering easier on the sampling Arnold has introduced something called "Min Pixel Width" (MPW).

The concept behind MPW is to make extremely thin hair strands automatically thicker in screen space while at the same time making them transparent. This helps the hair strand to render smoother and with less flickering in animation even at lower AA settings. The MPW transparency will only kick in if the given hair is thinner in screen space than the specified MPW value; for example, if all the hair strands in the frame are more than 1 pixel wide and the MPW value is set to 0.5, no transparency will be used at all. If the camera, for example, starts to zoom out from that point thus making all the hairs thinner in screen space, as soon as they go below 0.5 pixels the MPW system will kick in and automatically keep the hairs 0.5 pixels wide while making them more and more transparent the thinner/farther away they get.



The MPW parameter in the "Arnold Parameters" property

How many layers of hair strands will be made transparent depends on the "Auto Transparency" setting in the render options, as illustrated below adjusting this value has a great impact on both quality and render time. Setting the auto transparency to 0 will always disable MPW on all the hair in the scene.



MPW accuracy depends largely on the auto transparency

 Note:•MPW only works on hair in Ribbon mode.

Rendering examples

Below you will find the same scene rendered with different settings for the "MPW" value and the "Auto Transparency".

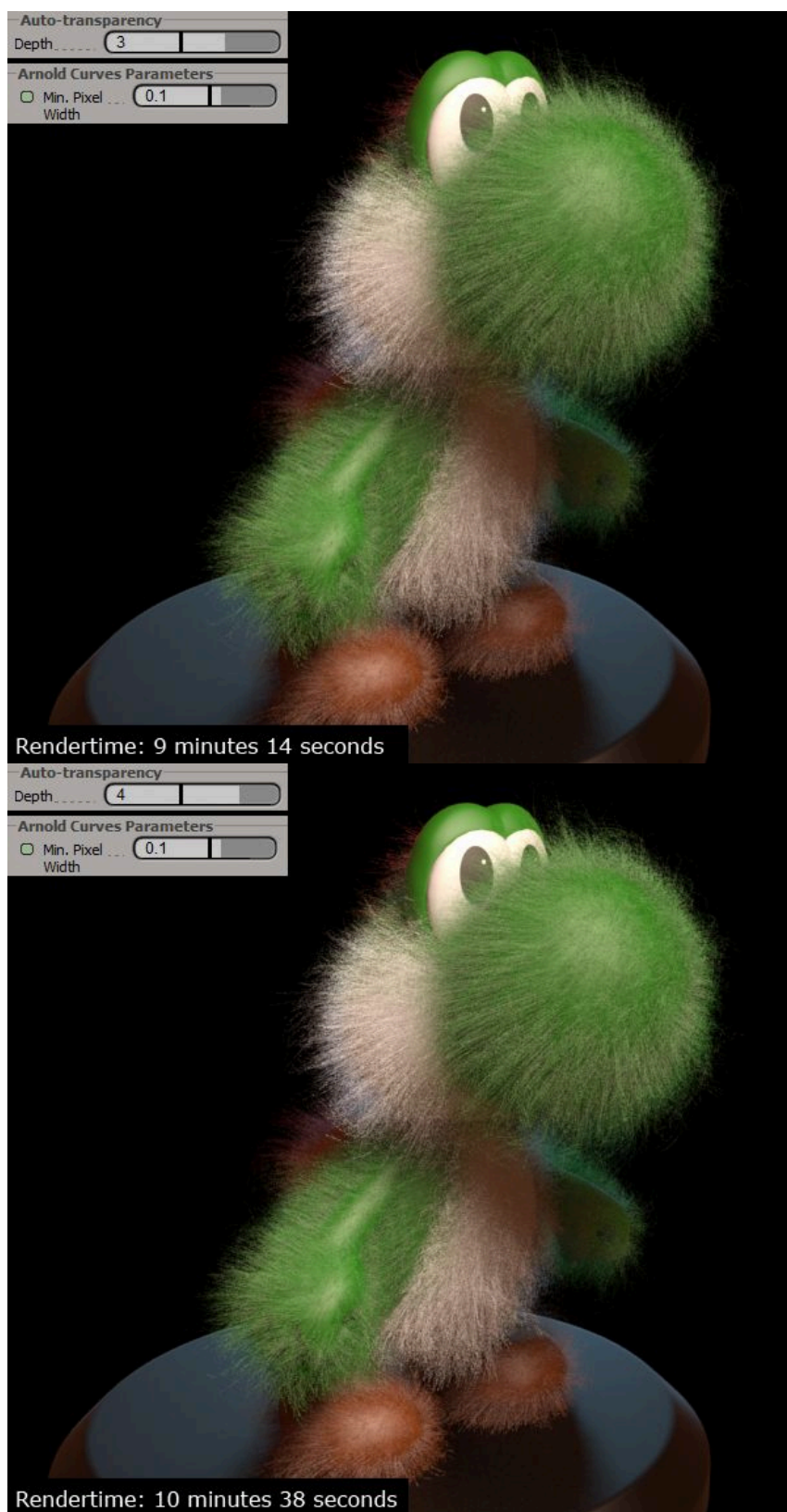
In all examples the following sample settings were used:

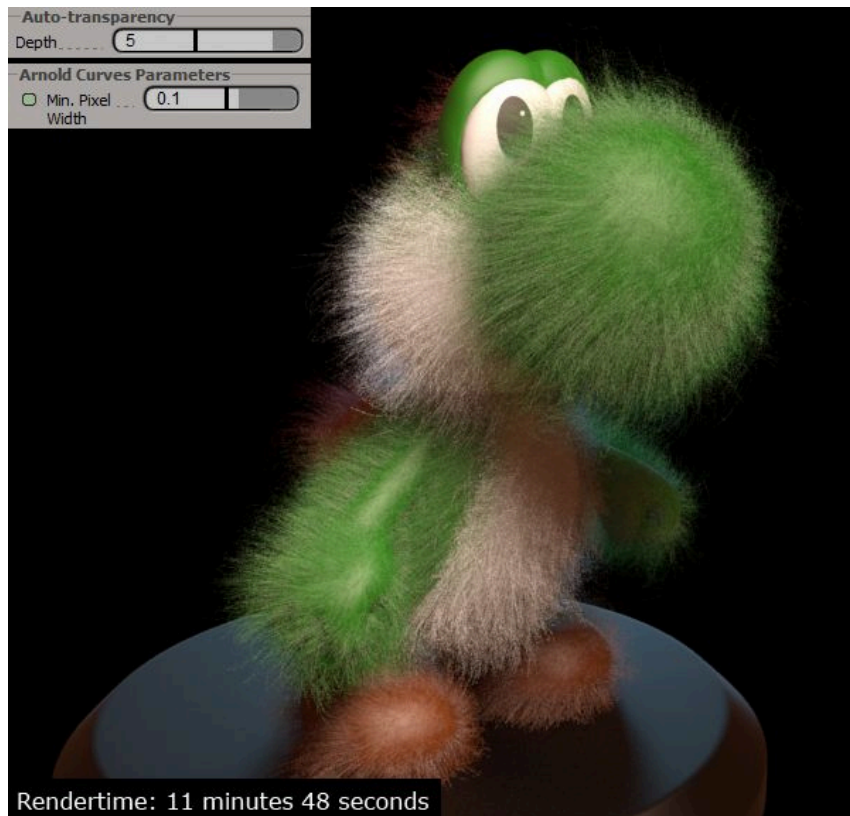
- AA Samples: 3
- Diffuse Samples: 3
- Specular Samples: 3
- Diffuse/GI Bounces: 1

These first set of images all use an MPW value of 0.1, note how the hair is still rather jaggy even with auto transparency set to 5, with these settings higher AA settings would be required to produce smooth results:

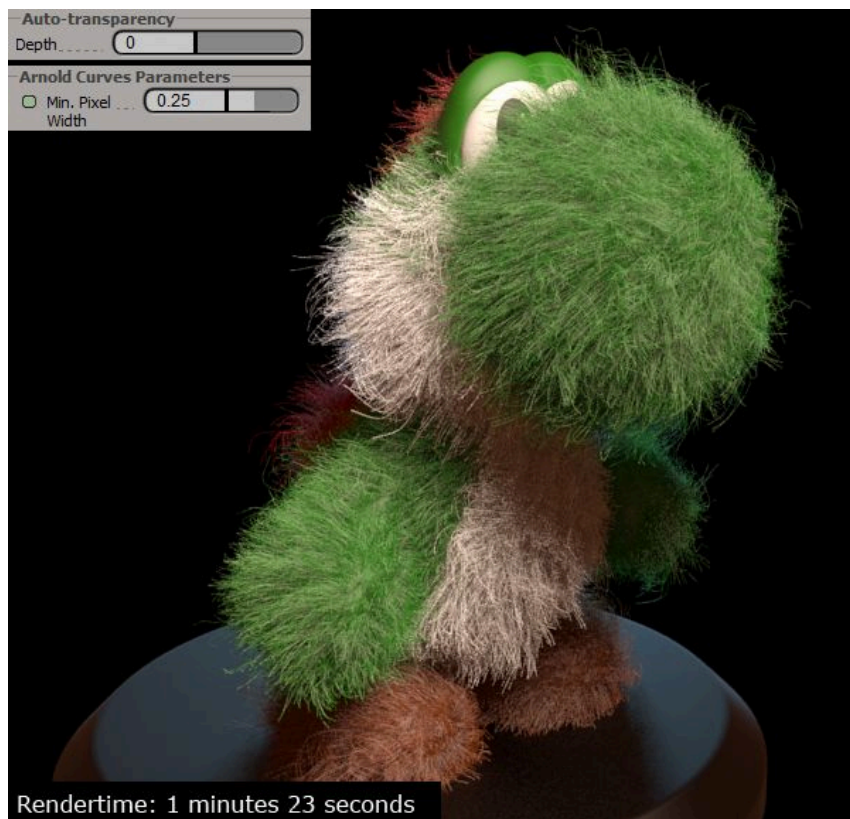


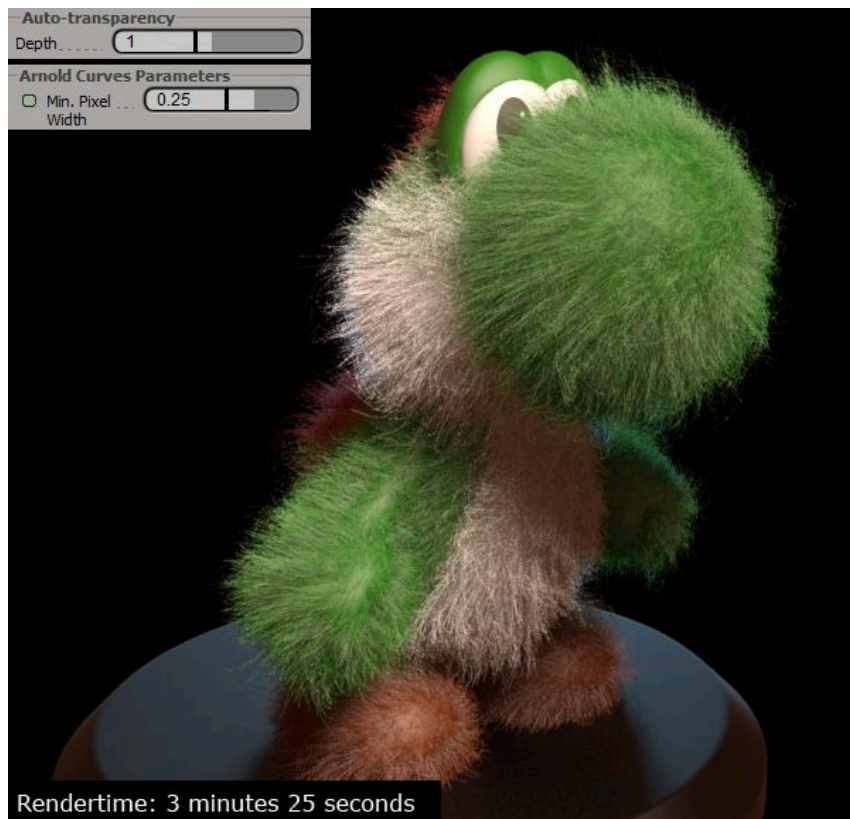




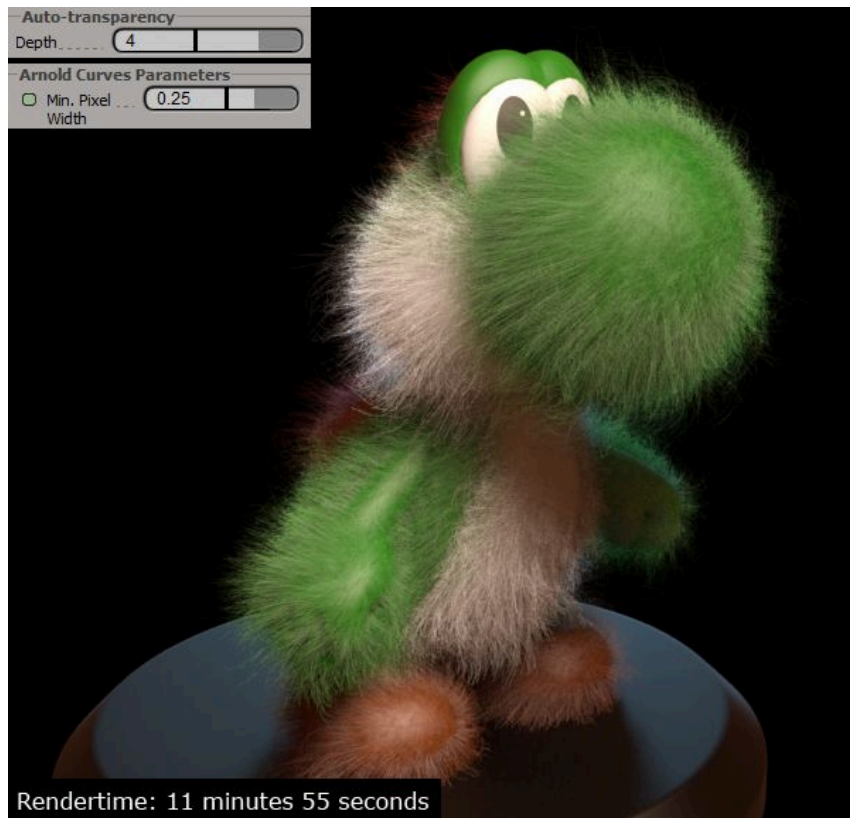


Here we are using an MPW value of 0.25, this is the default in SlttoA even when the Arnold Parameters property has not been applied to the hair, note how when auto transparency is increased the hair is a lot smoother than before even with the same AA settings:



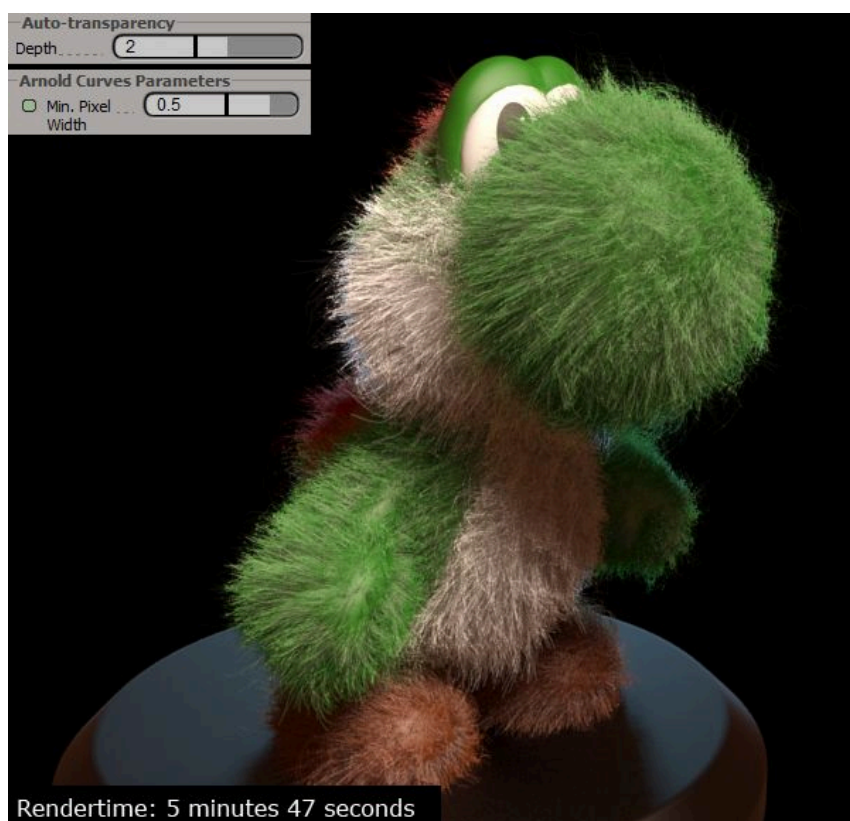


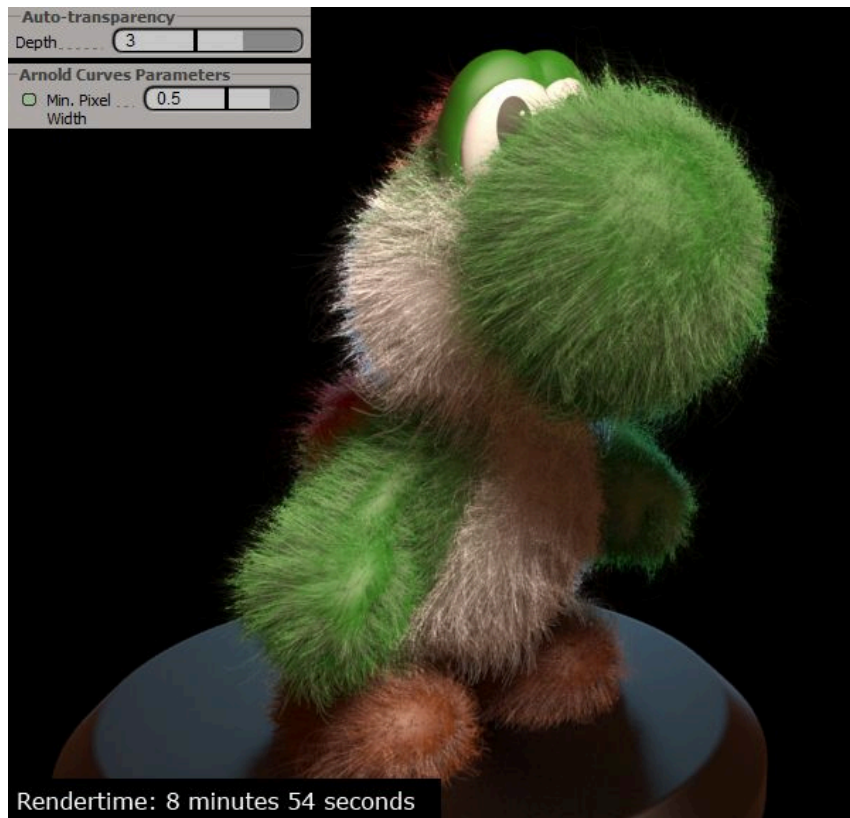


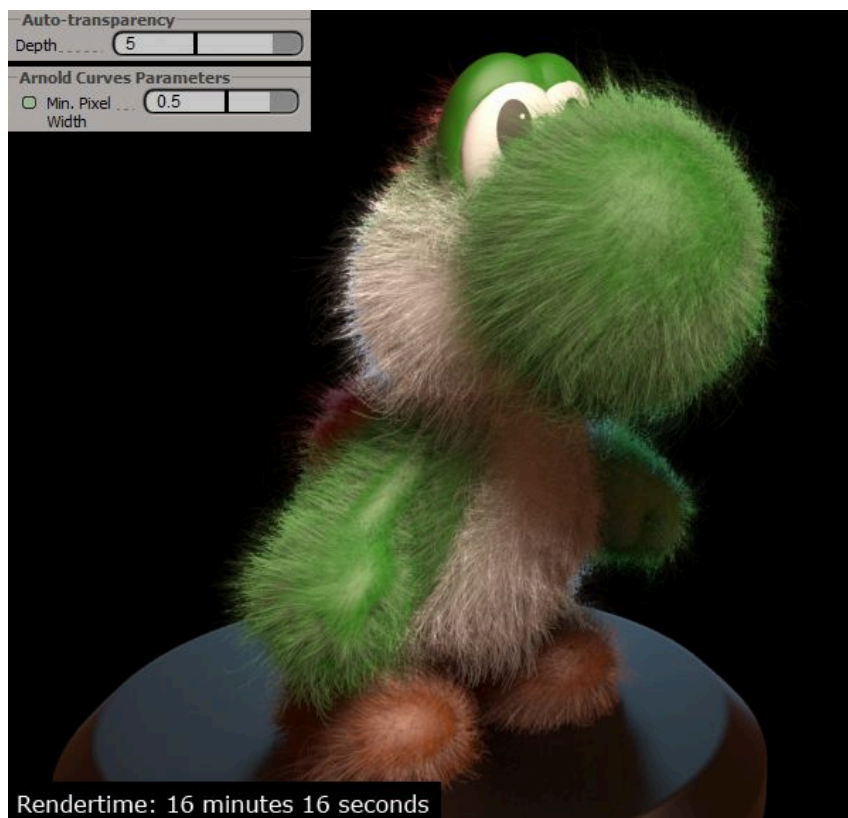


Finally we up the MPW value of 0.5, note that even when auto transparency is set to 0 the hairs will always be at least 0.5 pixels thick in screen space though not transparent.









•

Optimization

As seen in the examples above making the hairs transparent has a big impact on render times. For optimal rendering speed it is very important to use sensible values.

How smooth/slow hair renders depends on three things:

- Min Pixel Width
- Auto Transparency depth
- Number of Anti-Aliasing Samples

The higher the AA samples, the less you will need to rely on MPW to get smooth hair. If your scene is already using 10 AA samples to get smooth motion blur and DOF you can use a much smaller MPW value than if you were using only for example 4 AA samples.

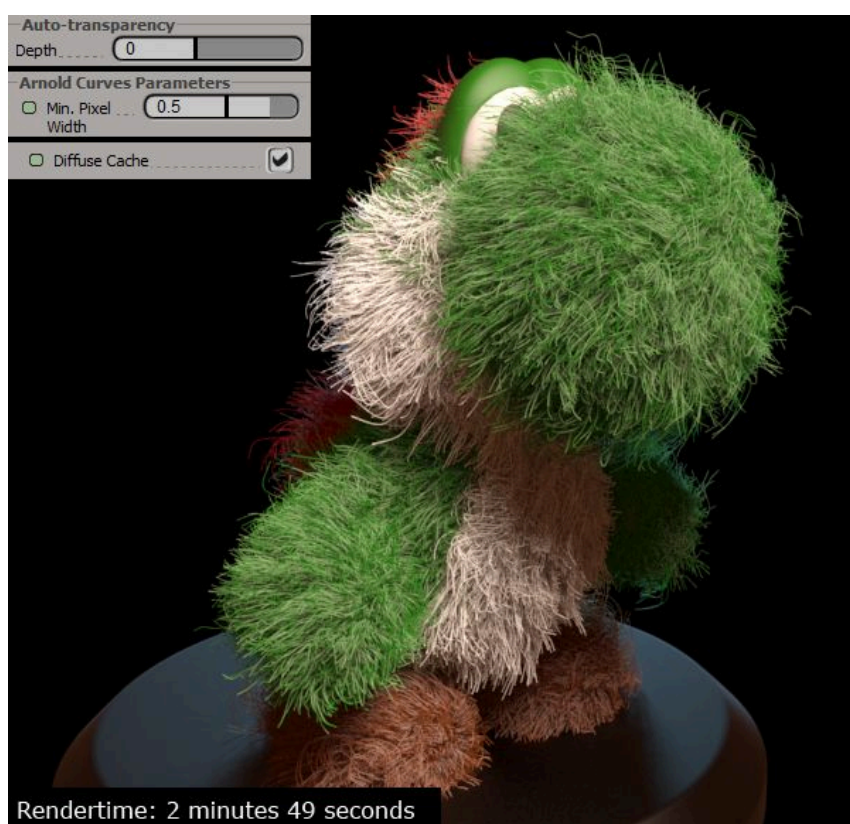
As mentioned above MPW will have an effect even if auto transparency is set to 0, so it can be useful as a "cap" for how small a hair is allowed to get even if you're not making the hair transparent, looking at the render times above thickening the hairs using MPW can have a positive effect on the render time when auto transparency is set to 0.

•

Diffuse Cache

The hair shader has a "Diffuse Cache" parameter. This switch makes Arnold to store direct diffuse lighting at hair vertices (technically: segment-end points). This cache is filled on-demand on the first lookup. Then GI rays use this cache as an approximation to the direct lighting.

- AA Samples: 5
- Diffuse Samples: 3
- Specular Samples: 3
- Diffuse/GI Bounces: 3





.

Rendering Glass Surfaces with Arnold

•



There are some common problems faced when rendering refractive glass surfaces with Arnold. There are a few guidelines that need to be followed when modeling surfaces where there is a liquid to glass interface. The problem lies in how the renderer determines if we are entering or exiting an object, and what the incoming and outgoing interfaces are.

The standard_surface shader assumes outward facing normals, that objects are embedded in air, and that there is no overlapping between them.



More information on nested dielectrics can be found [here](#).

We will demonstrate the results obtained from the various modeling approaches below:

- The liquid and container touch exactly.
- The liquid and container have a gap between them and therefore do not touch each other.
- The liquid and container overlap each other.
- We will also cover what material and render settings to use when rendering glass surfaces.

•

The tutorial is presented in the three following sections:

- [Modeling Glass and Liquid](#)
- [Shading Glass and Liquid](#)
- [Render Settings](#)

Modeling Glass and Liquid

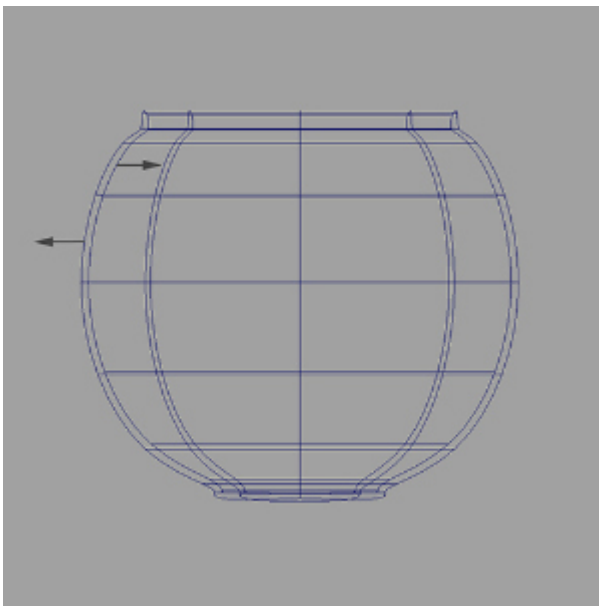


Ensure that you accurately model glass surfaces with thickness. Otherwise, the glass will not render correctly.

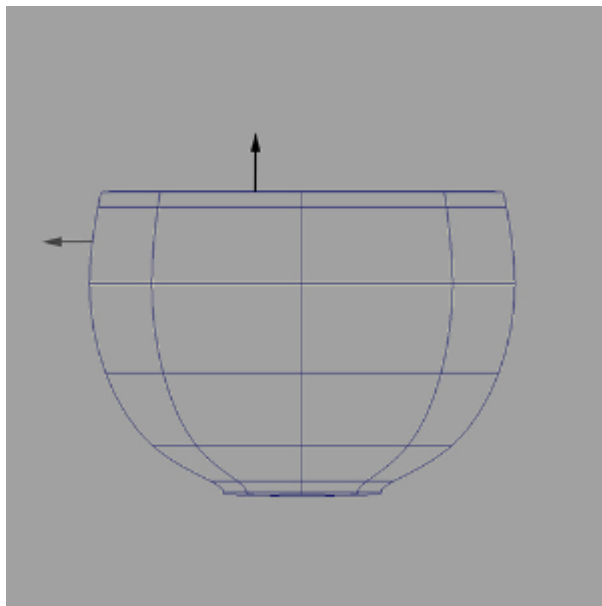
•

To accurately render refractive surfaces with Arnold, it is important that your model follows certain guidelines:

- The air to glass surface (IOR 1.5) has normals that are pointing out of the glass, covering the area where air directly touches the glass.
- The glass to liquid surface (IOR 1.33), has normals that are pointing out of the liquid, covering the area where air directly touches the liquid.



Glass bowl normal direction

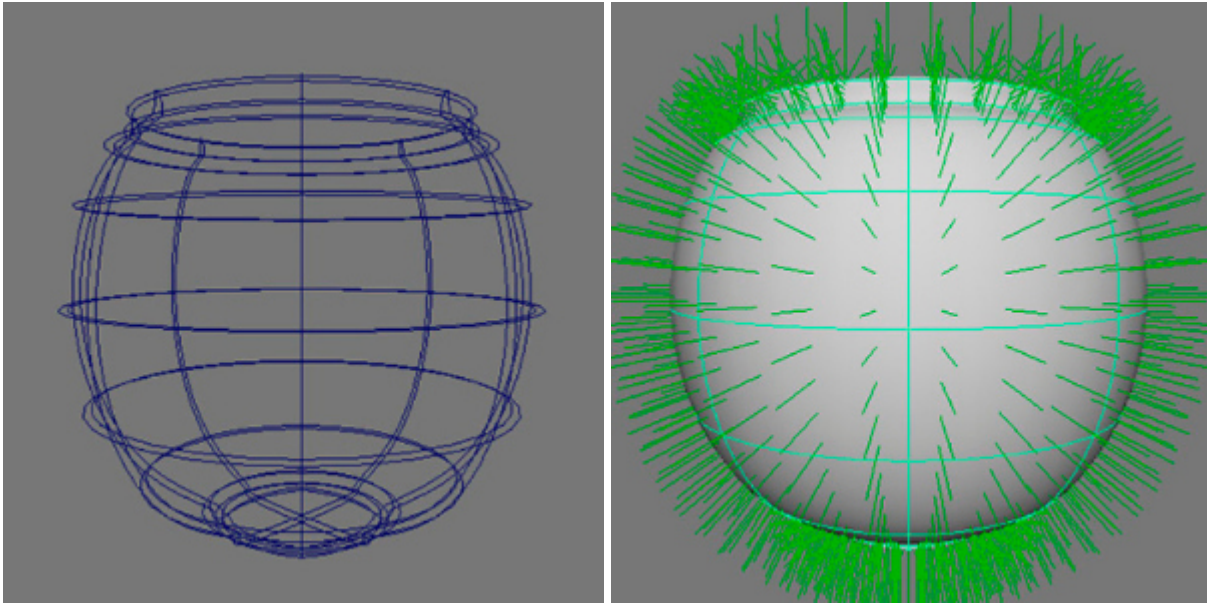


Liquid normal direction

A ray of light travels through the air (IOR = 1.0), enters the glass, and is refracted by the IOR of the glass (1.5). After traveling through the glass the ray leaves the glass and enters the liquid, i.e. it travels from one medium of IOR 1.5 to another medium of IOR 1.33.

Glass Surface Model

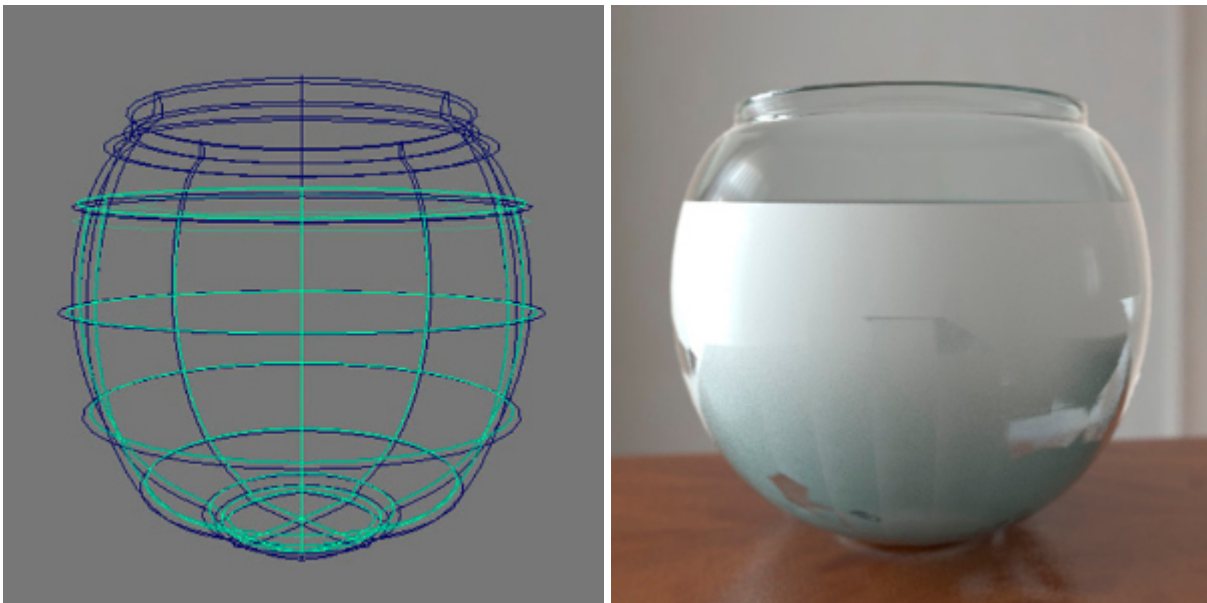
Below we have a simple glass bowl model. You can see in the image on the right that the normals of the mesh are correctly pointing outwards. The correct normal direction is important when rendering refractive objects with Arnold.



Surface normals are pointing outwards

Modeling Liquid -> Coincidental surfaces

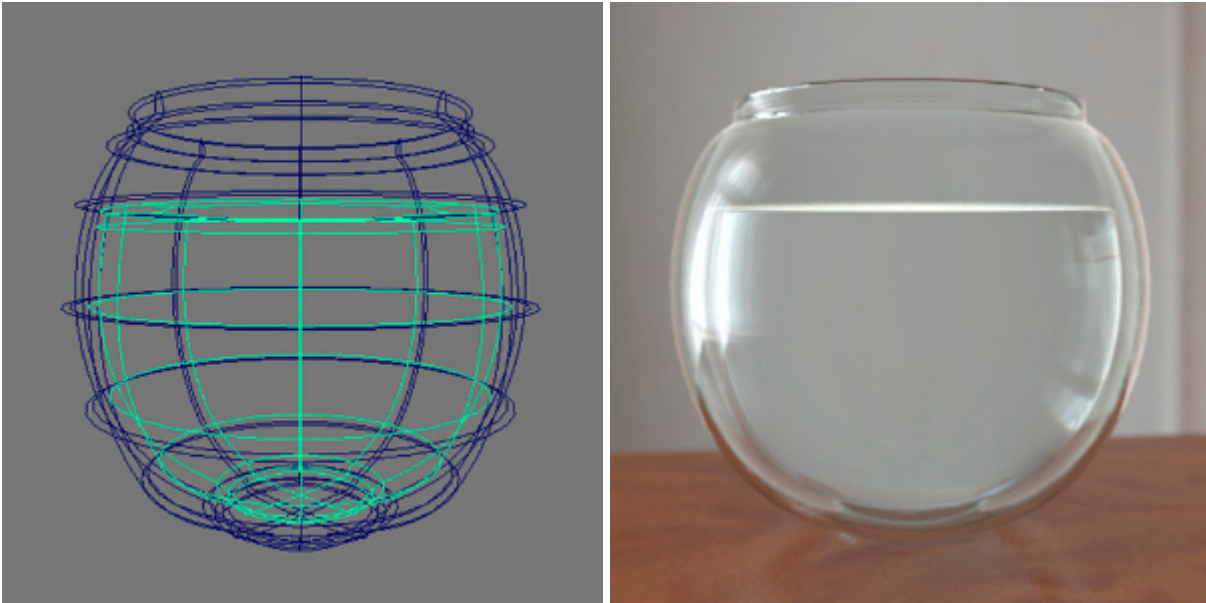
In the example below, the surfaces of both the glass and liquid are sitting on top of each other, and are therefore coincidental. The renderer has problems identifying between each surface, which will output as overlapping artifacts.



Glass and liquid surfaces are coincidental

Scale liquid down

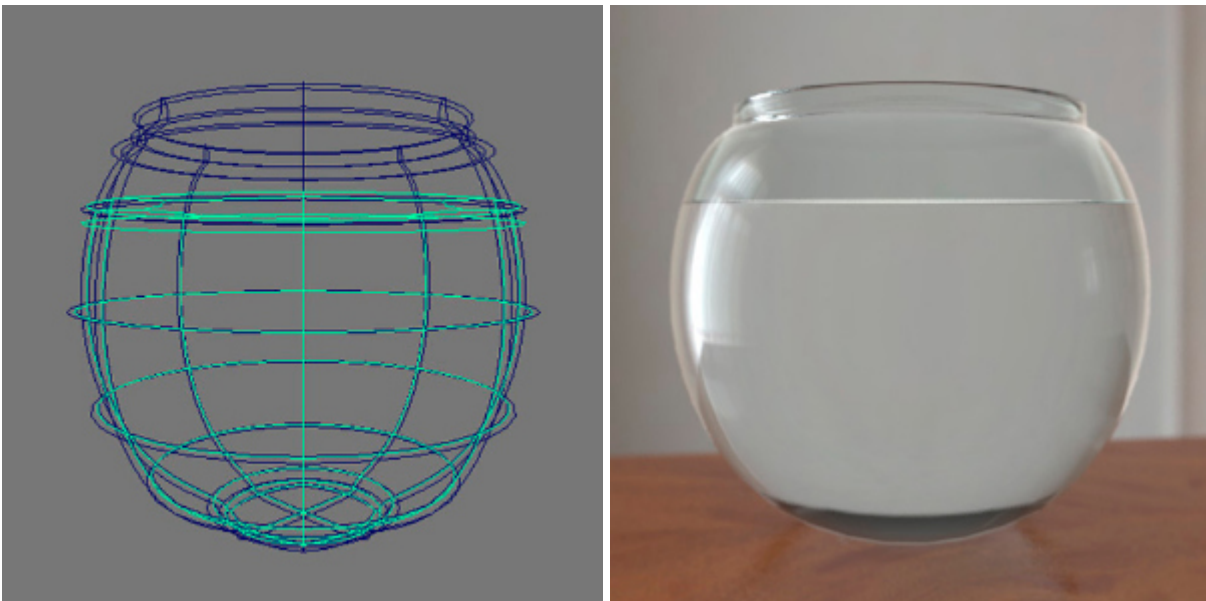
In this case, the liquid has been scaled down slightly. The gap between the glass and the liquid is very small. The resulting render does not look realistic.



Liquid is scaled down (looks wrong)

Scale liquid up

Here the scale of the liquid has been increased so that it overlaps the glass bowl geometry. This looks much better as the liquid appears to be touching the glass.



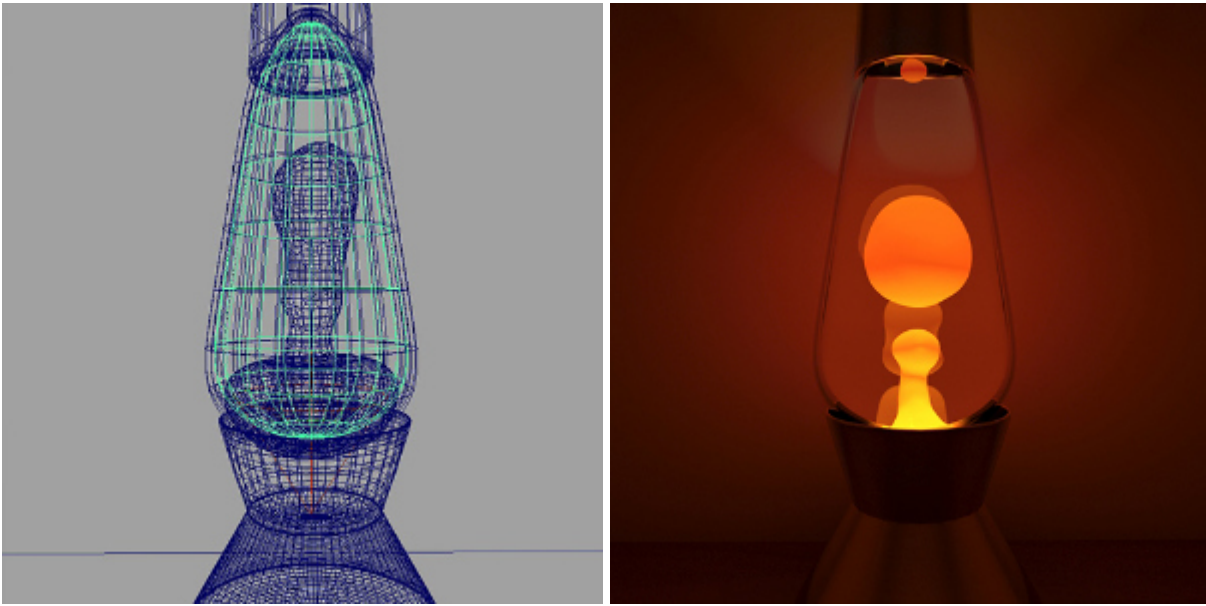
Liquid is scaled up



Note that even though this modeling approach does 'look' correct, you should be aware that this is not a physically based approach. In general, objects should be modeled as realistically as possible to attain correct results when rendering with Arnold.

Example 2

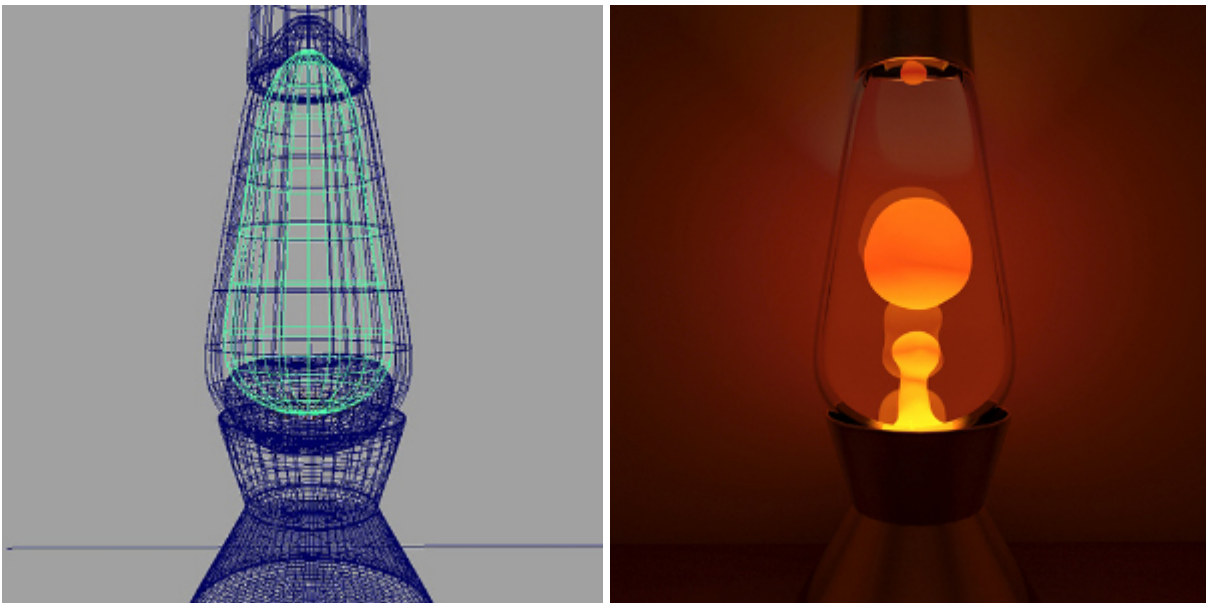
In the example below, the surfaces of both the glass and liquid are sitting on top of each other, and are therefore coincidental. You can see errors in the image on the right because the renderer has a problem identifying between each surface.



Glass and liquid surfaces are coincidental

•

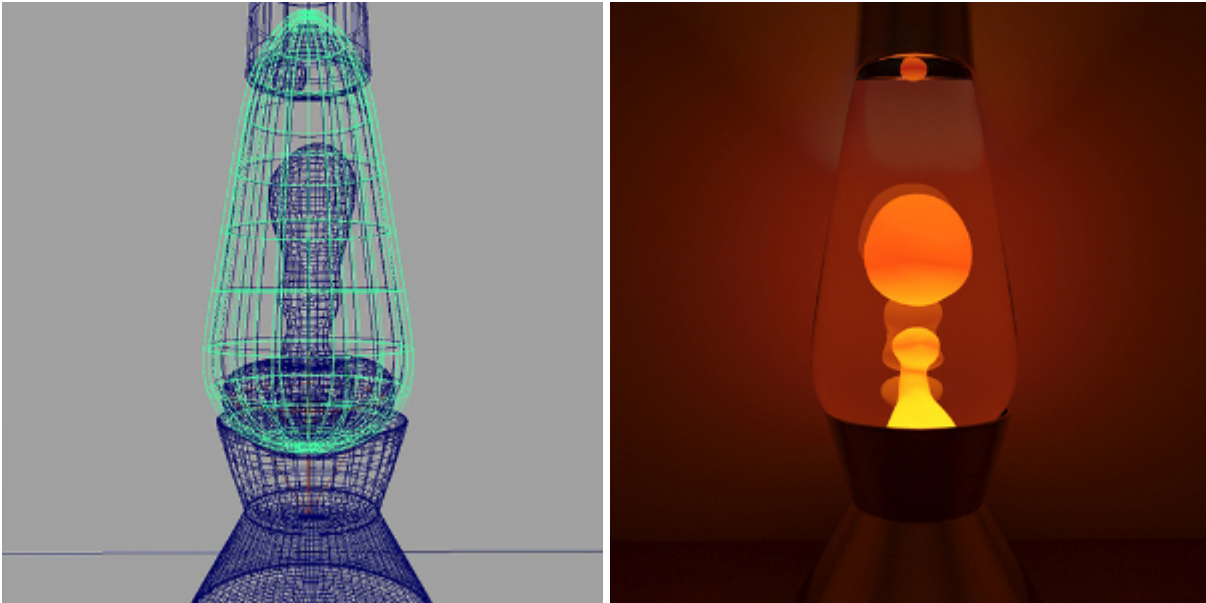
In this case, the liquid has been scaled down slightly. The gap between the glass and the liquid is very small. The resulting render does not look realistic.



Liquid is scaled down

•

Below, the scale of the liquid has been increased so that it overlaps the glass geometry. This looks much better as the liquid appears to be touching the glass.



Liquid is scaled up

Shading Glass and Liquid

Below are the material settings used to accurately shade the glass and water.

Opaque

MultiExcerpt named 'opaque ' was not found

The page: Shading Glass and Liquid was found, but the multiexcerpt named 'opaque ' was not found. Please check/update the page name used in the 'multiexcerpt-include macro.

Glass

Set the base_weight to 0 and increase the specular_weight to 1. Change the IOR to that of glass (1.5). Increase the transmission_weight to 1.

Transmission

Lowering the transmission_weight value attenuates the amount of light that passes through the volume of the glass (distance-based coloring). By default, transmission_color is white, which means no distance-based tint. Try lowering it from full white.



Transmission Color: Gray



Transmission Color: White

Water

Repeat the settings used for the glass material. However, make sure that the Index of Refraction is set to that of water (1.33).

Milky Liquid

To create a milky liquid, it helps to add small amounts of sub-surface scattering. Additionally, to make the liquid appear more vibrant, you could experiment by adding a small amount of emission_weight to it.



SSS Weight: 0.1



SSS Weight: 0.1. Emission 0.1

Render Settings



If you are experiencing long render times using glass surfaces, select the mesh and disable 'enable_internal_reflections' in the Advanced section of the standard_surface shader.

Transmission Depth

Correct transmission_depth settings are important for achieving realistic glass and water effects. Bear in mind that higher values will increase render time.

By default, Arnold uses a transmission_depth of 8.



2



4

The scene below contains many glass surfaces. Increasing the transmission_depth reveals more detail in the refracted glass.



2



4

Total Depth

When increasing ray depth settings, make sure that the total depth is set high enough to include the transmission_depth.

The total depth specifies the total maximum recursion depth of any ray in the scene (diffuse + transmission + specular < Total).

Final Render

Here is the final image. The transmission_depth has been left at the default value of 8. Higher values may be required to reduce further areas in the splashed water where the render appears solid black.



Shading a Rose



This tutorial explores how to shade a realistic looking rose material using the Standard Surface shader. We will look at some subtle (but important) shading attributes that make up the rose shader, including translucency and sub-surface scattering and some modeling considerations to bear in mind when rendering single-sided geometry.

Shading

The rose's Standard Surface shader has a base_color, specular, and SSS maps connected to it. A slightly modified duplicate color map has also been connected to the sub-surface color attribute. The backlighting effect is achieved by enabling Thin-Walled.

The animated gif below shows the effect that diffuse (base_color), specular (with high specular_roughness) and subsurface (SSS) scattering have on the look of the rose. The scene is lit with a directional light pointing towards the camera, thereby creating the translucent backlighting effect and a skydome_light with a sky HDR map connected to its color attribute.



Translucency

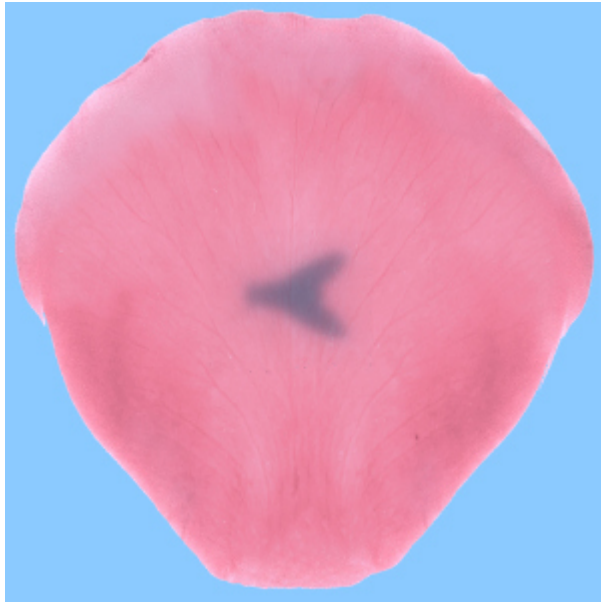
SSS combined with Thin Walled can replicate the effect of a translucent object being lit from behind (bear in mind this effect is more suited to single-sided objects). The images below show the difference when rendering the petal with increasing amounts of thickness. A fly model has been placed behind the petal, and the directional light is pointing towards the camera for maximum effect.



It is generally recommended that Thin Walled only be used with thin objects (single-sided geometry) as objects with thickness may render incorrectly. However, in this situation, it works fine with thickness (ensure that the diffuse ray depth level is above 1).



Mesh width scale: 1 unit



Mesh width scale: 10 units

Lighting

An HDRI is connected to the Color attribute of a Skydome light. The HDRI used in this example has been downloaded from www.hdrilabs.com. A directional light has also been used to add more sunlight into the scene.



Scene lit with a directional light and a sky shader with an HDR map

Shadow Terminator

Lighting the scene from behind with a strong directional light has introduced some artifacts at the shadow terminator. The current algorithm enabled by `shadow_terminator_fix` works well for convex surfaces like a sphere, but it can't fix self-shadowing in concave regions. There are ways to work around this.

Shadow Angle

Increasing the radius of the light (in this case `angle` because a directional light has been used for the sun) helps to reduce the effect of the shadow terminator (by softening the shadow). However, even when increasing the Angle to high amounts, the shadow terminator will still be apparent.



Shadow Angle 0



Shadow Angle 4

Subdivision Iterations

Another solution is to increase the number of `subdivision_iterations` of the petal geometry. The artifacts will become smaller.



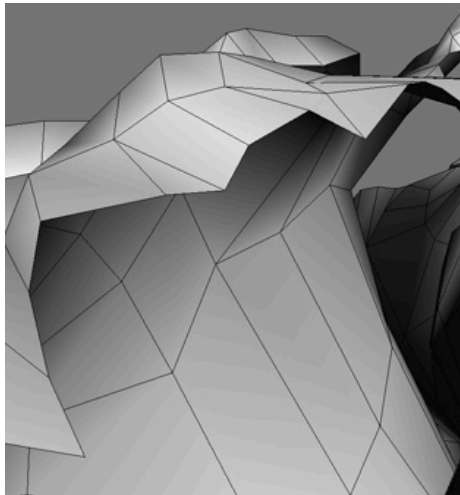
No subdivision



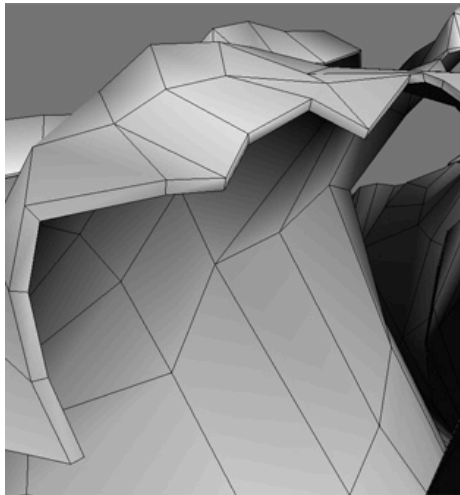
Subdivision Iterations 2

Extruded thickness

A better, more 'physically accurate' method is to add a small amount of thickness to the geometry by extruding the petals. The image on the left has petals that are only single-sided. When rendered, the petals appear paper-thin (more noticeable around the backscattered areas of the petals). The image on the right shows the results when the petals have extruded depth. The petals now appear to have some thickness when rendered.



Single sided (mesh has no thickness).



Mesh has extruded thickness.

Rendering

Indirect Diffuse Ray Depth

Increasing the `diffuse_ray_depth` value will add more bounced light into the scene. The rose appears much brighter after increasing the ray depth to 3. Be cautious when increasing the `diffuse_ray_depth` value because SSS is sensitive to this value. When you have many overlapping or enclosed surfaces (such as a rose), you will notice a visible effect when increasing the `diffuse_ray_depth` bounces to 2 or more.

Lowering the sub-surface scattering `value` will help to compensate for this increased brightness.



Diffuse ray depth 1



Diffuse ray depth 3

That concludes this tutorial on how to shade a rose. The techniques covered here can also be applied to other translucent materials such as skin, paper, and wax. •

Specular BRDF (Bidirectional Reflectance Distribution Function)

This tutorial explores various Specular settings found within the Standard Surface shader.

BRDF

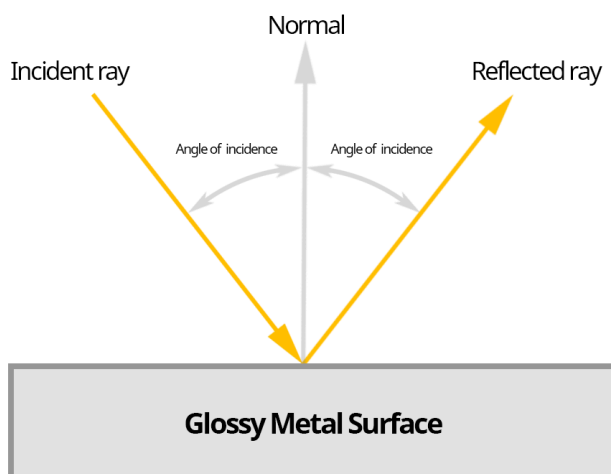
A bidirectional reflectance distribution function (BRDF) describes the reflectance properties of a surface by specifying the amount of radiance incident from one direction that is reflected into another direction, with respect to the surface normal.

The main characteristics of a physically plausible BRDF are the symmetry between the incident and reflected directions (Helmholtz reciprocity) and that the total reflected power for a given direction of incident radiation is less than or equal to the energy of the incident light (Energy conservation).

A Specular BRDF model is view dependant and can take into account anisotropy and Fresnel effects at grazing angles. Arnold uses a Cook-Torrance BRDF.

•

The light ray that hits a surface is called the Incident Ray and the angle that at which it hits is called the Angle of Incidence (as shown in the figure below). Light energy reflected back must be less than or equal to light incoming to a surface (we cannot "add" light). On the surface, the ray is either reflected or refracted, and it can eventually be absorbed by either medium.



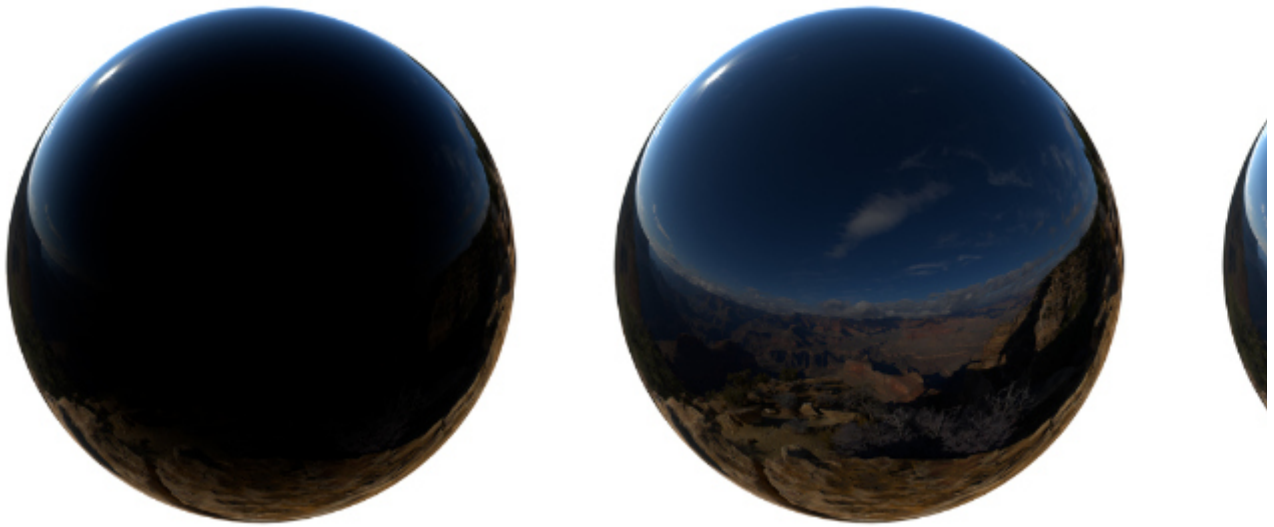
Angle of incidence, incident, and reflected rays

Reflectance at Normal

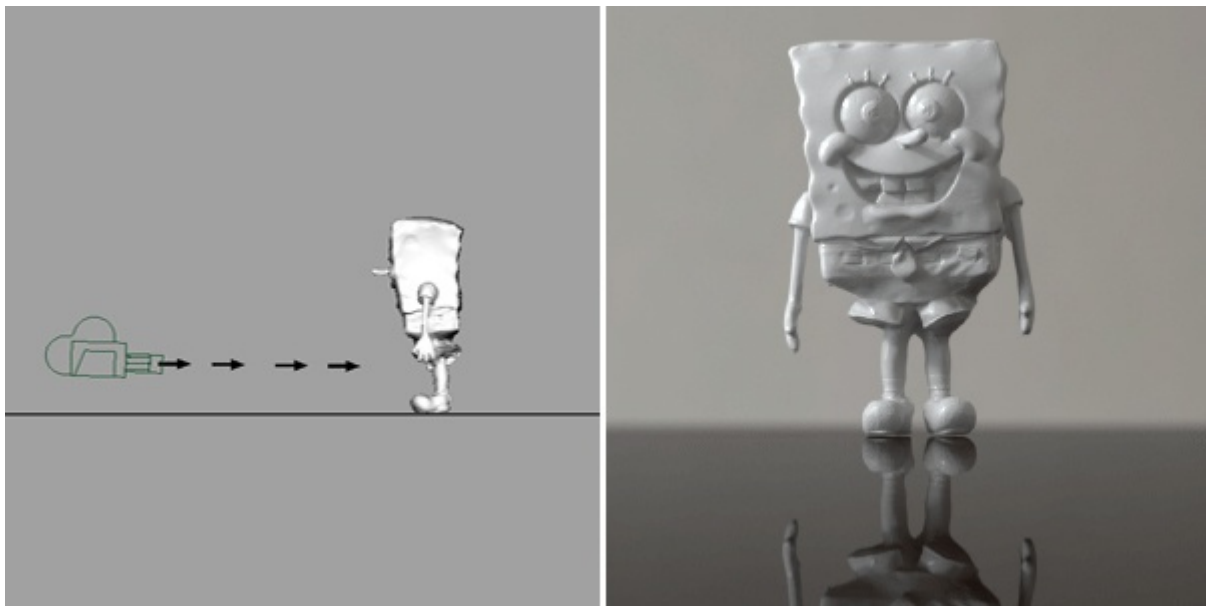


The Fresnel effect is named after the French physicist Augustin-Jean Fresnel who first described it. This effect states that the strength of reflections on a surface is dependent on the viewing angle. The amount of reflection increases on surfaces viewed at a glancing angle.

When IOR is greater than 1 in the Standard Surface shader, the reflectance of the object will be view dependent and vary obeying Fresnel equations. The Standard Surface shader uses Schlick's approximation of the fresnel equations and can be controlled by using the IOR of the material. At low values, the material behaves as a perfect dielectric, like plastic. With higher values, the material becomes a near-perfect conductor, or metallic surface.



Fresnel is present in all materials in some form, being the most common examples glass and water. The example below shows the effect of fresnel on a wooden table. Notice how the reflection on the table appears more reflective as the camera's viewing angle becomes shallower.



Variation of a Specular BRDF with respect to the view direction (rollover to view animation)

In the example below the cube has a Standard Surface shader assigned to using IOR. The Fresnel effect is visible. The specular glossy reflection appears brighter when the camera's viewing angle becomes shallower.



Rollover to view animation

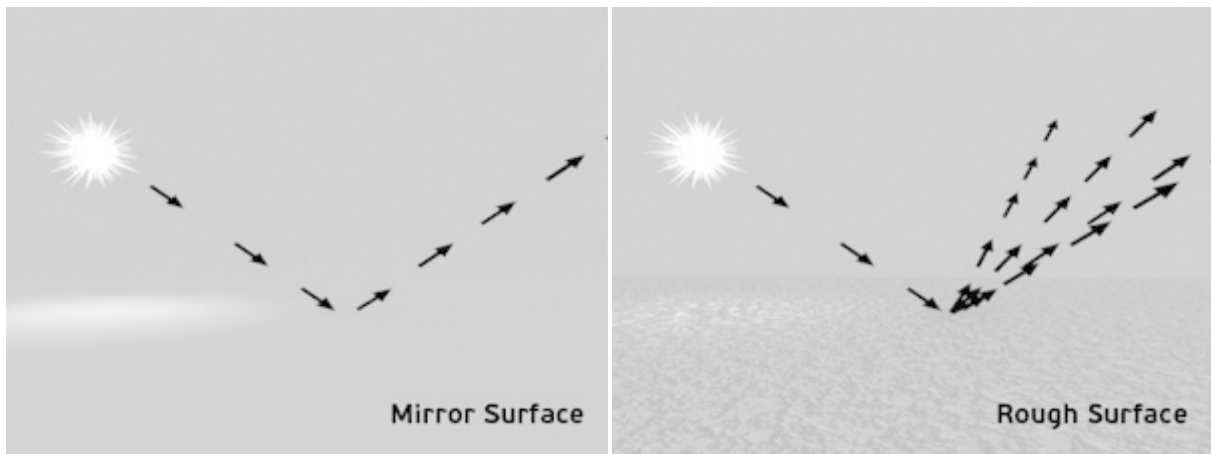
Micro-Facet Surfaces

Cook-Torrance is a micro-facet BRDF based on Micro-facet Models (See for instance "[Microfacet Models for Refraction through Rough Surfaces](#)" by Walter, Marschner, Li and Torrance, Eurographics 2007).

These models consider that surfaces that are not perfectly smooth and are composed of many very tiny facets, each of which is a perfect specular reflector.

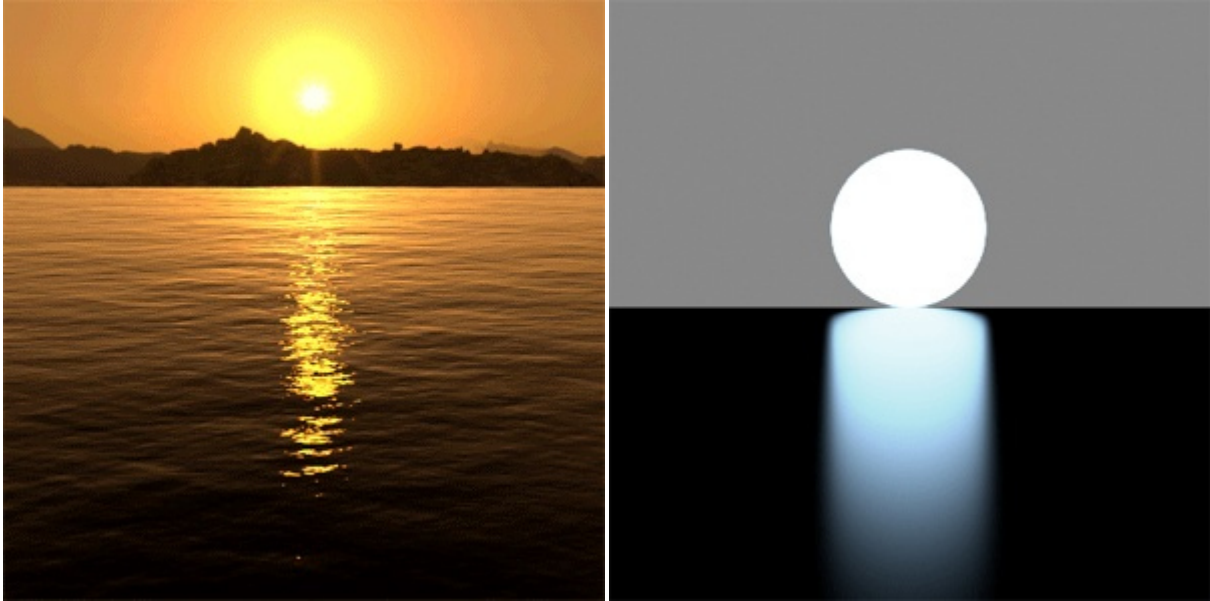
These microfacets have normals that are distributed about the normal of the approximating smooth surface. The degree to which microfacet normals differ from the smooth surface normal is determined by the roughness of the surface.

The animations below show how surface roughness can affect specular reflections.



Rollover to view animation

•
In the examples below, you can see that the specular reflection appears stretched when the surface becomes 'bumpy' or 'rougher'.



Rollover to view animation

Anisotropy and Roughness

An example of an anisotropic material would be asphalt concrete. This effect is evident on the surface of a wet road. Notice that the wet surface appears diffusely darker and specularly brighter. When you are far from the surface, the stretched anisotropic reflection is more obvious on the ground. However, when you get closer, the stretched reflection becomes smaller due to the change in viewing angle. The anisotropic highlight size depends on the viewing angle.



Dry, diffuse road surface vs. wet anisotropic road surface

The example animations below show the effect of specular roughness when using anisotropy (0, 0.5, 1). The roughness values have been animated from 0 to 1 to demonstrate the effect.



Anisotropy values left to right: 0, 0.5, 1 (rollover to view animations).

Anisotropy Example

The images below show the difference between rendering with and without anisotropy on a floor surface. Both shaders have Roughness set to 0.2.



Floor shader Anisotropy: 0



Floor shader Anisotropy: 0.8

Water Bottle



This page shows a set of more advanced tips for rendering liquids in a plastic bottle. It explores the various shading aspects required to achieve realistic results. It also looks at ways to use the 'Rayswitch' shader to minimize the calculation of certain rays. This way render times are reduced without sacrificing render quality.

There are various aspects of shading that need to be considered when tackling liquid and container surfaces. If physical accuracy is a priority, then the default Standard Surface shader settings should be sufficient. However, enabling Fresnel with Reflectance at Normal is essential to give the plastic a cleaner, more realistic look and is probably more suited to a photographic studio image.

If the speed of rendering is the primary consideration, then you may want to look into disabling some attributes (Enable Internal Reflections, Caustics, Bounce Factor 0). To reduce render times even further you should consider using a rayswitch shader that disables expensive rendering attributes such as Transmission roughness from secondary rays.

Before reading this page, it is recommended that you first look at the tutorial 'Rendering Glass Surfaces with Arnold' to be more familiar with the modeling requirements for rendering glass and liquid surfaces.

Scene Setup

The Transmission Ray Depth has been increased to 7 to allow enough rays to pass through both the plastic bottle and the liquid surface. The scene is lit with an Area Light with its Light Shape set to Cylinder. There is also a Sky in the scene.

Shading the Bottle

The examples below show the effect of changing various attributes of the Standard Surface shader used for the plastic bottle shader.

Enable Internal Reflections

Unchecking internal reflections will disable indirect specular and mirror perfect reflection computations in secondary bounces.

Bear in mind that even though disabling 'Enable Internal Reflections' can dramatically reduce render times, it will also affect the appearance of the plastic. Therefore, you should ask whether accurate quality or speed is more important before using this feature.



Enable Internal Reflections: On (17 mins 45 secs).

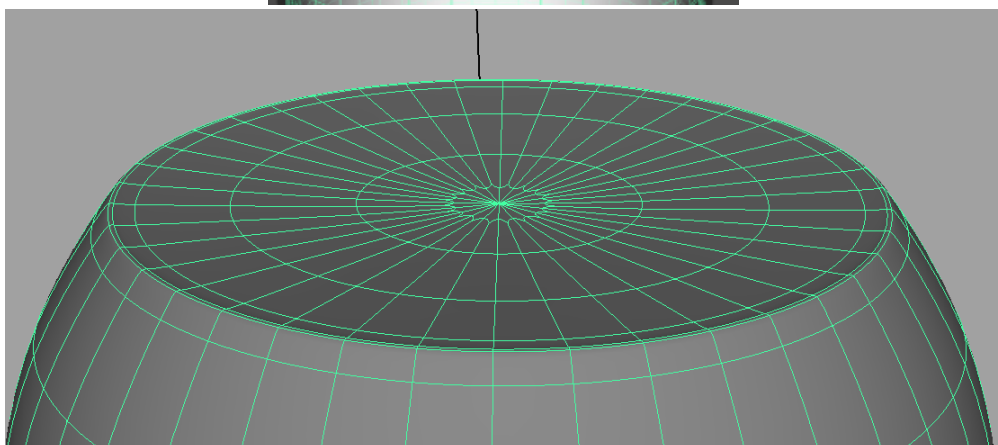
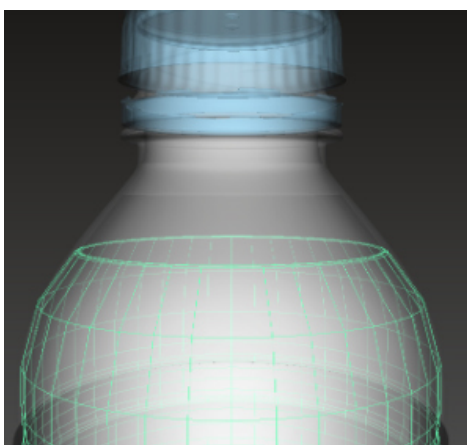


Enable Internal Reflections: Off (7 mins).

Shading the Water

Liquid Meniscus • ••

The top of the liquid has been modeled accurately with a meniscus. A meniscus is a curve in the upper surface of a liquid close to the surface of the container or another object, caused by surface tension. ••



Model the liquid meniscus. Ensure you have enough edge loops to pull the top of the surface down.

Note that the plastic shader will affect the light rays going through the plastic and will have an affect on the liquid inside. Therefore you should ensure that the Transmission•Ray Depth •settings are high enough for rays to travel through both the plastic and liquid surfaces. The•Index of Refraction •has been changed to that of water:•1.33. The•Base Weight •has been lowered to•0.9. •

The examples below show the•effect of changing various Standard attributes for the•water shader. •

Base Weight

By default, the water looks a little dark. Adding a small amount of•diffuse•helps to 'lighten' the appearance of the water (note that this is not a physically accurate approach). •



Base Weight: 0



Base Weight: 0.2

Specular

Specular •Weight• has been left at•0•as we do not want the water to show specular reflections inside the bottle.



Specular 0 (water looks correct).



Specular 1 (water looks incorrect).

Enable Internal Reflections

The images below show the difference between rendering the water surface with and without **Enable Internal Reflections**. The differences are subtle but can be seen mainly around the base of the bottle.



Enable Internal Reflections: On



Enable Internal Reflections: Off

Caustics

The **Caustics** menu can be found in the **Standard Surface** shader. This switch in the **Standard Surface** shader specifies whether the diffuse GI rays can 'see' specular reflection and transmission rays (by default only the direct and indirect diffuse lighting of the **Standard Surface** shader will be computed in the diffuse rays).

The images below show the difference between rendering the bottle and liquid with and without caustics.



Caustics Disabled



Caustics Enabled

Cloudy Water

To make the water appear cloudy, the 'Transmission Roughness' has been increased to 0.4. Increasing this value will add to the render times. A 'Rayswitch' shader has been used to keep render times to a minimum.

Rayswitch

The **Rayswitch** shader can help to reduce the number of rays calculated in the scene, thereby improving render times without sacrificing quality.

The primary rays need to refract because of the cloudy•roughness required•to make the liquid look cloudy. Whereas the secondary glossy, shadow and diffuse rays are using a second shader (Water_Secondary_Rays) that does not have specular/ transmission roughness (this decreases render time).

As you can see in the images below, there is little difference in render quality and yet the render times have been reduced dramatically.

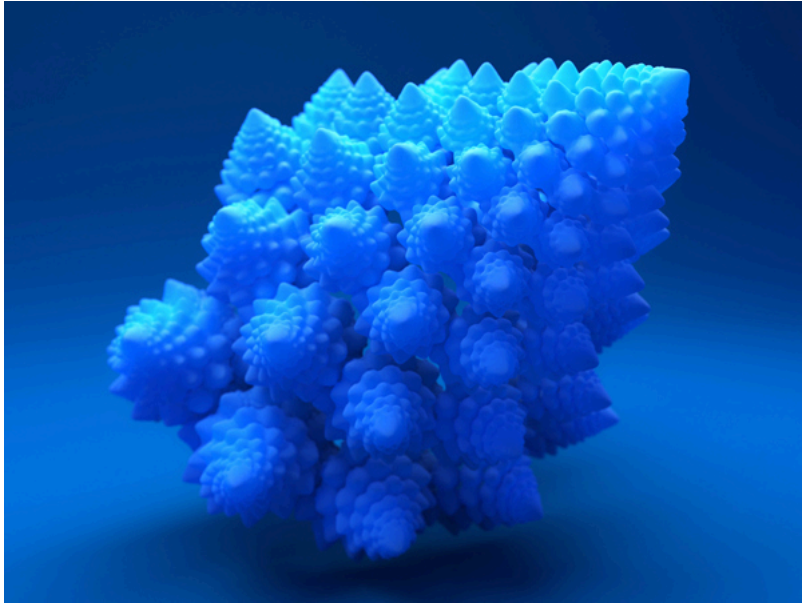


No Rayswitch shader: 31 mins 15 secs



Rayswitch shader: 16 mins 14 secs

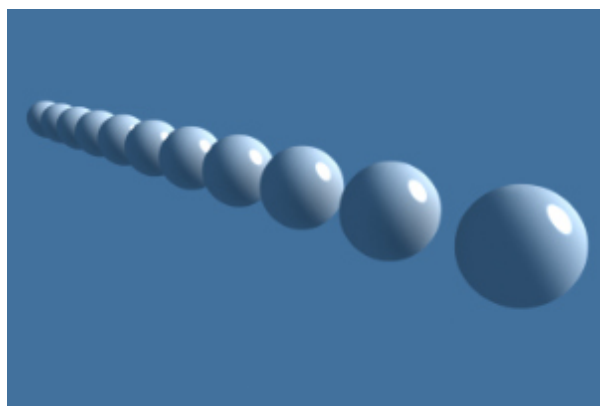
Rendering



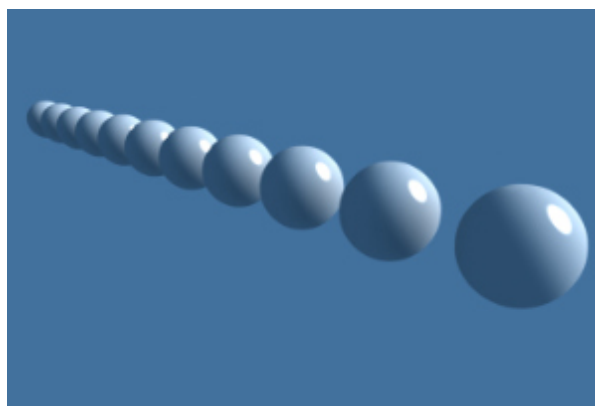
- [Depth of Field](#)
- [How to Render a Mandelbulb with SitoA](#)
- [ICE Time Shifted Instances](#)
- [Rendering Maya Fluids with Softimage](#)
- [Rendering an Ocean with Displacement](#)
- [Subdivision Tutorial](#)

Depth of Field

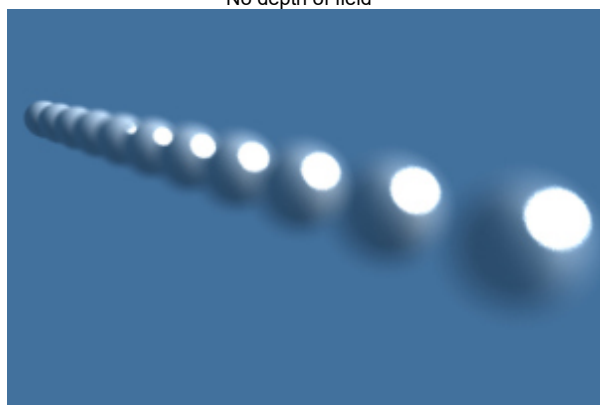
The following images demonstrate the effect of different camera depth of field values. The examples below use a sphere which has a radius of 1 (remember that scene scale size can affect the DOF values that you use).



No depth of field



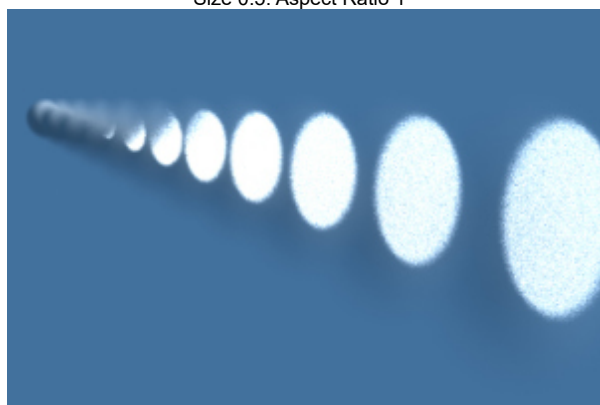
Size 0.01. Aspect Ratio 1



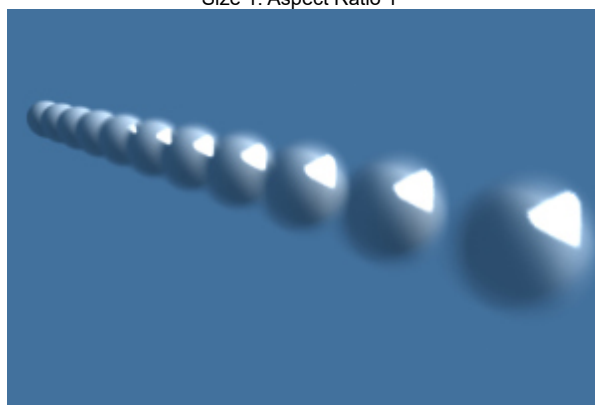
Size 0.5. Aspect Ratio 1



Size 1. Aspect Ratio 1



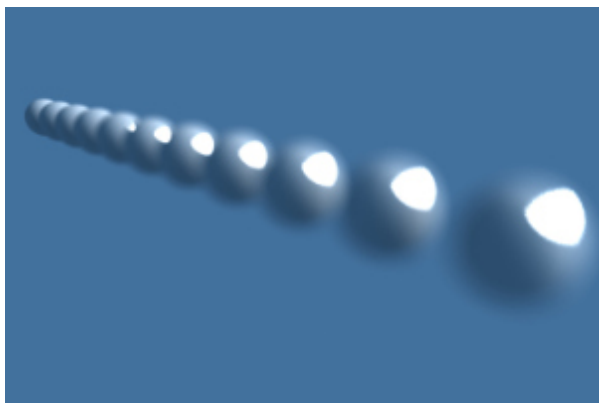
Size 1. Aspect Ratio 2



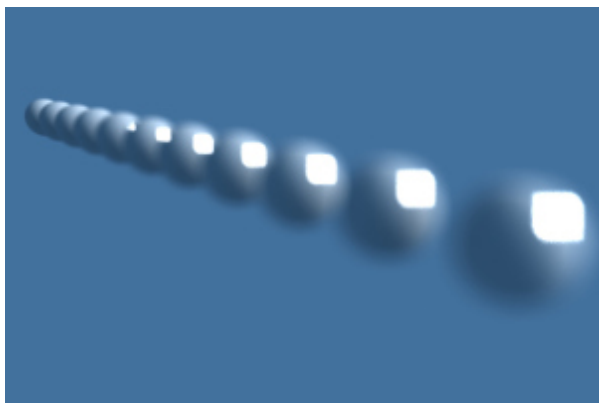
Size 0.5. Aspect Ratio 1. Blades 3



Size



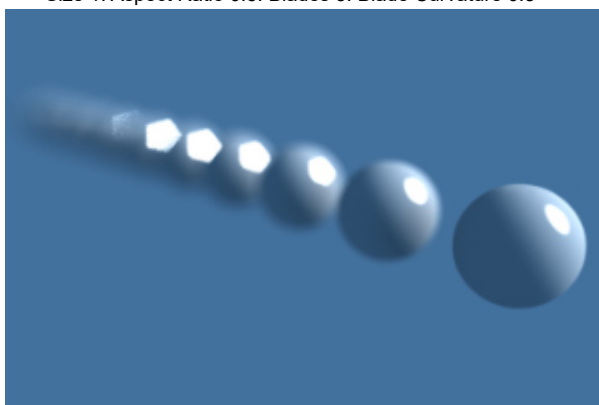
Size 1. Aspect Ratio 0.5. Blades 3. Blade Curvature 0.5



Size 1. Aspect Ratio 0.5. Blades 4

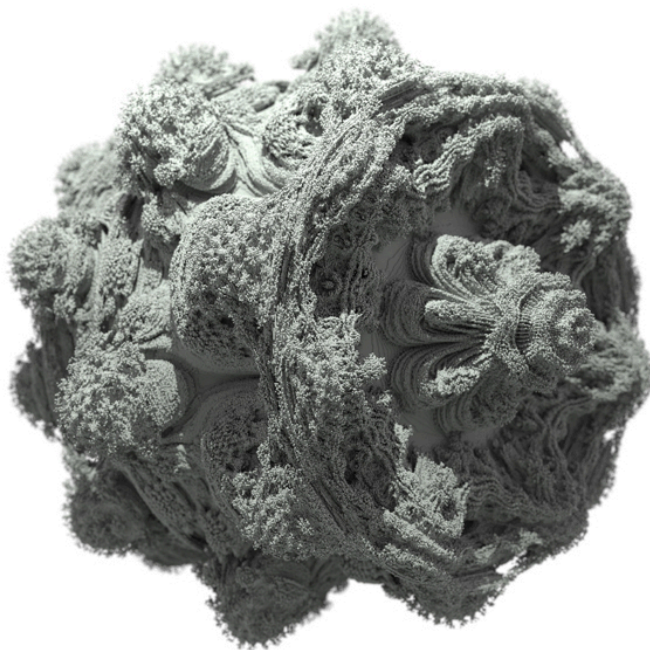


S

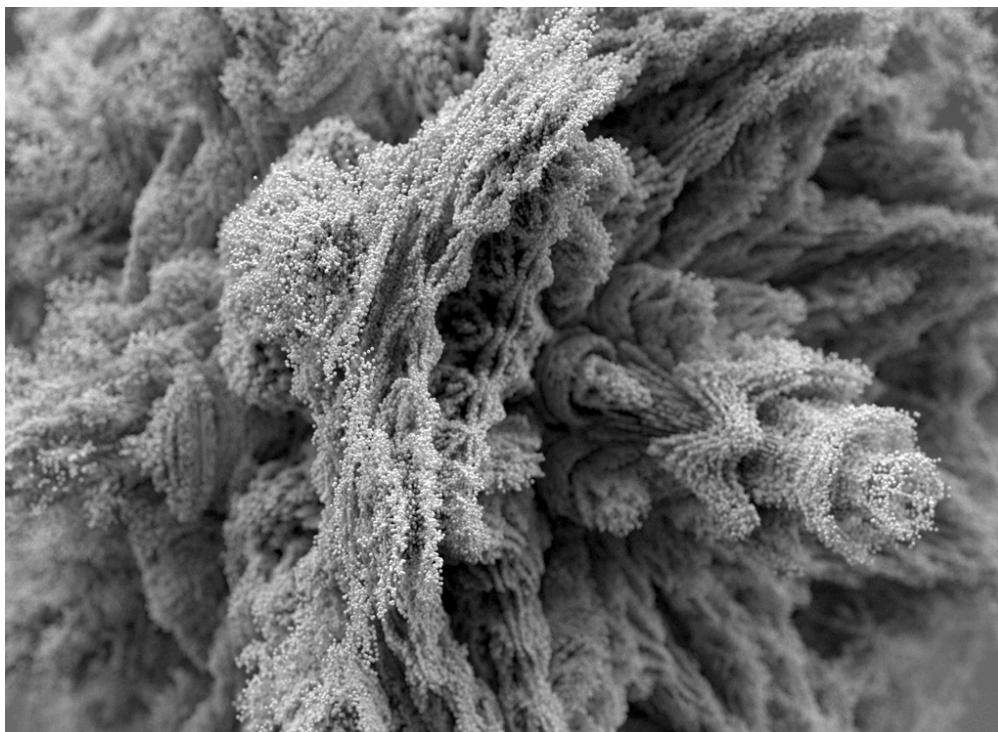


Distance 8.5. Size 1. Aspect Ratio 0.5. Blades 3. Blade Curvature -5

How to Render a Mandelbulb with SitoA



This short tutorial covers how to render a Mandelbulb via a .ass file. We are generating a "Mandelbulb" 3D version of the Mandelbrot and Julia sets. For more information about how to generate the Mandelbulb, see the [Large Datasets from Procedurals](#) tutorial.



A closeup of the Mandelbulb. You can clearly see that it is formed of thousands of tiny spheres.

A Julia set is the set of points on the complex plane that when squared and added iteratively to a constant (C), the 'prisoner' points fail to leave a circle of radius 2. There is a full Julia set for all possible values of C; C is a 2-dimensional value on the complex plane.

Julia sets come in 2 different types:

1. Completely connected.
2. A disconnected Cantor dust of points (for more on Cantor dust -- http://en.wikipedia.org/wiki/Cantor_set

If the center point (0,0) of a Julia set is a prisoner point, then the Julia set is completely connected; if it is not a prisoner point the Julia set is a Cantor dust.

The Mandelbrot set is the set of all fully connected Julia sets. It is generated by sampling the center point of all possible Julia sets. That means, for every point on a Mandelbrot set, you can generate a Julia set. Although both sets are continuous functions in the complex plane, they are usually rendered by sampling an even grid (pixels), and if the point is a prisoner point, it is rendered black.

The Mandelbulb extends this math into three dimensions. This generator works by sampling all points in a regular grid, and if the point is a prisoner point, it adds it to an AiPoints geometry node for rendering.

You will need the following files to create and render the Mandelbulb:

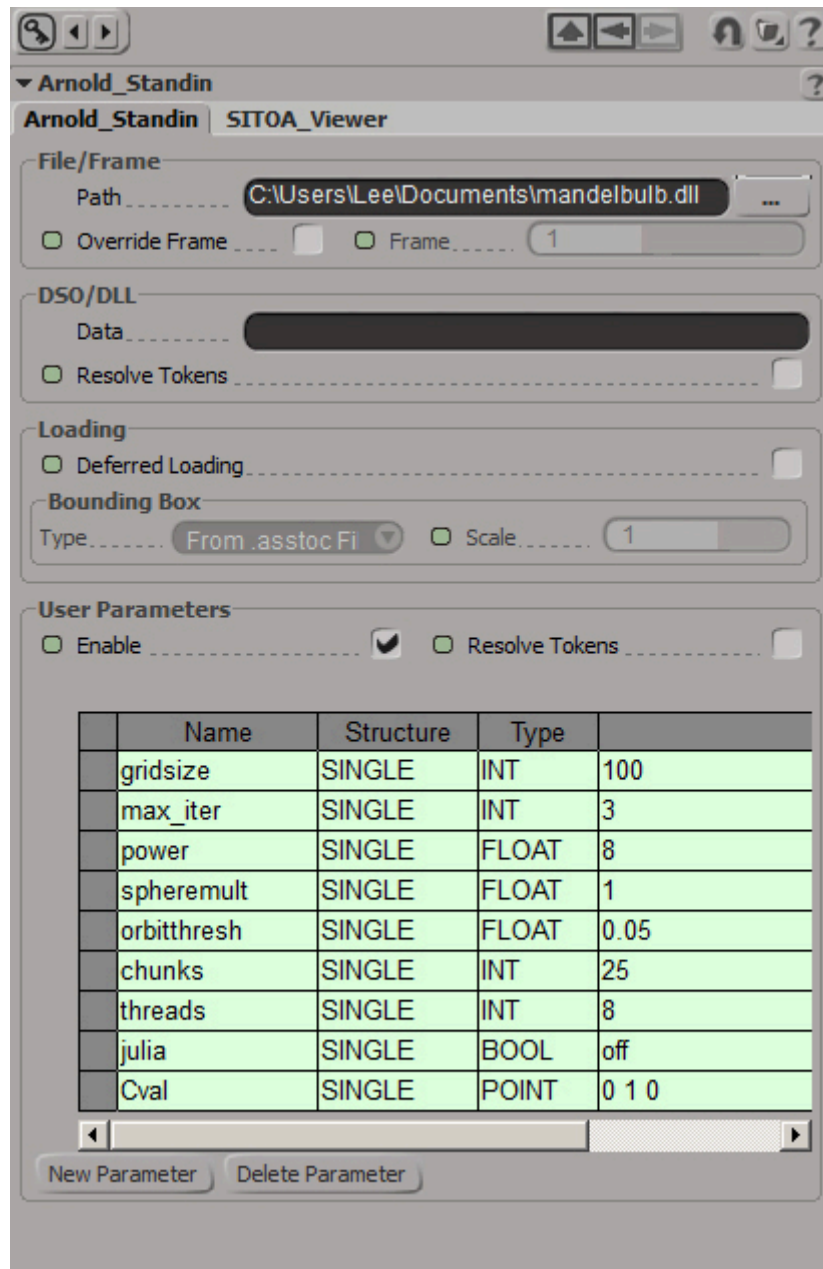
- [Compiled Mandelbulb procedural](#)
 - [Arnold 5: Windows](#) • [Mac](#) | [Linux](#)
 - [Arnold 6: All platforms](#)



Ensure that you are using an up to date version of Arnold before following this tutorial. Otherwise, you may encounter problems when rendering the Mandelbulb.

Ass file

- Start off by creating an Arnold Standin by creating a polygon cube and then selecting Arnold>Properties>Standin.
- Select the Standin and locate the Path of the '[mandelbulb.dll](#)' file.
- The image below shows the User Parameters that have been added to the Standin:

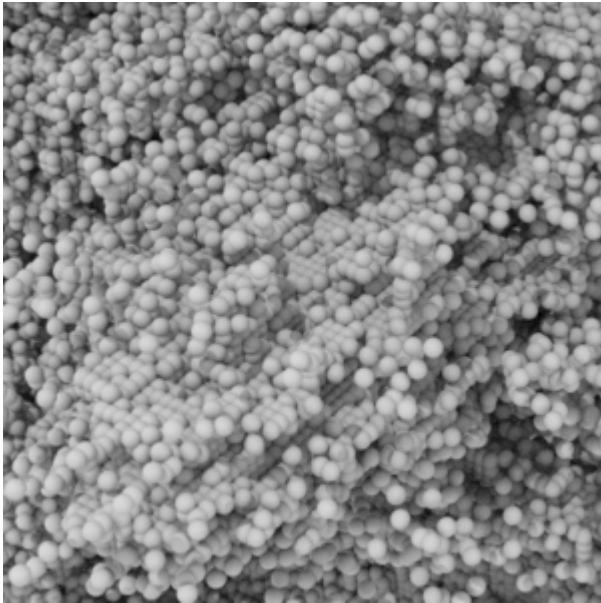


- Create a light and render the scene. It will take a moment before rendering in order to generate the Mandelbulb.

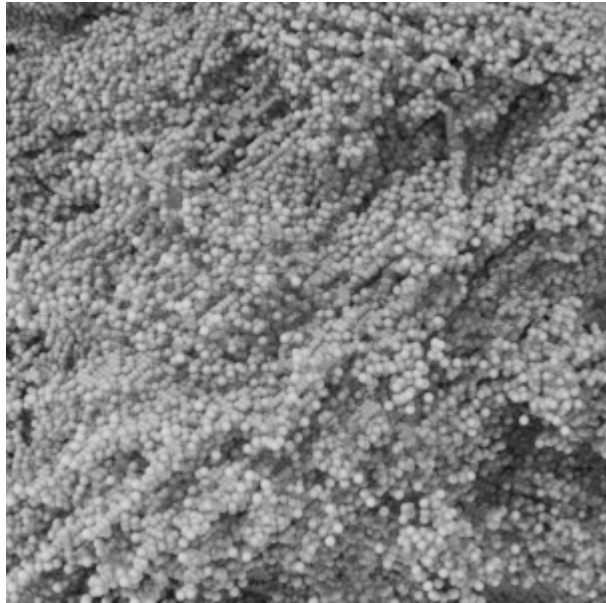
Gridsize

Gridsize is the number of cells in the grid, i.e. 800 x 800 x 800.

To increase the resolution of the Mandelbulb, you will need to increase the grid size. Higher numbers will create more spheres. Be careful when increasing this number as memory use and render times will increase.



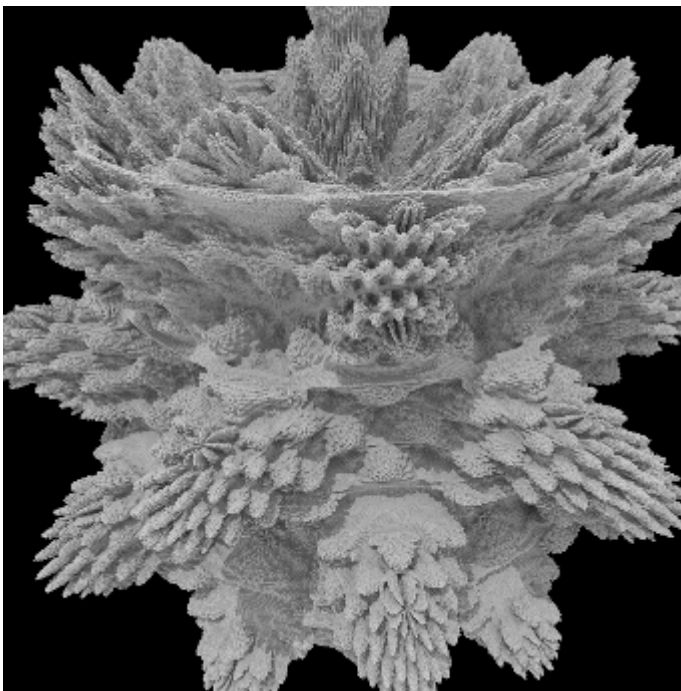
Gridsize: 400



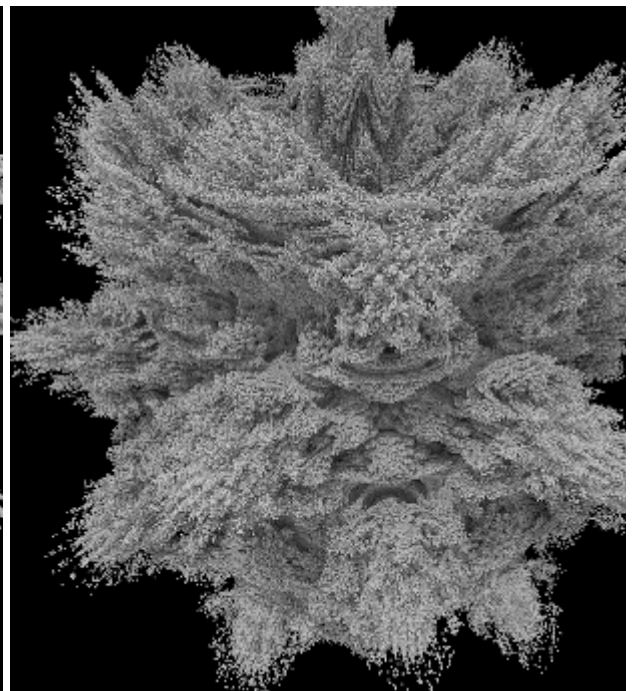
Gridsize: 800

Maximum iterations

Maximum number of iterations to try. If after this number of iterations, the point is still inside a sphere of radius 2, then it is assumed to be a prisoner point.



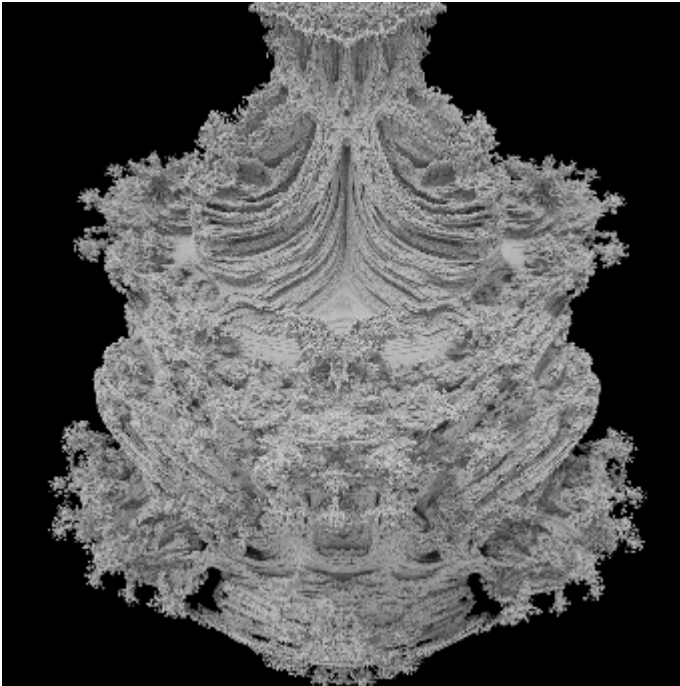
Max iterations: 3



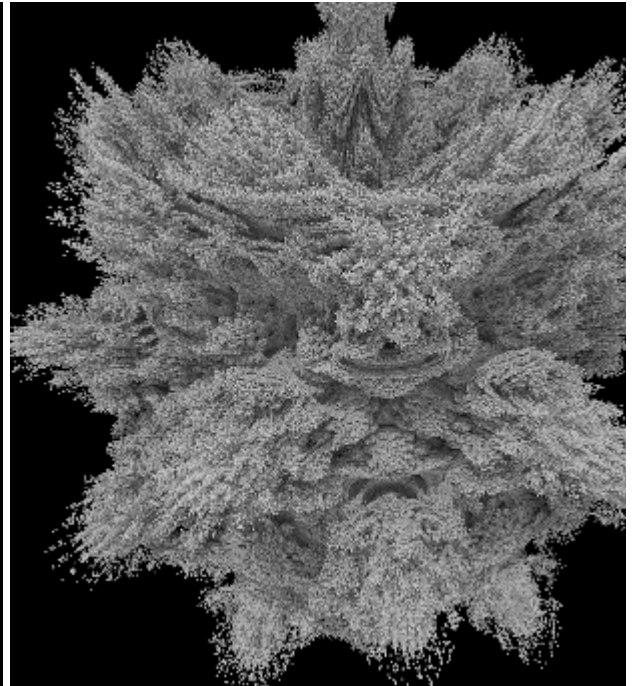
Max iterations: 9

Power

In two dimensions, Julia sets, and Mandelbrot sets are computed by iterating Z raised to a power of 2 added to C . In three dimensions, raising to a power of 8 gives more interesting results, but there is a control to try changing this if desired.



Power: 4



Power: 8

Spheremult

This is a multiplier for the radii of the spheres in each cell; larger spheres will blend better, but will make it 'puffy', smaller spheres will show the grid more.



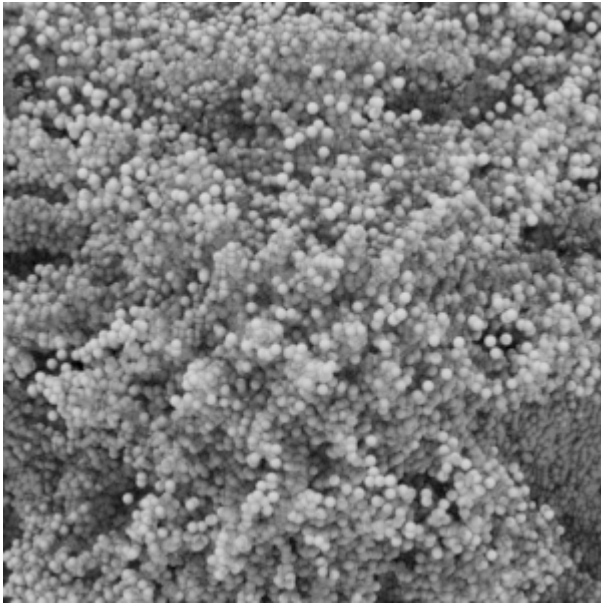
Spheremult: 1



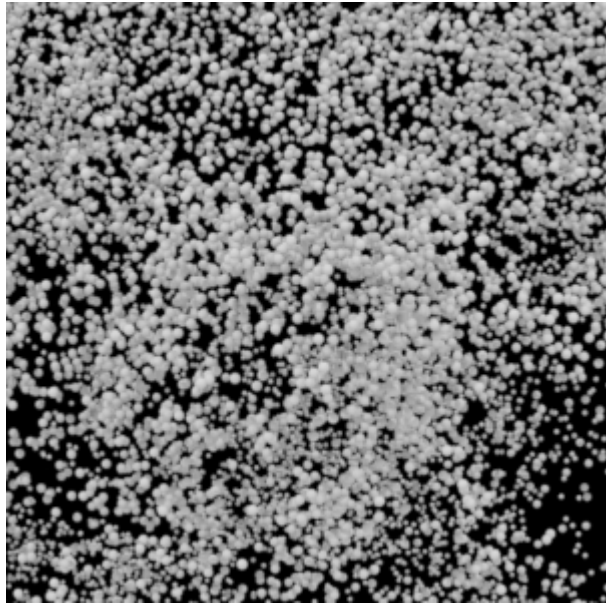
Spheremult: 2

Orbitthresh

If a point doesn't iterate out farther than this distance, it is assumed to be well inside the Mandelbulb and will not be added for rendering; this clears out a large hollow area in the center of the Mandelbulb, making the geometry a bit lighter.



Orbitthresh: 0.05



Orbitthresh: 2

Chunks

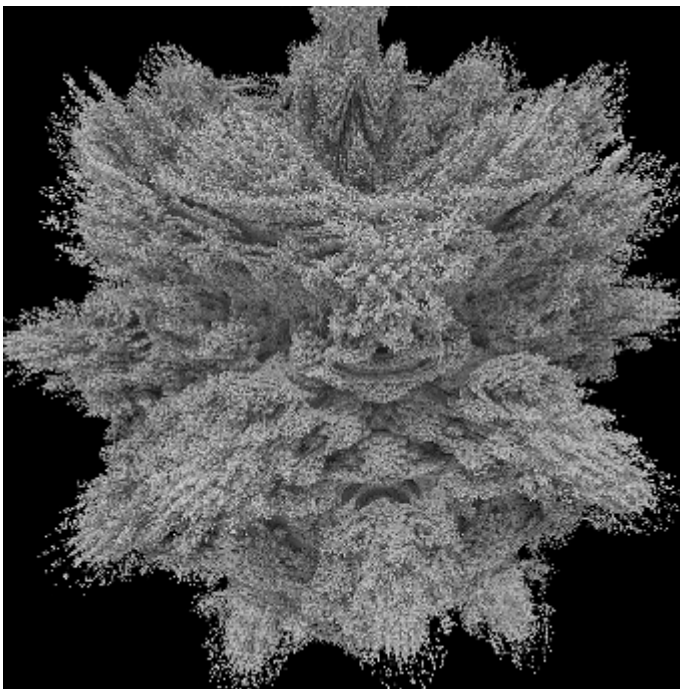
This is the number of chunks to break the Mandelbulb into, so you are not building the entire set in ram at once. Higher numbers should use less ram, with a saturation point at around 30.

Threads

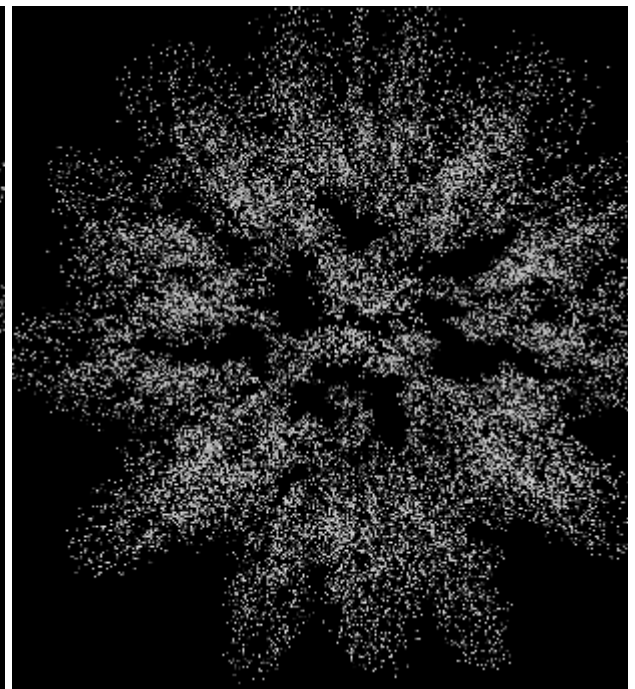
The number of threads to spawn when sampling values on the grid.

Julia

A switch to render as a Mandelbulb, or a julia set



Julia: Off



Julia: On

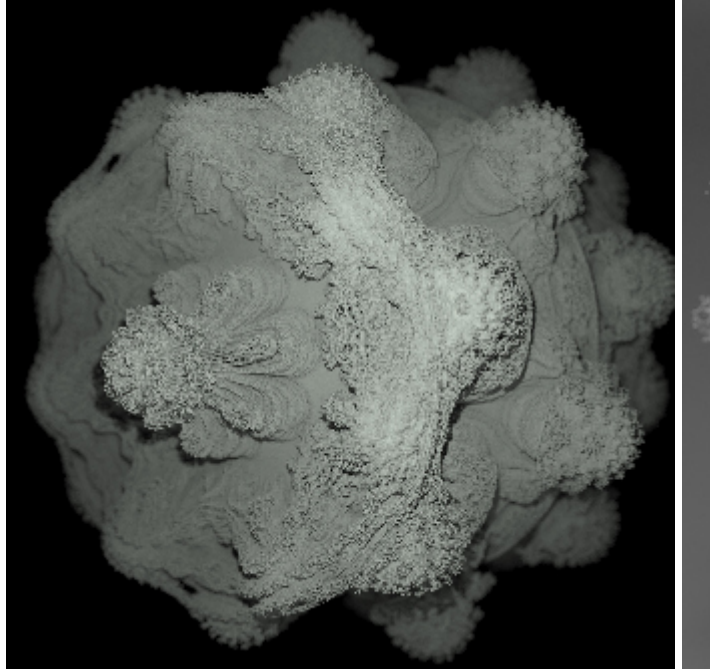
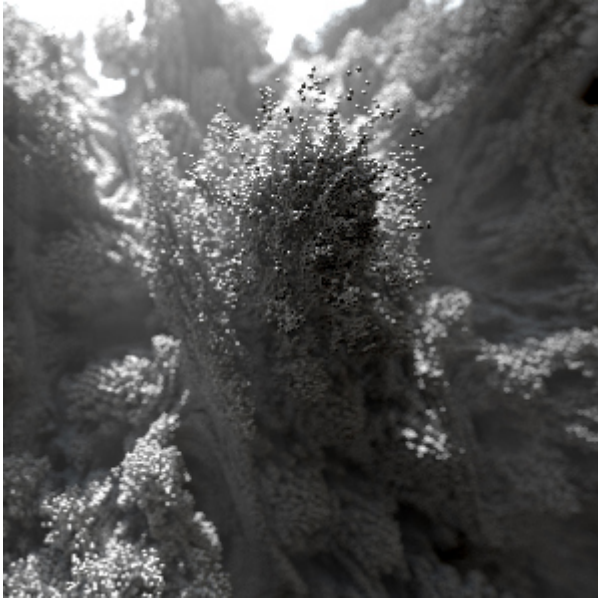
Cval

A value for C used to generate a julia set; ignored if julia == off.

Examples

That concludes this tutorial. Have a go at increasing the grid size in the .ass file and adding some more interesting lighting effects.

•



ICE Time Shifted Instances

MBC Korea reported a problem with a crowd scene that took way too much to start rendering in SlttoA.

The crowd was made of a set of warriors running through a field. A single warrior model was animated and instanced over a flow of particles.

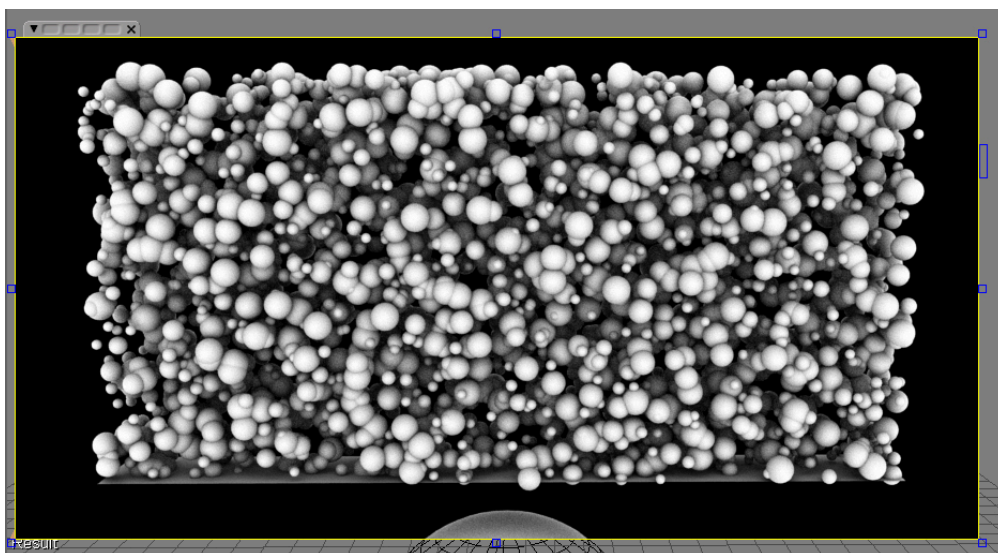
In order to displace the animation of each actor randomly, some of the ICE nodes (eg Set Instance Geometry or Control Instance Animation) were driven by a time randomizing node.

With massive crowds (>5000 actors), SlttoA was taking too long to translate/render the scene (the render itself was fast).

Also, if exporting to ass, the size of the file was way too big, and increasing proportionally to the number of actors.

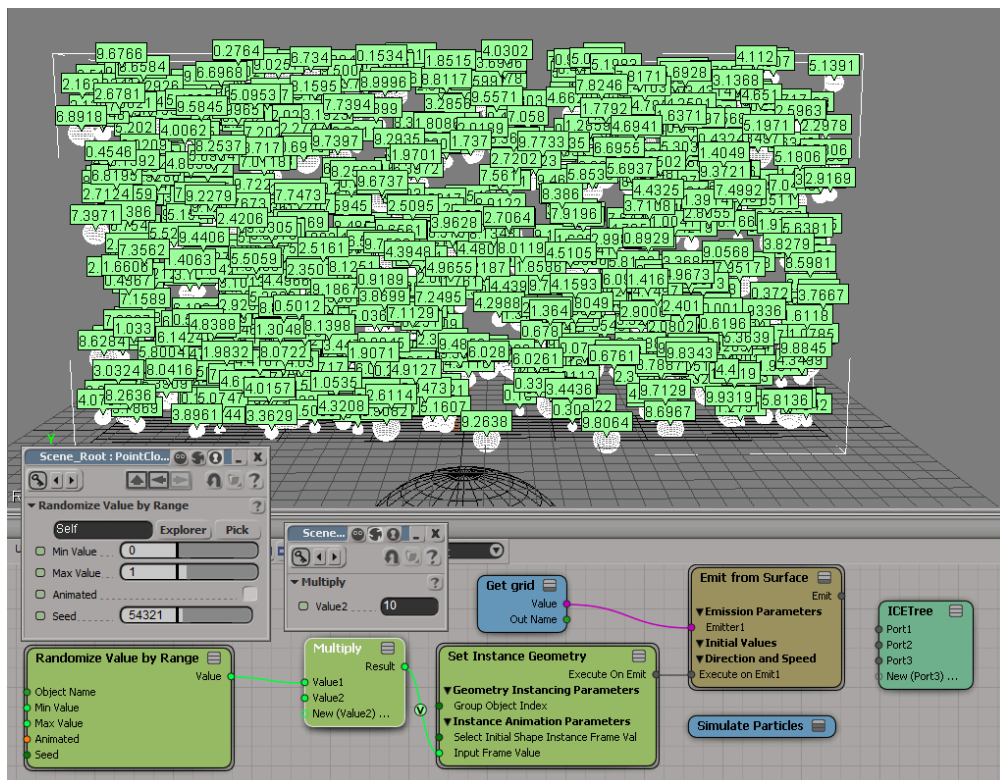
In order to explain the problem and how to fix it, we can use a very basic scene.

We have a sphere animated by a push operator from frame 0 to 10 and instanced over a simple point cloud.



The Input Frame Value of the instancing node is driven by a randomizer (0..1) that then is multiplied by 10, so resulting in a random value between 0 and 10. As you can see, no two instances share the same time of evaluation (meaning that the randomizing node works well!).

This scene takes long to start rendering, and the waiting time keeps increasing with larger amounts of particles.



When exporting to Arnold, SltToA checks for each instance if the instanced geometry has already been exported. If so, an instance is created, pointing to the instanced geo. Else, the instanced geometry is exported as a polymesh.

The existence check is performed against the object name and the evaluation time.

Also, keep in mind that Arnold does not support any kind of animation. A polymesh is exported with constant values for each vertex, and the animation values for a given frame only contribute to the motion blur evaluation.

So, here comes the problem. In the example above, no two objects share the same evaluation time, so each sphere is exported as a standalone shape.

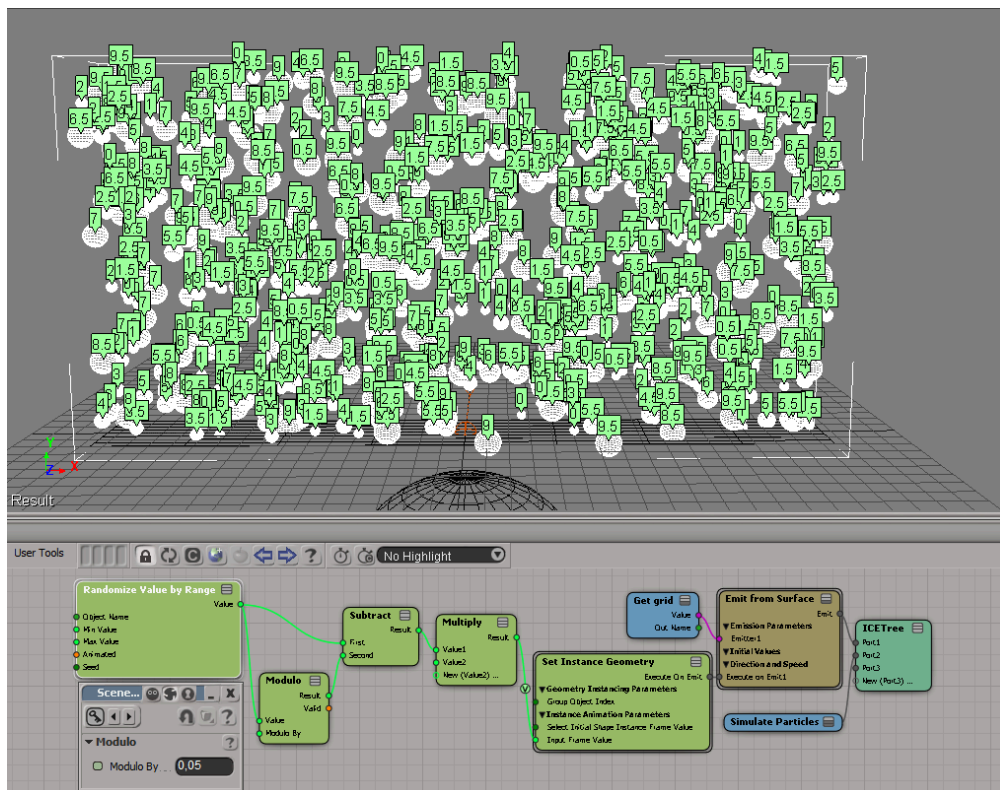
The time for a render to start in Softimage (or the size of the exported ass file) is simply proportional to the number of instances.

The workaround we found is to quantize the continuous random value (0..1) to a finite set with a given step.

In the new icetree below, we map the continuous 0..1 range into a step function, by subtracting its Modulo(0.05) from the random value.

So, for example, a value of 0.267 is rounded to 0.25. The possible outputs of the leftmost Subtract node are then 0, 0.05, 0.1, 0.15, etc.

Since then they are then multiplied by 10, they round up to 0, 0.5, 1, 1.5, etc., as showing now in the viewport.



So, what happens now is that SltToA exports only 20 spheres as polymeshes. The other instances find a matching time exported already, and so they are rendered as an Arnold instance.

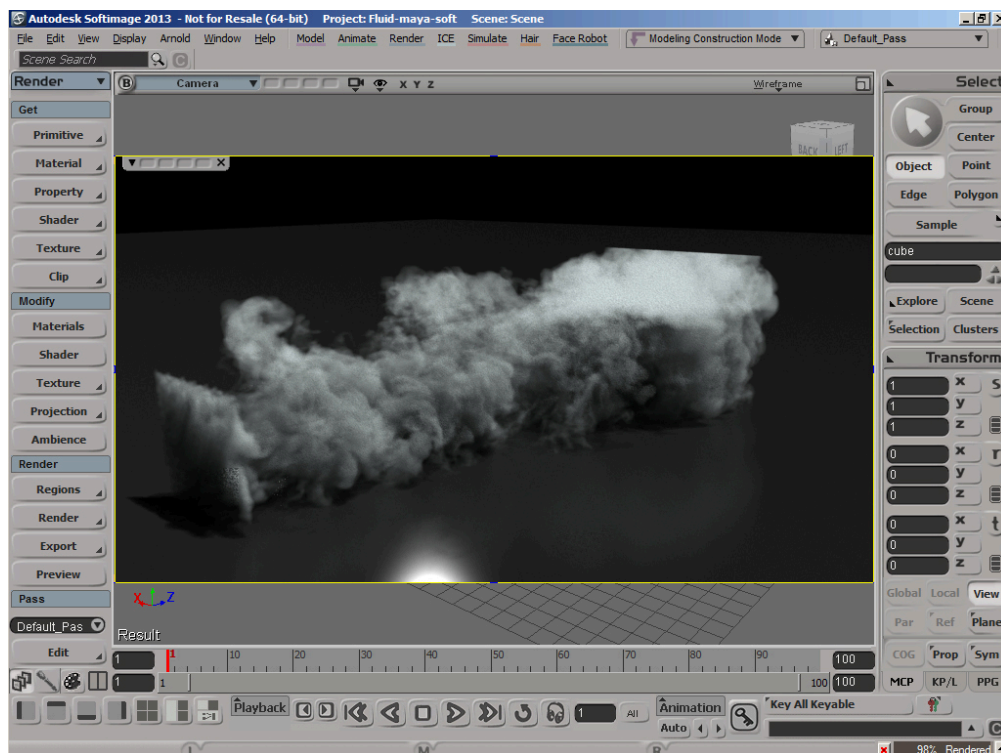
At this point, you can have your pointcloud growing considerably, yet having a very small time required for the render to start, and small ass files exported.

Of course, we are reducing the granularity of the animation, since we'll have only 20 "types" of spheres randomly distributed, but for most cases, this will still look well.

If not, you can simply decrease the modulo value until the wanted detail. 0.01 will make SltToA export 100 spheres, and so on.

Steven Caron then posted a [Round to Nearest Decimal](#) compound which can be used generically or right before you set the ShapeInstanceTime attribute.

Rendering Maya Fluids with Softimage



Fluid exported from Maya as an ASS file and rendered with SitoA as a Stand In

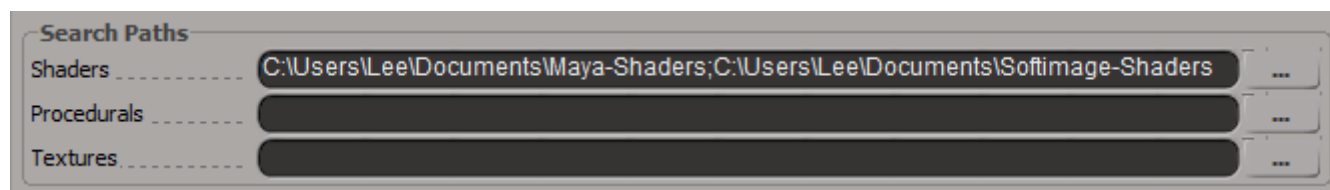
It is possible to render Maya fluids as a Stand In with SitoA. This short tutorial will guide you through the process.

- First, you must export the fluid from Maya as an ASS file. To export the fluid from Maya, select the fluid and select StandIn>Export from the main Arnold menu.
- Download an up-to-date build for MtoA. The file that you will need is 'mtoa_shaders.dll'. It can be found in the 'Shaders' folder in the MtoA folder.

i MtoA shaders should be compatible with the SitoA shaders. Ensure that you are using up-to-date builds for both SitoA and MtoA, otherwise, the Stand In may not render correctly.

Shader Paths

- You must add the location of the Maya shader path (where 'mtoa_shaders.dll' is located) to the Softimage shader path found in the System tab in the Arnold Render Options (add a semicolon in-between the paths).



Shader paths set for both Maya and Softimage shaders

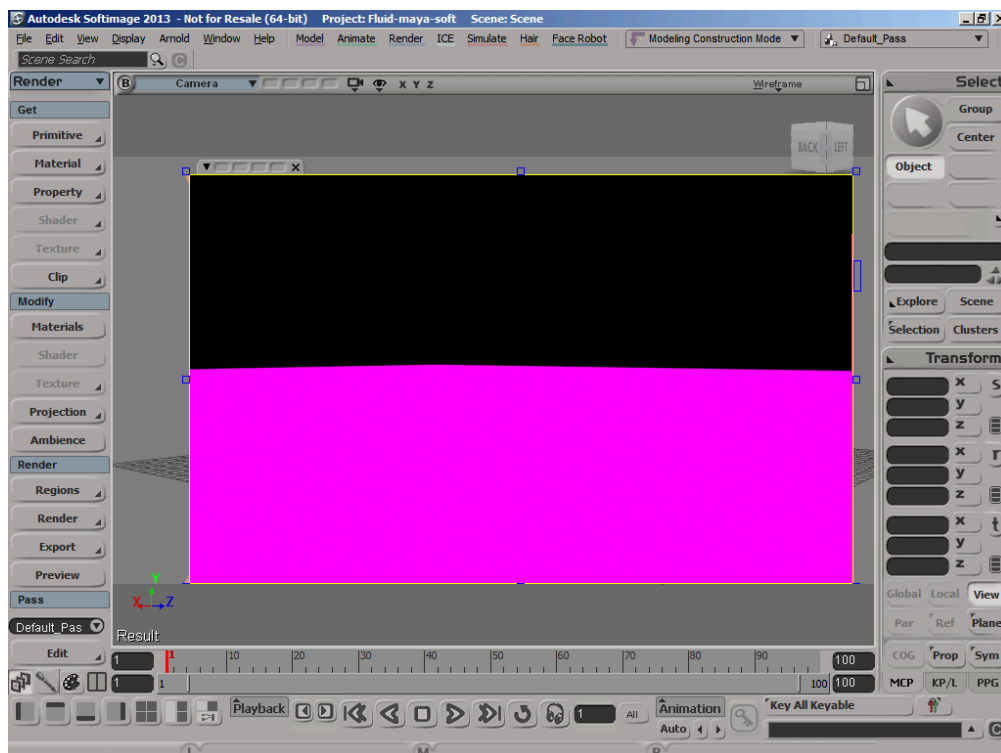
- In Softimage, create a **Stand In** and open the fluid ASS file that was exported from Maya. Render the scene and you should see the fluid.

Workflow issues

- If you are getting the following warning, then your shader 'Search Paths' are not set up correctly:

' WARNING : [arnold] [ass] line 126: node "mayaFluid" is not installed

- If the render appears pink it is because you do not have the 'Shaders Path' set correctly or either the versions of the MtoA or SitoA plugins are not up-to-date with each other.



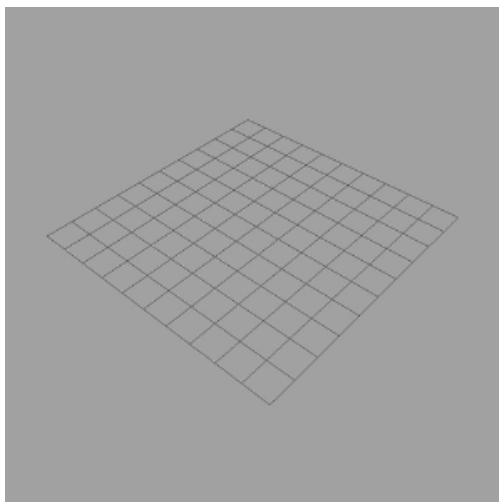
Rendering an Ocean with Displacement



This tutorial covers how to render an ocean scene using nothing but a polygon plane and an HDR image connected to a skydome_light. The detail for the ocean shader will come from a vector displacement map. There are a variety of attributes to set when using displacement mapping. However, it is very simple to set up and get convincing results using the right settings.

The ocean displacement map can be downloaded [here](#).

- Start off by creating a polygon plane that can be used as the ocean surface:

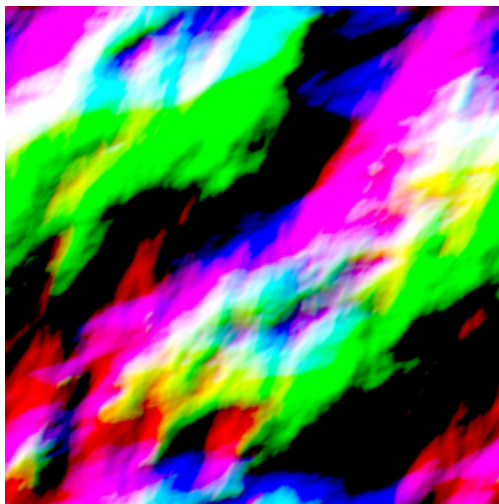


Ocean Material

- Create a `standard_surface` shader and assign it to the polygon plane. Set the `base_weight` to 0. Increase the `specular_weight` to around 1. Increase the `transmission_weight` to 1 (this will make the water transparent). Finally set the `specular_IOR` to that of water (1.33).

Vector Displacement Map

The vector displacement map used is a 32-bit EXR image. This means it is able to retain the highest quality for the vector displacement effect. To get a more realistic wave effect, we will use a vector displacement map to displace the sea waves. This will displace the sea geometry around a direction different from the normal, unlike a traditional displacement map which displaces the geometry in one direction only.



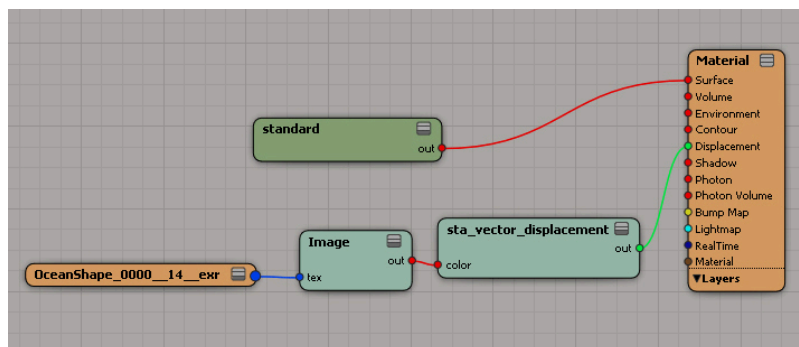
Object Space Vector map used to create ocean displacement (click to download)



If the effect of the waves looks too sharp you could try blurring the vector displacement map in order to get smoother waves.

- Create an `sta_vector_displacement` and connect it to the displacement of the Ai Standard material. Connect the Ocean image vector map to the color parameter of

the `sta_vector_displacement` as in the image below:

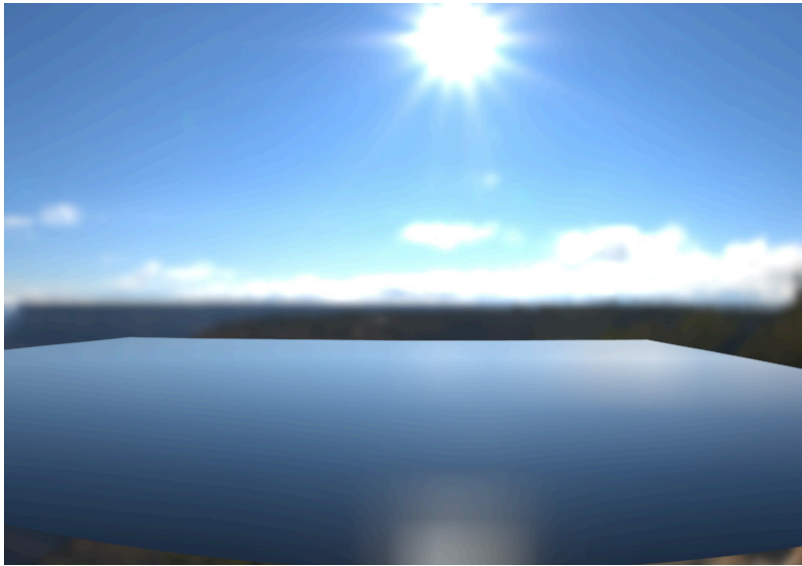


Skydome light

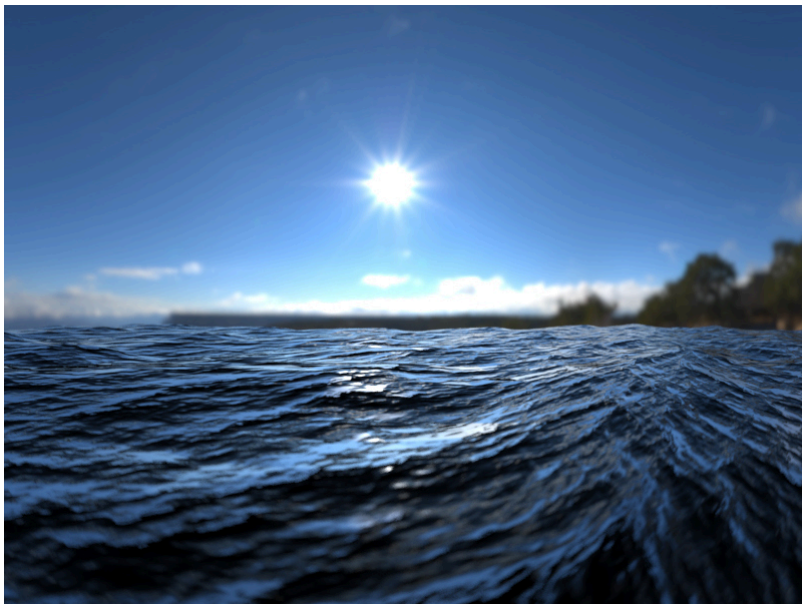
Create a Skydome light and connect a HDR map to its Color attribute.

Subdivision settings

- Render the scene. There is no displacement! That is because we don't have enough subdivisions in the geometry of the sea plane.

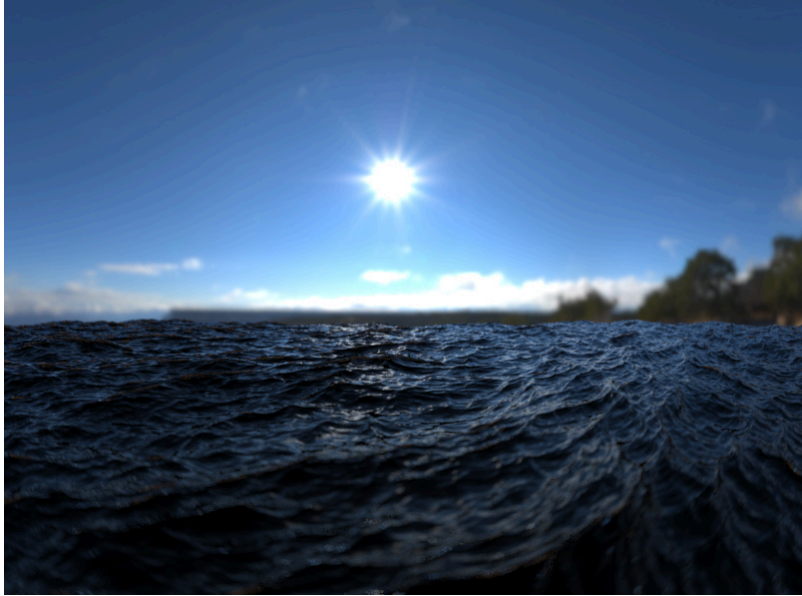


- Select the plane and under the Arnold attributes, change the subdivision_type to catclark and increase the number of iterations. Start off with a low number like 3:



subdivision_type : catclark ,subdivision_iterations : 3

- The sea looks ok, but it really needs more detail. Increase the subdivision_iterations to 8. Be careful not to increase this value too much as memory use and render times can increase dramatically with each increment (each iteration quadruples the geometry).



subdivision_iterations : 8. More detail can be seen in the sea waves.

Lighting

- Finally, add a [Distant•light](#) to the scene. We can use this to exaggerate the effect of the sun on the sea surface, thereby bringing out the highlights on the sea surface.



Rendered with a directional light

Increasing the [Angle](#) attribute of the directional light will increase the angular size of the light, thereby producing a larger specular reflection on the sea surface.



Angle: 5

Transmission

Light that penetrates the ocean surface is 'refracted' due to the fact that light travels faster in the air than in water. When the light reaches the water it will be scattered or absorbed by particles in the ocean. As the ocean becomes deeper (denser), the water will decrease the depth of light penetration. We can replicate this effect quite easily using the transmission attribute in the standard_surface shader. The effect can be seen when placing an object inside the ocean surface. We can see that the longer light travels inside the mesh, the more it is affected by the blue transmission_color.

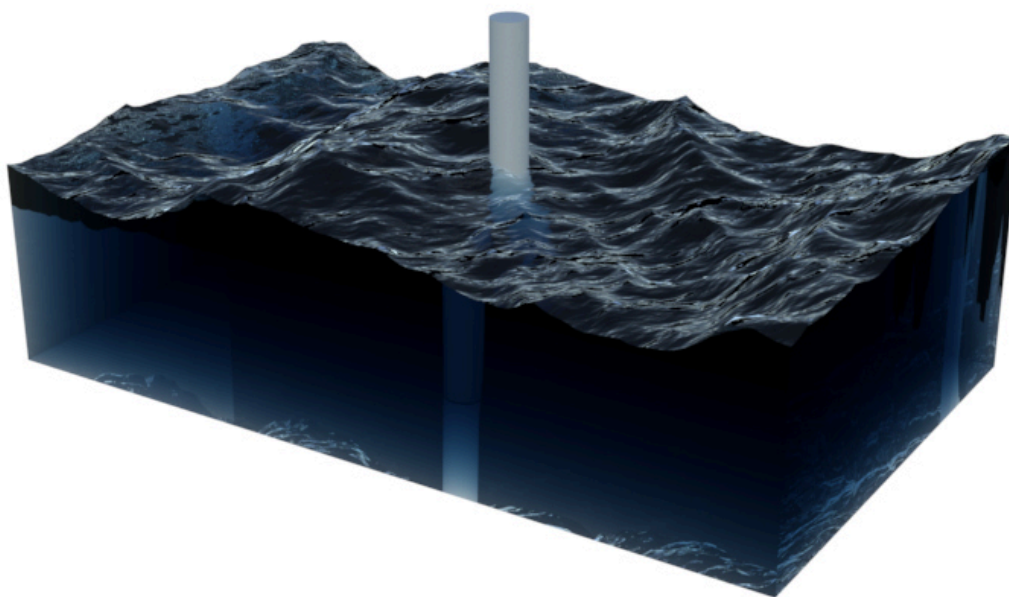


transmission_color: white



transmission_color: blue

When applied to the ocean shader we can see the effect that transmission_color and depth has on the white cylinder inside the closed surface ocean below:



Dark Patches

The ocean shader is not fully transparent. This means that non-glancing rays will not reflect and will instead refract and go through the water (like in real life), but since it's not transparent, we instead terminate the rays and get black.

Shading normals can cause problems when the shading normal points away from the ray even though the geometric normal points towards the ray. In this case, the shading (Auto Bump) normal says that we are hitting the backside of the water and should reflect down, while of course in actuality the ray is hitting the top of the water mesh.

At grazing angles, real water is nearly 100% reflective, but even so, it may take several bounces for the light to get out of the valley behind a wave crest, and even after several bounces it may reflect something that is higher up from the horizon than what the surrounding water reflects.

A more realistic look is to make the water transparent and then place a plane below the water that has the color of ocean water. In the example images below the ocean floor's Standard shader has an Emission Scale of 1 and a blue/green Emission Color. Increasing the Subdivision Iterations has also helped to reduce the number of black patches on the displaced sea waves.

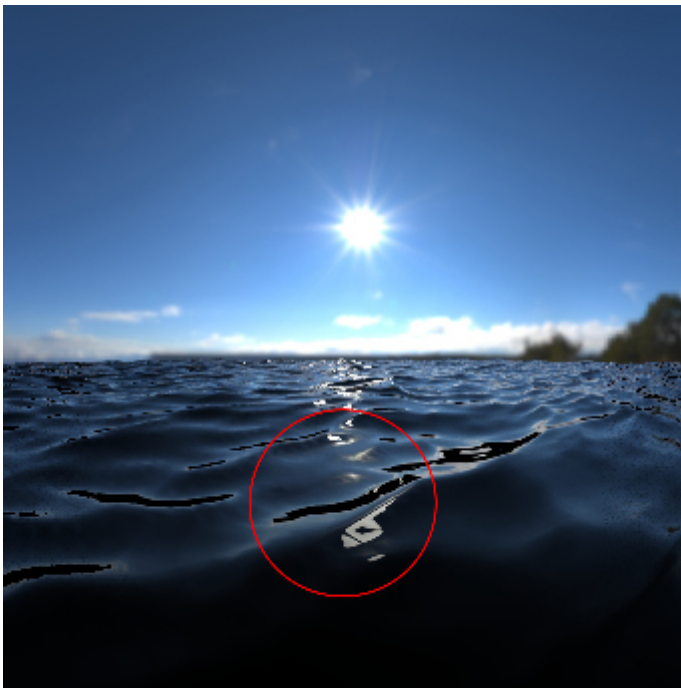


subdivision_iterations: 3. Black patches are visible



Increasing the subdivision_iterations to 8 reduces the problem

The dark patches behind the wave crests (self-reflection) can also be improved by increasing the specular_ray_depth in the render settings.



specular_ray_depth: 1



specular_ray_depth: 2

Conclusion

That concludes this tutorial on how to create a believable ocean scene. Using the correct displacement settings in conjunction with a high-quality 32-bit vector displacement and the correct Standard shader settings should give you something that looks realistic. Why not try adding depth of field or change the camera lens to something like a fish eye in order to add more visual interest to the scene.

Below you can see how much more detail can be achieved by using a vector displacement map to create the wave effects compared to using regular geometry:



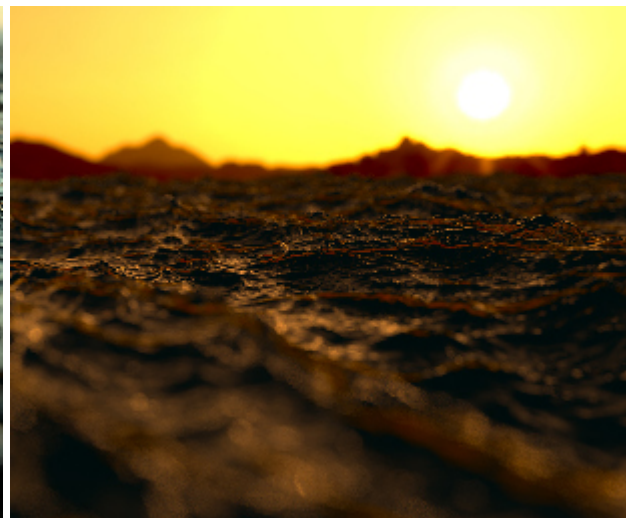
Sea waves with polygon geometry



A vector displacement map can give far more detail than using geometry

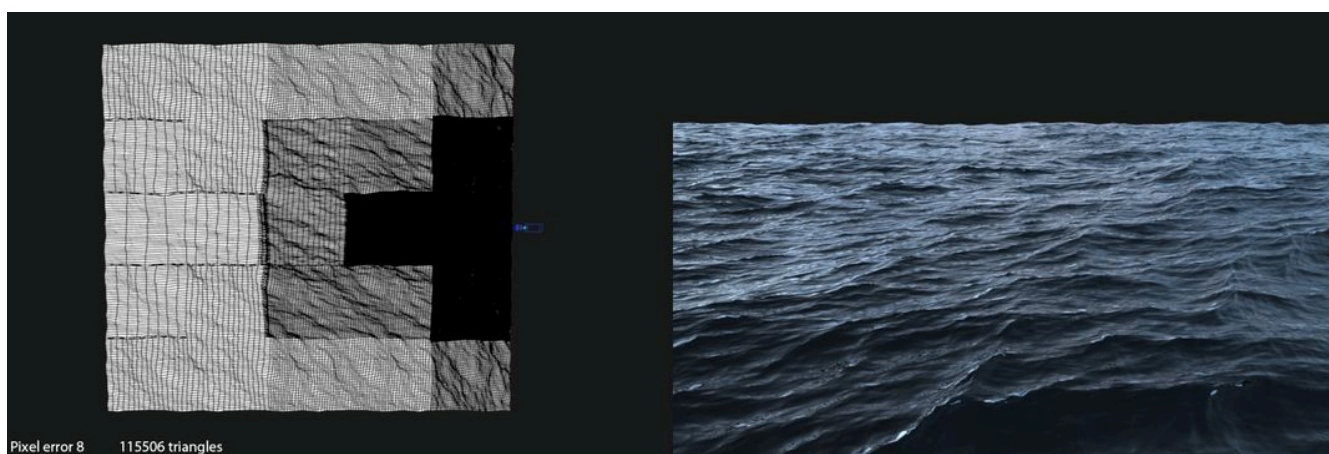
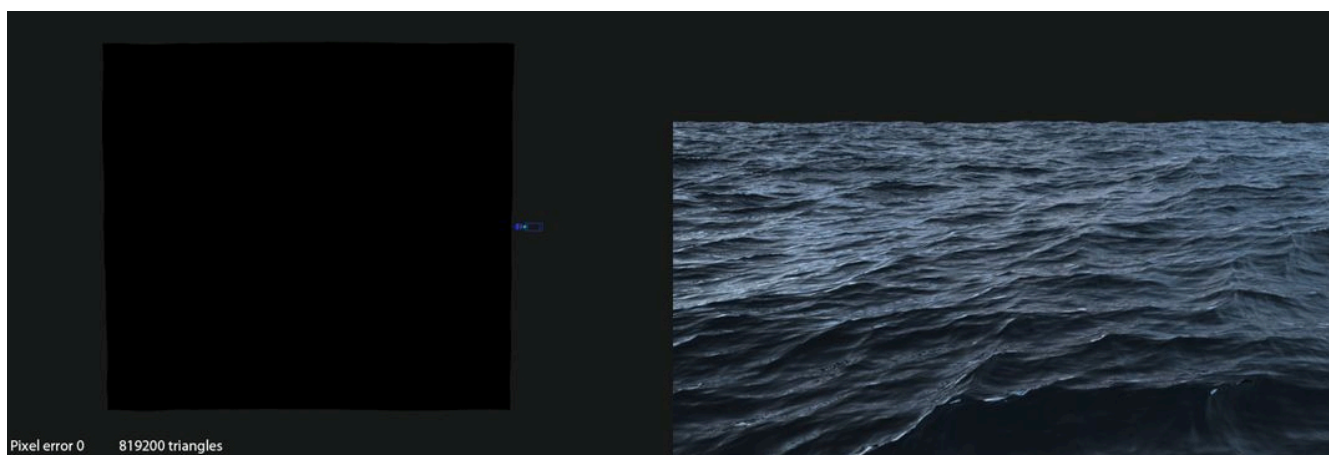
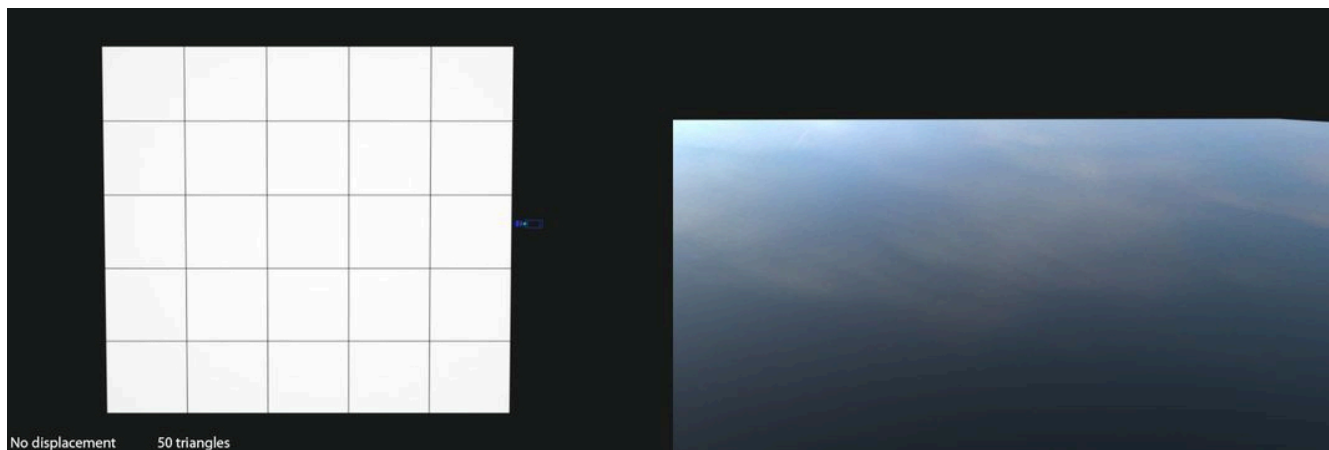


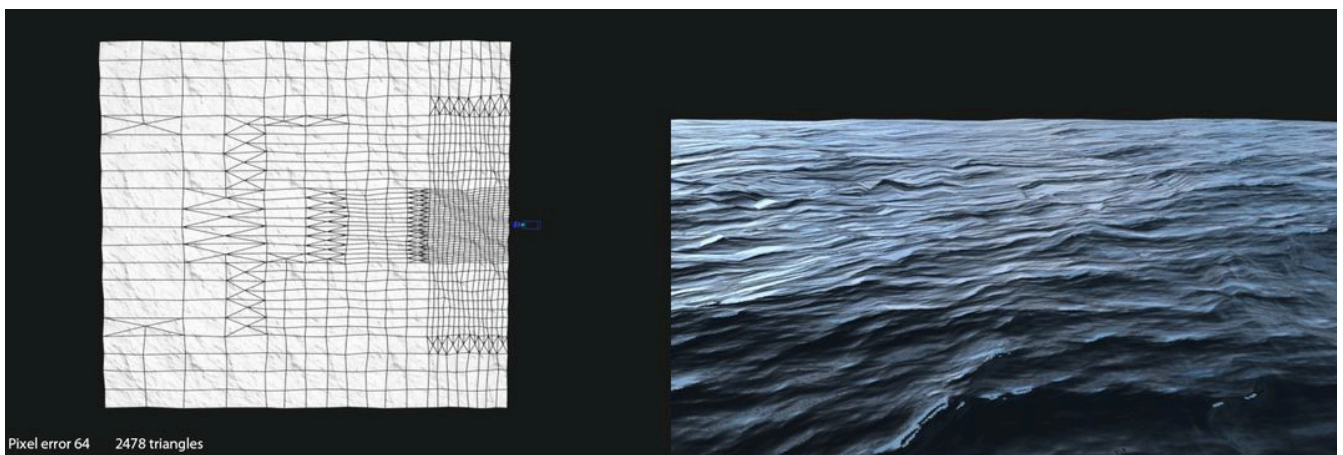
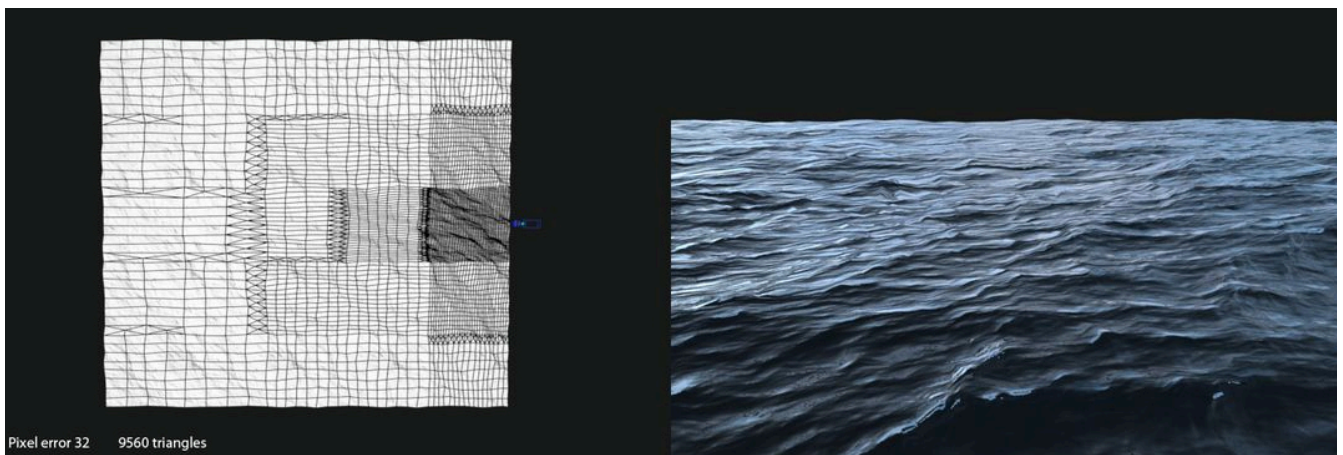
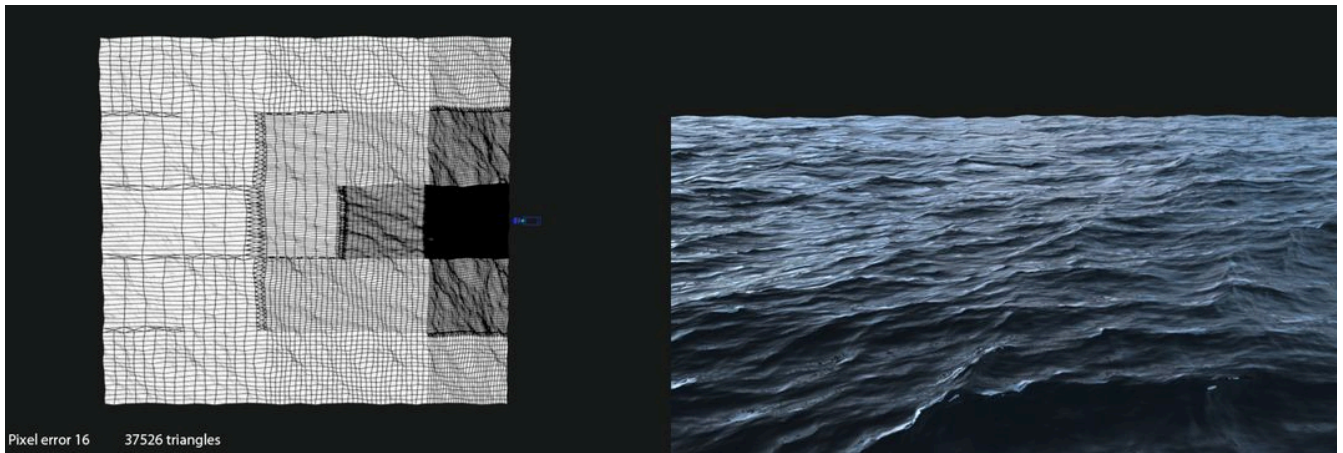
Animation of displacement value



Further examples

Subdivision Tutorial





Removing Noise

Types Of Noise

The first step in removing noise from your renders is to identify where it is coming from. Noise can be caused by:

Insufficient Sampling:

- Motion Blur
- Depth Of Field
- Diffuse
- Specular
- Shadow
- Indirect Specular
- Transmission
- SSS
- Atmospheric Volume

Other:

- Fireflies
- Non-energy conserving shaders, networks or settings

.

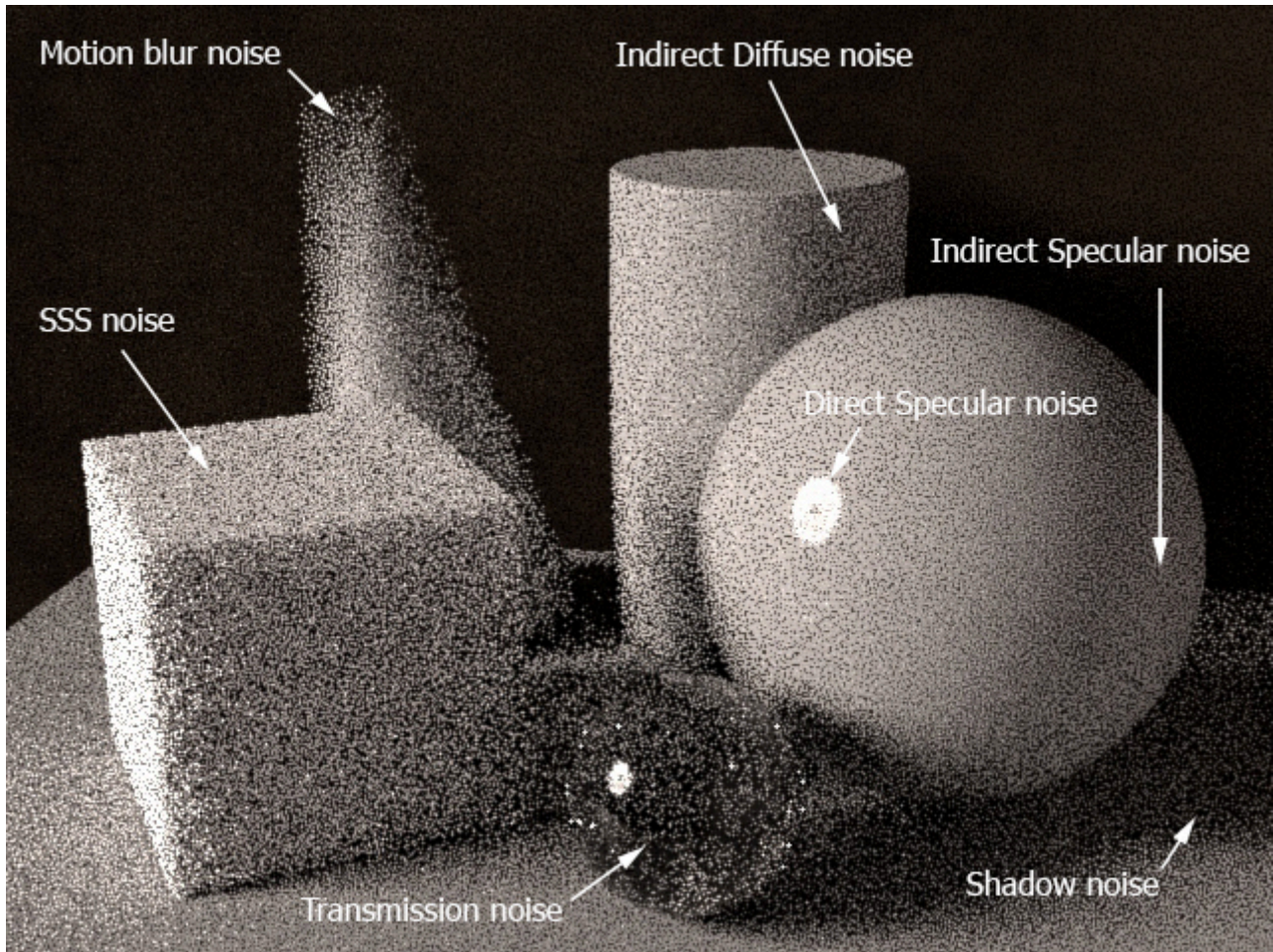
Noise nearly always comes from insufficient sampling, but increasing sampling for the wrong rays can make the render times increase without helping to remove the noise. As an artist is usually working to a render time limit or amount of rays the aim is to allocate those rays as effectively as possible to minimize the noise in the most efficient manner. So if the Camera (AA) samples have to be increased to remove DOF noise, the other settings must be lowered to keep render times manageable. However, if DOF or motion blur is not a concern, then increasing Camera (AA) samples would fix all noise elsewhere but would also slow render times from the unnecessary rays.



For a procedure for removing noise, follow the [Removing Noise Workflow](#) diagram.

.

Here is a very noisy scene where all of the samples have been set to 1.

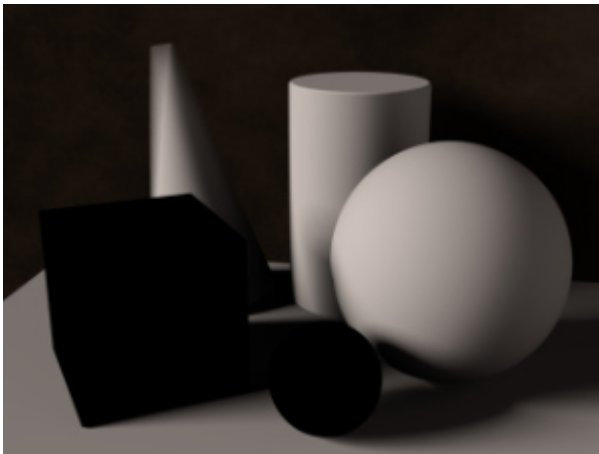


All samples: 1 (rollover image for final sample settings)

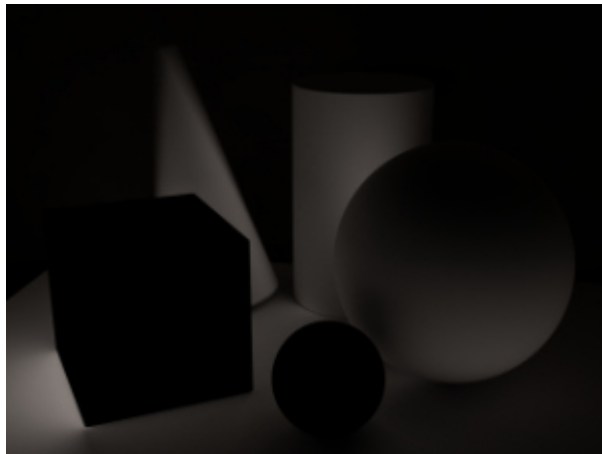
-
- Camera (AA) samples: 1
- Diffuse samples: 1
- Specular samples: 1
- Transmission samples: 1
- SSS samples: 1
- Light samples: 1
-

The most efficient method for identifying noise is to render AOVs. This can help save time by reducing the need to adjust settings unnecessarily.

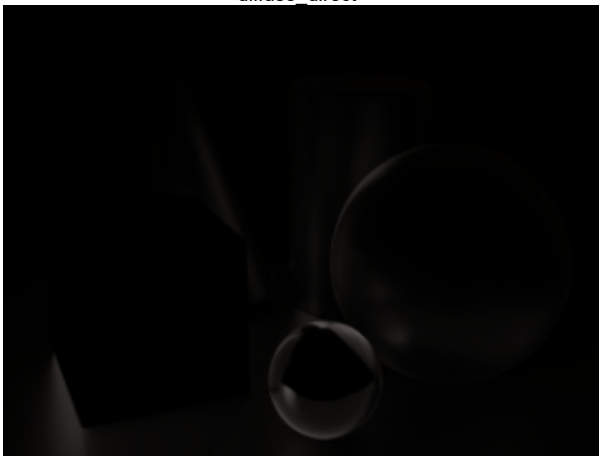
AOVs



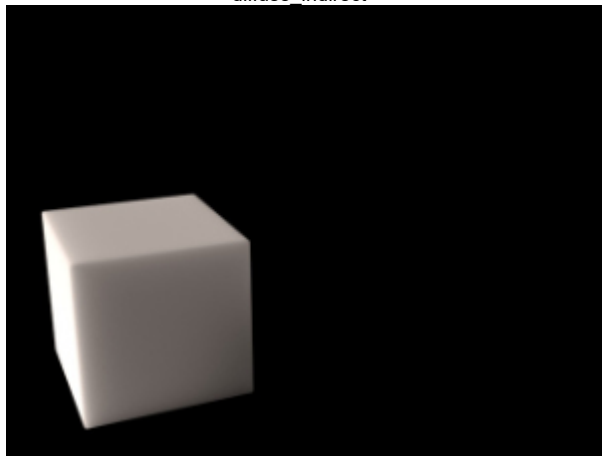
diffuse_direct



diffuse_indirect



specular_indirect



sss

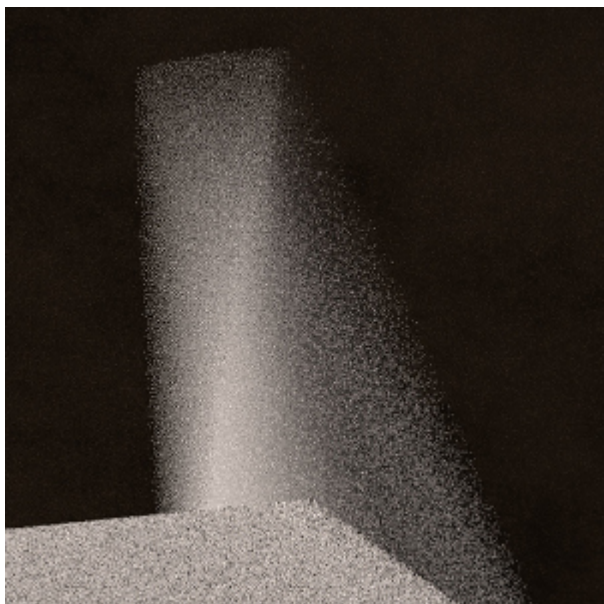


In the examples below the smoother option is shown with samples of 10. This is a very high value and is only used here because the Camera (AA) samples and all other samples are at 1 for an illustrative effect to contrast the noisy and smooth areas. Normally the Camera (AA) samples would be around 4-8, and as it acts as a multiplier, similar values would be needed for the other samples.

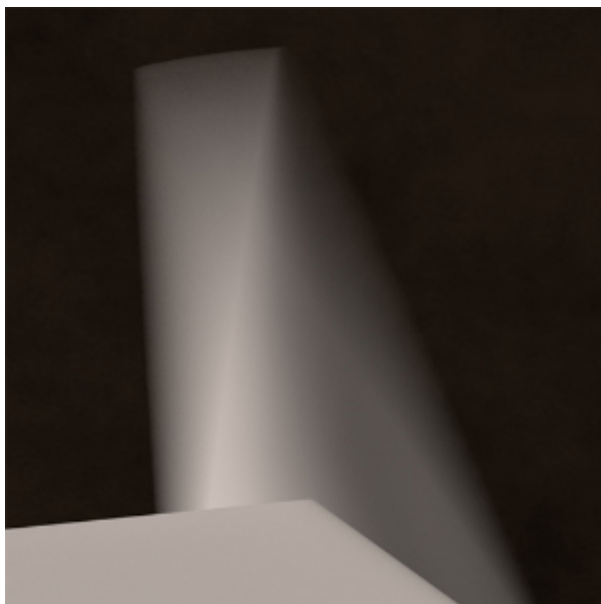
Motion Blur And Depth Of Field Noise

Motion blur noise shows up in the trails of moving geometry while DOF noise appears in the out of focus areas.

Motion Blur

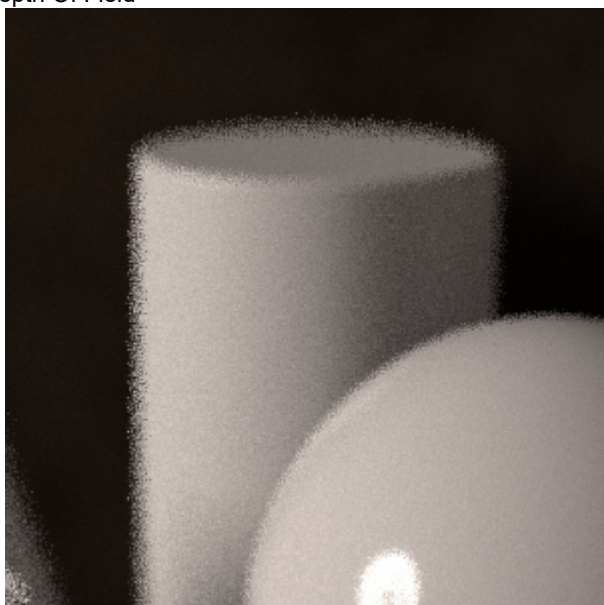


All samples: 1



Camera (AA) samples: 10

Depth Of Field

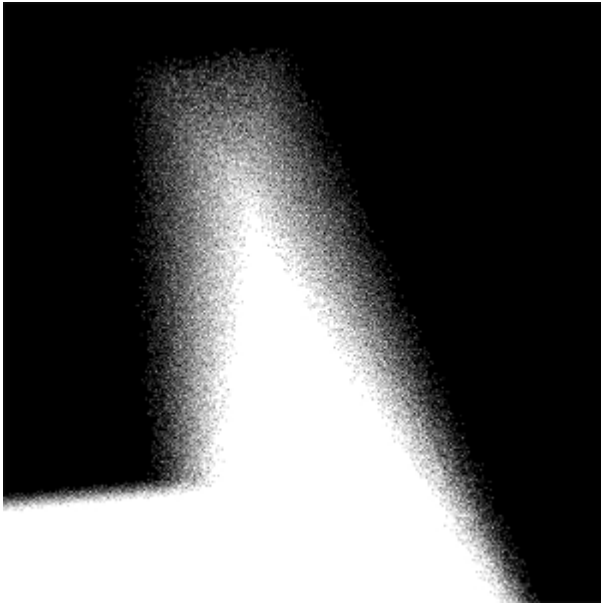


Camera (AA) samples: 1

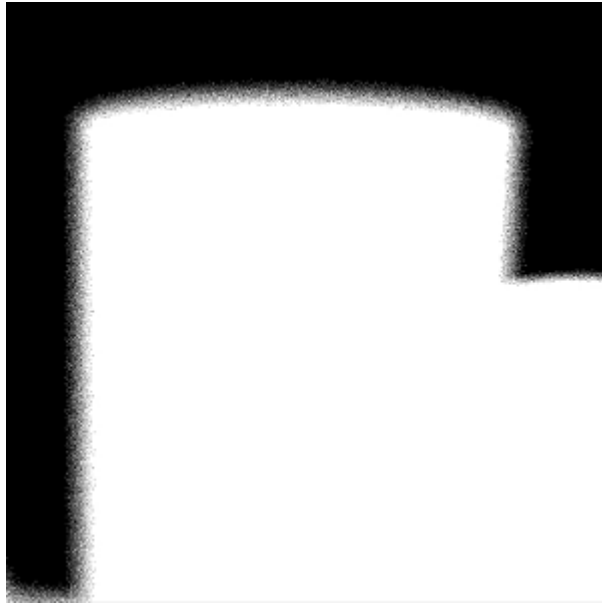


Camera (AA) samples: 10

Both the depth of field and motion blur can be confirmed by looking at the alpha channel to see if noise is present.



Motion blur alpha



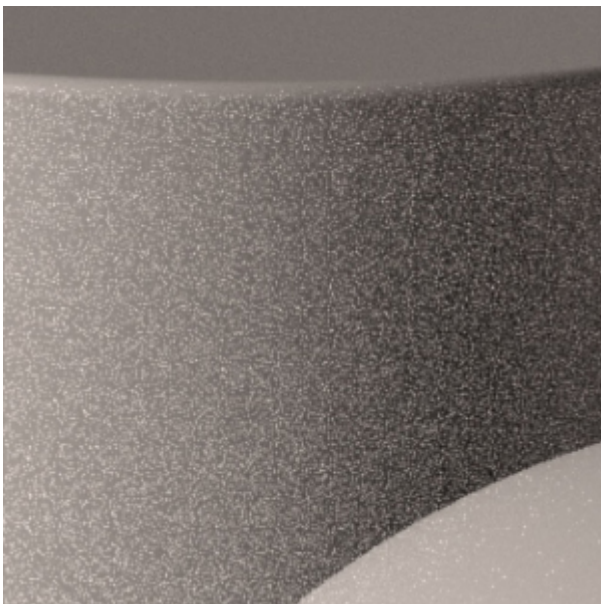
Depth of field alpha

•

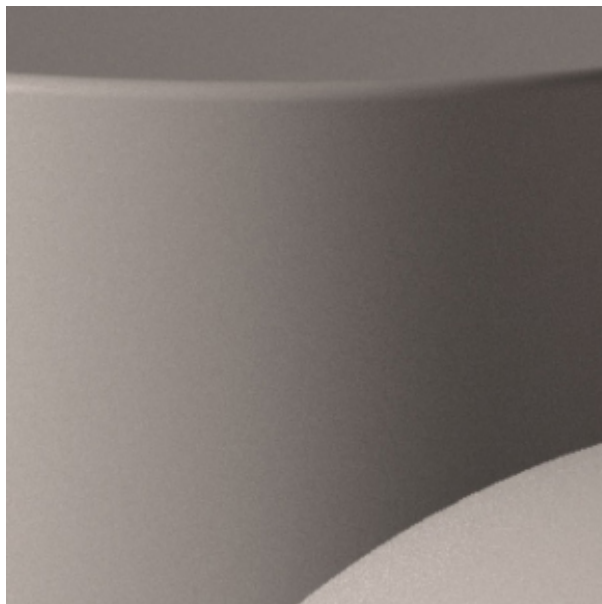
Motion blur and depth of field noise are caused by insufficient camera rays and therefore can only be solved by increasing Camera (AA) samples. The actual number of Camera (AA) samples is the square of this number. Camera (AA) samples of 4 results in 16 rays being cast. Note that increasing Camera (AA) samples will also increase the other samples meaning that they should be decreased to compensate.

Increasing the Camera (AA) samples will have a dramatic effect on render times. If motion blur or DOF is not a problem, then Camera (AA) samples should be the last consideration for fixing the other noise types.

Diffuse Noise



All samples: 1

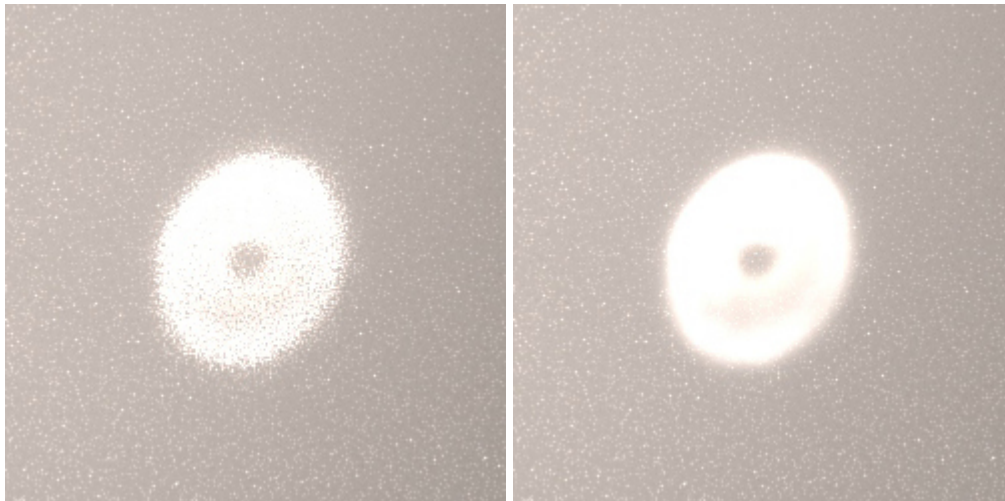


Diffuse samples: 10.

Indirect diffuse noise is a common type of noise. The easiest way to confirm the cause is to check the indirect diffuse AOV. Another method is to set diffuse samples to 0 which will turn off indirect diffuse. If the noise disappears, then it is created by indirect diffuse.

When diffuse samples are more than zero, camera rays intersecting with diffuse surfaces fire indirect diffuse rays. The rays are fired in random directions within a hemispherical spread. Noise is introduced when there are insufficient rays to resolve the range of values from the environment. The noise can be removed by increasing diffuse samples.

Specular Noise



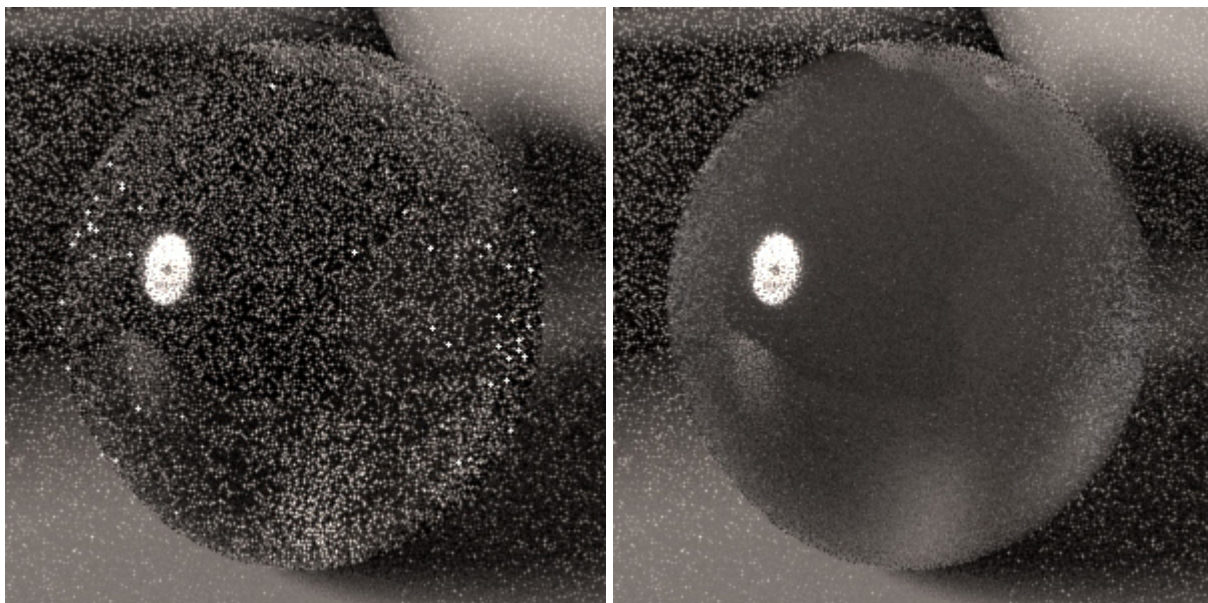
All samples: 1

Specular samples: 10

Indirect specular noise occurs when the `specular_roughness` parameter > 0 . The easiest way to confirm the cause is to check the indirect specular AOV. Another method is to set specular samples to 0 which will remove blurred reflections. If the noise disappears, then it is created by indirect specular.

Indirect specular noise is caused by a lack of specular samples. These samples control the number of rays fired when computing the reflected indirect-radiance integrated over the hemisphere weighted by a specular BRDF. The exact number of rays is the square of this value. Increase this number to reduce the indirect specular noise. Remember that the sampling is done for each Camera (AA) sample, so high values for both Camera (AA) samples and specular samples will tend to result in slow renders.

Transmission Noise



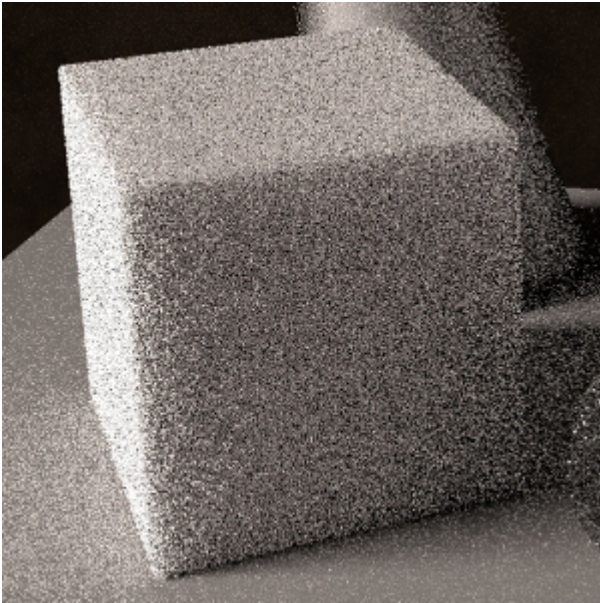
All samples: 1

Transmission samples: 10. All other samples: 1.

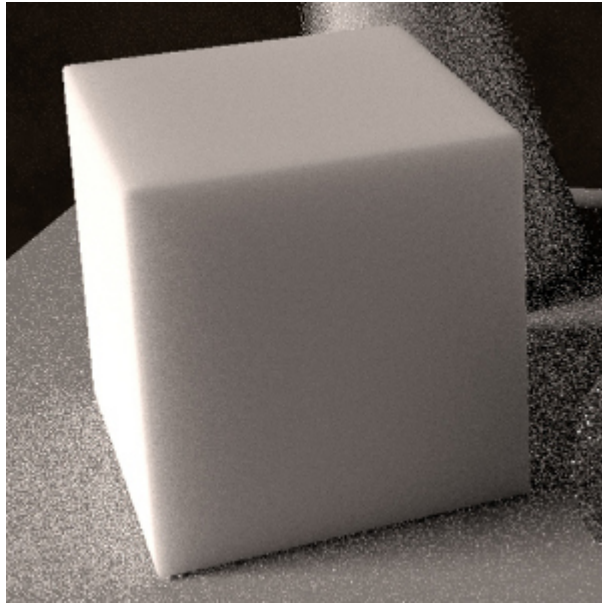
Transmission noise is noticeable in the blurred refraction of a transparent object when `specular_roughness` > 0 . The easiest way to confirm this is to check the transmission AOV. Another method is to set transmission samples to 0 which will remove blurred refractions. If the noise disappears, then it is created by transmission.

Transmission noise is caused by a lack of transmission samples. Transmission samples control the number of samples used to simulate the microfacet-based glossy refraction evaluations. The exact number of rays is the square of this value. Increase this number to reduce noise in shaders using transmission.

SSS Noise



All samples: 1

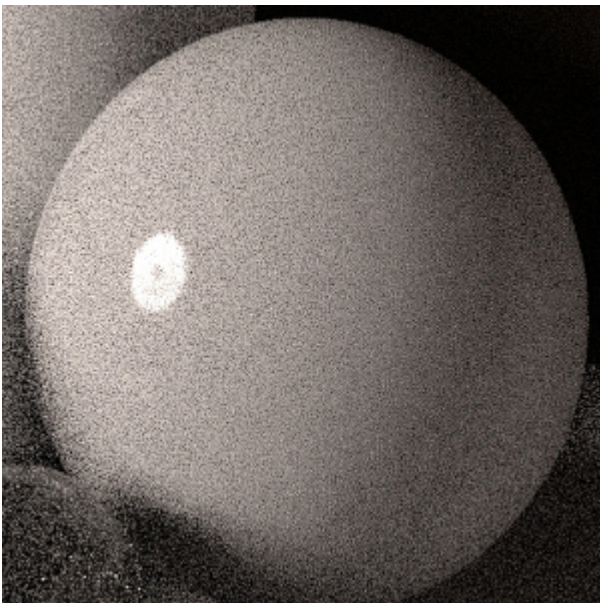


SSS samples: 10

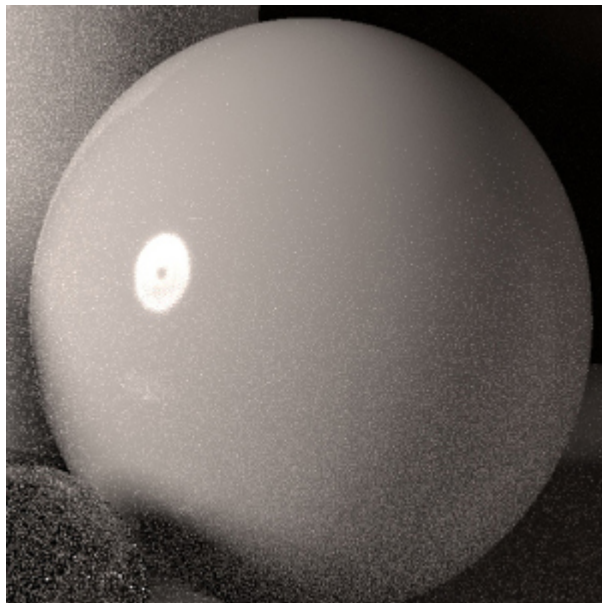
Subsurface-scattering (SSS) noise will occur on surfaces that are using the [standard_surface](#) shader with SSS enabled. To confirm, check the SSS AOV. The SSS noise can be removed by increasing SSS samples.

Direct Specular And Shadow Noise

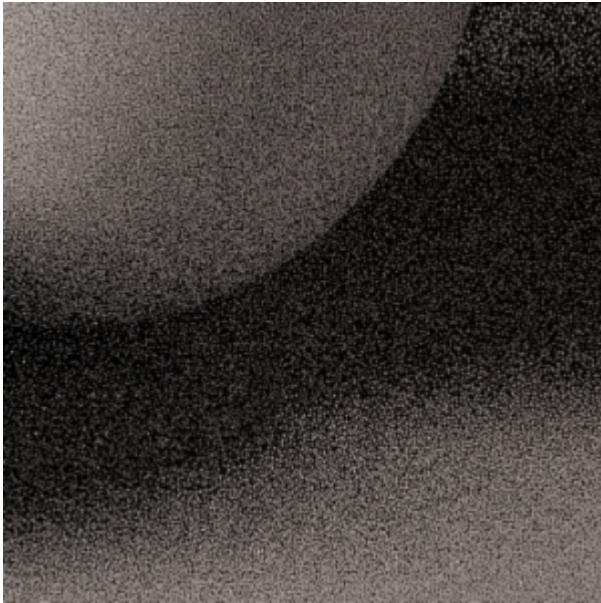
Noise in the direct specular and shadow is caused by a lack of light samples. Normally a small number of samples will be required to remove direct specular noise, but more samples may be required to remove shadow noise. The larger the radius a light source has, the softer the shadows will be, and therefore more samples will be required to remove shadow noise.



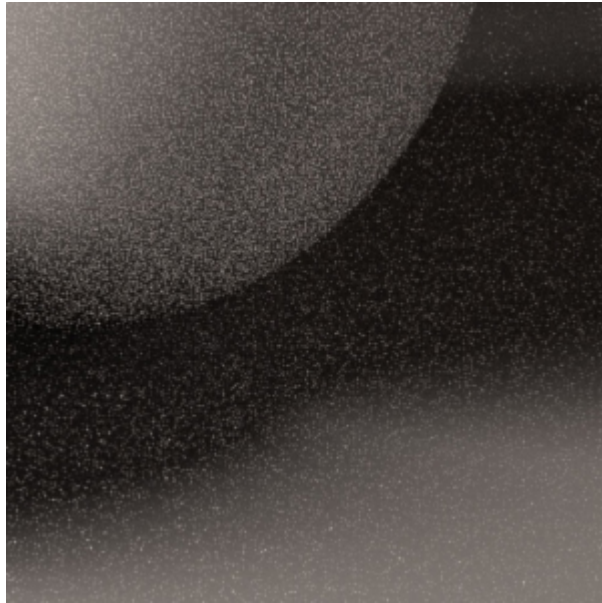
Light samples: 1



Light samples: 10



Light samples: 1



Light samples: 10



Light samples between 3-4 should be sufficient for most scenes.

•

Shadow noise is often mixed up with indirect diffuse noise, particularly for lights with large radii because their shadows will be softer. Look at the direct_diffuse AOV to see if the shadows are noisy. Alternatively, set diffuse samples and specular samples to 0. This will remove GI which will allow you to single out the direct lighting contribution. It can be tricky to identify direct specular noise as it could be mistaken for indirect specular noise. Direct specular is the reflection of the light itself on the surface. This will help you distinguish it from the reflections of surrounding objects based on color and intensity.

•

Noise can also appear on the specular highlights of thin geometry that use a shader with a high specular_weight value. Increasing the number of Camera (AA) samples can help reduce this effect to a point. However, even with high Camera (AA) samples, anti-aliasing artifacts can appear. Increasing the specular_roughness value helps to reduce this type of noise. Increasing the size of the light and reducing its intensity can also help in these situations.

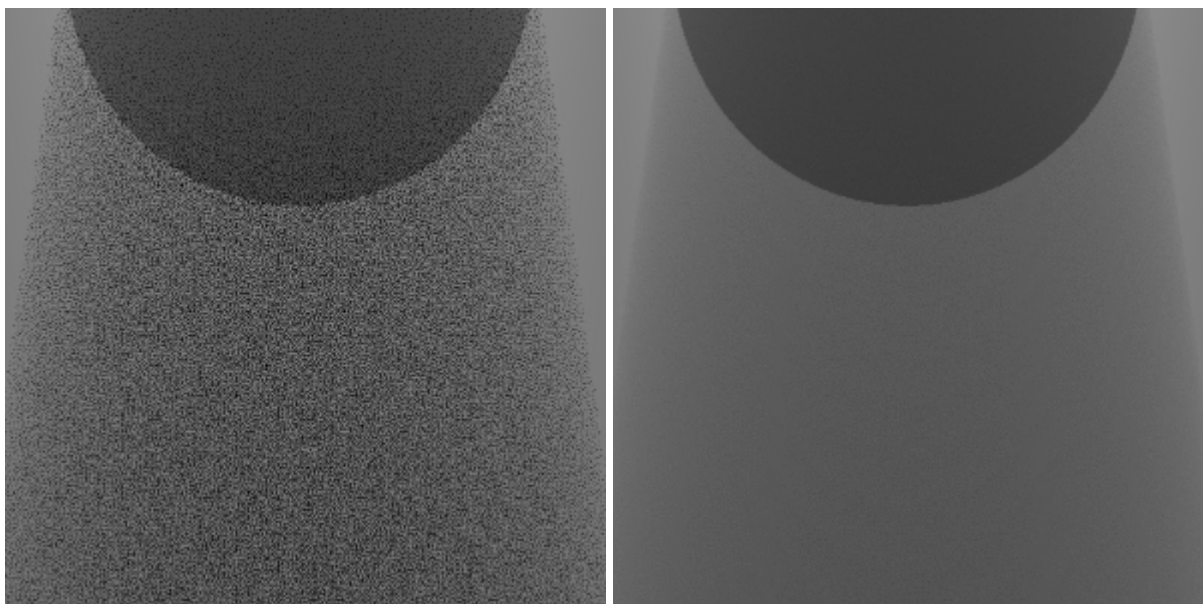


Specular Roughness: 0.2



Specular Roughness: 0.5

Atmospheric Scattering Noise



All samples: 1

Atmosphere Volume samples: 10

Atmosphere_volume noise will occur within the shadowed regions of a beam of light. This noise is caused by a lack of samples in the Atmosphere_volume shader. The samples are distributed according to the volume density. More samples will refine the quality of the solution.

Fireflies And Sparks

Fireflies (bright 'spike' type noise) are typically generated from the reflection of a strong light on a shiny, low-roughness glossy surface. Usually, hundreds of samples participate in a final pixel color. If one of those is a high-valued specular ray (coming from the reflection of a strong light), then there will be many samples with low values and a single sample with a value in the thousands. That single sample will make the whole pixel become white (a "firefly").

Clamping

Sample clamping can help eliminate this kind of noise. If you clamp high sample values, a single stray sample will now be diluted and not affect the final color as much. However, this will affect the final dynamic range of the render. Use with caution.

Light Decay Filter

Fireflies can also be caused by indirect diffuse illumination from very bright hotspots, in particular when the light sources are very close (or even touch) nearby geometry, such as a light's cabinet or enclosure. The inverse square decay in the light will make those nearby surfaces extremely bright. A workaround is to add a light decay filter with a very low near start value to avoid samples very close to the light. Again, this should be used with caution.

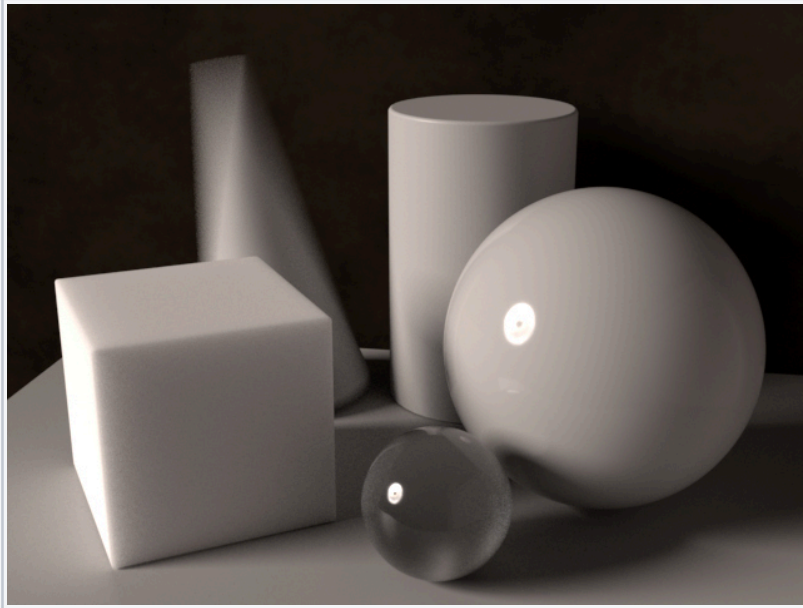
Other Considerations

- Noise could come from things not visible in the render (behind the camera).
- Noise could be caused/exacerbated by non-energy conserving shaders, networks or settings.
- Another method to reduce noise is to remove the cause and fake it with special lights. For example, a character whose face is lit by bounce light only: it will be much less noisy to add a specific bounce light.
- Arnold can remove noise easily when it knows where lights are, by sampling the lights directly, but will have problems if there are bright "directional" patches not tagged as lights that contribute significantly to the scene's lighting.

Clean Renders

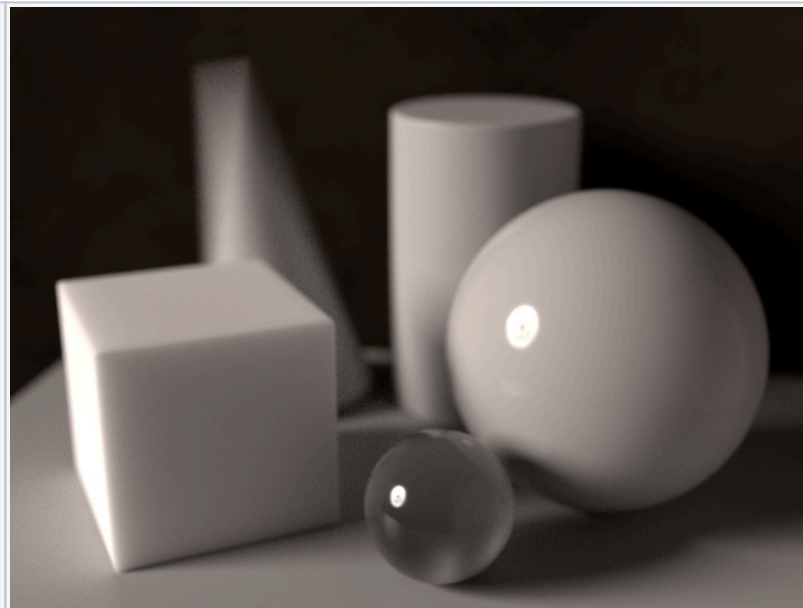
No DOF

- Camera (AA) samples: 7
- Diffuse samples: 3
- Specular samples: 2
- Transmission samples: 6
- SSS samples: 4
- Light samples: 3



With DOF

- Camera (AA) samples: 15
- Diffuse samples: 1
- Specular samples: 1
- Transmission samples: 4
- SSS samples: 3
- Light samples: 3

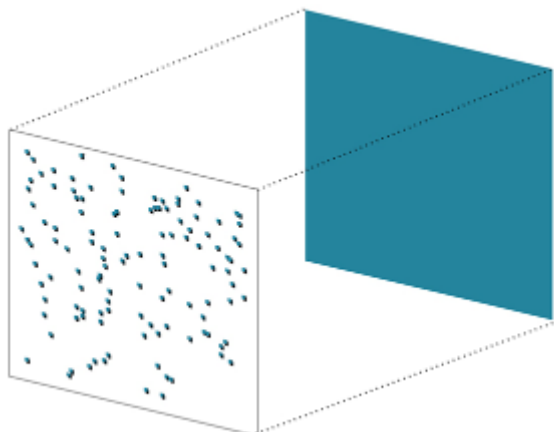


When trying to identify noise in your renders, it is useful to render and view AOVs. This enables you to isolate the type of noise and adjust the relevant samples. A general guide to identifying and prioritizing the order in which common noise types can be found in the table below. Note that this is a general guide and that every scene is different.

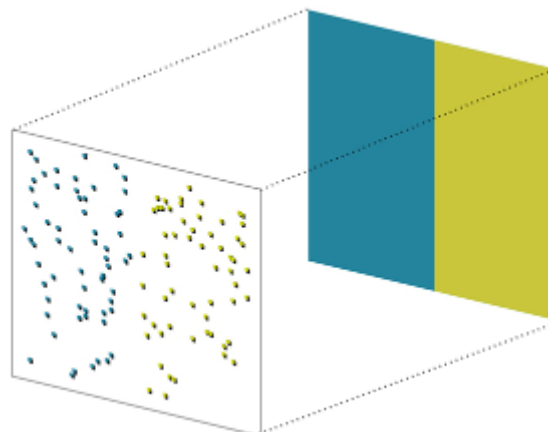
Noise Visible in	Samples to Adjust
Alpha channel	Camera (AA) samples
Indirect Diffuse	Diffuse samples
Direct Specular (specular noise)	Light samples
Direct Diffuse (shadow noise)	Light samples
Indirect Specular	Specular samples
Transmission	Transmission samples
SSS (direct and indirect)	SSS (direct and indirect) samples
Volume	Volume samples (note that there are also volume samples in lights too)

Noise and Sampling

If you have a scene that consists of a simple color and you want to know the average color of the photo, it is easy to see what the color is at any one point by using a low number of samples and then averaging the result.

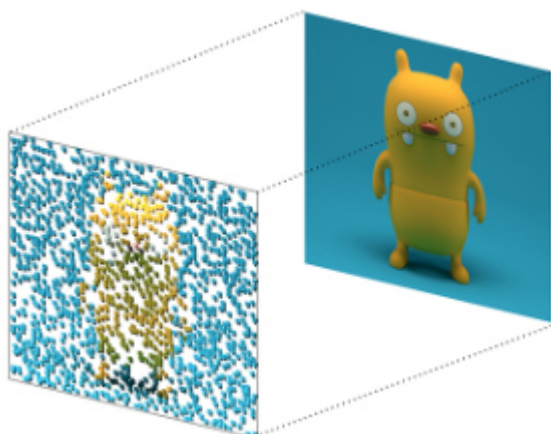


If the scene is all one color, it only requires a few samples to get the correct average

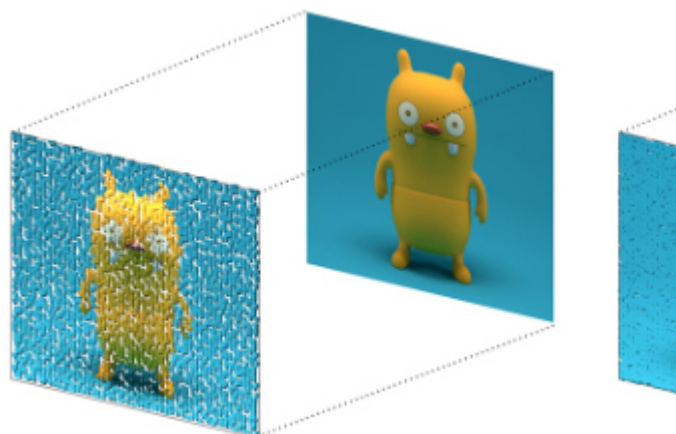


If the scene is half one color and half another, more samples will still give you a good average

However, if you have a more complex scene (with a character for example), and use only a few samples, sometimes half of them will be yellow and half of them blue. Sometimes all the samples will be one color or the other. Sometimes, by coincidence, all of the samples will be on the yellow character, despite only a part of the scene being the yellow character. A way to improve the average is by increasing the number of Camera (AA) samples.



3 Camera (AA) samples. Low quality.



4 Camera (AA) samples. Medium quality.

Increasing the number of Camera (AA) samples resolves the noise and improves the quality of the image.

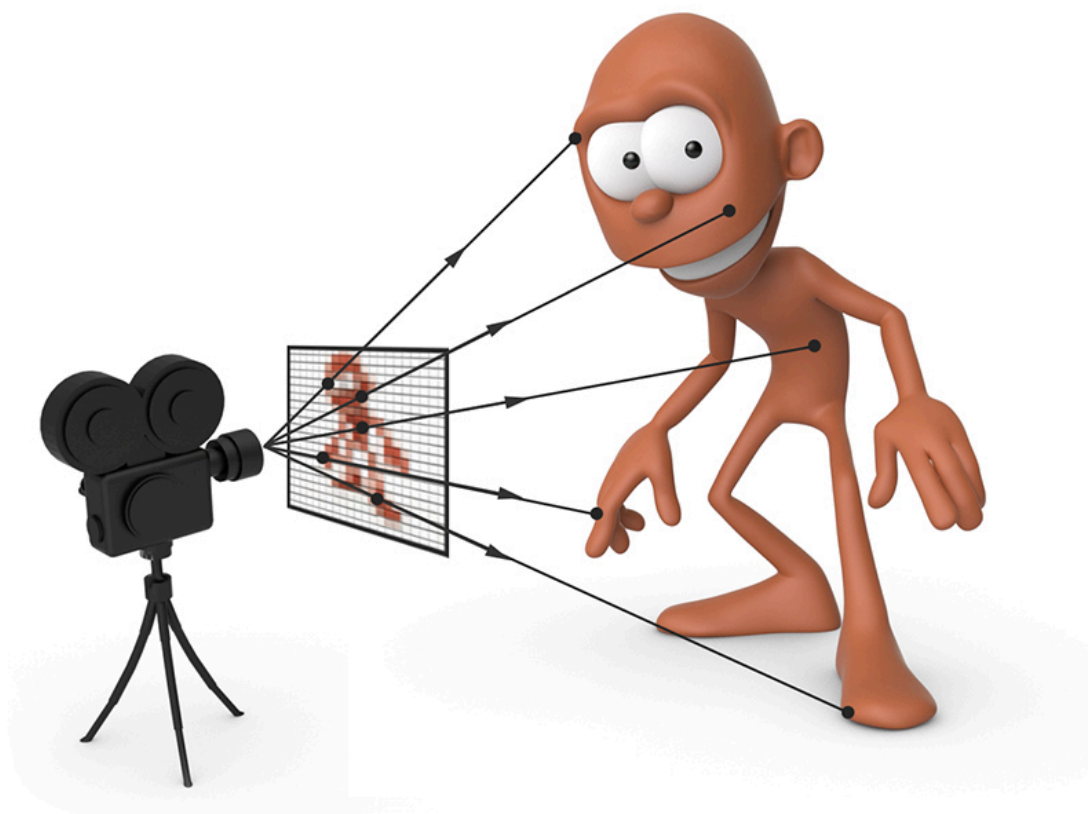


Further information about sampling can be found [here](#) and [here](#).

What is Sampling?

When working with a renderer like [Arnold](#), it is useful to have a basic understanding of the core principles behind ray-trace rendering. Producing a realistic-looking image of a virtual scene requires simulating the propagation of light in the scene all the way from the light sources to the camera. To determine the color of each image pixel, Arnold collects information from the scene geometry, shaders, lights, etc., and traces a number of random light transport paths that connect the objects seen through the pixel to the light sources - a process called 'sampling'. The quality of the resulting image largely depends on the number of paths, or samples, generated for each pixel.

When rendering an image, Arnold must determine a color value for each pixel by examining the scene. Arnold achieves this by sending out a number of rays from the camera's position until they hit an object in the scene. Every time a ray hits an object, it will perform some calculations that will eventually return a piece of information about the object (its color, for instance). This process can basically be described as 'sampling' the pixels in the virtual camera's image plane.



Renders computed with too few samples in each pixel can be noisy. Increasing the number of samples per pixel gives a less noisy image which better represents the actual scene. The images below show a comparison when increasing the number of Camera (AA) samples in a scene. Not only does increasing Camera (AA) samples reduce aliasing noise, but it also adds more secondary ray samples which also reduces noise in the lighting.



1 Camera (AA): 1x1=1 sample per pixel (image appears noisy)



6 Camera (AA): 6x6=36 (noise is reduced)

Camera (AA) Sampling •

So, what is camera sampling? Basically, a number of rays are shot from the camera through each required pixel of the rendered screen window into the scene. These are called 'primary rays', but sometimes they are referred to as 'eye' or 'camera rays' (Camera (AA) (anti-aliasing)) as they are cast from the rendering view. Sometimes they are called 'pixel samples'.

The Camera (AA) value controls the pixel supersampling rate or the number of rays per pixel that will be traced from the camera. The higher the number of samples, the better the anti-aliasing quality, but the longer the render times.

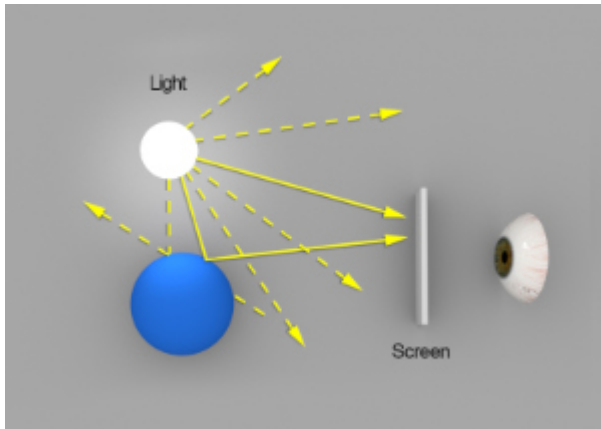
Note that this process is not linear, as, for each of these sampling rates, the actual number of samples taken is the square of the input value. For example, if Camera (AA) samples is 3, it means that $3 \times 3 = 9$ samples will be used for anti-aliasing. If diffuse samples is 2, then $2 \times 2 = 4$ samples will be used for the Diffuse GI. The same applies to the other values.

These Camera (AA) samples could be described as primary rays (or pixel samples) and determine the overall quality of the image being rendered. Increasing the Camera (AA) samples improves the overall quality of the image, but it can often be wasteful because it increases the sampling rate for everything. When the source of noise is identified in the scene, it is more efficient to set the sampling values to increase the number of only those specific rays. If for example, there is a scene with lots of motion blur, a larger Camera (AA) sample is required, and therefore the other sample values can be reduced. Likewise, increasing Camera (AA) can also benefit indirect lighting, and therefore not as many diffuse samples would be required. However, if you were to reduce noise on the skin of a character, it would be more efficient to have higher SSS samples and not such a high value for Camera (AA) samples.

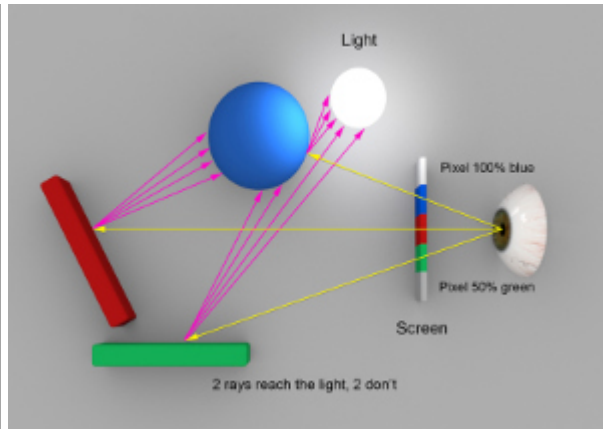
Light Sampling •

Direct lighting rays can be described as rays which deal with lights. These rays travel from a position in the scene toward the various light sources. These rays determine if a surface is in shadow, and if not, lighting information can be calculated.

Noise from lights can sometimes be difficult to diagnose, particularly if it is a light with a large area or size. In these cases, it can sometimes be mistaken for indirect diffuse noise. This highlights the necessity for testing different noise ray types. If the issue is shadow noise, then we can simply toggle ignore shadows in the Arnold render settings, and the noise will completely resolve. The images below show how a light is traced in Arnold.



Some light rays hit an object while others do not



Light sampling in Arnold

The same noise problems described earlier can occur with direct and indirect rays. Noise from direct light sources usually appears in specular highlights or in large, soft shadows from area lights. If this is the case, the number of light samples needs to be increased. The images below show the difference in the shadow noise when increasing the number of light samples from 1 to 3.

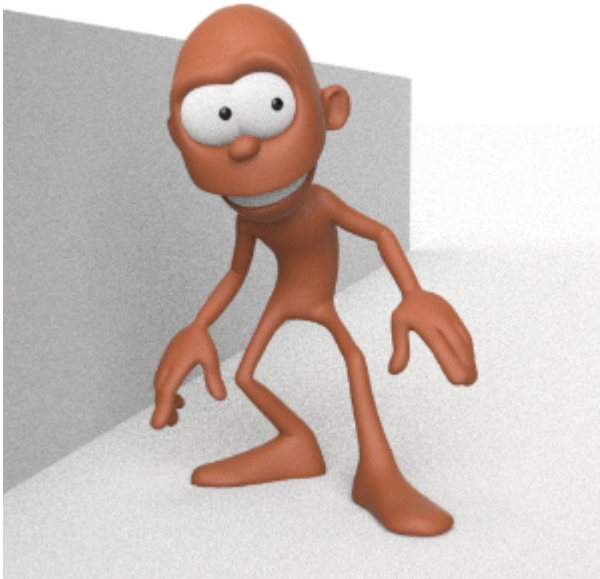


1 light sample. Very noisy.

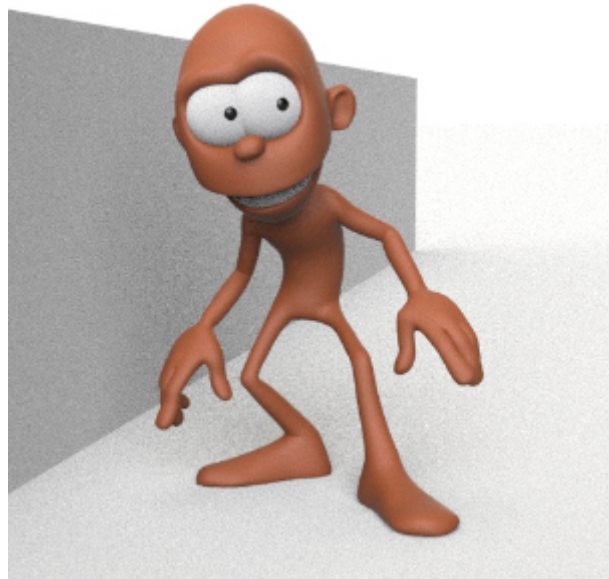


3 light samples. Noise is improved.

It can be useful to separate one source of noise from another when trying to evaluate noise in a scene. Disabling diffuse rays (0) can help identify which type of noise is present. When attempting to optimize the number of indirect diffuse rays in a scene, rather than increasing the number of diffuse rays (render time will double for each diffuse sample), the source of the noise will need to be isolated first.



Both light noise and Diffuse noise (difficult to tell which is which).



Diffuse set to 0 (disabled). Light noise is easier to identify.

Diffuse Ray Sampling



Indirect (diffuse) lighting from emissive green cube. Rollover image to view direct lighting only (diffuse samples: 0)

Indirect diffuse rays are rays that interact with objects and their surface shaders. Therefore, rays travel in the scene in directions determined by the shader assigned to the object. Transmission rays travel through objects. When a ray hits a reflective object, such as a mirror, only one reflection ray is traced from that point in a direction determined by the incoming ray. In contrast, diffuse rays are sampled randomly over the hemisphere around the hit point.

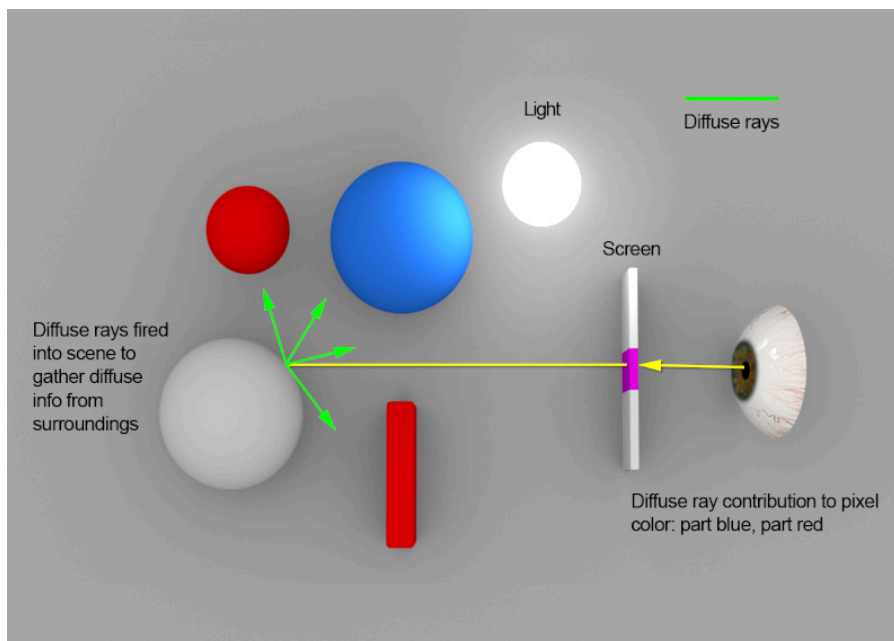
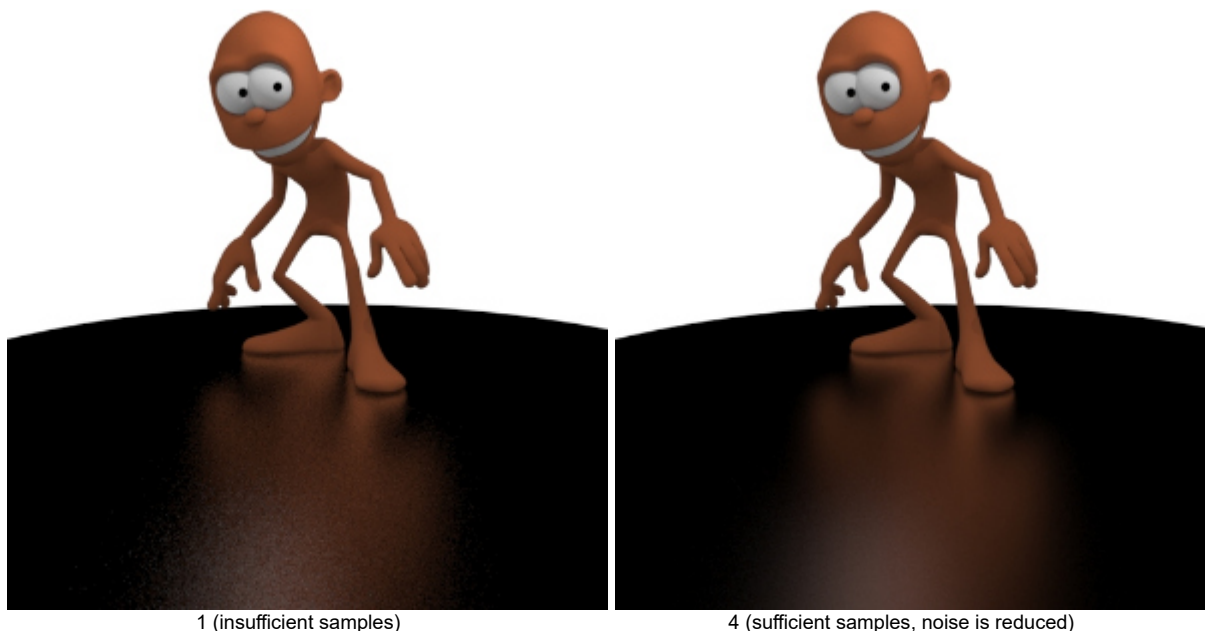


Diagram showing how diffuse rays are propagated in an Arnold render

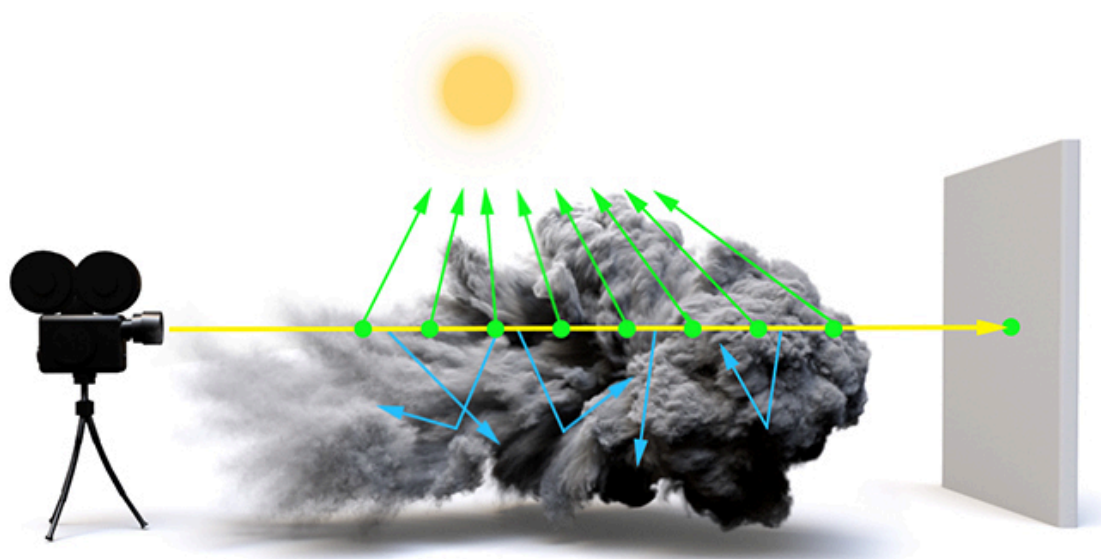
Specular Ray Sampling

When rendering glossy specular surfaces, noise can also be a problem, as with diffuse rays. Bright hotspots can cause noise in indirect specular samples. For example, wide specular reflections of small, bright lights. If the issue is noise in a specular highlight, it needs to be confirmed that the source is the direct light and not a secondary ray type (such as specular). This is easy to achieve by setting the `GI_diffuse_depth`, `GI_specular_depth` to zero (this essentially turns off all global illumination). If the noise is still there, we know it is the direct specular component of the illumination model. In most cases, it can be resolved by increasing the number of specular samples.



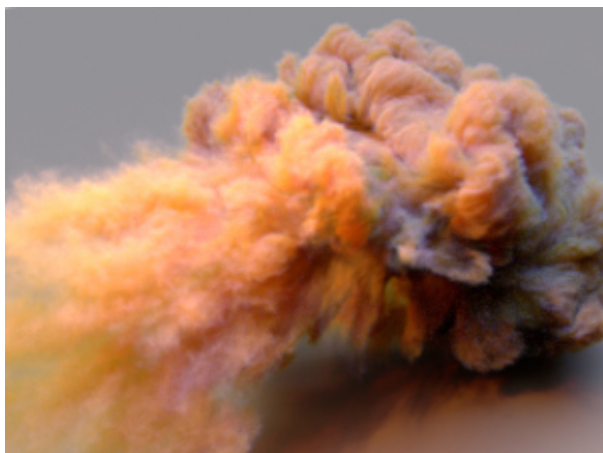
Volumes •

When sampling volumes, direct Camera (AA) rays sample the volume multiple times as they travel through it. Indirect rays (`volume_indirect`) behave similarly, sent multiple times as the ray 'steps' (`volume_step_size`) through the volumetric object. Therefore, sampling a volume usually takes longer to calculate than sampling a surface. •

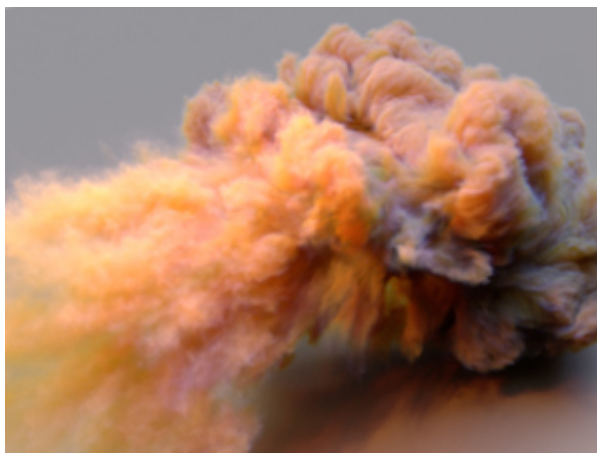


Volume sampling. Direct lighting (green). Indirect lighting (blue).

At each step the ray travels, it evaluates the shader and accumulates the density of the volume. When these density values become more erratic throughout the volume, nearby rays can calculate substantially different values, introducing noise into the render. In this case, it may be necessary to render using more light volume samples or to reduce the step size in the volume. Even with low sampling values, it can be expensive to render clean images with volumes.



Light volume samples: 1



Light volume samples: 6

There are three things to consider when rendering volumes; raymarching, direct light, and indirect light sampling, all of which have different sampling settings.

Step Size

The step_size of the ray marching process. Too low a sampling rate here will cause the volume rendering to miss small details in the volume, making them noisy, making them thinner than they should be, or making a bad estimate of the exact position of the 'surface' of a 'solid' volume like a pyroclastic cloud. A bad step size has a snowball effect that can make the rest of the sampling process noisier. This is pretty easy to test by looking at the alpha channel, or by adding some emission to the volume and turning off all lighting. If the emission/alpha channel is noisy at the desired Camera (AA) setting, then the step_size is probably too large.



Step size correct



Artifacts appear in the alpha when the step size is too large

Step_size is actually in object-space and not in world-space. This is so that the sampling quality remains the same upon scaling of the volume's transformation. However, unreasonably large step sizes should be avoided. Otherwise, the volume will appear thinned and washed out. In the example images below the volume has been scaled up to 100 units to exaggerate this effect (1/25/50 are relative to the size of this volume).



1



25



Generally, the step_size needs to be as large as possible before visibly noticeable artifacts occur. Low step sizes will increase render times. For example, when the step size is 0.1, and the volume is 10 units big in world space, there are approximately 100 primary samples, and therefore the volume shader is called 100 times.

Direct Lighting

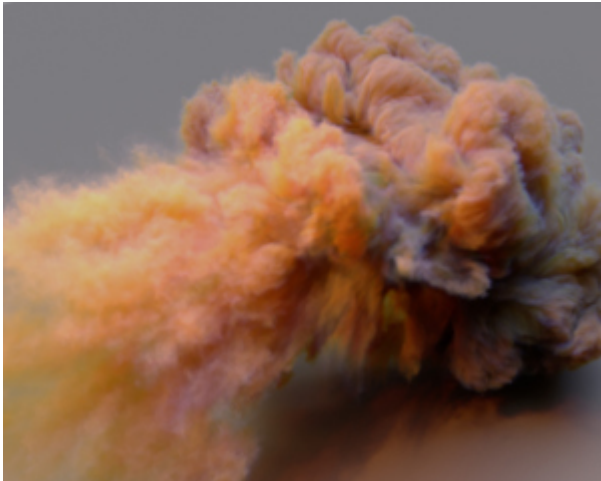
When the step size is correct, noise can come from either direct or indirect lighting. Direct lighting is pretty easy to detect. The indirect light needs to be toggled off, and if the render is noisy, each light needs to be rendered one at a time until the light that is responsible for the noise is found, and turn up their volume_samples. If the direct lighting noise doesn't significantly decrease as volume_samples increases, then it's probably due to a too-low step_size, or simply difficult to sample light interactions (highly anisotropic volumes, heavily textured mesh_lights, quadratic falloff, etc.). In this case, there's not much to do except to eliminate the samples via clamping or filtering.

Indirect Lighting

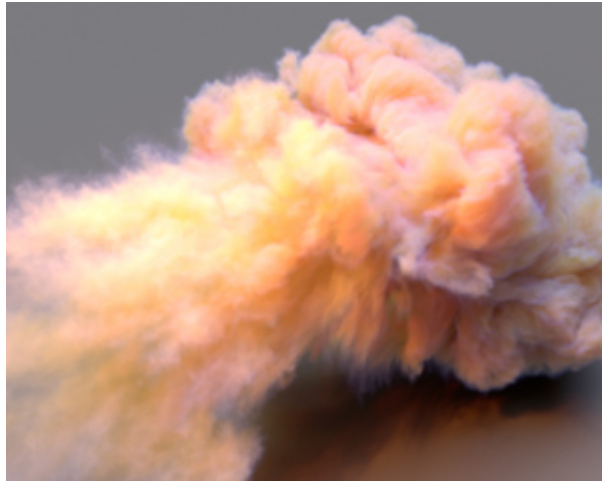
Indirect noise is also easy to identify. `GI_volume_depth` of a noisy volume needs to be disabled, and if the render is no longer noisy, then the problem is indirect sampling. If the quality needs to be improved, the number of `volume_indirect_samples` needs to be increased. If increasing this doesn't reduce noise, then the noise could again be from difficult to sample light interactions (multiple scattering, anisotropic volumes, lights embedded in the volume, etc.), and as mentioned before there's not much to do except to eliminate the samples via clamping or filtering.

Volume ray depth

Increasing the `volume_ray_depth` can make a dramatic difference to the appearance of a volume. However, be aware that increasing the volume ray depth will increase the number of multiple scattering bounces within the volume (defaults to 0), and therefore render times will increase significantly.



0



6

-
-
-
-

 Further information about sampling can be found [here](#).

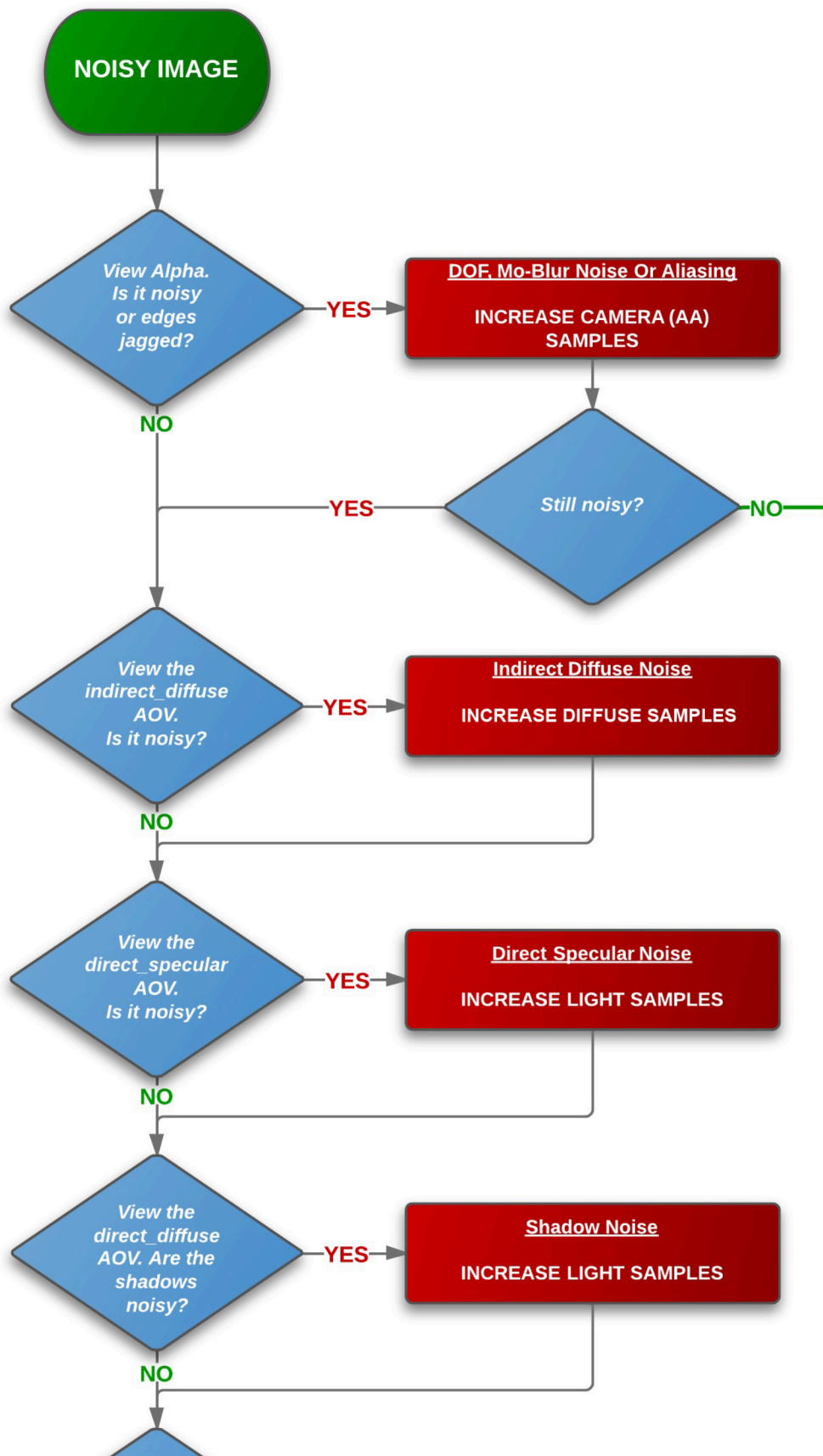
 Further information about removing noise can be found [here](#).

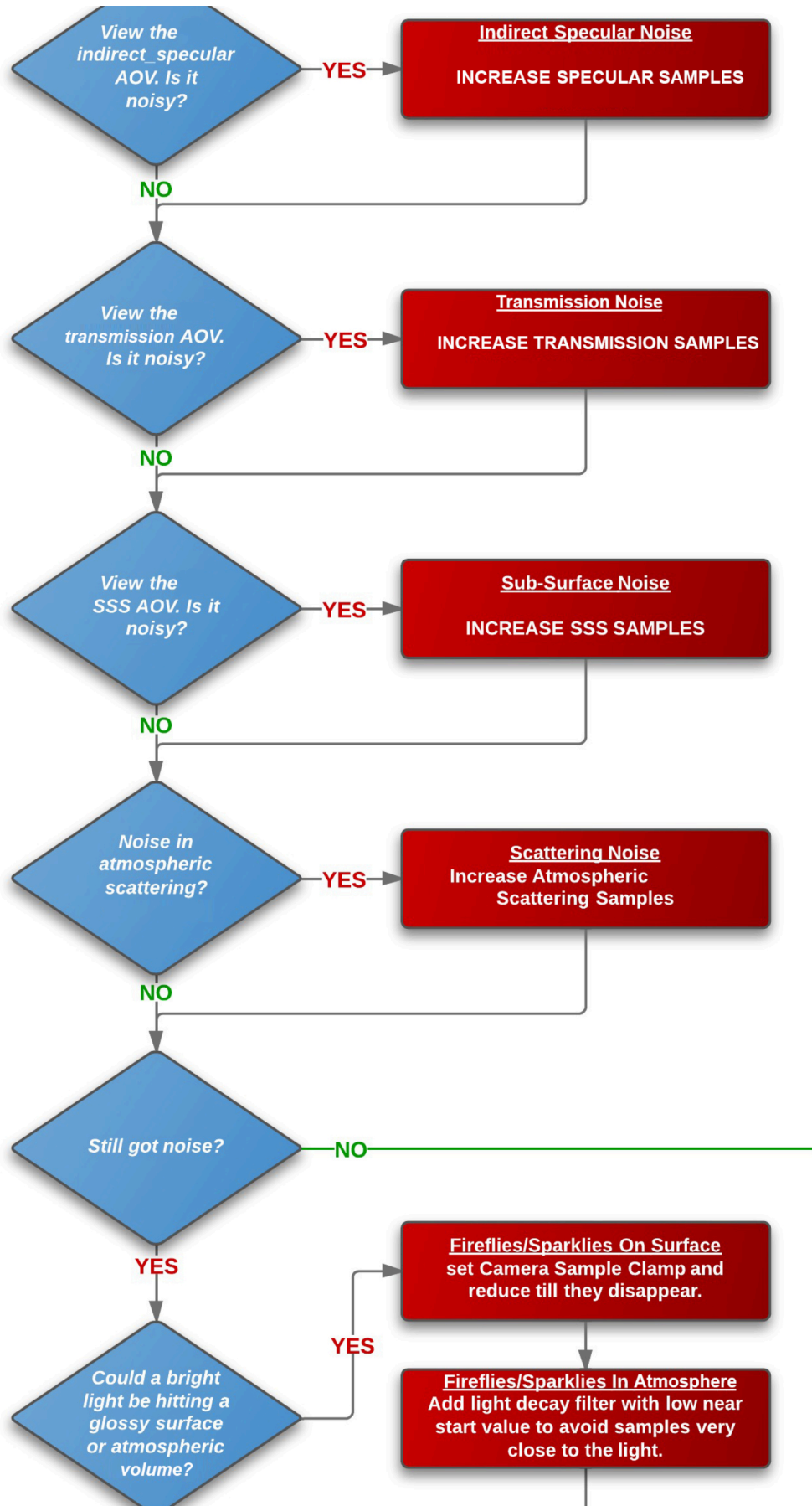
-
-
-

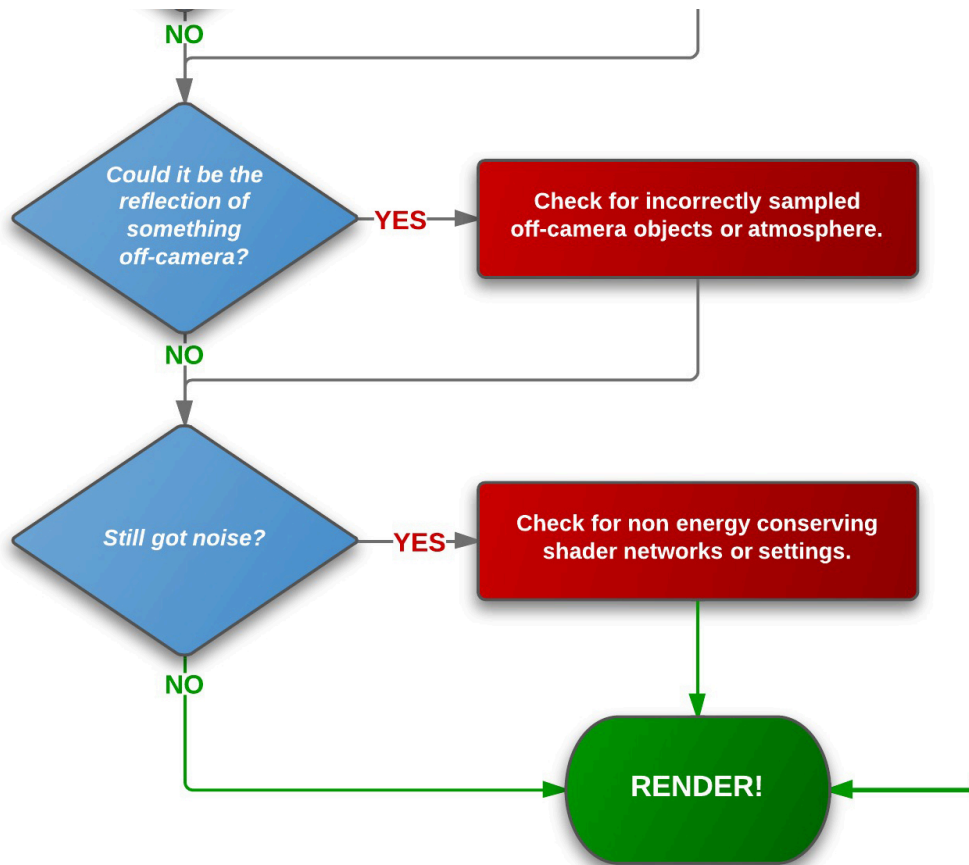
Pepe model by Daniel M. Lara (Pepeland)

Removing Noise Workflow

The diagram below shows a good method for identifying the type of noise in your scene and which sampling parameter to use to improve it.







•
•

Getting rid of noise

! This section is mostly obsolete, you should be using the [Skydome](#) light instead.

Using the Ray Switch Node

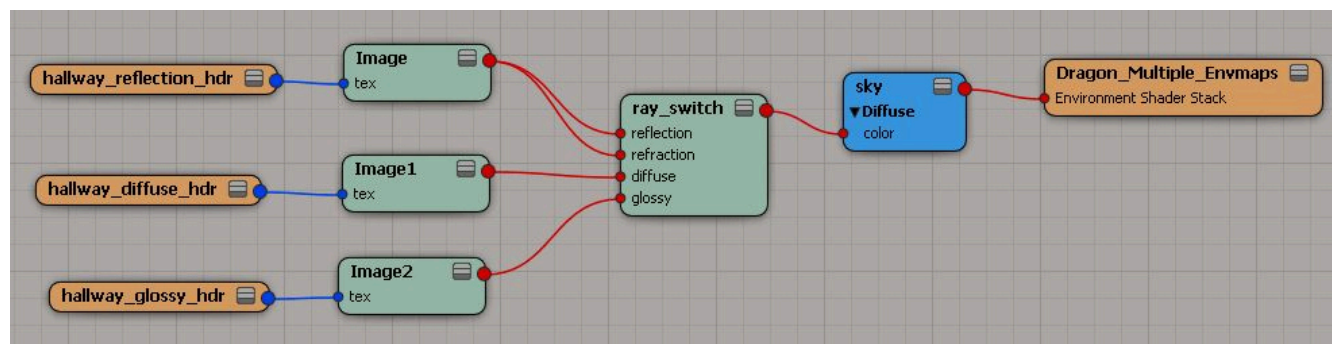
Using a very high contrast HDR image for lighting a scene can produce noise which can be hard to get rid of simply by increasing the AA or Hemi/specular samples. An HDR image contains a tiny sun disc whose pixel values are in the hundreds of thousands or even millions, is going to need a lot of samples for Arnold's brute-force hemispherical sampling to find. On the other hand, an HDR map of a thick overcast day with no sun at sight is very easy to sample.

In order to fix the excessive sampling noise, one could simply blur the environment map so that it's easier for the GI and specular rays to hit bright areas. However, that would cause the render to lose all the nice highlights seen in the reflections from the environment map. Instead, we will be using different environment maps for each ray type in the scene.

Typically, you will need 2 or 3 different versions of the same environment:

- One high-res and sharp map for reflection and transmission rays
- One mid-res and slightly blurred map for specular rays
- One low-res and very blurred map for diffuse GI rays

For this purpose, you can use the "Ray_Switch" on the pass rendertree. Below is an image illustrating such a setup:



Rendertree for the sky shader using different images for each ray type

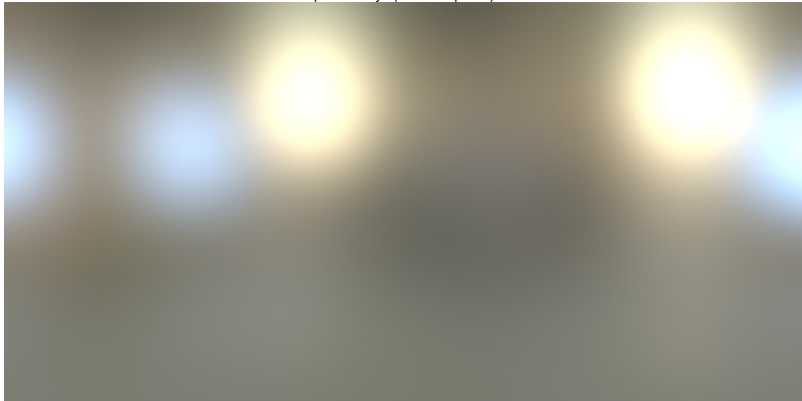
And here are the images used for the different ray types, using a Gaussian blur in PhotoShop:



HDRI used for Specular/Transmission-rays (3000x1500 pixels)



Specular rays (800x400 pixels)



Diffuse/GI rays (right, 200x100 pixels)

- Depending on the dynamic range of the HDRI you may need to blur more or less to achieve optimal results. Also, how blurry the map used for specular rays should depend on the material; for very sharp materials (almost mirror-like) you don't need as much blurring, but if your specular reflections are very soft you should use a bigger blur.

•

Here's the resulting image from this setup:



Dragon model lit with sky shader using separate HDRI's for each ray type

Note how with this setup the noise is now gone and the illumination is very similar to the version using a single high-res map for all ray types.

Different Amounts of Blur for the Diffuse Environment Map

i Note: this section is mostly obsolete, you should be using the Skydome light instead.

The following images illustrate the effect of varying the amount of blur in the HDRI's used for diffuse GI rays. To make things simpler, all the illumination comes from the HDRI - there is no explicit key light. As expected, the more you blur, the lower the noise, at the cost of reduced lighting detail (e.g., softer shadows):



Unblurred, high-res HDRI map, 3 diffuse samples, render time 5 mins



Unblurred, high-res HDRI map, 12 diffuse samples, render time 40 mins; sharp shadows are preserved, but still a bit noisy even at such high sample settings



Blurred HDRI map, 3 diffuse samples, render time 5 mins; this renders fast and has low noise, though shadows start getting a little softer.



Extremely blurred HDRI map, 3 diffuse samples, render time 5 mins; noise-free, but note how most of the lighting detail is lost.

Other ways to reduce noise

As a last resource, if you find that the render is still noisy despite using the methods described above, you can try clamping the pixel samples in the render options.

- By default, pixel samples are unclamped and can have infinite brightness values. If your environment map, for example, has a sun with a value of say 5000 that will cause a lot of noise even if it's blurred. Typically this isn't a problem for the diffuse/GI environment map as it's extremely blurry, however, the map used for Specular rays might need to stay relatively sharp and can still introduce noise in some cases.

Below are two renders of the same dragon model used above, with a purely glossy shader (no diffuse component):



Dragon with samples unclamped



Clamping set to 2

Clamping the pixel samples should be used with care as it's obviously reducing the dynamic range of the render. However, it can be an effective way to remove certain types of noise that might otherwise be difficult to get rid of.

-
-



Arnold Render Options

SltoA has its own Render Options panel to control Arnold-specific features such as multi-threading, sampling quality, etc. These Render Options can be global to all passes or apply to a specific pass only.

To open this panel, from the Softimage menu, select Explorer / Passes or Render / Renderer Options (or bring up the Render Manager via Render / Render Manager):

The menus provided on each tab beneath Arnold Render Options are described in more detail in the topics below:

- [System](#)
- [Output](#)
- [Sampling](#)
- [Motion Blur](#)
- [Depth of Field](#)
- [Subdivision](#)
- [Ray Depth](#)
- [Textures - settings](#)
- [Diagnostics](#)
- [ASS Archives](#)

•

•

System

The System tab gives access to general Arnold system-wide controls.

System Output Sampling Motion Blur
Subdivision Ray Depth Textures Diagnostics

ASS Archives

Multi-threading
Autodetect ☒
Number of Threads

Buckets
Scanning Size

Progressive Refinement
-3 ☒ -2 ☒ -1 ☒ 1 ☒

Scene Rebuild Mode

Licensing
Skip License Check ☐
Abort On License Fail ☐

Error Handling
Abort On Error ☒
NaN Error Color R 0.000 G 1.000 B 0.000
Texture Error Color R 1.000 G 0.000 B 0.000

Search Paths
Plugins ...
Procedurals ...
Textures ...

User Options
Options ...
Resolve Tokens ☐

Reset Options

Multi-threading

Autodetect

Use an optimal number of render threads based on the number of hardware processing cores.

Number of Threads

If autodetect is disabled, this will let you manually set the number of rendering threads. Negative numbers are also allowed. If specifying 0 threads means use all cores on a machine, then negative numbers can mean use all but that many cores. For example, threads=-2 means use all but 2 cores, while threads=2 means only use 2 cores. This is useful when you want to leave one or two cores for other tasks. One example of this is so that Softimage can be more responsive while Arnold is rendering in the Render View.

Buckets

Scanning

The spatial order in which the image buckets (i.e. threads) are processed. By default, buckets start in the center of the image and proceed outwards in a spiral pattern.

Size

The size of the image buckets. The default size is 64x64 pixels, which is a good compromise; bigger buckets use more memory, while smaller buckets may perform redundant computations and filtering and thus render slower but give initial faster feedback.

Progressive Refinement

-3..1: the AA steps for iterative progressive render region.



.

Scene Rebuild Mode

This dropdown lets you select the scene rebuilding policy during an IPR session. Options are:

- Auto : SltOA tries to detect the type of change happening, and destroys the scene when the event is incompatible with IPR. The scene is also destroyed on a frame change.
- Always : the scene is always destroyed and rebuilt, for any change (for example, when orbiting the camera).
- Manual: the scene is never destroyed, it's up to the user to call the destroying command. The scene is destroyed on a frame change.
- Fly-through : the scene is destroyed on an incompatible change (for example when moving a point) but not on a frame change. On such an event, SltOA updates the camera, all the kinematics, lights, and shaders.

In Manual mode, when an incompatible event is detected, a message is logged in the script editor, warning the user about the type of change. Fly-through works also when rendering a pass, and in batch mode. It's ignored when exporting a frame or sequence of frames to .ass.

Licensing

Skip License check

Switch this on to avoid taking an Arnold license from the license server. This will render your image with a watermark.

Abort On License Fail

If on, rendering will stop if the license is not detected when the render begins.

Error Handling

Abort On Error

If set, rendering will stop as soon as an error is detected. This is the recommended setting. In general, you should not ignore important error messages, or you'll risk crashes, rendering artifacts and undefined behavior.

NaN Error Color

The color that will be corresponded to a NaN (Not a Number) value.

Texture Error Color : the color rendered in case a texture error is detected (for instance if the texture is not found on disk).

Search Paths

Lets you configure various search paths: shader plug-ins, procedurals and texture maps. You can enter an explicit path in the text box for any of these options. Use the Browse (...) button to choose a location.

Plugins

Defines a location to search for plugins. •Required if importing .ass files from other plugins.

Procedurals

The directory where to search for dynamically loaded Arnold procedural nodes.

Textures

The directory where to search for texture maps. This works in conjunction with the "Save Texture Paths" option in the ASS tab.



All the search fields can contain more than a path, separated by a ';' char. Environment variable are also supported. For more information, see [SlttoA Paths](#)

User Options

Options

This string allows you to set unexposed attributes of the Arnold's options node. See [User Options](#) for more details.

Resolve Tokens

If on, the Softimage tokens (if any), are resolved for the •Options string.

Reset Options

By pushing Reset All Parameters To Their Default Value , you can reset all the rendering options to their default values. The command will ask you for confirmation before proceeding, but, once run, you will not be able to undo it.

-
-
-

Output

The Output tab gives access to output driver settings:

•

The screenshot shows the Arnold Output tab settings panel. It is organized into several sections:

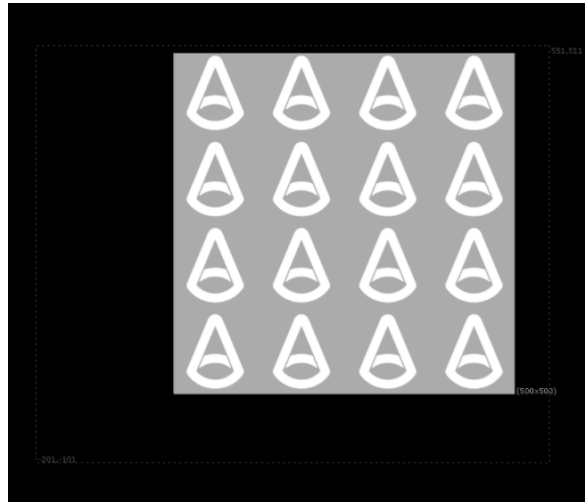
- Overscan (top, bottom, left, right):** Includes an 'Enable' checkbox and four input fields, each with a green square icon and the value '10'.
- Color Space:** A dropdown menu set to 'auto'.
- TIFF:** Includes 'Unpremultiplied Alpha' checkbox, 'Format' dropdown (set to 'tiled'), 'Compression' dropdown (set to 'lzw'), and an 'Append' checkbox.
- EXR:** Includes 'Format' dropdown (set to 'tiled'), 'Compression' dropdown (set to 'zip'), 'Preserve Layer Name' checkbox, 'Autocrop' checkbox, and an 'Append' checkbox.
- Metadata Name/Type/Value:** A section with three rows of input fields and dropdown menus (INT, FLOAT, STRING). Below are 'Add' and 'Remove' buttons.
- Deep EXR:** Includes 'Enable' checkbox, 'Subpixel Merge' checkbox (checked), 'Use RGB Opacity' checkbox, and two sub-sections: 'Alpha' and 'Depth'. Each has a 'Tolerance' input field (set to 0.01) and a 'Half Precision' checkbox.
- Dither LDR Images:** A checkbox at the bottom, which is checked.

Overscan

Overscan extends the render region beyond the regular image coordinates. It can be either a percentage or a pixel value. You can mix % and pixel values at the same time, however, if you don't specify the unit it will default to pixels. It can have:

- One value such as 10px to specify an equal margin on every side.
- Two values, such as 10px 5px, to specify the top/bottom (first value) and right/left (second value) margin.
- Three values, such as 10px 5px 2px, to specify the top (first value), right/left (second value) and bottom (third value) margin.
- Four values, such as 10px 5px 2px 1px, to specify the top, right, bottom, and left margins respectively.

E.g. a ten-pixel overscan for a 640x480 image in all directions can be achieved with options region_min_x, region_min_y, region_max_x, region_max_y of -10, -10, 649, 489. The example below shows a pixel value of 10, 50, 100, 200 (top, right, bottom, left) in the 'Overscan' field:



Overscan set to 10, 50, 100, 200 and the result viewed in Nuke



There is a way to view the output image of a render in progress. You must do the following:

- Set your output to be tiled EXR, zip is fine.
- Set the bucket scanning method to 'top'.
- Use a viewer like imf_disp.

Color Space

Arnold has a built-in color manager. It implements the open-source [OpenColorIO \(OCIO\)](#) color management system and uses an OCIO v2 config shipped with Arnold. The color manager is configured to leverage the industry standard [ACES](#) color encoding system, setting the rendering color space to [ACEScg](#). If no color manager is linked to in `options.color_manager`, the built-in one will take over. The default OCIO color manager points at the default OCIO config.



The name of the default OCIO color manager node is called `ai_default_color_manager_ocio`.

Dither • LDR Images

Applies a small amount of noise to the final pixel values. This helps minimize banding artifacts apparent on 8-bit output image file formats.

-
-



TIFF

Unpremultiplied Alpha •

Specifies whether the main RGB channel is premultiplied with the alpha channel (the default) or not. It only works with 8-bit output images.

Format

Allows you to save the file in scanline or tiled mode. Scanline may work better with composition packages like Nuke or Fusion.

Compression

The compression mode of the image file.

Append

If true Arnold will preserve previously rendered tiles and only work on the missing ones, appending them to the output files. If no image is present the render will proceed as normal creating a new image.

EXR

Format

Allows you to save the file in scanline or tiled mode. Scanline may work better with composition packages like Nuke or Fusion.

Compression

The compression mode of the image file.

Append

If true Arnold will preserve previously rendered tiles and only work on the missing ones, appending them to the output files. If no image is present the render will proceed as normal creating a new image.

Preserve Layer Name

If checked, the data of the different AOVs will be stored in the EXR in layers with the names of the AOV, and not the RGB

Autocrop

Autocrop removes pixels where alpha and all other channels are zero. It contains a bounding box around the content of the image, which informs a compositing application which part of the image needs to be identified and which part can be skipped. This is particularly useful when you have a CG element that is not filling the whole frame (see image below). Only a small part of the image will be processed in the compositing application. Therefore it will only load the data window and skip the rest.



The autocropped area that has been identified in a compositing application

Metadata

This group lets you add a set of generic metadata to all the EXR image files. Each data must have a name, a type (int, float, point2, string, matrix) and a value.

For point2 and matrix types, the values must be separated by a space (my_point POINT2 3 4).

Metadata names can contain spaces. In this case, you must use single or double quotes as delimiters ("my float" 1.0).

You can add a line with the Add button, remove the bottom line with the Remove button.



Starting from 3.11 (Arnold 4.2.11.0) the metadata apply to deep EXR outputs as well.

-
-
-



.

Deep EXR

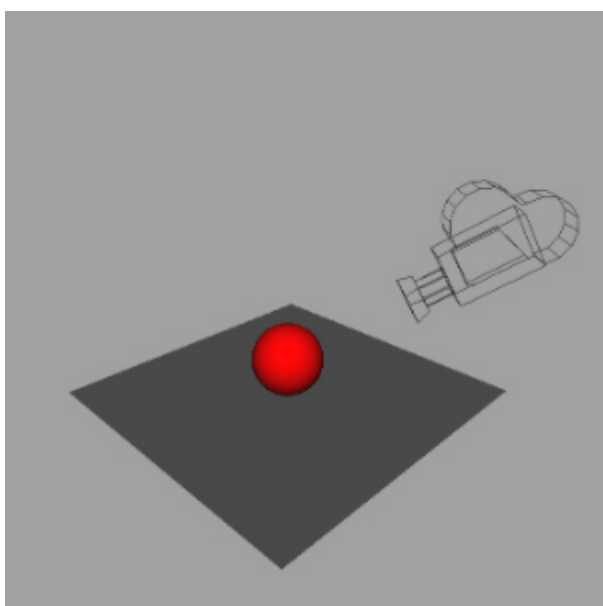
Deep EXR files store a variable-length list of samples stored with multiple values at different depths for each pixel.



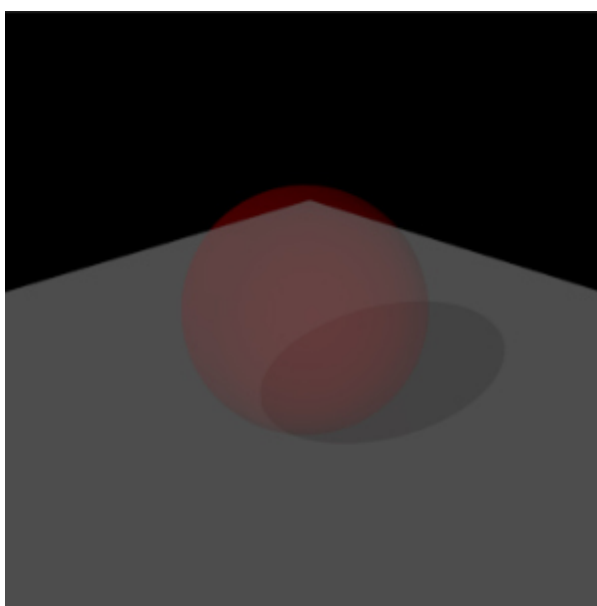
Volumes are visible in deep renders. However, older atmosphere shaders and volumetric mattes are not supported yet.

The following example demonstrates working with Deep EXR files.

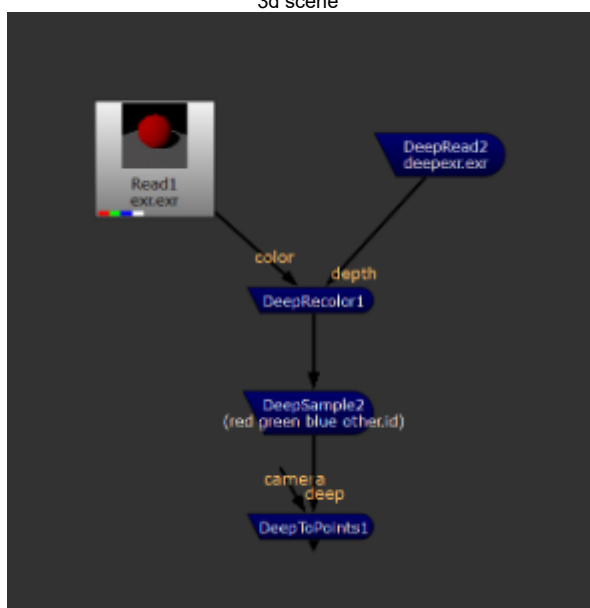
- First, change the output format to deepEXR. Render the scene and open it in Nuke using a **DeepRead** node.
- In Nuke, create a **DeepRecolor** node. Connect the depth input of the **DeepRecolor** node to the deep EXR. Next, connect the color input of the **DeepRecolor** node with the 2D EXR render.
- Connect a **DeepSample** node to the **DeepRead** node. The **DeepSample** node will give you a list of samples for each pixel (see images below).
- Create a **DeepToPoints** node and connect **deep** to the **Deep Sample** node.



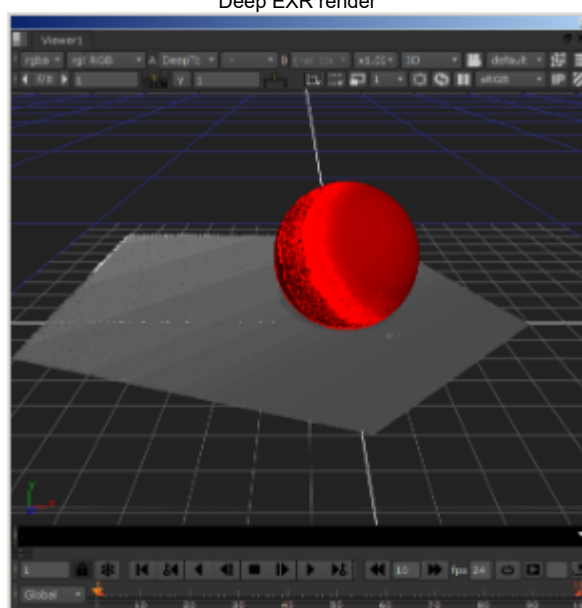
3d scene



Deep EXR render

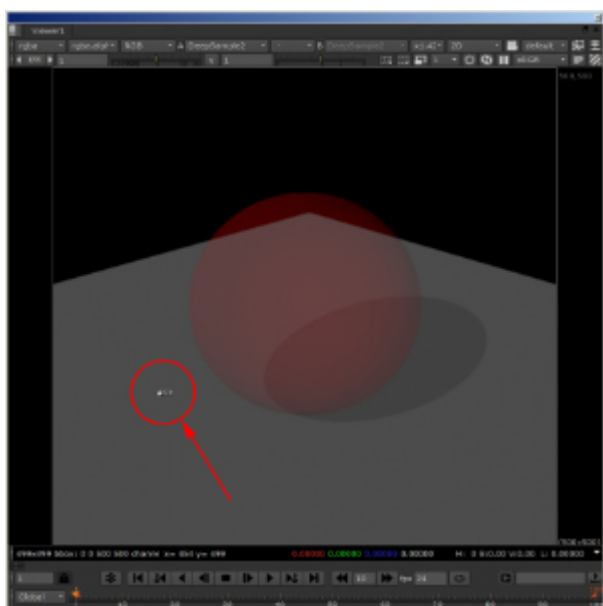


Node graph in Nuke

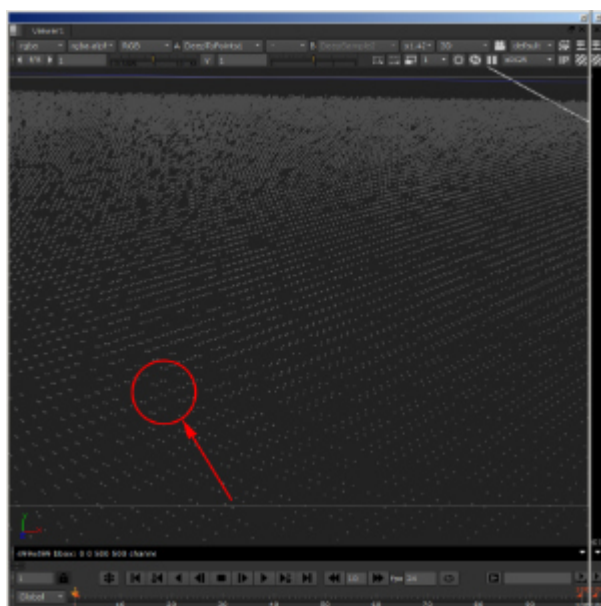


3d view in Nuke

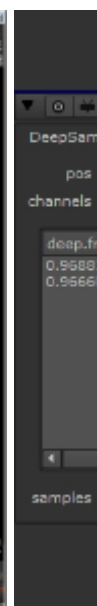
- Select the •Deep Sample •node. Drag the 'pos' dot in the 2d viewer where you want to view the samples. You should see that there are two samples in the •Deep Sample node.



Deep sample viewed in 2d



2 Deep samples viewable in 3d



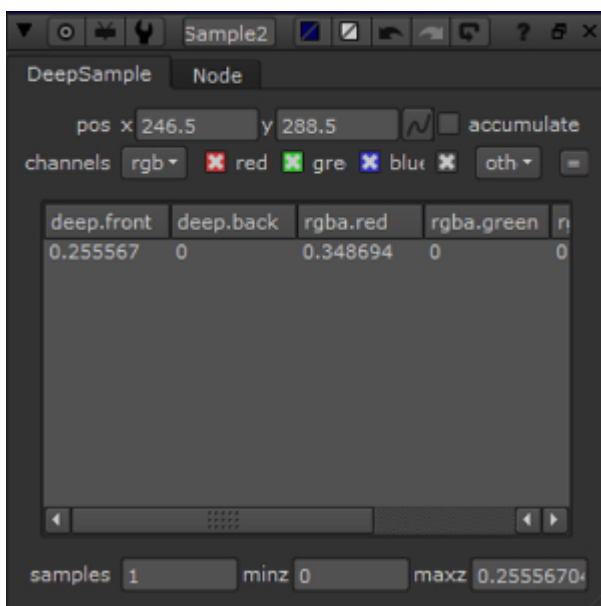
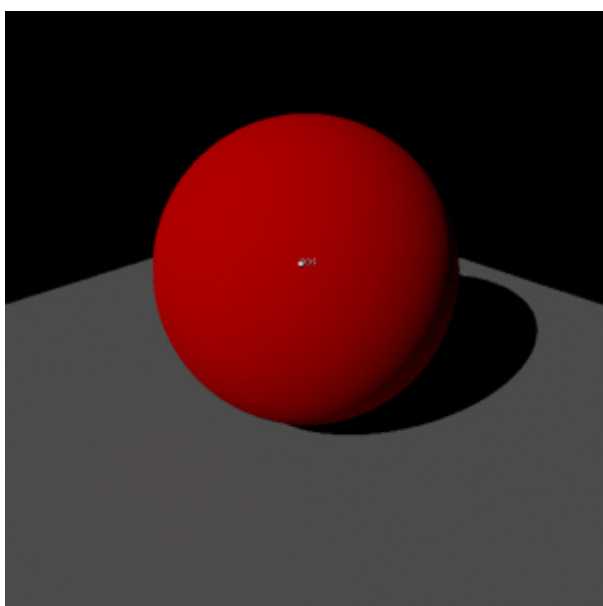
Enable

If enabled, all the exr output files of the current pass are written as deep images. So, you cannot have some flat and some deep AOVs at the same time.

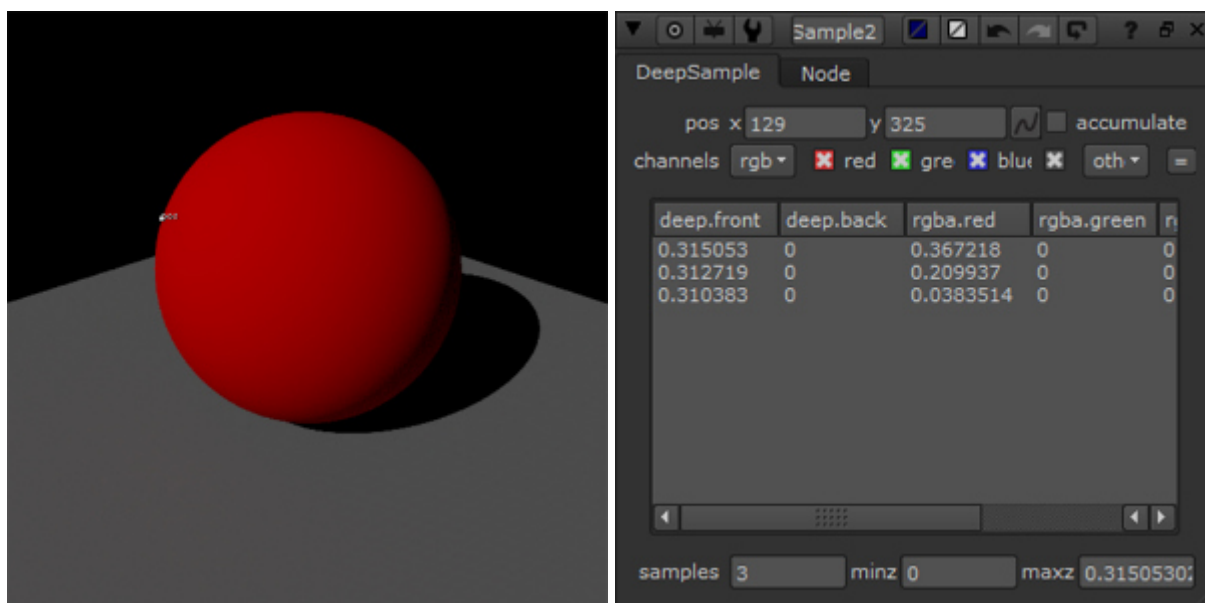
Subpixel Merge

Subpixel merge takes into account screen space resolution to further compress the file without losing too much accuracy, keeping more deep samples where they are needed. Note that adding large 'Tolerance Values' will override the default behavior of the 'Subpixel Merge'.

Subpixel merge's effect is obvious when we view a pixel at the edge of the sphere compared to a pixel that is in the center. The pixel at the edge of the sphere will get more deep samples to preserve the detail in the antialiased edge:



One sample is visible in the center of the red sphere



Three samples are visible on the edge of the red sphere

i If Subpixel Merge is disabled, the total number of samples will increase dramatically.

•

i If you need to disable all sample merging, you can do so by setting all tolerances to 0 and turning off Subpixel Merge.

If you increase any of the samples values in the render settings, for example the Camera samples, the total number of samples for each pixel will be the same.

Use RGB Opacity

By default a single channel alpha is used, setting this to true uses RGB opacity, saved as RA, GA, BA channels. Nuke only expects an alpha channel, so this might require further work in comp.

Alpha / Depth Tolerance

If the opacity / depth difference between two samples along the depth is smaller than the respective threshold, samples will be merged.

Alpha / Depth Half Precision

If on, stores alpha / depth with 16 bit precision. Beneficial in reducing file size.

Layers Tolerance / Filtering •

Here you can set the tolerance and disable filtering for each AOV. The half precision mode depends on the Softimage framebuffer setting (Half float for half precision, Float for full precision).

In order for two samples to merge, their samples should be under all tolerances for all of the values.

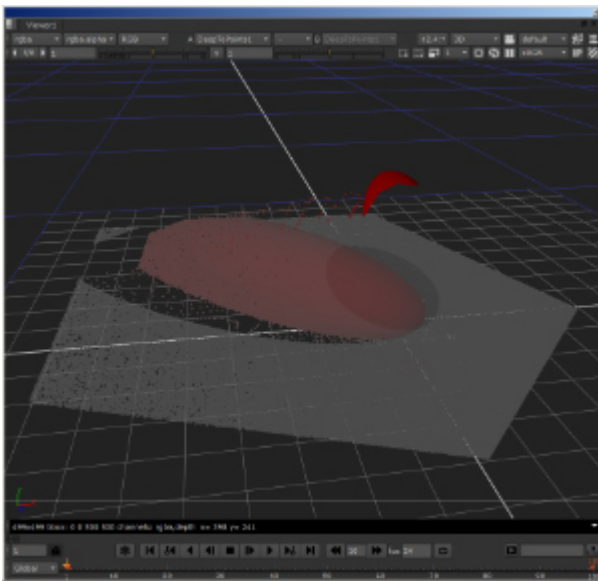
For example, if you have two samples in the beauty:

1: Depth: 0.9, Alpha: 1, RGB: 1, 1, 1

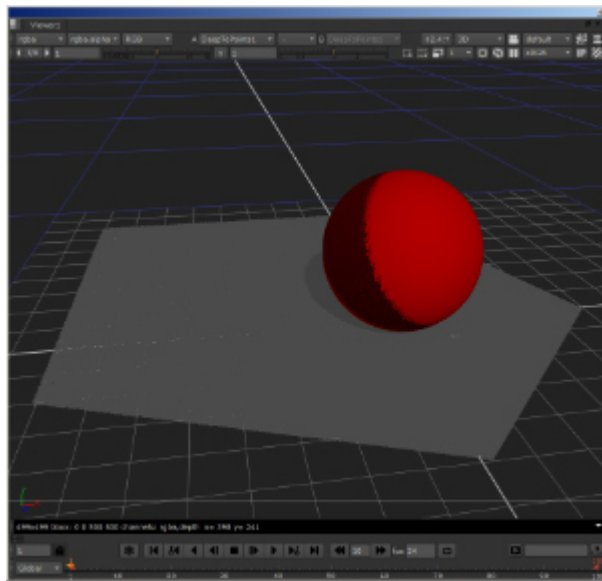
2: Depth: 0.8, Alpha: 0.9, RGB: 0.9, 0.9, 0.8

and you have these tolerance values: Depth: 0.15, Alpha: 0.15, Beauty: 0.15.

Depth and Alpha is beneath the tolerance value, but RGB is not, so they will not merge. With RGB the tolerance is checked against each component: R, G, and B.



Tolerance values are too high



Samples are correctly beneath the tolerance value

General

Dither Amplitude

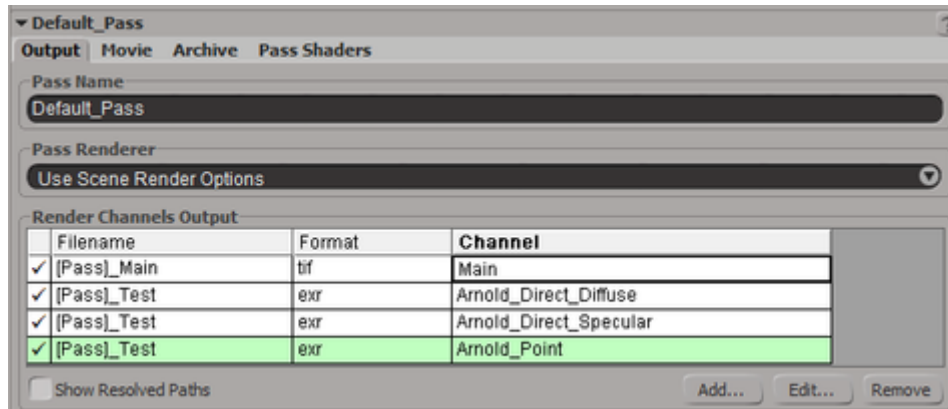
Applies a small amount of noise to the final pixel values. This helps minimize banding artifacts apparent on 8-bit output image file formats.

-
-
-

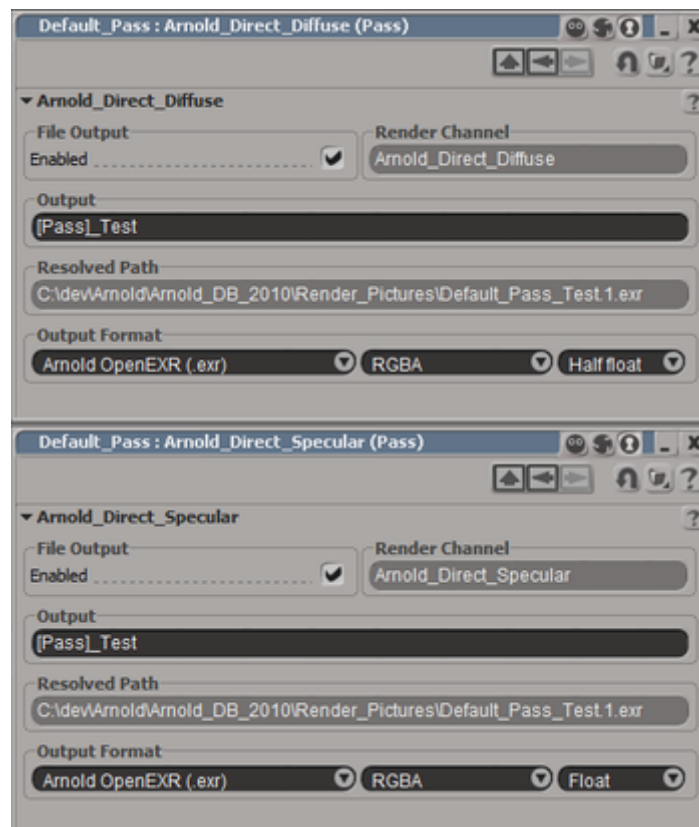
Multi-Layer EXR

You can have many AOVs (aka render channels) writing into the same EXR output file as layers. In SItOA, we detect the output filename for the render channels. If several render channels are of EXR type, and they share the same output filename, they will be layered into the same EXR file. There are no extra options to tune.

With the following setup, all the listed AOVs, except the main channel, are written into the same [Pass]_Test EXR file. •



The only limitation is that the bit depth (float/half float) must be the same for all the layers. If two or more EXR channels (with the same output filename) have different bit depth, the bit depth of the first channel in the list is used for the others also, and a warning is logged. •



In this case, for instance, the following warning is fired: •

INFO : [sitoa] Bit Depth for Arnold_Direct_Specular (float32) differs from Arnold_Direct_Diffuse (float16). Defaulting to float16



The length of the layers for EXR's must not exceed 31 characters, as from the file format specifications. If at least one longer name is detected, SltOA will log a message, but will not abort rendering. However, you may get wrong results in the written data.

-
-

Sampling

These settings control the sampling quality of the rendered images. Increasing the sampling rates reduces the amount of noise in the images, but at the expense of increased rendering time.

You will notice that this evolution is not linear, as, for each of these sampling rates, the actual number of samples is the square of the input value. For example, if Camera (AA) samples are 3, it means that $3 \times 3 = 9$ samples will be used for Anti-Aliasing. If diffuse samples is 2, then $2 \times 2 = 4$ samples will be used for the GI. The same applies to the other values.

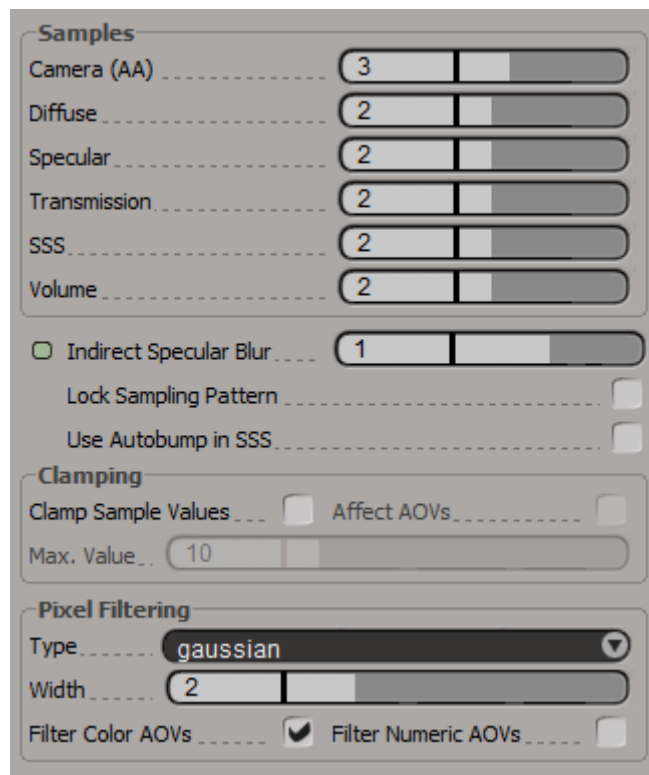
Diffuse, specular, transmission, SSS, and volume sampling rates are expressed for each Camera (AA) sample. This AA sampling rate can, therefore, be considered as a global multiplier for all the other ones. In this example, the total amount of diffuse samples per-pixel is, therefore, $9 \times 4 = 36$.

The different per-pixel sample counts are listed at the top of the section.

The distribution of SSS samples are lit as if they were a diffuse shading effect after a split, so 1 sample to each direct light and 1 indirect sample at each SSS probe/randomwalk that successfully hits a point of the surface. Arnold will only "split" a path once when recursing through the various types of bounces and scattering events. So for example, if you have a diffuse GI sample setting of 3, and the camera ray hits a diffuse surface first, this means that from that moment on there will only 9 GI paths traversing the scene, even if these paths recursively hit the diffuse surface, or hit a transmissive or SSS surface with a different sampling rate, no more than the 9 paths will be spawned (with some caveats)

 Further information about sampling can be found [here](#).

 More information about working with sampling and removing noise can be found [here](#).



Sampling is divided into the following sections:

- [Samples](#)
- [Clamping](#)
- [Sample Filtering](#)

Extra Attributes

Lock Sampling Pattern

Locks the AA_seed so that the sampling noise won't change with the frame number (avoiding the film grain look).

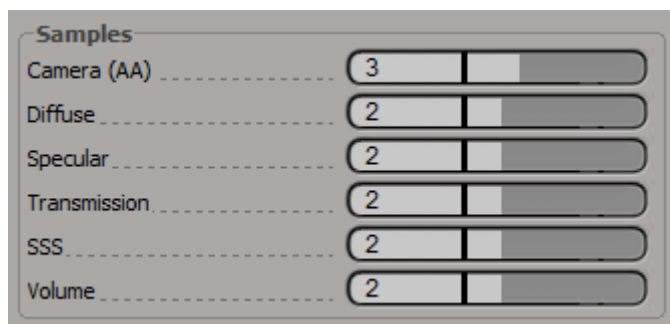
-

Use Autobump in SSS

If enabled, Arnold takes into account the effect that autobump has when computing the raytraced BSSRDF's result. This helps capture the high frequency details of the surface more accurately, at the expense of slightly longer render times.

-

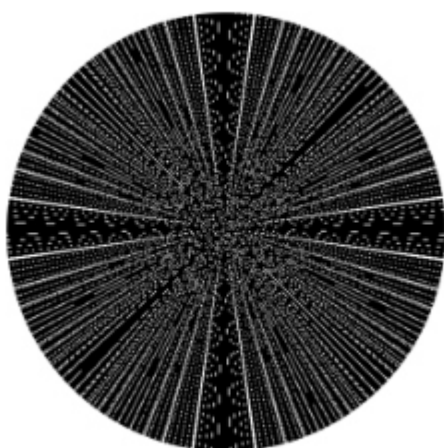
Samples



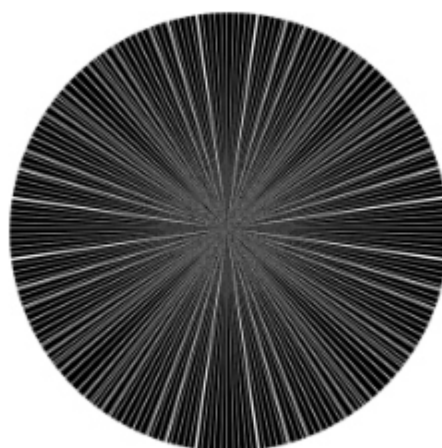
i More information about sampling and removing noise can be found [here](#).

Camera (AA)

Supersampling control over the number of rays per pixel that will be traced from the camera. The higher the number of samples, the better the anti-aliasing quality, and the longer the render times. The exact number of rays per pixel is the square of this value. For example, a Camera (AA) samples value of 3 means $3 \times 3 = 9$ -pixel samples. In practice, you may consider using a value of 4 for medium quality, 8 for high quality, and (rarely) 16 for super-high quality. This control acts as a global multiplier of all the different rays, multiplying the number of diffuse and specular rays. Motion blur and depth of field quality can only be improved by increasing Camera (AA) samples.



1



5

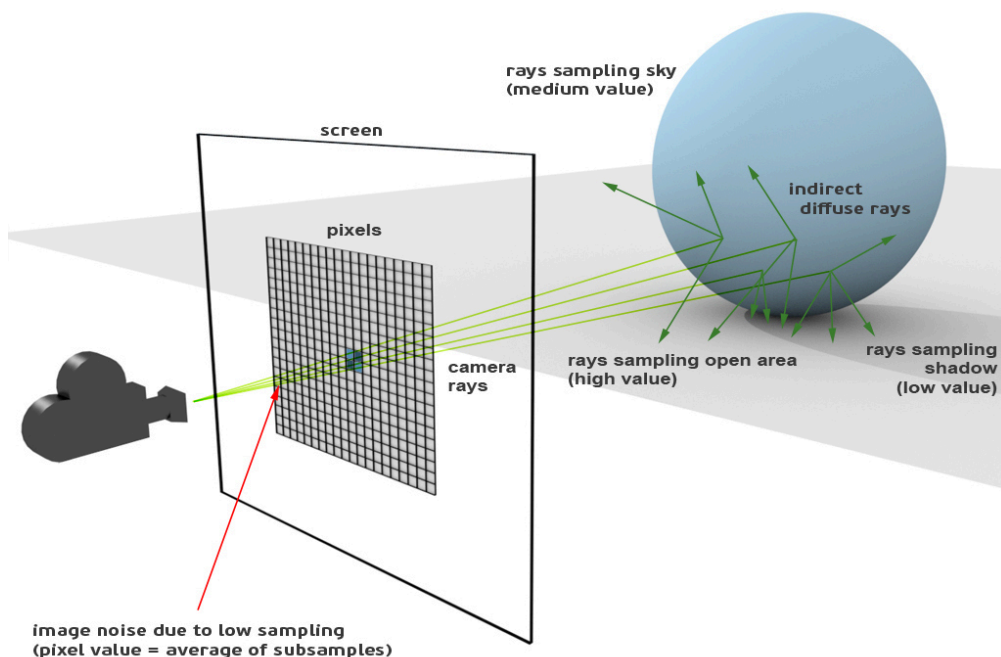
i Camera (AA) samples multiply diffuse, specular, and light samples after being squared. For example, 6 Camera (AA) samples and 6 specular samples = $6^2 \times 6^2 = 1296$ rays per pixel for the diffuse, and another 1296 rays per pixel for the indirect specular. Therefore when you increase the Camera (AA) samples to get better geometric anti-aliasing, you should decrease the others to compensate.

Diffuse

Controls the number of rays fired when computing the reflected indirect-radiance integrated over the hemisphere. The exact number of hemispherical rays is the square of this value. Increase this number to reduce the indirect diffuse noise. Remember that the diffuse sampling is done for each Camera (AA) sample, so high values for both Camera (AA) samples and diffuse samples will tend to result in slow renders.

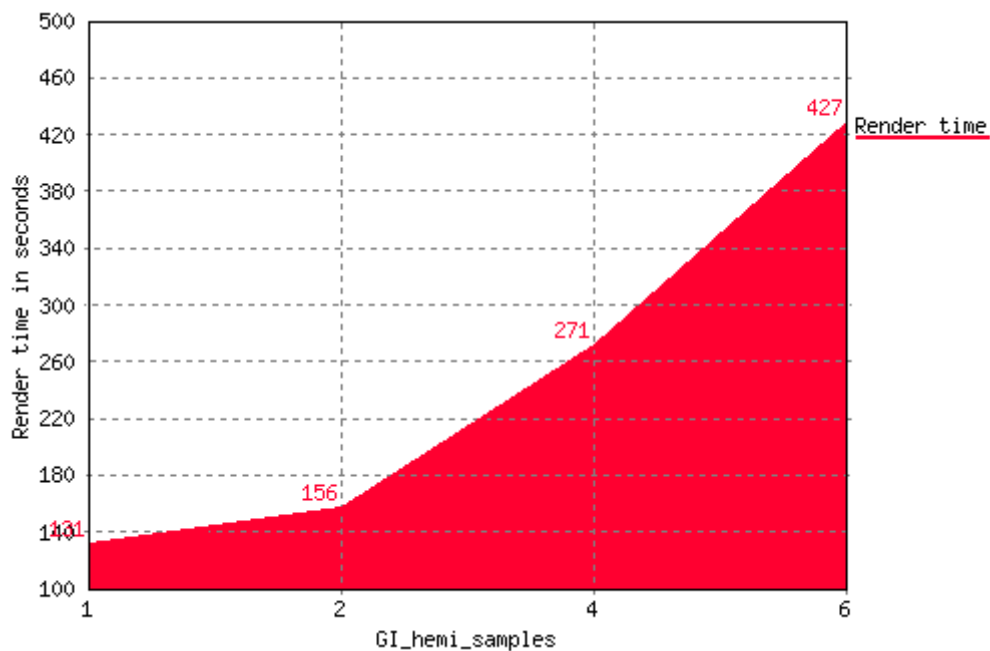
When diffuse samples are more than zero, camera rays intersecting with diffuse surfaces fire indirect diffuse rays. The rays are fired in random directions within a hemispherical spread. Noise is introduced when there are insufficient rays to resolve the range of values from the environment.

Increasing the number of diffuse samples will increase the number of diffuse rays fired from a point:•



Indirect Diffuse Ray sampling & Indirect Diffuse Noise

The table below shows the effect of increasing the number of diffuse samples (GI_diffuse_samples) to resolve indirect diffuse noise:



Diffuse samples (GI_diffuse_samples): 1 2 4 6. Render time: 131 156 271 427

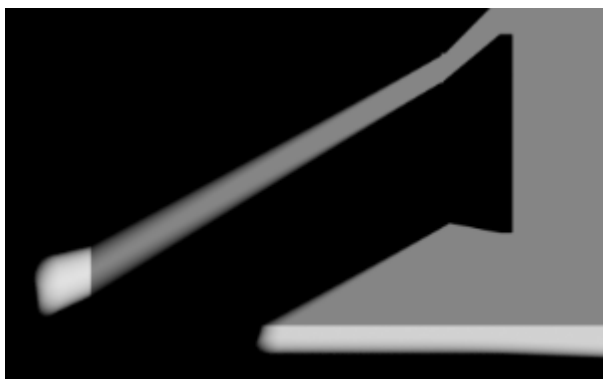
This shows the performance impact when increasing the number of diffuse samples (GI_diffuse_samples). Because indirect diffuse rays are so prevalent, this can get expensive. In this example, the performance hit from 1 to 6 samples was over 320%.••

Indirect diffuse noise

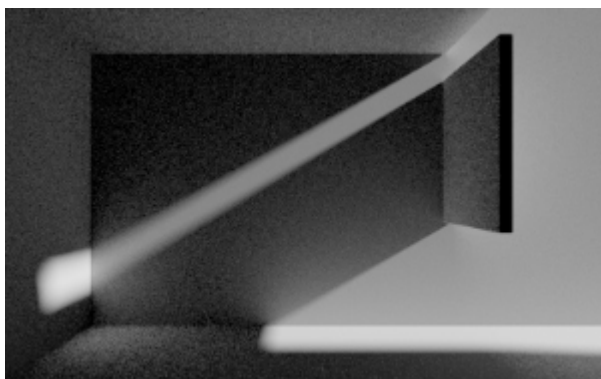
This is one of the most common causes of noise and can have a number of different sources. It manifests as granularity in the scene, usually in shadowed areas.

There are a couple of different methods to determine indirect diffuse noise. If you've rendered AOV's you can check the indirect diffuse AOV; if noise is present in this AOV only, you can be quite certain this ray type is responsible. You can check if an area of noise is created by indirect diffuse noise by turning diffuse samples to zero; this will effectively turn off indirect diffuse. If this ray type is responsible, then the noise will disappear. If the image darkens with the indirect diffuse gone, but the noise is still present, indirect diffuse rays are not responsible for the noise.

In the example below a directional light is pointing into an enclosed space. With **diffuse samples** set to 0, no light can bounce off of the surfaces, and therefore there is no indirect light in the scene. Increasing the **diffuse samples** to 1 allows diffuse rays to bounce around the scene. However, it produces a noisy result, especially in the corners of the scene. Increasing the **diffuse samples** to 3 gives an improved result. It is good practice to use this value sparingly. Increase it incrementally and see if you notice any difference in the quality of the indirect diffuse component.



0 (only direct light with no GI diffuse bounces)



1 (noise due to low number of samples)



3 (incr



Remember that the diffuse sampling is done for each Camera (AA) sample, so high values for both Camera (AA) samples and diffuse samples will tend to result in slow renders.



Diffuse Surfaces Through Reflections

In scenes where you have both directly visible diffuse (bath) and diffuse visible through reflections/refractions (chrome reflection), increasing the diffuse samples will only help to improve the directly visible diffuse noise. Whereas increasing the Camera (AA) samples will improve everything uniformly (the entire image).



Camera (AA): 5. Diffuse: 1



Camera (AA): 5. Diffuse: 5. Noise still visible in reflection.

Diffuse samples only affect directly visible diffuse surfaces and nothing else

Specular

Controls the number of rays fired when computing the reflected indirect-radiance integrated over the hemisphere weighted by a specular BRDF. The exact number of rays is the square of this value. Increase this number to reduce the indirect specular noise (soft/blurry reflections). Remember that the specular sampling is done for each Camera (AA) sample, so high values for both Camera (AA) samples and specular samples will tend to result in slow renders.

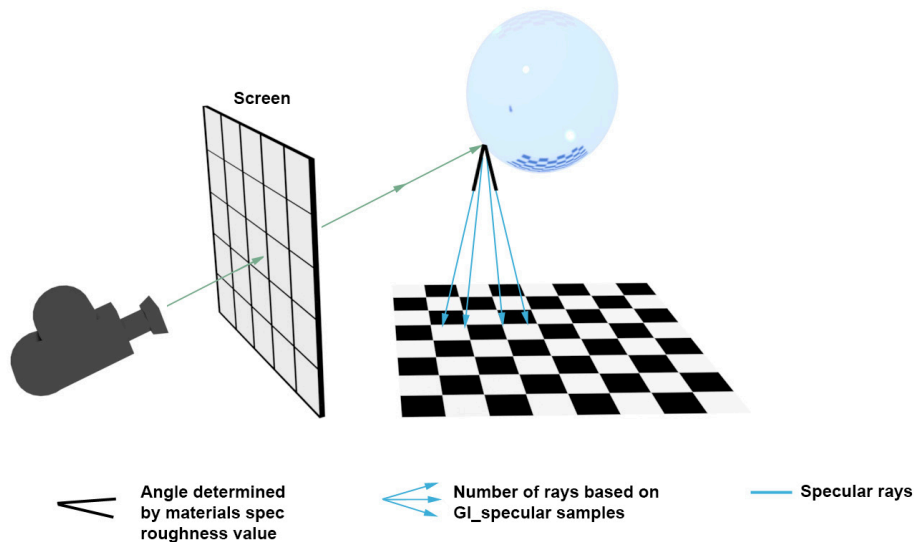
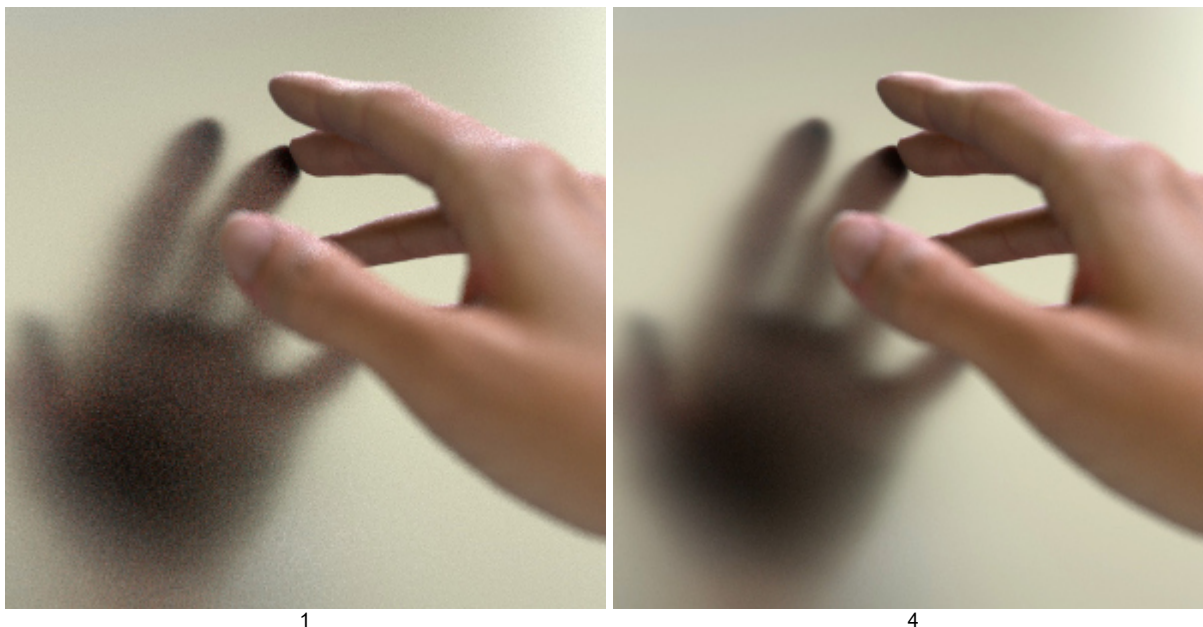


Diagram showing how specular reflection rays are propagated in an Arnold render

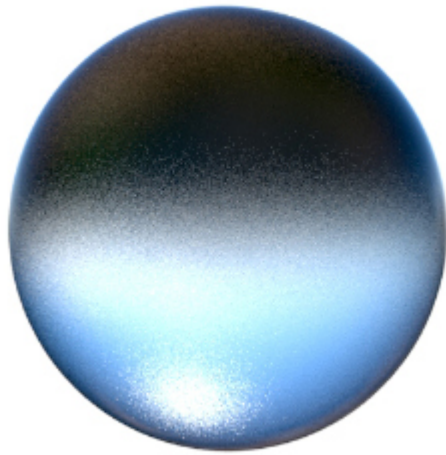
In the example below the mirrored surface has a high specular_weight and specular_roughness values. In the image on the left, you can see that there are not enough GI_specular_samples and therefore there is noise in the mirror. Increasing the GI_specular_samples gives a better result.



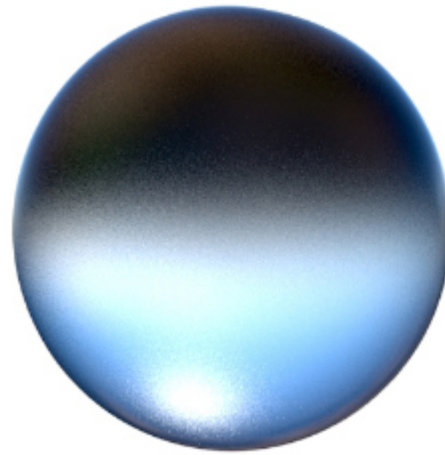
If you reduce the number of GI_specular_samples to zero and the specular_ray_depth to zero and the noise disappears, then the noise is due to specular reflections.

Transmission

Controls the number of samples used to simulate the microfacet-based transmission evaluations. Increase this value to resolve any noise in the transmission. If you switch this parameter to zero, the GI_transmission_depth to zero, and the noise disappears, you will know that the noise is due to transmission.



2



4

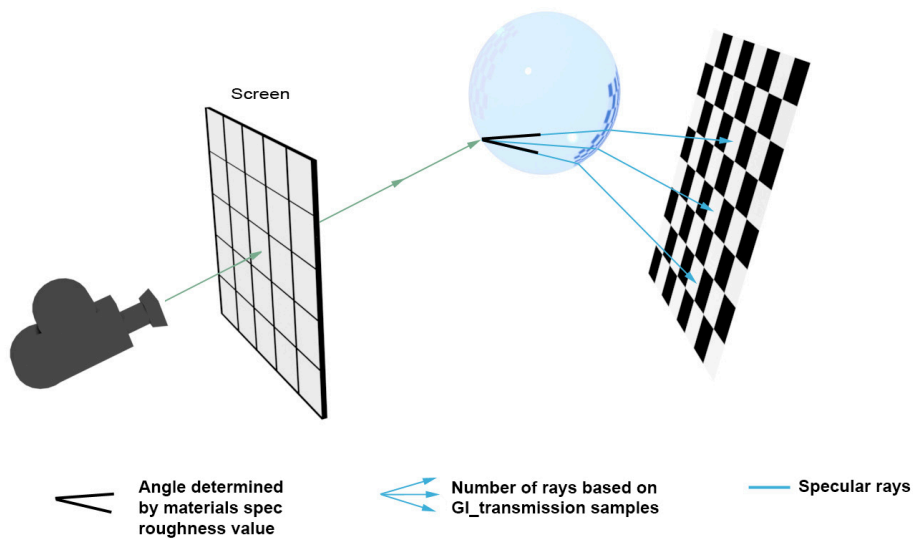
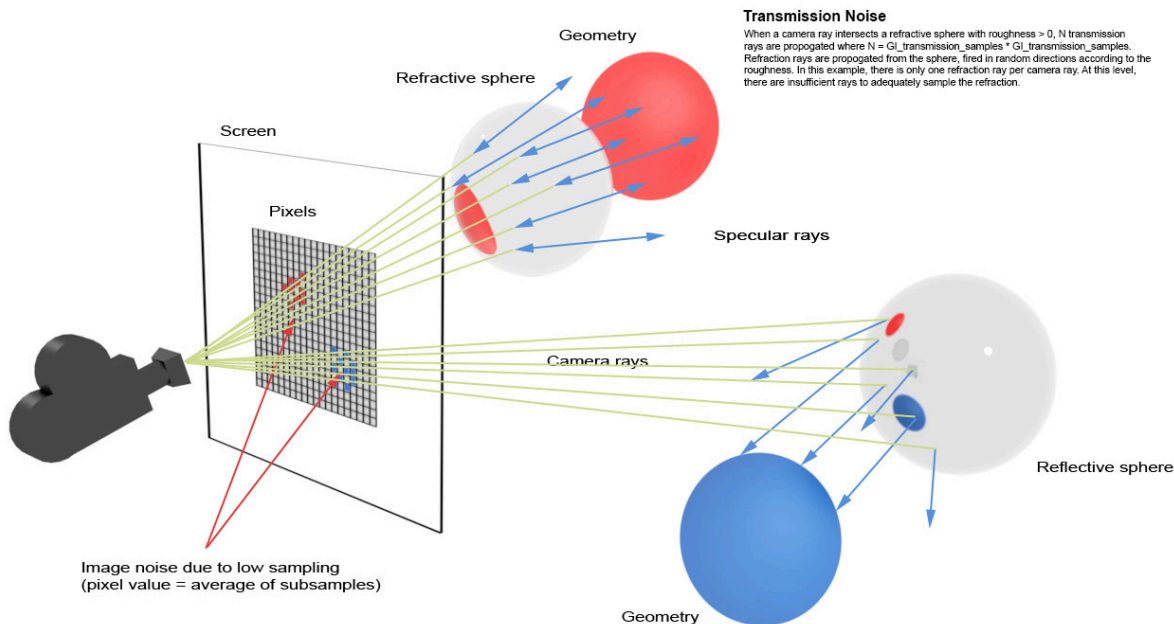


Diagram showing how transmission rays are propagated in an Arnold render

Specular Ray Sampling and Transmission Noise

Camera (AA) samples: 1, Specular samples: 1, Transmission samples: 1



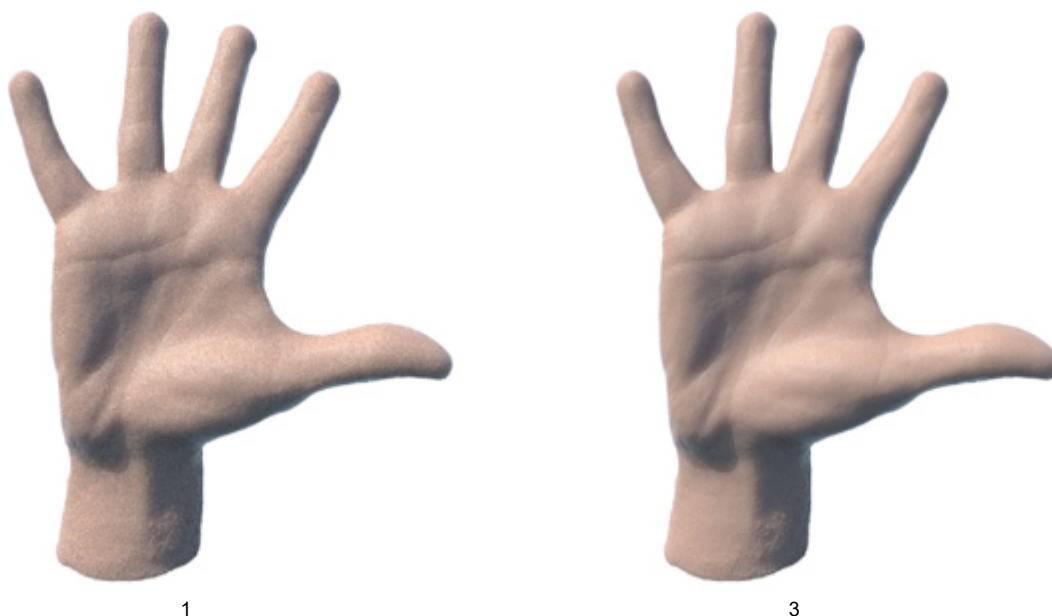
Specular Reflection Noise

When a camera ray intersects a reflective (specular) sphere with roughness > 0, N specular rays are propagated where $N = GI_specular_samples * GI_specular_samples$. Specular rays are propagated from the sphere, fired in random directions according to the roughness. In this example, there is only one specular ray per camera ray. At this level, there are insufficient rays to adequately sample the refraction.

While it is normally a straightforward matter to determine whether it is a specular reflection or transmission causing the noise, we need to confirm that specular rays are responsible for a given type of noise: If you've rendered AOV's you can check the indirect diffuse AOV; if noise is present in this AOV only, you can be quite certain this ray type is responsible. Also, you can switch the `GI_specular_samples` and `GI_transmission_samples` and `specular_ray_depth` types to zero in the Arnold render settings. Again, this essentially turns off specular rays. If the specular rays are responsible, the noise will disappear with this test. If the specular component disappears but the noise is still there, specular rays are not responsible.

SSS

This value controls the number of lighting samples (direct and indirect) that will be taken to estimate lighting within a radius of the point being shaded to compute sub-surface scattering. •Higher values produce a cleaner solution but will take longer to render.



Some additional noise in indirect specular and diffuse GI originating from an object with SSS is expected, in particular, if diffuse samples is set lower than the SSS samples setting. To combat this type of noise you can try using higher Camera (AA) settings and lower specular, diffuse, and SSS sampling rates or increase the diffuse/specular samples. Increasing the SSS samples will only make the subsurface effect have less noise in-camera, specular reflection, and transmission rays.

In the images below you can see that there is noise in the darker areas of the eye socket. Increasing the diffuse samples will reduce this type of noise.



2



5



Note that to have SSS values spread across multiple objects, for example from a face to an eyeball, you will need to use 'SSS Set Name'.

Volume

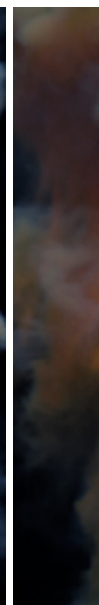
Controls the number of sample rays that get fired to compute indirect lighting of the volume. Like the other sampling rate controls (Camera (AA), light samples, diffuse samples, etc.), the number of actual samples is squared, so a setting of 3 fires $3 \times 3 = 9$ rays. Setting it to 0 turns off the indirect lighting of the volume. Note that indirect volume lighting is tied to the 'Volume' ray depth render option, and therefore there must be at least 1 volume bounce for indirect lighting to be computed.



0 (volume ray depth 1) - No indirect volume lighting

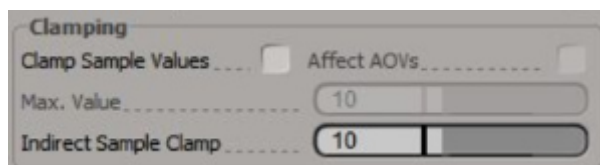


1 (volume ray depth 1) - Volume has indirect lighting but needs more samples



4 (volume ray depth 1)

Clamping



i Pixel clamping can be useful in scenes that contain 'spike noise' or 'fireflies': isolated, super bright pixels that jump around from frame to frame.

Clamp Sample Values

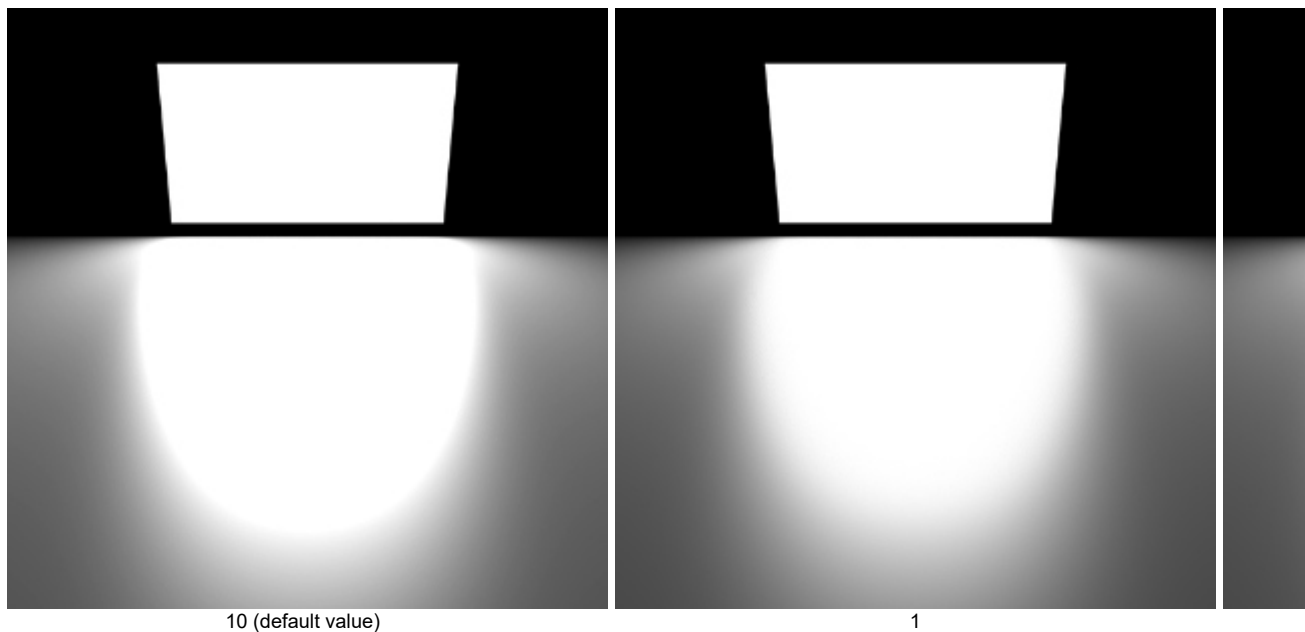
Enables/disables clamping.

Affect AOVs

With this control enabled the pixel samples of the AOVs will also be clamped. AOV clamping will affect every RGB and RGBA (except the A component) AOV. Currently, there is no way to tell Arnold which AOV's to clamp and which ones to preserve.

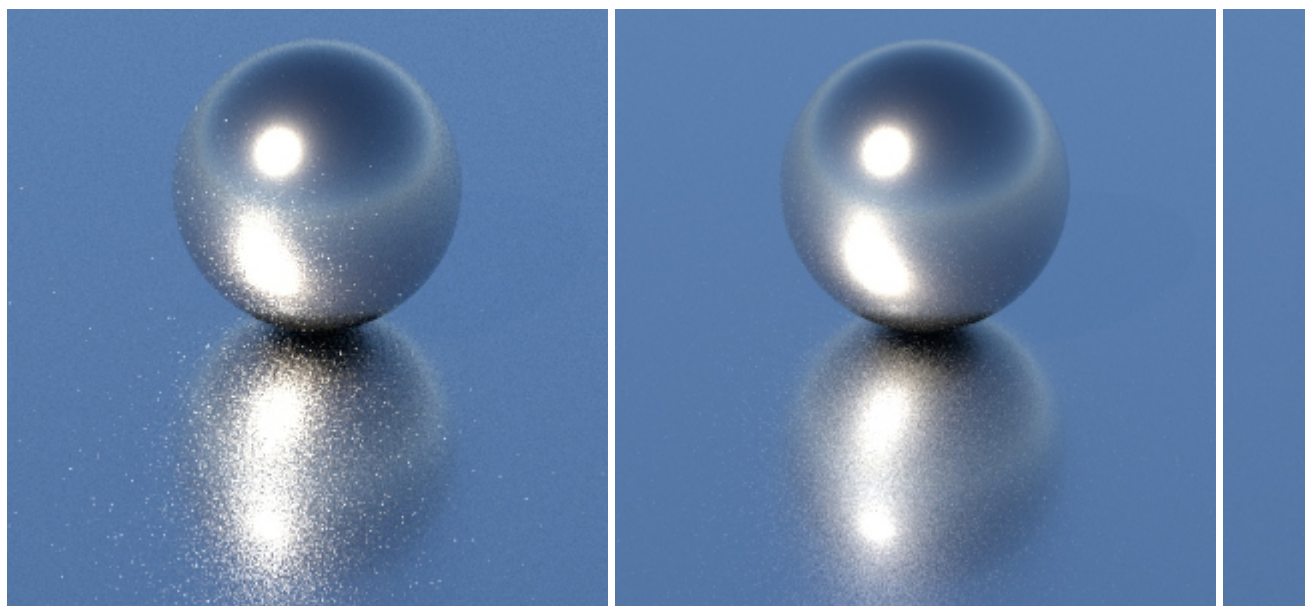
Max Value

If enabled, this control will clamp pixel samples to this specified maximum value. This can make it easier to anti-alias certain high-dynamic-range effects such as bright motion-blurred specular streaks (at the cost of reduced contrast). What is clamped is the resulting output value of the renderer computations of each pixel sample. Example: If the max_clamp_value is set to 2, no pixels will have an RGB value greater than 2.0, 2.0, 2.0



Indirect Sample Clamp

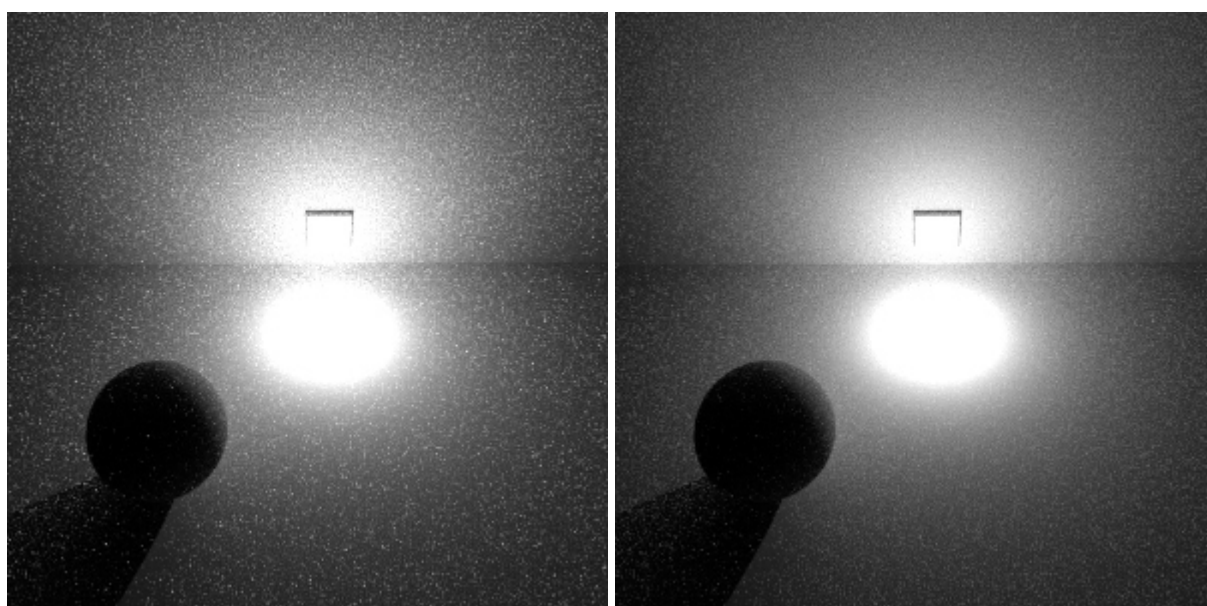
The threshold to clamp away fireflies from indirect light samples and reduce noise. This works similarly to AA_sample_clamp but preserves specular highlights from direct lighting. Lower values result in more aggressive noise reduction, possibly at the expense of dynamic range.



10 (default value)

4

This can also help with diffuse scenes when a light is very close to the ceiling/walls.



10 (default value)

1

Sample Filtering

Filter Type

The filter type used for averaging individual subpixel samples into a final pixel color. Most of these filters are available for legacy purposes only. The complete list of filters is:

- - Blackman-Harris
 - Box
 - Catrom
 - Gaussian
 - Mitnet
 - Triangle
 - Variance



•We recommend using the Gaussian (width 2.0) or Blackman-Harris filters.

Choosing a pixel filter is very much a personal choice. Some people prefer one over the other, but ultimately it's just slightly different amounts of blurring. While more blurring means less aliasing, it is also at the cost of detail loss.

It is best to avoid using a box filter (which is like disabling filtering) as well as using a filter width so small that you end up with practically no filtering.

It is important to note that all renderers don't implement filters in the same way, even if they have the same names. What Arnold calls a gaussian may not be the same as your Photoshop gaussian or 3Delight/PRMan/MentalRay. There can be slight differences in implementation, differences in the cutoff of filters whose support is infinite, etc. So don't assume that your filtering know-how can be transferred from one renderer to another verbatim. However, Arnold tries to replicate the standard filters from RenderMan.

The default gaussian-2.0 is slightly blurrier than others, but you could just reduce its width from 2.0 to 1.8-1.9 to get it sharper.

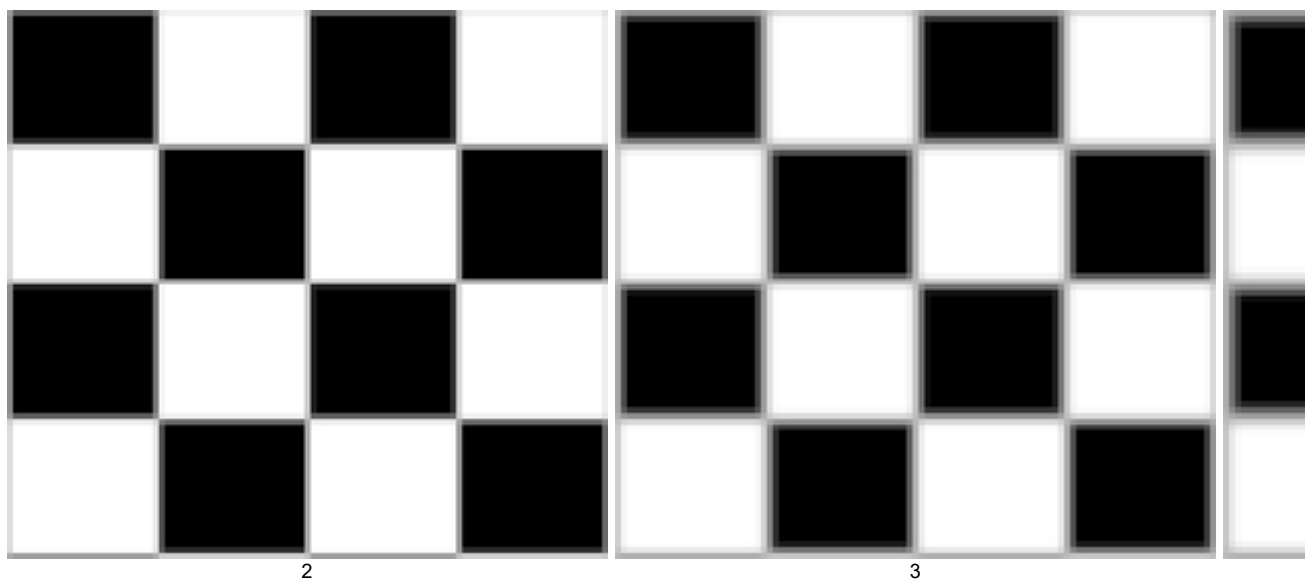
The catrom filter is one of the few filters that has negative weights and therefore may give you trouble when rendering to unclamped floating-point formats.

The choice of pixel filter should have zero render time impact. What could have a slight effect on render time is the width of the filter; too big a width and you have to consider many subpixel samples when filtering each pixel. For example, the catrom filter has a hardcoded width of 4 pixels compared to the 2 pixels of the gaussian filter which may slightly increase the render time.

If sharper renders are important, you should instead make sure that [options.texture_max_sharpen](#) is set to 1.5.

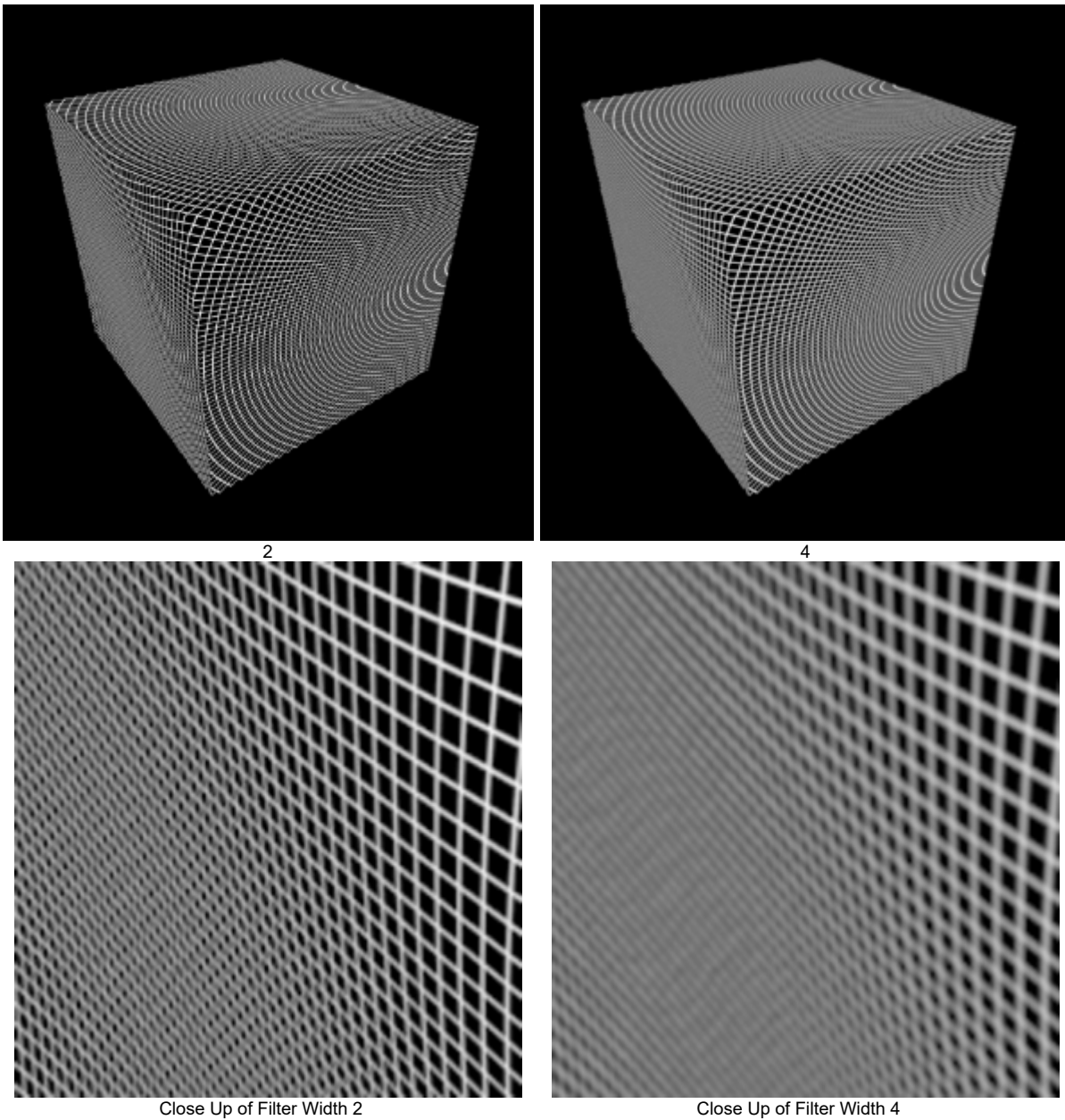
Width

The width of the subpixel sample averaging filter, in pixels. For circularly symmetric filters, this is the diameter of the filter's support. •



This example demonstrates the effect•filter width•can have on moire effects in your images. Increasing the filter width setting helps to reduce the moire effect.•

This is more noticeable on the left side and top of the cube.•



Filter Color AOVs, Filter Numeric AOVs

Lets you enable/disable filtering for all the color or all the numeric output render channels.

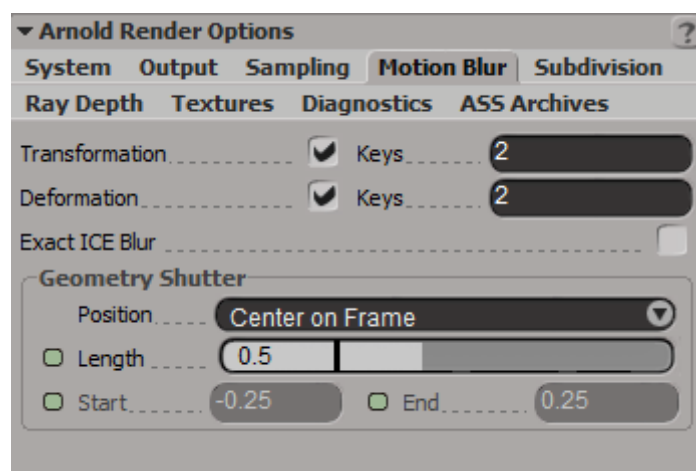
-
-

Motion Blur



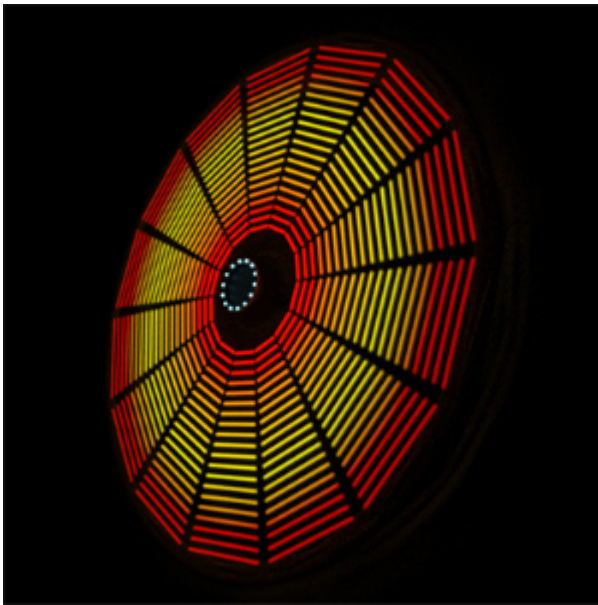
These settings control the amount, type and quality of motion blur. Arnold can apply motion blur to cameras, objects and lights. All together, these settings define the times at which the scene is evaluated to collect the points, normals, etc. that are then processed when tracing the scene. Extra camera-specific motion blur settings can be tuned in the motion blur tab of the [camera options](#) property.

 Increasing the number of [Camera \(AA\)](#) samples is the only way to decrease motion blur noise.

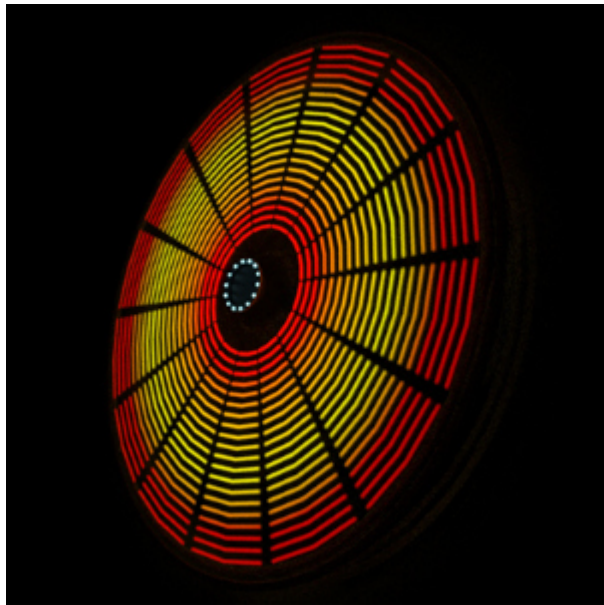


Transformation

The number of motion keys for matrix transformation. Transformation motion blur is calculated based on a linear interpolation of an object transform matrix, between successive motion keys. Increasing this value will add extra steps, which can improve the blurred result. The default is 2, which results in straight lines of blur between shutter start and shutter end. For situations where there is a direct movement, such as translation and rotation with constant speed, two steps are enough. However, if the object moves erratically during the exposure interval or performs any other kind of non-linear movement, you will need to increase the number of keys. Increasing the number of Transformation Keys does not usually have much effect on rendering times, however you may notice a small increase.

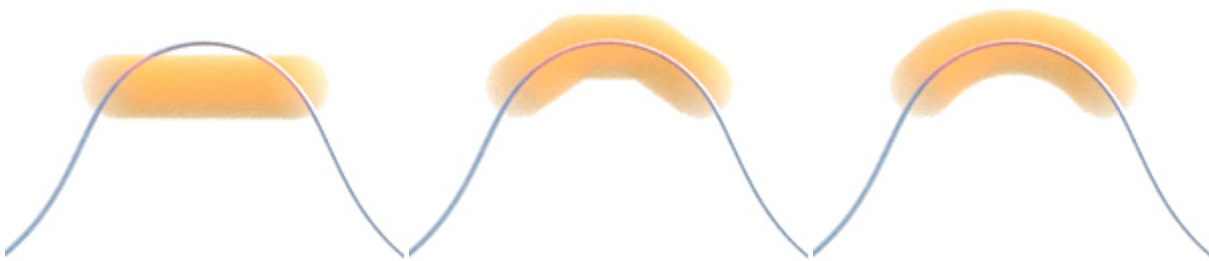


Keys: 2



Keys: 3

The effect is more apparent when animating a deforming object along a motion path:



Transformation Keys 2 (rollover image).

Transformation Keys 4 (rollover image).

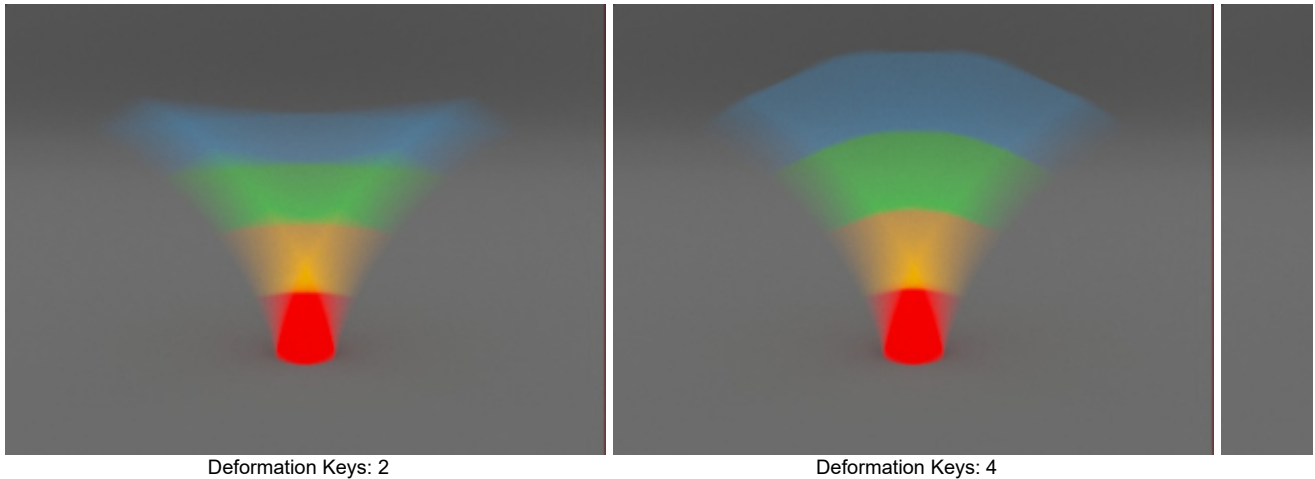
Transformation Keys 8 (rollover image).



Increasing the number of keys does not usually have much effect on rendering times, although it requires more memory to store the additional geometry at multiple times, especially for large polygon meshes.

Deformation

Here you can enable deformation motion blur, and set the number of Keys for it. Deformation blur takes into account geometry deformation, and should only be enabled when objects in the scene change shape quickly enough, since this option uses more memory and renders slower.



Deformation Keys: 2

Deformation Keys: 4

Exact ICE Blur

This option applies to pointclouds. By default, pointclouds are blurred according to the `PointVelocity` attribute, or to the `StrandVelocity` attribute (for strands). Such attributes define the velocity at the current frame, and are not sub-sampled by the ICE module. As a result, motion blur on pointclouds is pretty fast to compute, but linear.

For a more detailed result, with this flag you can tell the exporter to sample the pointcloud at the actual motion blur times, thus ignoring the velocity attributes. However, this requires caching the simulation to disk. •See [ICE Motion Blur](#) for details.

Geometry•Shutter

Define when and how the camera shutter opens and closes. The amount of blur is proportional to the time that the shutter remains open (i.e. shutter length or shutter end minus shutter start).

Position

Controls when the camera shutter opens and closes relative to the frame being rendered. Options are Start on Frame, Center on Frame, End of Frame, Custom .



Pool ball positioned in center

Start On Frame

Length

If `Position` is `Start on Frame`, `Center on Frame`, or `End of Frame`, defines the `length`, in frames, of the motion blur trails. Normally this won't be bigger than 1 frame unless you are looking for an exaggerated effect. •0 means no motion blur.

•



Example render using a long shutter length

Start and End

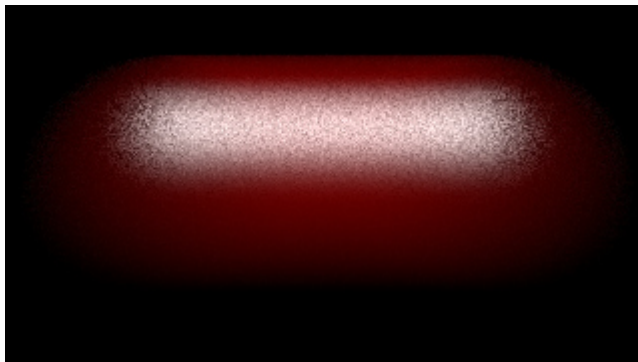
If Position is Custom, these are the start and end time, within the current frame, where the camera shutter opens and closes.



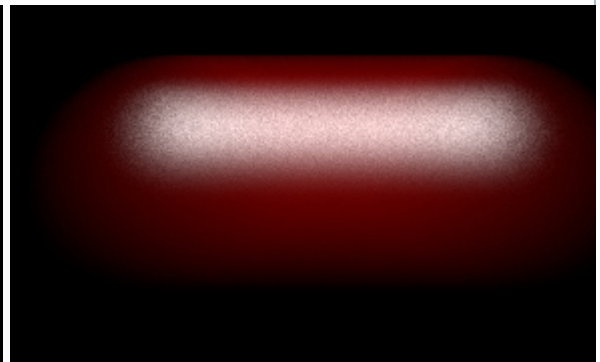
Custom Shutter Angles rendered at frame 0

i Motion Blur Noise

Values higher than 1 produce intense motion blur, but may also cause artifacts to appear in the rendered image. Increasing the number of AA Samples will reduce noise within the motion blur trail (see example below).



Camera (AA) samples: 3

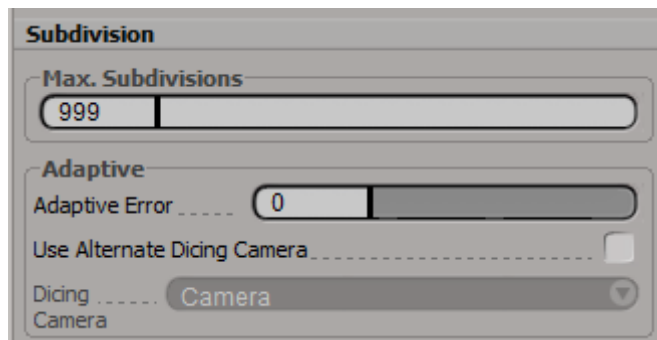


Camera (AA) samples: 6

 Global motion blur settings can be overridden locally by applying an [Arnold Parameters](#) property on a given set of objects.

Subdivision

These settings control the renderer's tessellation of subdivision surfaces for an object.



Settings

Max. Subdivisions

This value sets an upper limit on the number of subdivision iterations for all objects. By default this is set to a very high value (999) which in practice has no effect. Setting this to a low value such as 1 or 2 can be useful when debugging scenes that take a long time to subdivide/tessellate.

Adaptive

With adaptive subdivision, tessellation density across an object's surface varies depending on the distance of each surface patch to the camera as well as the image resolution (i.e., it is view-dependent). This will generally create far less polygons than subdividing to a fixed number of user-specified iterations.

Adaptive• Error

The maximum allowable difference for the chosen metric (`subdiv_adaptive_metric`) in the chosen space (`subdiv_adaptive_space`). The smaller the error, the closer to the limit surface a mesh will be and the less inter-frame "popping" when the subdivision level jumps, at the expense of using more polygons. A value of 0 disables adaptive subdivision, reverting to uniform subdivision, which can be more stable in animation. Examples:

- a value of 1.0 with `subdiv_adaptive_space` set to "raster" and `subdiv_adaptive_metric` set to "edge_length" will subdivide until all edges are shorter than 1 pixel.
- a value of 1.0 with `subdiv_adaptive_space` set to "object" and `subdiv_adaptive_metric` set to "flatness" will subdivide until the tessellation is closer than 1 unit in object space to the limit surface.

See examples•[here](#).

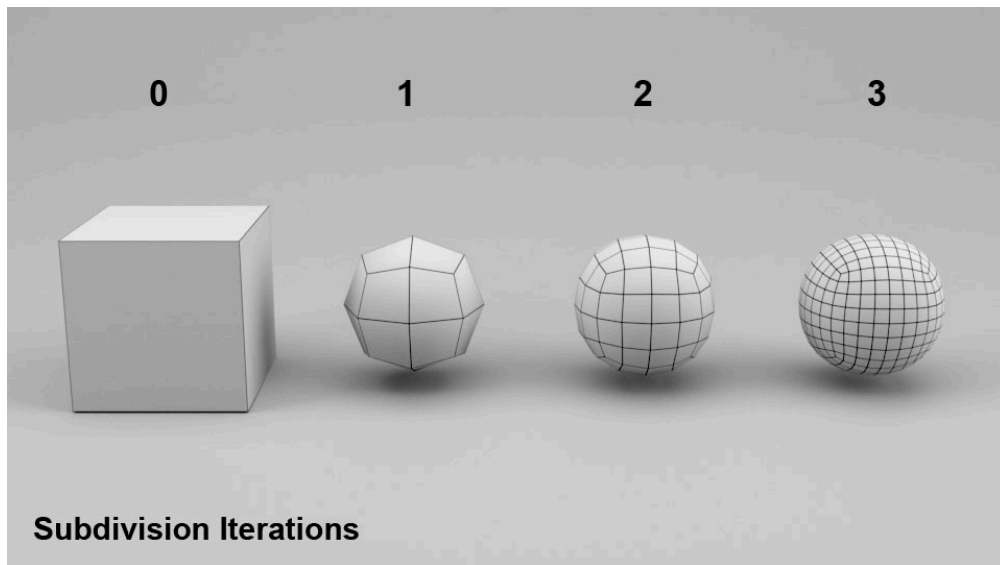
Use Alternate Dicing Camera

When enabled, the user provides a specific camera that will be used for all dicing (subdivision) calculations during Adaptive Subdivision (in other words, the tessellation of the object will not vary as the main camera is moved). This can be useful to fix objectionable flickering introduced by Adaptive Tessellation with certain camera moves. By default this is disabled, and should only be used when necessary, and with a carefully chosen position for the dicing camera.

Dicing Camera

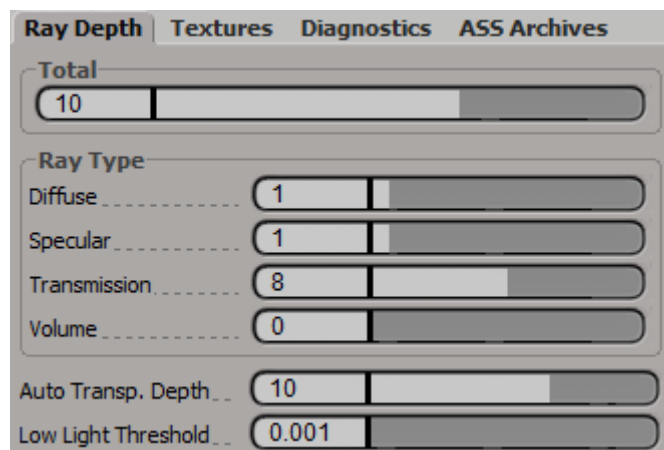
The camera that will be used for dicing calculations if Use Alternate Dicing Camera is enabled. If you set a static dicing camera you can still get the benefits from adaptive subdivision (higher polygon detail closer to camera) while getting a tessellation that does not change from frame to frame.

See examples•[here](#).



Cube subdivided with 0, 1, 2 and 3 iterations

Ray Depth



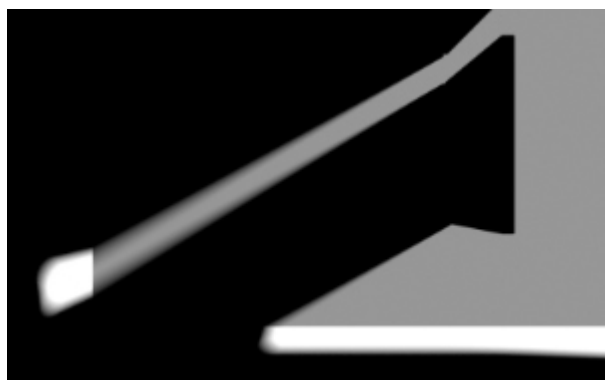
Ray Depth allows you to configure settings that limit the ray recursion based on ray type. Higher values will increase render time.

Total

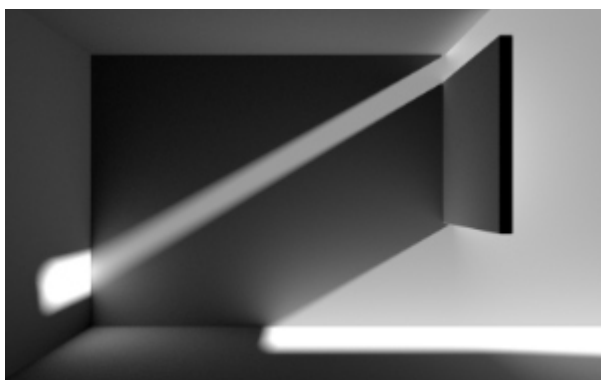
Specifies the total maximum recursion depth of any ray in the scene (diffuse + transmission + specular $\leq$ Total).

Diffuse

Defines the maximum ray diffuse depth bounces. Zero diffuse is equal to disabling diffuse illumination. Increasing the depth will add more bounced light to the scene, which can be especially noticeable in interiors. You may notice subtle differences when increasing the diffuse bounces incrementally. For example, the top of the cube has no light on it with the diffuse set to 1, whereas it is visible with the diffuse set to 2.



0. No bounced light.

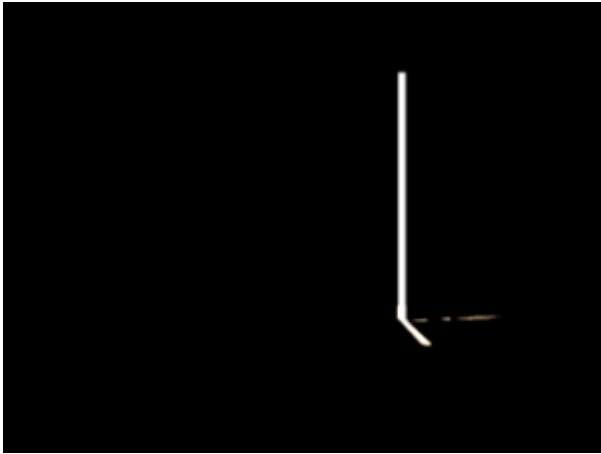


1. Light is bouncing round the scene but is still quite dark in areas.



2. More light bouncing off the floor behind the door.

Below is an example showing the dramatic difference that diffuse ray depth can have in an interior scene. Note the effect of the bounced light on the floor behind the door when rendering with ray depth 2 compared to ray depth 1 and 0.



0 (direct lighting only)



1 (default)

We can see a difference in the appearance (and render time) when increasing the ray depth in the production scenes below:•



Diffuse ray depth : 1 (default). Roll over image to view 3.

The current Arnold code will only take one random sample after a split has happened in the ray tree, causing the regions behind transmission /specular reflections and diffuse to have much fewer shadow samples than those that are viewed directly by the camera. Splitting occurs after the first bounce.

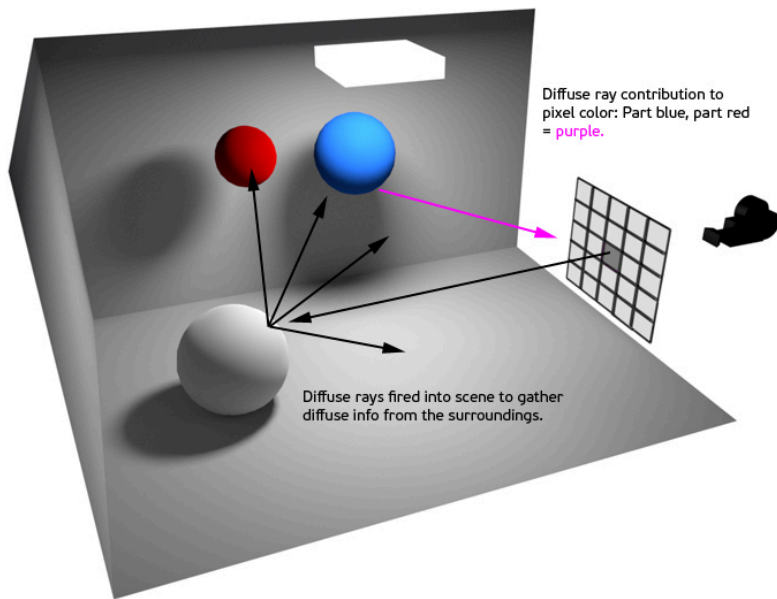
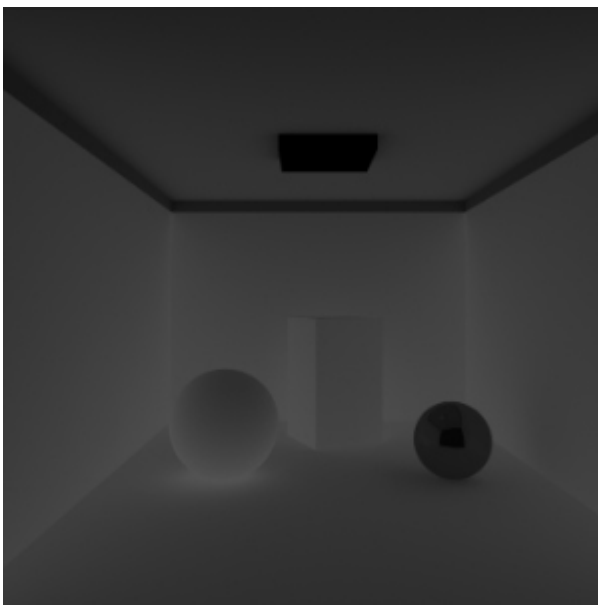


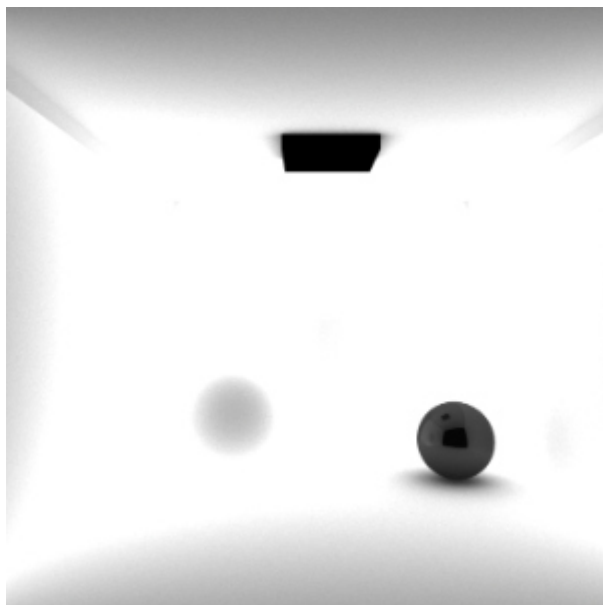
Diagram showing how diffuse rays are propagated in an Arnold render.

You can see the effect that increasing the diffuse ray depth has on the number of rays per pixel and how that impacts render time•by checking the render log. •

Rendering using the 'raycount' AOV shows•the total number of rays that are created/traced to compute a certain pixel (diffuse,•specular•and shadow rays are shown). •



Ray Depth 1 - 1206 rays/pixel - 5 min 29 secs

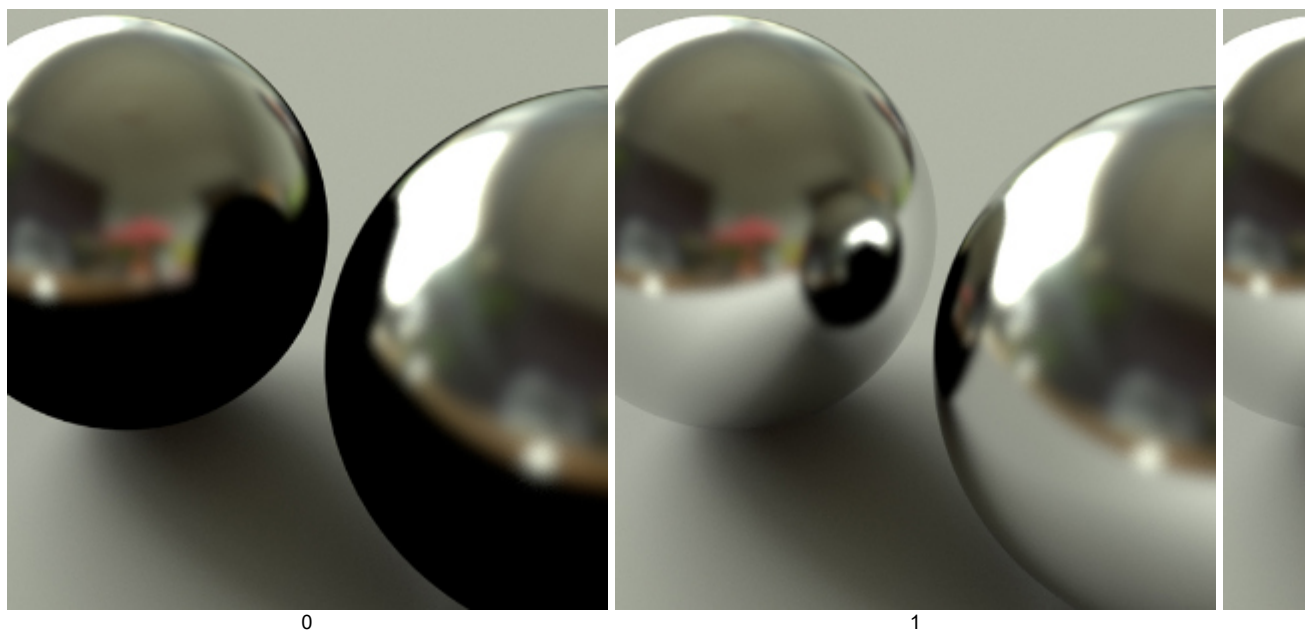


Ray Depth 4 - 2897 rays/pixel - 15 mins 2 secs

Images rendered using the 'builtin' raycount AOV

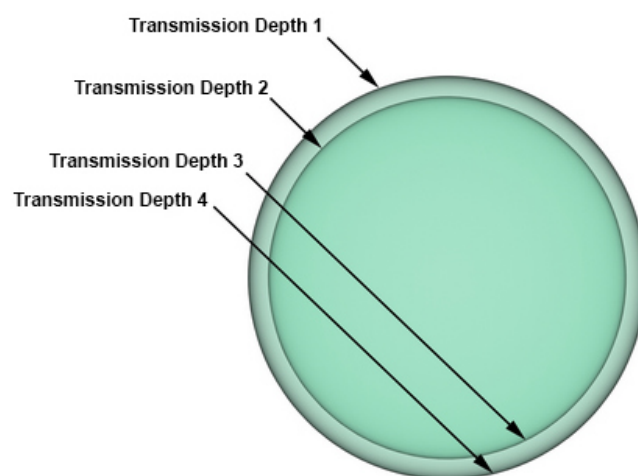
Specular

Defines the maximum number of times a ray can be specularly reflected. •Scenes with many specular surfaces may require higher values to look correct. A minimum value of 1 is necessary to get any specular reflections. •



Transmission

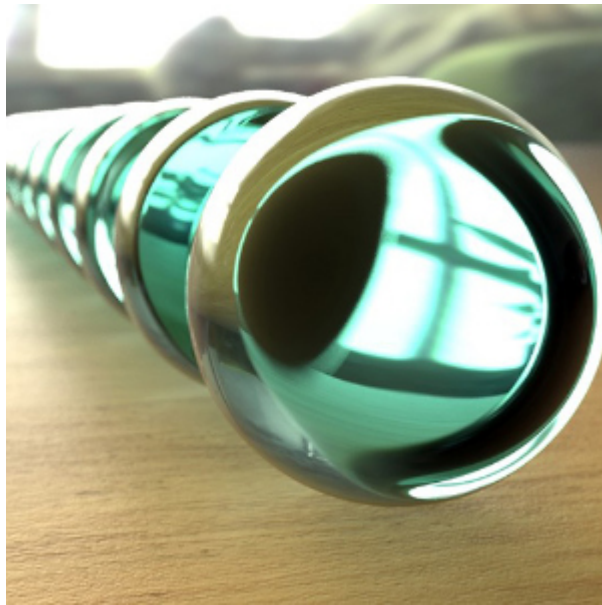
The maximum number of times a ray can be refracted. Scenes with many refractive surfaces may require higher values to look correct. •



Glass surface with double-sided thickness.



2



8

Volume

This parameter sets the number of multiple scattering bounces within a volume (defaults to 0). This is useful when rendering volumes such as clouds for which multiple scattering has a large influence on their appearance.



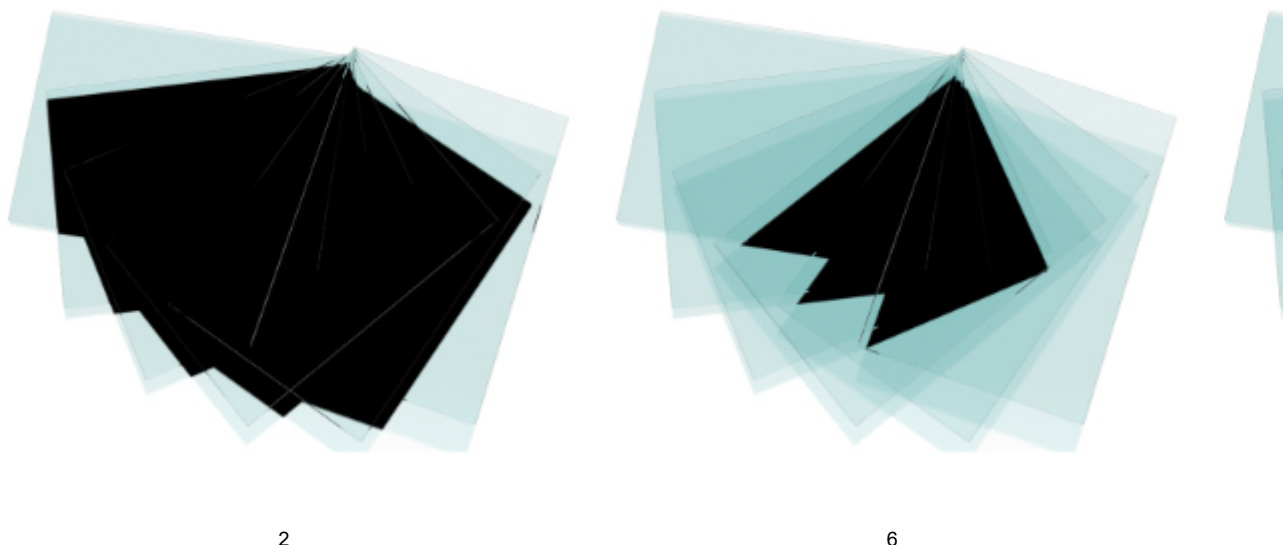
0



5

Transparency Depth

The number of allowed transparency hits. With 0 objects will be treated as opaque. The example below consists of six glass cubes that are sitting on top of each other. Arnold returns black when there is an insufficient number of rays due to the limit imposed by the transparency_depth. Increasing this value allows more rays to pass through transparent surfaces. In this case, a transparency_depth of 12 is enough to get a good result.



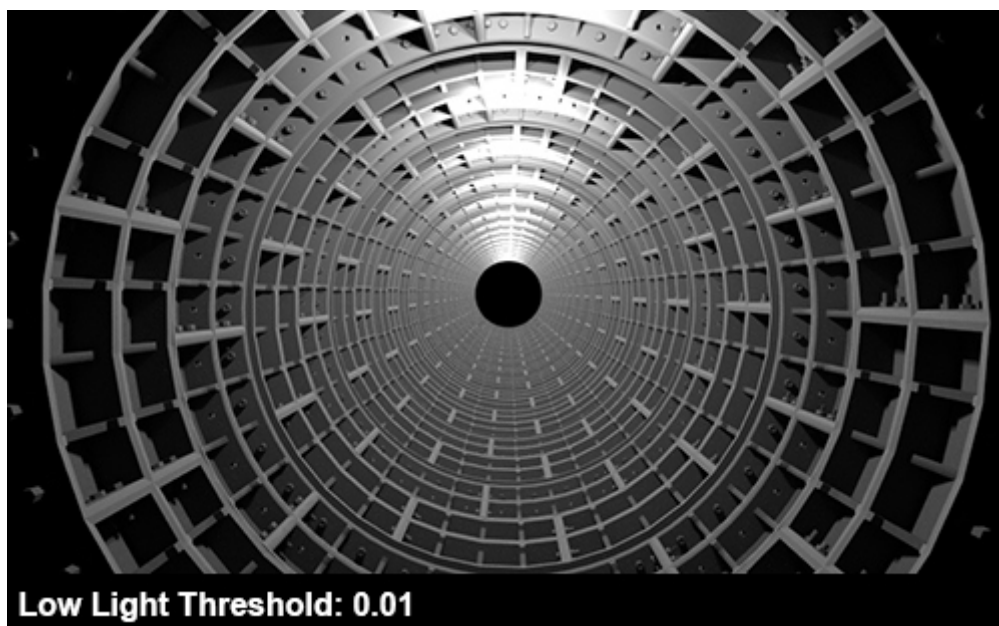
Low Light Threshold

Raising this value can speed up rendering by allowing Arnold to ignore tracing a shadow ray for light samples whose light contribution is below a certain value. The point of `low_light_threshold` is to save casting shadow rays when Arnold knows that the error from not casting that ray is below a certain amount. This makes sense because below a certain threshold there will be no perceptible difference between shadowed and unshadowed areas.

Roughly, what is taken into account to discard a shadow ray is its maximum possible incoming irradiance, which is linearly related to the light's intensity and decreases with distance. The specific number will vary depending on your exposure and light intensity values, but the default value is reasonable for the default exposure/light intensity values. This will typically help more in scenes with lots of lights with a finite area of influence, like a cityscape for example. It will help much less with a single strong source of light.

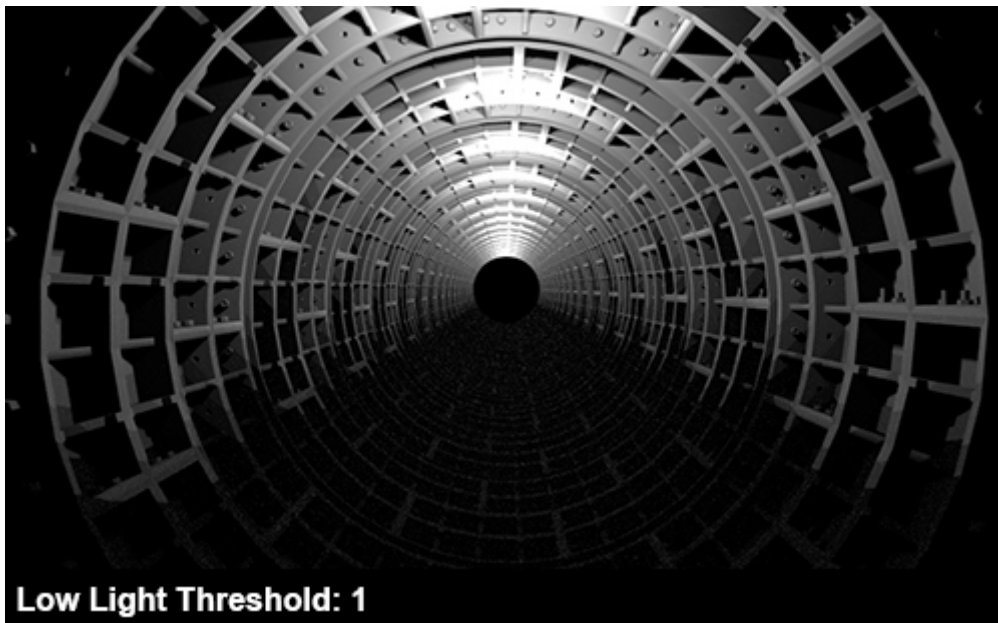
You should never be able to see the boundary when this optimization kicks in. If you can see it, you need to make the threshold lower. The aim is to prune shadow rays that don't contribute to the final image

In the example below, increasing the `low_light_threshold` to 0.1 shows a good speedup in render time (in this case, more than twice as fast) with little noticeable difference in the image.

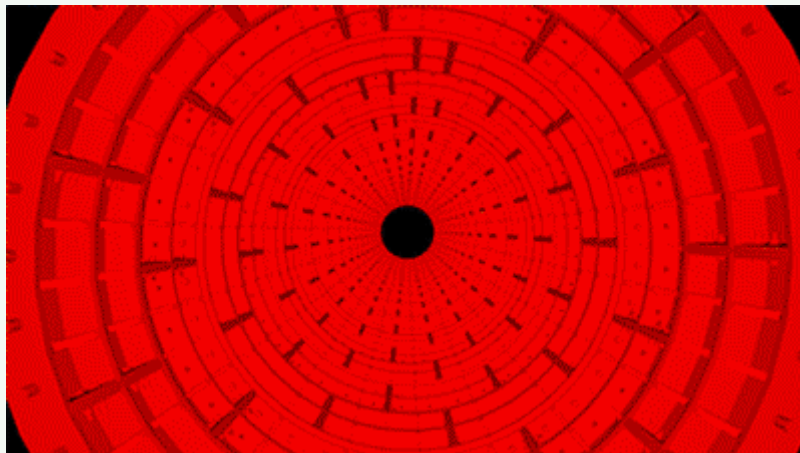


A slightly higher threshold can offer significant rendering speed increases without noticeably darkening the image (r
ollover image).

•
However, increasing the `low_light_threshold` value too much can have an adverse effect on the lighting:



The Utility shader's `nlights` color mode can be used to see how many lights are influencing a shading point when adjusting the `low_light_threshold`.

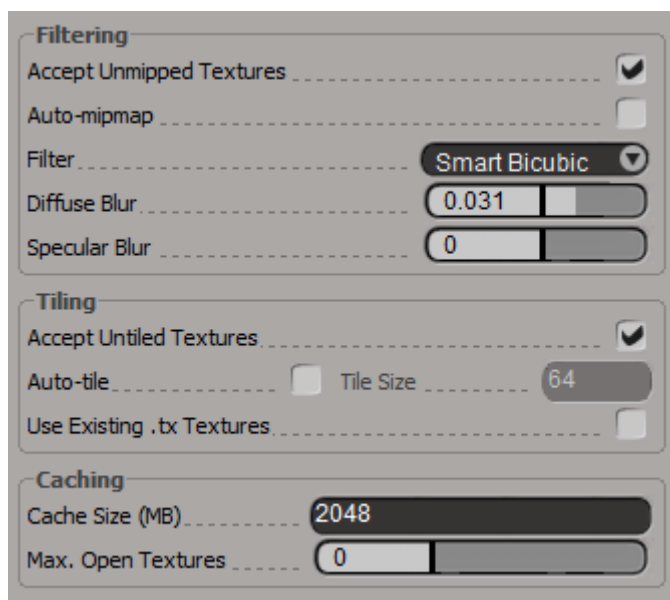


`low_light_threshold` : 0 to 1. Red = all lights, blue = low values, black = no lights.

Textures - settings

These settings control how texture files are handled by Arnold.

•



Filtering

•

Accept Unmipped Textures

High-resolution unmipped texture maps are very inefficient to render because the highest resolution level must be loaded into memory regardless of the distance rather than a lower resolution level. When this flag is disabled, any attempt at loading an unmipped file will produce an error and abort the renderer.

Auto-mipmap

If a texture map file is not natively mipmapped (e.g. a JPEG file), enabling this option will trigger the generation of a mipmap on demand. The output will be stored in memory and put in the global texture cache. This mipmap generation increases render time, especially for scenes with many high-resolution textures. To avoid this performance slowdown, it is recommended to use texture file formats that natively support mipmapping (such as TIFF and EXR), where the mipmaps are preprocessed before the render and stored in the file. This preprocessing can be done with the [maketx](#) utility.



Disabling 'options.texture_automip' will result in unfiltered non-mipmapped textures.

Filter

The type of texel interpolation used for texture lookups. Bicubic produces the highest quality, at a slight increase in render time.

Diffuse Blur

MultiExcerpt named 'texture_diffuse_blur' was not found

The page: options was found, but the multiexcerpt named 'texture_diffuse_blur' was not found. Please check/update the page name used in the 'multiexcerpt-include macro.

Specular Blur ••

MultiExcerpt named 'texture_specular_blur' was not found

The page: options was found, but the multiexcerpt named 'texture_specular_blur ' was not found. Please check/update the page name used in the 'multiexcerpt-include macro.

Tiling

Accept Untiled Textures

High-resolution untiled texture maps are very inefficient to render because they must be loaded entirely into memory as soon as the renderer needs to access the first texel. When this flag is disabled, any attempt at loading an untiled file will produce an error and abort the renderer.

Auto-tile

If a texture map file is stored in scanline mode (e.g. a JPEG file), enabling this option will trigger the generation of tiles on demand. The output will be stored in memory and put in the global texture cache. This process increases render time, especially for scenes with many high-resolution textures. To avoid this performance slowdown, it is recommended to use texture file formats that natively support tile mode (such as TIFF and EXR). You can create tile textures with the [maketx](#) utility.

Tile Size

This is the size of the tiles when using auto-tiling. Bigger values mean less frequent texture loads, but use more memory.

Use Existing .tx Textures

If an untiled texture is requested (e.g. concrete.jpg) but a .tx file with the same filename is found (e.g. concrete.tx), the renderer will load the .tx file instead.

Caching

Cache Size (MB)

The maximum amount of memory to be used for texture caching. Arnold uses a tile-based cache with an LRU (Least Recently Used) type algorithm, where the least recently used tiles are discarded when the texture cache is full.

Max. Open Textures

The maximum number of files that the texture system keeps open at any given time to avoid excessive closing and re-opening of files when caching individual texture tiles. Increasing this number may result in slightly better texture caching performance. On the other hand, if this value is higher than the maximum number of concurrently open files supported by the operating system (e.g. Linux vs Windows), some texture lookups may fail and return red. This is set by default to 0, which means that the maximum number of texture files that can be simultaneously opened is automatically computed by Arnold using a heuristic that tries to guess what the optimal number would be for each particular machine. We expect that the majority of users will be able to leave this at 0 and get the best performance.

•

Diagnostics

Arnold can generate a log reporting all the steps needed to render an image, as well as any warnings and errors encountered by the renderer. This log also provides detailed statistics that are useful for debugging, optimizing and benchmarking renders. The log can be sent to the Softimage console or to a file.

Logs

Console ☒ File ☐

Verbosity ... **Warnings**

Detailed Texture Statistics ☐

Max. Warning Messages ... **5**

Output Path

[Project Path]\Arnold_Logs ...

Resolved Path

C:\Program Files\Autodesk\Softimage 2015\Data\XSI_

Ignore

Texture Maps ☐

Shaders ☐

Atmosphere Shaders ☐

Lights ☐

Shadows ☐

Subdivision ☐

Displacement ☐

Bump ☐

Normal Smoothing ☐

Motion Blur ☐

Depth of Field ☐

Sub-Surface Scattering ☐

Hair ☐

ICE Point Clouds ☐

Procedurals ☐

User Options ☐

Matte Properties ☐

Logs

Console

Specify whether logs are to be written to the console.

File

Specify whether logs are to be written to file (the location of the file is specified in the Output Path setting below).

Verbosity

Specifies the amount of information in the log messages (Warnings being least and Debug most) and whether log messages are sent to the console or written to a file.

- Errors (default): Show errors only.
- Warning + Info: Show warnings, statistics and any info that does not slow down rendering.
- Debug: Shows all information. Take care when using this mode as render times may increase.

Detailed Texture Statistics

Enable very useful per texture statistics, with time spent on texture queries, amount of texture data read, memory consumption, etc

Max. Warning Messages

Limits the number of warning messages that are sent to the console.

Output Path

The output path for log files (if logging to file is enabled above). You can use environment variables or Softimage tags like [Project Path].

Resolved Path

Shows the resolved path where log files will be saved.

•

Ignore

This group of switches allows you to disable a number of important rendering features. By selectively disabling some features, you can get a feel for where the renderer is spending most of the time, which helps optimizing scenes. This is also useful for isolating errors and artifacts when debugging. Note that not all of these options are interactive - some of them may require the entire scene to be refreshed/exported.

Textures

Ignore all textures when rendering.



Disabled



Enabled

Shaders

Ignore all shaders when rendering, Arnold will render with a simple n_dot_eye shader instead.



Disabled

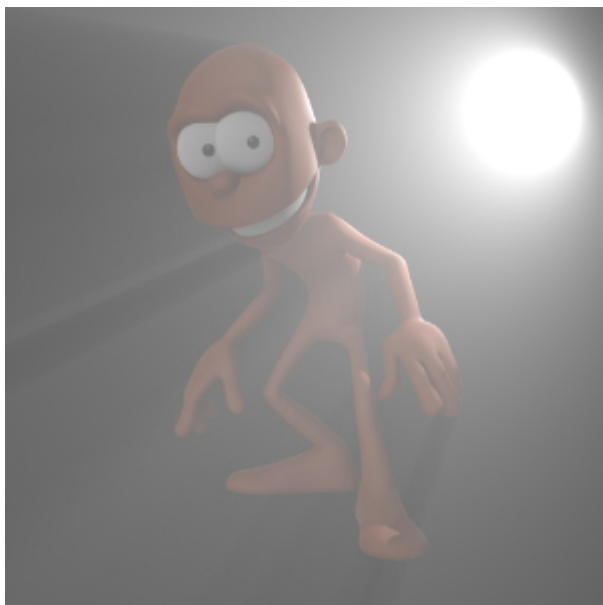


Enabled

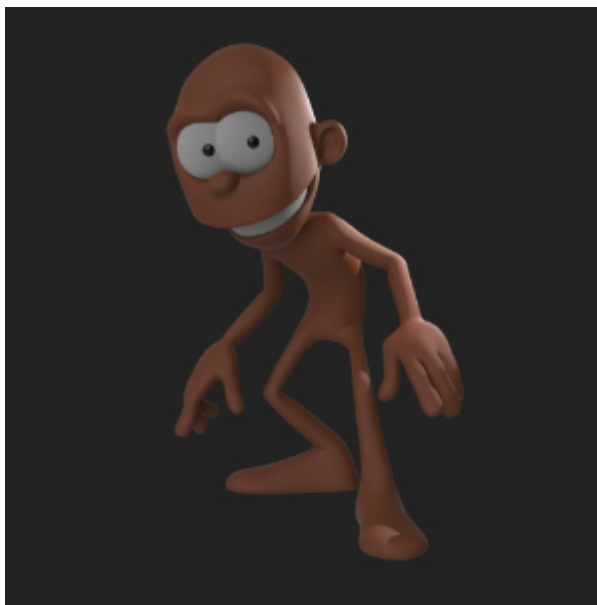
Atmosphere

•

Ignore atmospheric shaders.



Disabled



Enabled

Lights

Ignore all light sources.



Disabled



Enabled

Shadows

Don't perform shadow calculations.



Disabled



Enabled

Subdivision

No objects will be subdivided. This parameter is not interactive.



Disabled



Enabled

Displacement

Displacements will be ignored. This parameter is not interactive.



Disabled



Enabled

Motion Blur

Ignores all motion keys.

Sub-Surface Scattering

Ignore subsurface scattering.



Disabled



Enabled

Hair

Ignore hair objects.

ICE Point Clouds

Ignore ICE objects.

Procedurals

Ignore stand-ins.

User Options ••

Ignore setting user options (if there are any applied on objects, camera, lights, or globally to the rendering options).

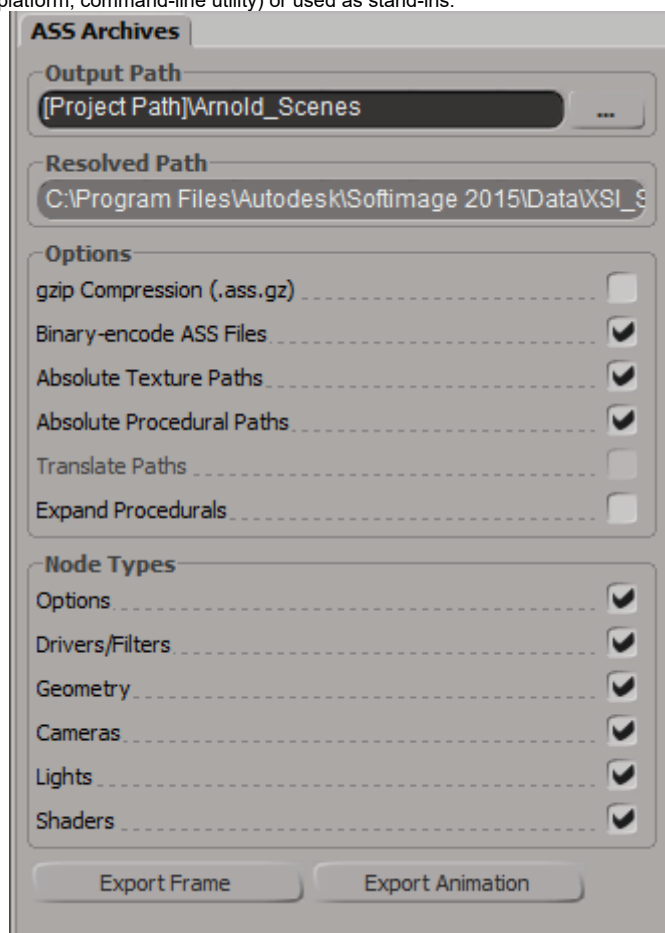
Matte Properties

Ignore all the Arnold matte properties in the scene. •

•

ASS Archives

Arnold's native scene definition format is stored in files with the .ass extension. SltoA lets you save scenes to .ass files that can later be rendered outside Softimage with kick (a multi-platform, command-line utility) or used as stand-ins.



Output Path

The output path for the exported .files. You can use environment variables or Softimage tokens like [Project Path].

Resolved Path

Shows the resolved output path, after variable/token expansion.

gzip Compression (.ass.gz)•

Exports the scene directly to gzip-compressed files with the .ass.gz extension. You can expect compression rates in the 4-5x range.

Binary-encoded ASS Files

Specify whether binary encoding is used to compress large data arrays into a more compact ASCII representation, leading to smaller files and faster load times, while still being mostly human-readable. In addition, the binary encoding has exact 32-bit precision, whereas without this binary output floating point values are truncated into at most 8 ASCII digits (e.g. 1234.5678). The encoded arrays are indicated by prefixing the array type with "b64" as in the example below:

[illegible]

Absolute Texture Paths

Enable/disable use of absolute texture paths.

Absolute Procedural Paths

Enable/disable use of absolute procedural paths.

Translate Paths

Perform appropriate file path translation when exporting to .ass files intended to be rendered on a different operating system. (see Cross-platform Path Translation in the [SltoA Paths](#) topic for more information).

Expand Procedurals

If there are stand-ins, their content is expanded in the written file.

Node Types•••

Allows you to filter the export nodes by type: •Options, •Drivers/Filters , Geometry , Cameras , Lights , Shaders

Buttons

Export Frame

Exports to file the current timeline frame, according to the previous settings.

Export Animation

Exports to file the whole animation (one file per frame), according to the previous settings.



When exporting a scene to .ass, SltoA will also add 2 extra fields as part of the options node: the current frame and the fps rate, both as constant floating point user data.

If you edit an exported .ass, they will show as•

```
options
{
  name options
  ...
  declare frame constant FLOAT
  frame 1
  declare fps constant FLOAT
  fps 29.9700298
}
```

-
-



Shapes

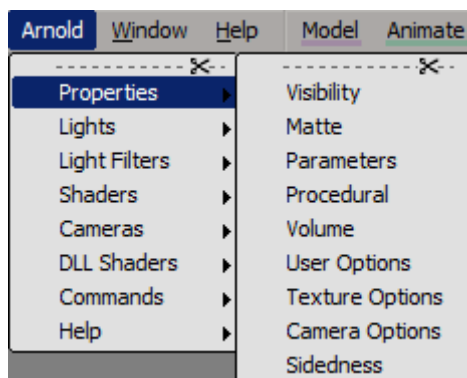
There are Arnold settings attached to each object. The following pages describe these specific Arnold settings:

-

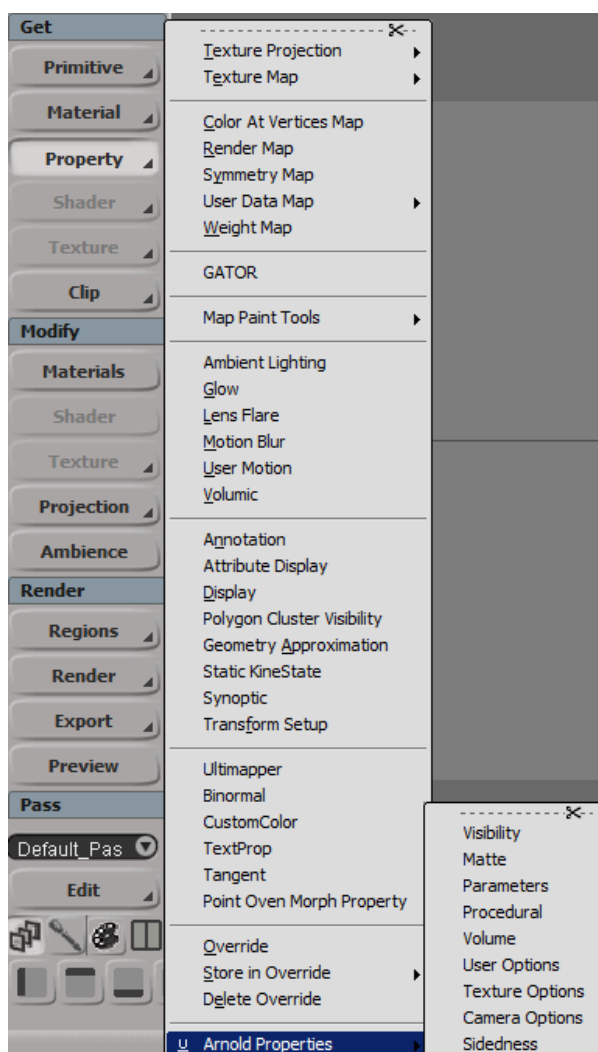
- [Arnold Properties](#)
- [ICE](#)
- [Volumes](#)

Arnold Properties

The Arnold renderer has several additional parameters which control specific features that don't have a direct equivalent in Softimage. These parameters are grouped under some properties, that can be applied from the Arnold->Properties top menu.



or from Get->Property > Arnold Properties. Select the objects you want to apply the property to, then the property itself.



•Geometry :

- Arnold Visibility
- Arnold Matte



- [Arnold Parameters](#)•(common to all geometries)•
- [Arnold Parameters](#)•(hair and ICE strands only)
- [Arnold•Procedurals](#)
- [Arnold Sidedness](#)•

Geometry, cameras, and light s:

- [Arnold User Options](#)

Texture s:

- [Arnold Texture Options](#)

Cameras

- [Arnold Camera Options](#)

Volume

- [Arnold Volume](#)

•

Arnold Parameters

With this property, you can set several attributes of the exported Arnold shape, such as SSS, motion blur, and displacement.

To apply it, select one or several objects and run either:

- Get->Property->Arnold Properties->Parameters
- Arnold->Properties->Parameters
- [SITOA_AddParametersProperties](#) (script editor)

Arnold_Parameters

Shape

- ☐ Opaque
- ☐ Invert Normals
- ☐ Self Shadows
- ☐ Receive Shadows
- Export Bind Pose (Pref)
- Smooth Subdiv Tangents

SSS Set Name

Trace Sets

Motion Blur

- Override Keys
- Transformation ☒ Keys 2
- Deformation ☒ Keys 2
- Use ICE PointVelocity

Displacement

- ☐ Height 1
- ☐ Bounds Padding 0
- ☐ Zero Value 0
- ☐ AutoBump

Subdivision

- ☐ Additional Iterations 0
- ☐ Adaptive Subdivision
- ☐ Adaptive Error 2
- Adaptive Metric Auto
- Adaptive Space Raster

Volume

- ☐ Step Size 0.1
- ☐ Padding 0

Shape

Opaque

By default, objects are opaque, which lets the renderer take certain shortcuts and optimizations for maximum ray tracing speed. When this option is unchecked, the object is flagged as "possibly transparent", and Arnold will perform extra computations to support transparency and transparent shadows, according to the shader's opacity settings.

Invert Normals

Inverts the normals of the mesh.

Self Shadows

Determines whether or not the object casts shadows on itself.



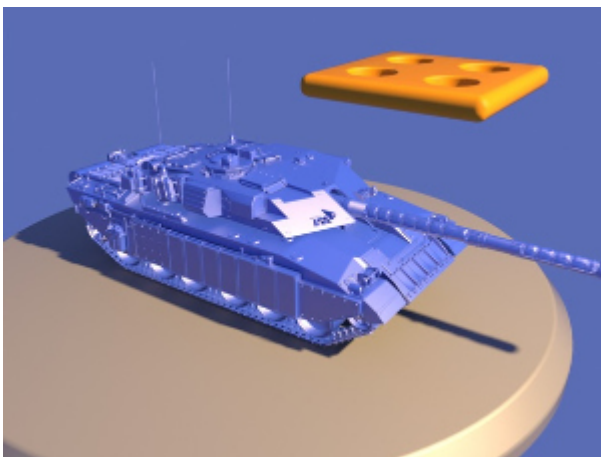
Disabled



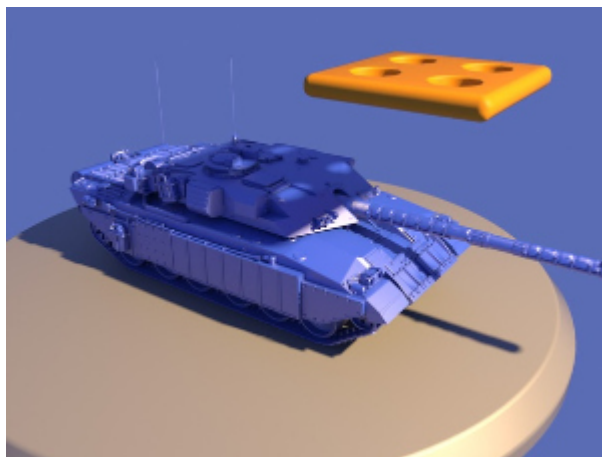
Enabled

Receive Shadows

Determines whether or not the object picks up shadows from other objects.



Tank with Receive Shadows disabled



Tank with Receive Shadows enabled

Export Bind Pose (Pref)•

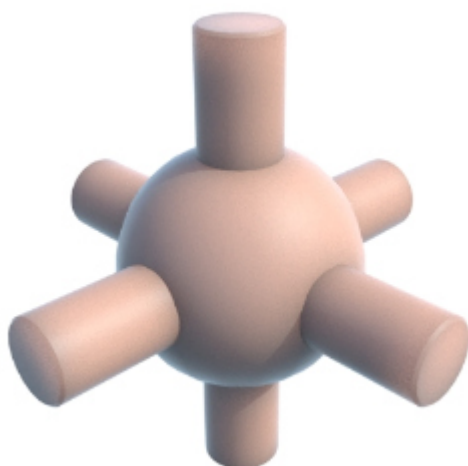
This option can be used to make procedurals stick and avoid SSS flickering. Remember that deformations need to be above the modeling stack from where the bind pose is exported.

Smooth Subdiv Tangents

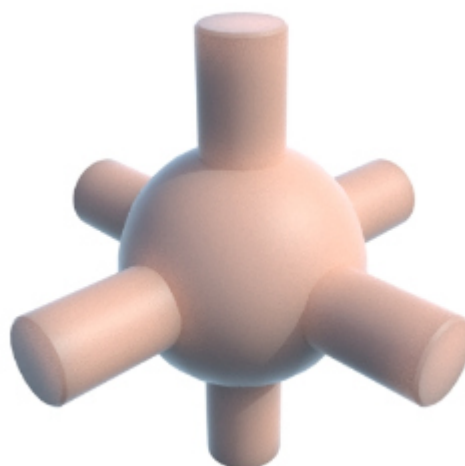
This option makes it possible to remove the faceted look from anisotropic speculars. The object must have a valid UV map for this option to work.

SSS Set Name

It is possible to tag multiple objects as belonging to the same SSS 'set' so that illumination will blur across object boundaries. A common use case might be blurring between teeth and gum geometry. It is enabled by adding the constant `•STRING•userdata•sss_setname•` to the same value on the objects in the set.



No Set Name



Objects In Same Set

Trace Sets

The strings used to tag the current object, in order to set the inclusiveness or exclusiveness when tracing this object. This option works in conjunction with shaders supporting trace sets, like [trace_set](#).

Motion Blur

These parameters allow you to override the number of motion blur keys per object, or to disable motion blur.

Override Keys

If on, the global motion blur settings are over-ridden by the following:

Transformation Keys

Number of keys for the transformation motion blur

Deformation Keys

Number of keys for the deformation motion blur

Transformation, Deformation

If the Transformation and/or Deformation motion blur is enabled in the rendering options, you can disable it for this object by unticking these flags.

Use ICE PointVelocity

If the ICE PointVelocity attribute is available, then it is used to motion-blur the object.



**Note**

By default, ICE does not push out attributes that are not used in the object's rendertree. To make such attributes available to SltoA, you can either

- Read the attribute in the rendertree by a dummy shader (say, a mixer with null weight)
- Display the attribute in the viewport (this is the fastest way, although you may get annoyed by the extra text)
- Make the ICE attribute pass through a Log Value . This is the advised method

Displacement

Height

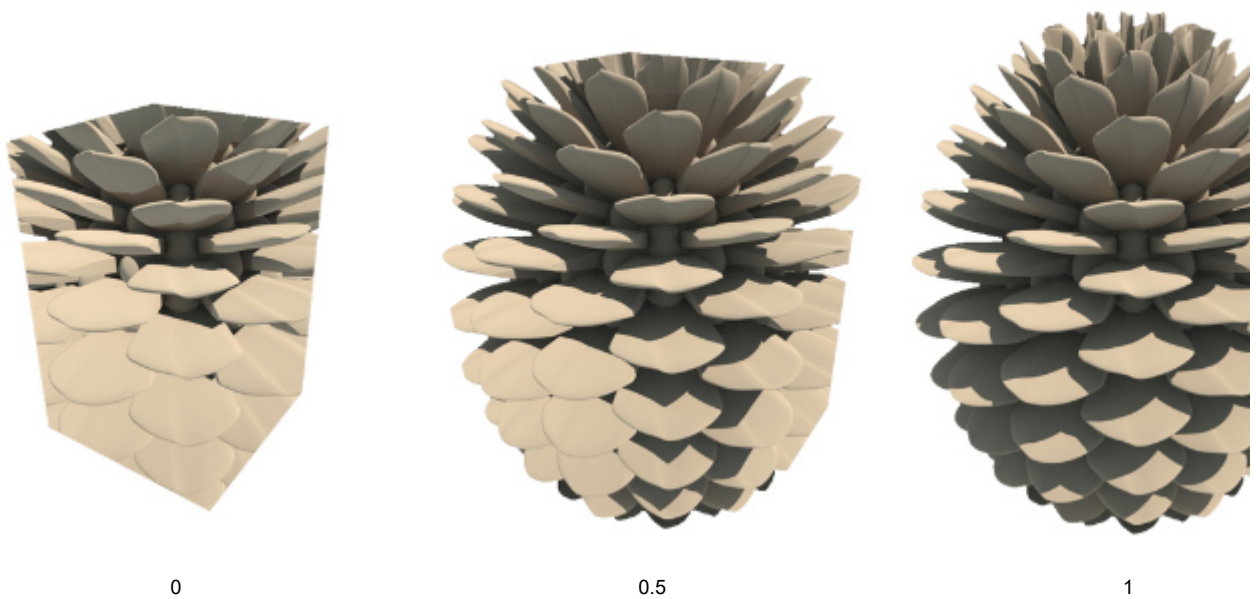
Controls the amount of displacement. Displacement_height can have either positive or negative values. This attribute only applies to normal displacement. •You can use this value to compensate for any inconsistencies between the exported displacement map and the low-resolution geometry.



Bounds Padding

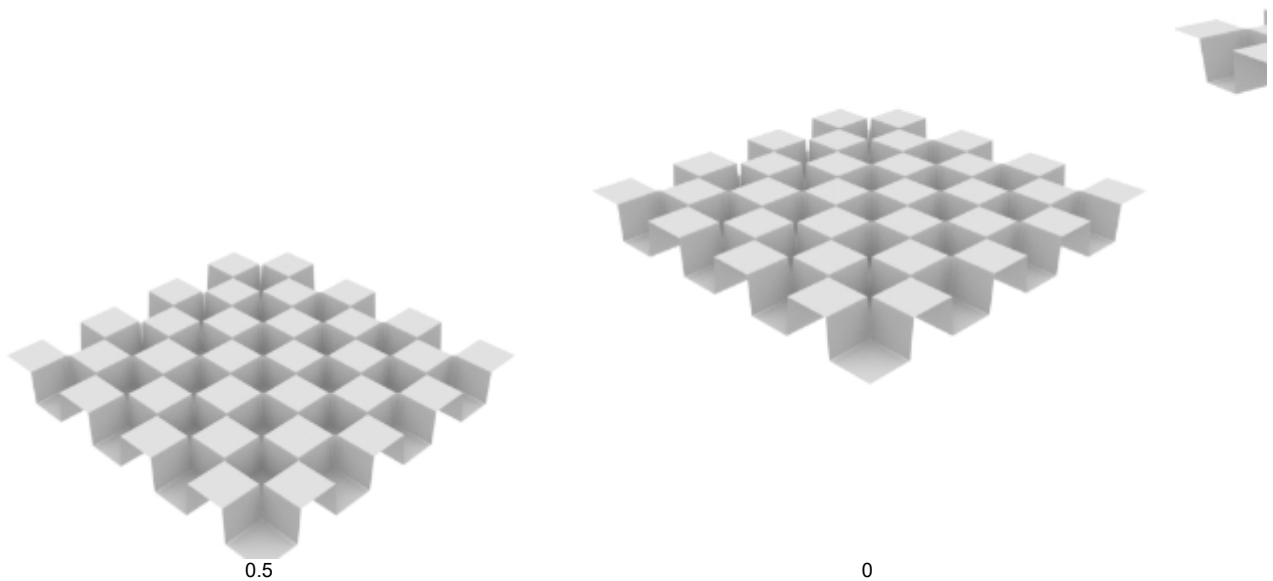
Padding defines how much to extend the bounding box of the object so that it can include any additional displacement coming from the displacement shader. When the bounding box is hit first by a ray, the displacement will be computed, so an unnecessarily high value will decrease the rendering efficiency. On the other hand, a low value could result in a clipping of the displaced mesh.

The proper workflow for displacement in Arnold is to have the shader give the final displacement value and then to offset the bounding box with the bounds padding attribute.



Zero Value

This is a floating-point value that is applied as a shift to the displacement amount. It defines the value of the displacement map that is considered to be zero displacement. This value can vary depending on how the displacement map has been generated.



Autobump

Autobump puts the high frequencies of a displacement map into the bump attribute so that you do not need as many **Subdivision Iteration** values. Autobump is visible to camera rays only by default. The visibility parameters let you make autobump visible to other rays (eg specular and transmission) however that can increase render times.



The autobump algorithm needs UV coordinates to compute surface tangents. Make sure your polymesh has a UV set applied.

Technical information:

When Autobump is enabled, Arnold makes a copy of all of the vertices of a mesh before displacement (let's call that the "reference" mesh, or Pref). Prior to shading at some surface point on the displaced surface P, the equivalent Pref for that point is found on the non-displaced surface, and the displacement shader is evaluated there (at Pref) to estimate what would be the equivalent normal at P if we had subdivided the polymesh at an insanely high tessellation rate.

The main difference between Arnold's Autobump and using the displacement shader for bump mapping is that Autobump has access to Pref whereas bump2d does not and would be executing the displacement shader on already-displaced points which could "compound" the displacement amounts.

The only extra storage is for copying P prior to displacement. There is no analysis of the displacement map; Arnold displaces vertices purely based on where they "land" in the displacement map (or procedural) regardless of whether it happens to "hit" a high-frequency spike or not.



Disabled (subdivision iterations 2)

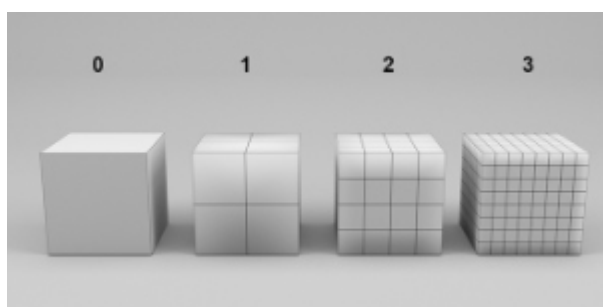


Disabled (subdivision iterations 5)

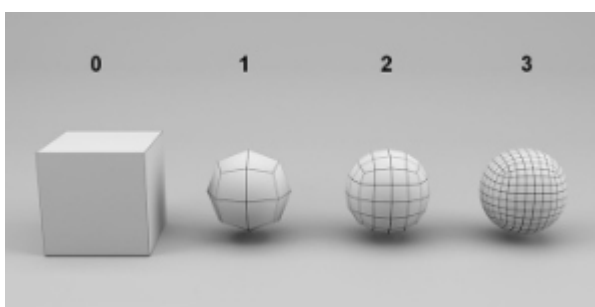
Subdivision

Additional Iterations

The maximum number of subdivision rounds applied to the mesh. When subdiv_pixel_error is 0 the number of rounds will be exact instead of a maximum.



Subdivision Type: Linear



Subdivision Type: Catclark

Adaptive Subdivision

If on, adaptive subdivision is triggered, depending on the next two parameters.

Adaptive• Error

The maximum allowable difference for the chosen metric (subdiv_adaptive_metric) in the chosen space (subdiv_adaptive_space). The smaller the error, the closer to the limit surface a mesh will be and the less inter-frame "popping" when the subdivision level jumps, at the expense of using more polygons. A value of 0 disables adaptive subdivision, reverting to uniform subdivision, which can be more stable in animation.

Examples:

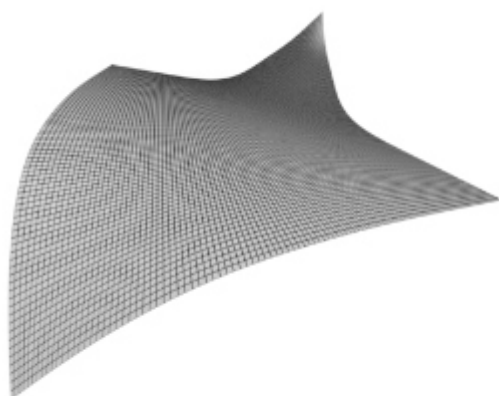
- a value of 1.0 with `subdiv_adaptive_space` set to "raster" and `subdiv_adaptive_metric` set to "edge_length" will subdivide until all edges are shorter than 1 pixel.
- a value of 1.0 with `subdiv_adaptive_space` set to "object" and `subdiv_adaptive_metric` set to "flatness" will subdivide until the tessellation is closer than 1 unit in object space to the limit surface.

See examples [here](#).

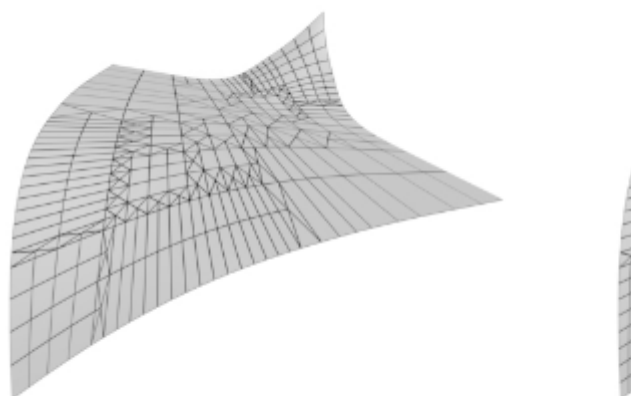
Adaptive Metric

The metric used to determine the amount of error between a given subdivision level and the limit surface. •

- `edge_length` : no final edge will be longer than `subdiv_adaptive_error`.
- `flatness` : the distance to the limit surface will be smaller than `subdiv_adaptive_error`.
- `auto` : in general, flatness will produce better approximation for the same number of tessellated triangles. But since subdivision tessellation does not take into account displacement, "flatness" might stop way too early and miss a lot of displacement detail (think of a displaced plane: a single quad will represent the plane exactly, but will not have enough vertices to show the displacement's detail). To avoid this issue "auto" uses the "edge_length" metric if the polymesh has a displacement shader assigned, otherwise, it will use the "flatness" metric.



Non-adaptive



Flatness

Adaptive Space

Adaptive subdivision in raster space is problematic when instancing: A tessellation that is good for one instance will not be good for another further away. Using object space subdivision will ensure that all instances will subdivide to the proper level.

Raster

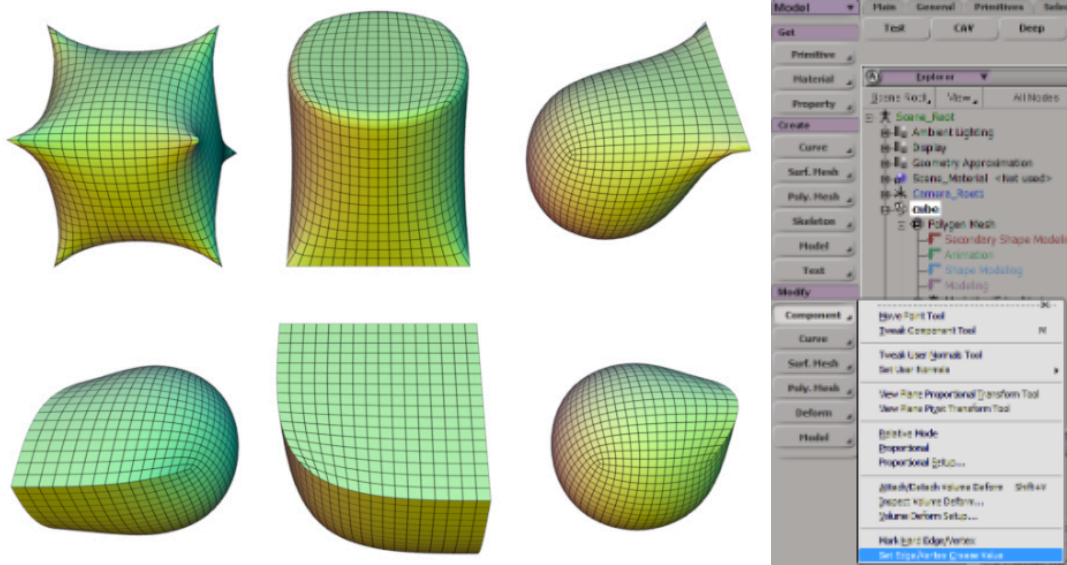
Corresponds to adaptive subdivision in raster space. •

Object •

This method corresponds to adaptive subdivision in object space. This is useful for instances or when adaptive tessellation is desirable regardless of a specific camera. Arnold will warn when raster-space tessellation is used on more than one visible instance.

Subdivision Soft Creases and Vertex Creases ••

OpenSubdiv-compatible edge and vertex soft creases are supported. You can control the sharpness with the `creaseweight` and `creaseweight_point` vertex and point attributes. A crease sharpness value bigger than `polymesh.subdiv_iterations` will de facto encode an infinitely sharp crease. If `crease_sharpness` is not present, all creases will be assumed to be infinitely sharp.



Subdivision creases using the standard Softimage settings

Volume



Polymesh rendered as a volume (rollover image).

noise connected (through range shader) to displacement of standard_volume.

step_size: 0.1, volume_padding: 0.05.

When the Step Size is positive, it turns a polymesh into a volume and its value is used as the base step for raymarching the volume.



The mesh should be closed and not have anything self-intersecting, otherwise, the volume may not render correctly.

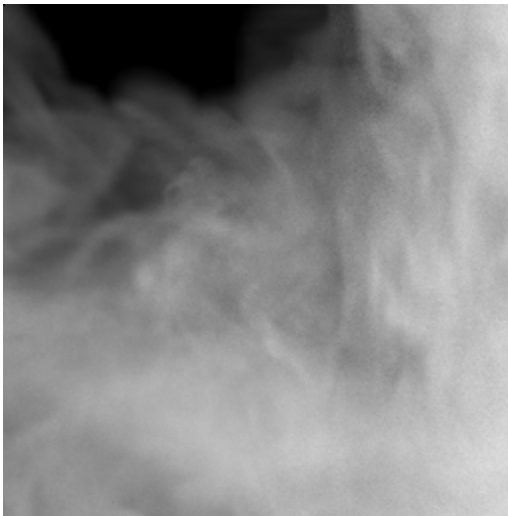


You must add a `step_size` value and assign a `standard_volume` shader for the mesh to render as a volume.

A tutorial that covers how to render a polymesh as a volume can be found [here](#).

Step Size

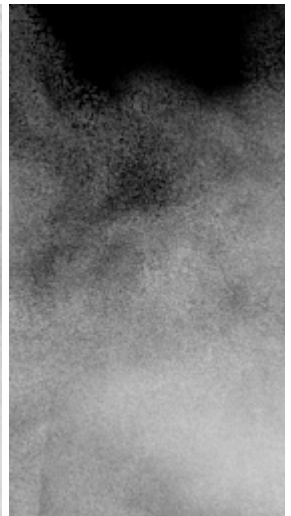
Sets the size for sampling inside the volume. It should be small enough to capture the smallest feature in the volume (aka the Nyquist limit). If the size is too large, aliasing, bias or banding will appear. Smaller (and therefore more accurate) sizes will increase render times while larger sizes will render more quickly.



0.1



2



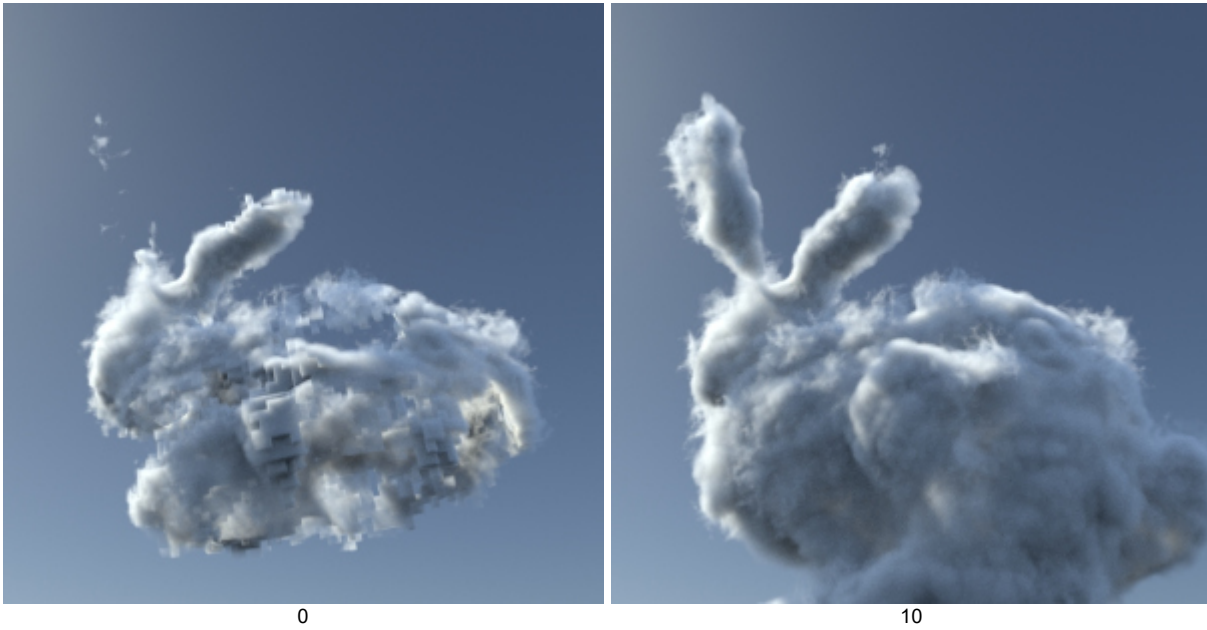
5

Padding

Enlarge the volume by `Padding`. This is useful when displacing a volume with a noise for example. Note that this slack applies not only to the bounding box but also to the voxels used for ray extents inside the bounding box. When velocity motion blur is enabled, some extra padding is computed, based on the maximum value of the velocity field, and the maximum of the velocity padding and user padding is applied. When padding exceeds the bounding box maximum dimension, the ray extents acceleration makes little sense and is disabled, so you should set the padding as tightly as possible.



For best performance, this should be set as low as possible, while still covering the maximum displacement amount.



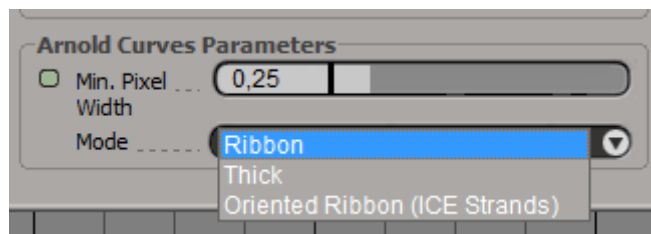
The amount of padding to add to the volume bounds for displacement

-
-

Arnold Parameters (Hair and ICE strands)

These parameters are specific to hair objects and ICE strands.

•



•

Min. Pixel Width

If this value is non-zero, curves with a small on-screen width will be automatically enlarged so that they are at least the specified size in pixels. The enlargement fraction is then used in the hair shader to adjust the opacity so that the visible thickness of the hair remains the same. For a given number of AA samples, this makes it a lot easier to antialias fine hair, at the expense of render time (because of the additional transparency/depth complexity). Good values are in the range of 0.2 to 0.7. Values closer to 0 are faster to render but need more AA samples. So if your scene already uses very high AA settings, you should use a low value like 0.1. For best results, you may need to increase the auto-transparency depth, and/or lower the auto-transparency threshold, but watch the effect on render times. Note that this parameter currently works with the ribbon mode only.



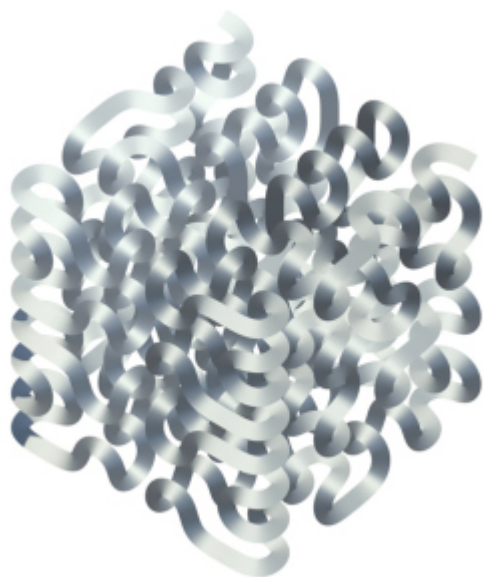
0.1



1

Mode

There are three algorithms for rendering curves in Arnold. •



Ribbon



Thick

Ribbon

Ribbon mode is recommended for fine geometry such as realistic hair, fur or fields of grass. These curves are rendered as camera-facing flat ribbons. For secondary and shadow rays, they face the incoming ray direction. This mode doesn't look so good for very wide hairs or dramatic zoom-ins because of the flat appearance. This mode works best with a proper hair shader (perhaps based on a Kay-Kajiya or Marschner specular model).

Thick

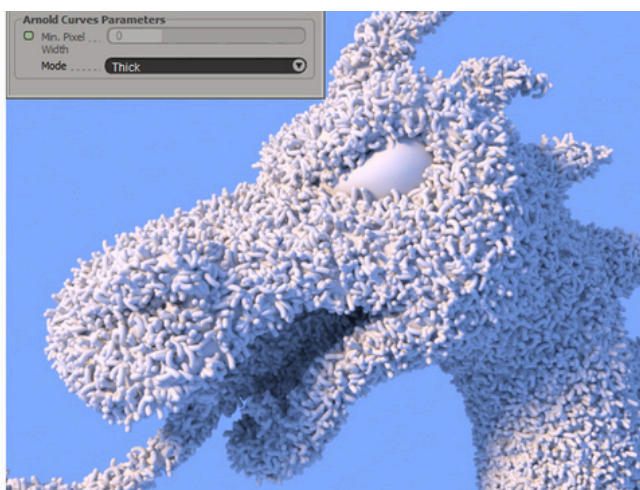
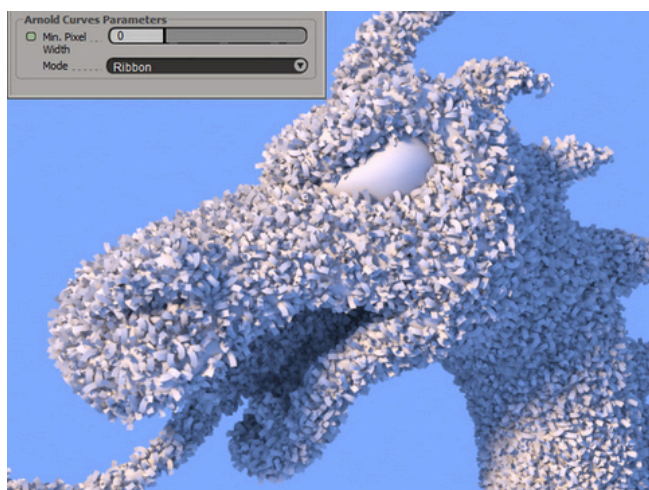
Thick mode resembles spaghetti. It has a circular cross-section and a normal vector that varies across the width of the hair. Thick hairs look great when zoomed in, and are especially useful for effects work, but their varying normals make them more difficult to antialias when they are small. You can use any shader with this rendering mode, including lambert, phong, etc.

Oriented

Oriented mode allows the curves to face in a given direction at each point. This is more useful for modeling blades of grass, tape, and so on.



A more detailed tutorial about Hair in SltA is available [here](#).



Hairs in ribbon mode (left) and thick mode (right)



•

Camera Options

To apply it, select one or several perspective cameras, and run either

- Get->Property->Arnold Properties->Camera Options
- Arnold->Properties->Camera Options
- SITOA_AddCameraOptionsProperties• (script editor).

Starting with 3.1, this property defines the type of the Arnold camera. If not applied, the standard settings of the Softimage camera are used to define the Arnold perspective camera.

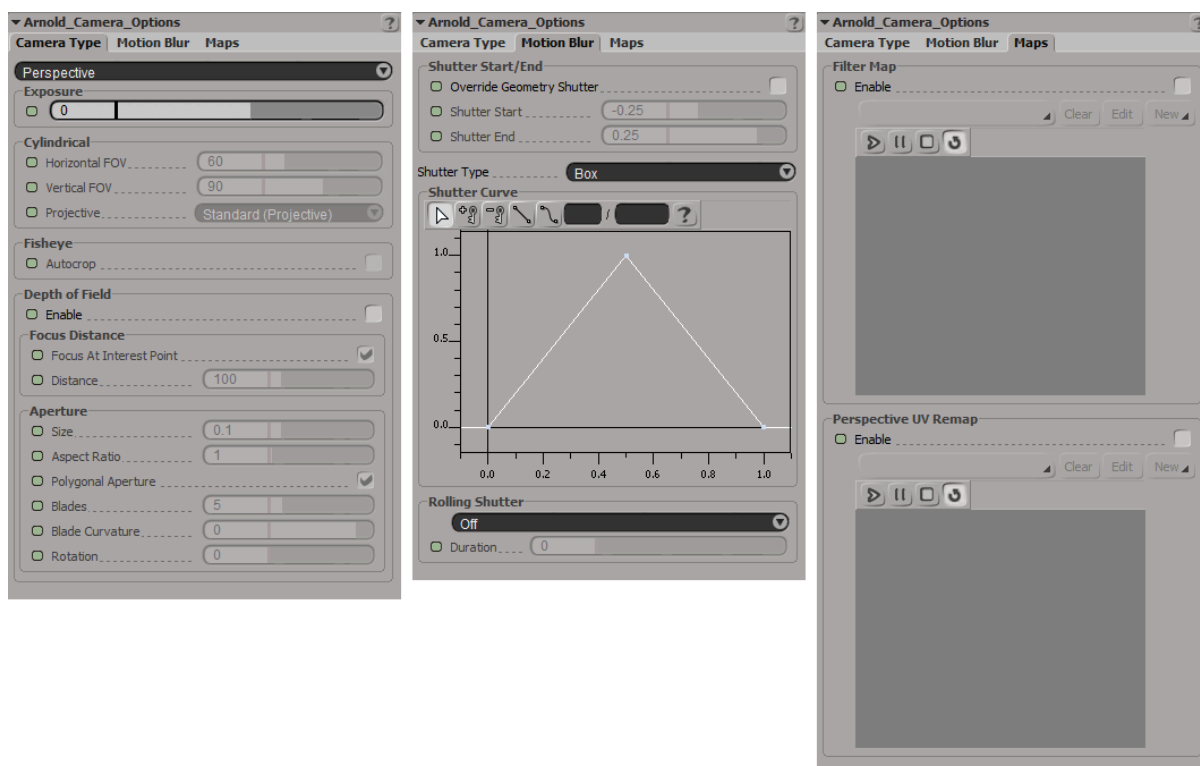
However, if you want to define a different kind of camera (cylindrical, fisheye, spherical), you have to use this property.

This is also mandatory if you want to access Arnold-specific controls, like those defining depth of field, aperture size, and type, advanced motion blur, and remapping.

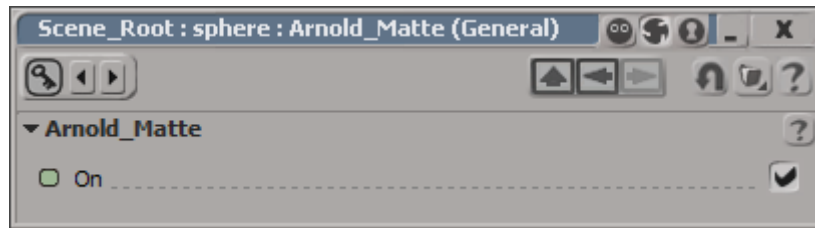
Lens shaders, previously used to set camera-specific attributes, have been discontinued.

For a complete description of the options, see the [camera](#) page.

•



Matte



.

To apply it, select one or several objects, or a group, and run either

- Get->Property->Arnold Properties->Matte
- Arnold->Properties->Matte
- SITOA_AddMatteProperties*(script editor).

.

This property can work in either Matte or Mask Mode. It is enabled by the On check.



All of the matte properties in the scene can be disabled by the •Matte Properties •flag in the •Ignore •group of the rendering options.

.

When enabled, this turns an object or group into a holdout matte. It affects primary rays only. The matte color is always black, and the opacity of the object is preserved. Shaders will not run on the object anymore and it will output all-black (including the alpha), except if the opaque parameter is off, in which case shaders will be run just to compute the opacity. Note that even AOVs output by its shaders in this case will be black.



The Matte attribute does not matte out the AOV contributions of built-in AOVs such as P, N, and raycount.



Disabled



Enabled



Standard Surface shader with 50% opacity



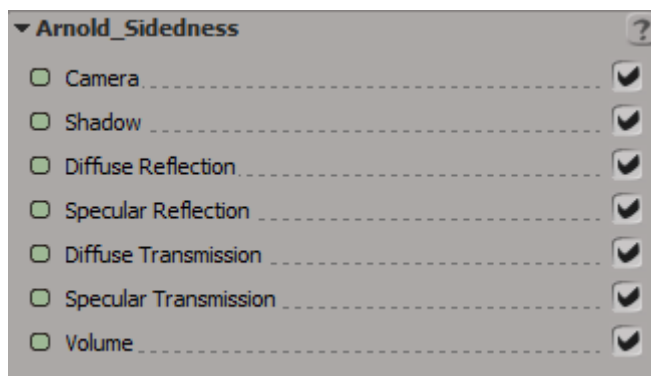
Alpha mask showing 50% opacity



There is also the option to create a matte with a Matte shader.

-
-

Sidedness



•

Just like you can disable the visibility for specific ray types, you can also change an object's sidedness depending on the ray type. By default, objects are double-sided for all rays. By disabling any of these checkboxes, the object will become single-sided, which means that those parts of the object whose normal vector points away from the incoming ray direction will not be traced, depending on the ray type.

To apply it, select one or several objects and run either

- Get->Property->Arnold Properties->Sidedness
- Arnold->Properties->Sidedness
- [SITOA_AddSidednessProperties](#) (script editor).

Camera

Camera rays (i.e. primary or view rays).

Shadow

Shadow rays fired in the direct lighting calculations.

Diffuse •Reflection

Indirect diffuse rays (i.e. global illumination, hemi, or GI rays).

Specular Reflection

Indirect specular rays (i.e. specular reflection rays).

Diffuse Transmission

Object is visible in diffuse transmission and subsurface scattering. •

Specular Transmission

Object is visible in sharp and glossy specular transmission (refraction).

•

Procedurals

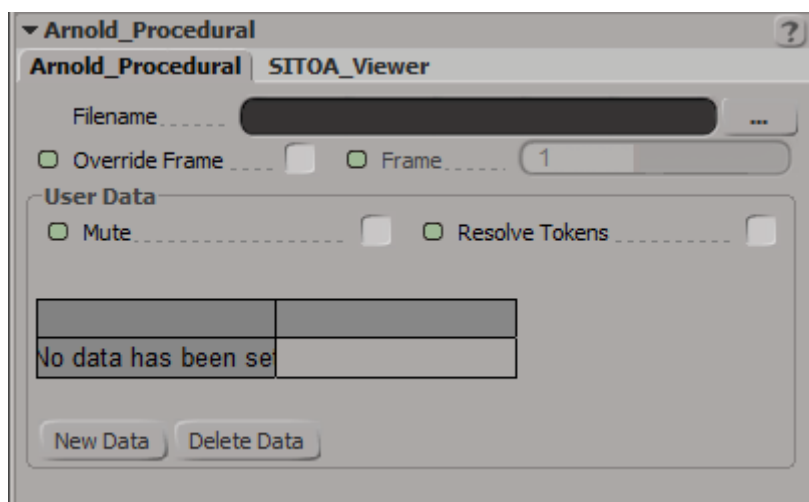


Duck models are instanced Arnold Procedurals

• In SItOA you can export any object as an **ASS** file. A procedural references the ASS file on disk (or other procedural formats supported by Arnold such as **.obj**, **.ply**). Arnold Procedurals (previously called "stand-ins") allow you to keep your working scene light and workable by deferring the loading of geometry data until render time. The object is exported as an Arnold procedural node.

• To apply it, select an object and run either:

- Property->Arnold Properties->Procedural
- Arnold->Properties->Procedural
- [SITOA_AddProceduralProperties](#) (script editor).



[Filename](#)

The full path to the file to be loaded. Valid extensions are *.ass, *.ass.gz, *.obj, *.obj.gz, *.dll (Windows), *.so (Linux). The path string supports the use of Softimage tokens. For example, *c:\temp\dummy.[Frame].ass is resolved with the current rendering frame when rendering a sequence. Tokens support padding and shifting. So, *c:\temp\dummy.[Frame #4 +2].ass is resolved as *c:\temp\dummy.0022.ass at frame 20.

Override Frame, Frame

If Override Frame is on, the Frame value is used to resolve the Path's [Frame] token (if it exists). For example, you can have Frame defined by an expression and still benefit from the Path padding and shifting syntax. If Path is not a sequence, these parameters are not used.

User Data

If enabled (Mute off), each row of the grid can define a user data by its Name, Structure, Type, and Value. You can add a row by clicking New Data, or delete an existing one by selecting it and clicking Delete Data. Value can contain Softimage tokens, that are resolved if Resolve Tokens is enabled.

A Structure can be either SINGLE or ARRAY, the Type can be BOOL, INT, VECTOR, POINT, etc.

The syntax to use for the Value depends on the structure and the type.

- If the structure is SINGLE, and the type is made of a single element (bool, int, float) just type the plain value. Instead, if the type is a vector, point, etc, each field must be separated by a white space.
- If the structure is ARRAY, the array members must be separated by a ',' character.

As an example, here is a list of valid entries, and the corresponding section of the exported ass file.

Name	Structure	Type	Value
BoolSimple	SINGLE	BOOL	1
VecSimple	SINGLE	VECTOR	1 2 3
VecArray	ARRAY	VECTOR	4 5 6, 7 8 9, 10 11 12
FloatArray	ARRAY	FLOAT	13.0, 14.0
StringArray	ARRAY	STRING	One,Two,Three

New Parameter Delete Parameter

```
procedural
{
• name•cube
• dso•"C:/temp/someArchive.ass"
• min•-4•-1•-4
• max•4•1•4
• matrix•
• 1•0•0•0
• 0•1•0•0
• 0•0•1•0
• 0•0•0•1•
• declare•BoolSimple•constant•BOOL
• BoolSimple•on
• declare•VecSimple•constant•VECTOR
• VecSimple•1•2•3
• declare•VecArray•constant•ARRAY•VECTOR
• VecArray•3•1•VECTOR•4•5•6•7•8•9•10•11•12•
• declare•FloatArray•constant•ARRAY•FLOAT
• FloatArray•2•1•FLOAT•13•14•
• declare•StringArray•constant•ARRAY•STRING
```

```
• StringArray•3•1•STRING•• "One" "Two" "Three"
}
```

Procedurals as ICE instance shape

A typical usage of the Procedural•property is to replace an ICE instance shape with complex geometry, as shown below. Because of the number of objects in the model, having all that geometry in the scene would make the export time rather long. Referencing the archive, on the other hand, makes the render start in a matter of seconds.

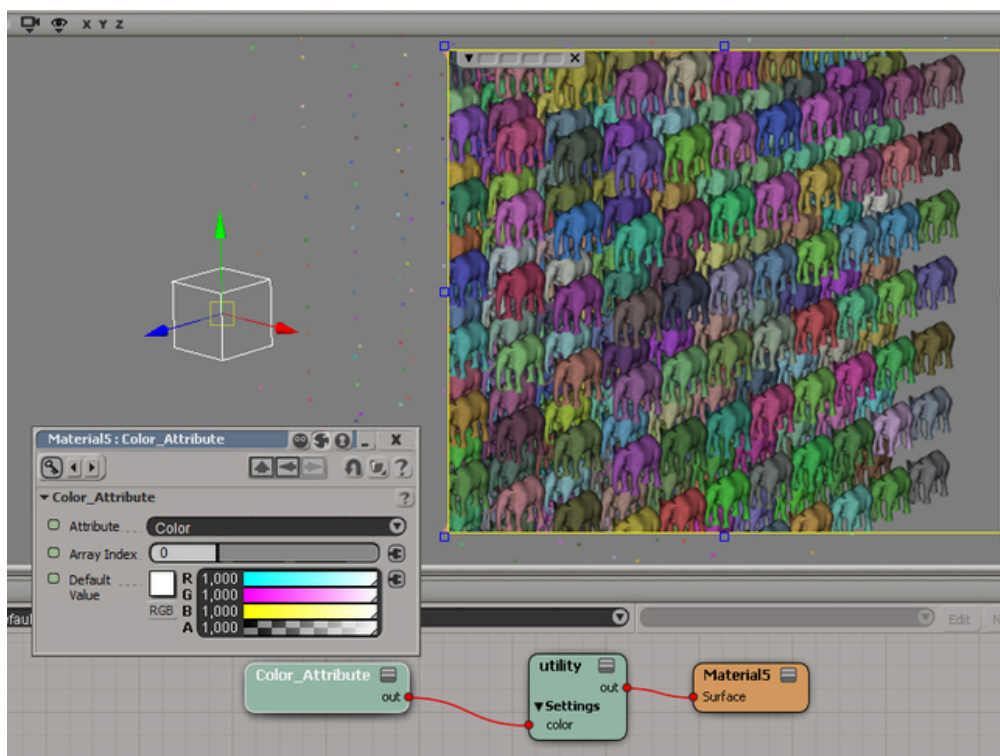
For more information, see the [ICE page](#).

 If you want motion blur on a Procedural, you will have to export the Procedural with motion blur enabled.

Materials

We use the following convention: if the material of the placeholder object is called Scene_Material or Procedural_Material, then the original shaders are used, i.e. the shaders stored in the .ass (or other types of procedural).

Else, the material applied to the placeholder propagates to the Procedural, allowing you to, for instance, render an occlusion or matte pass from the Procedurals without having to export an .ass file with that material already applied. In the image below, we override the native elephant's material with a utility shader, getting its color from the Color attribute set by ICE.

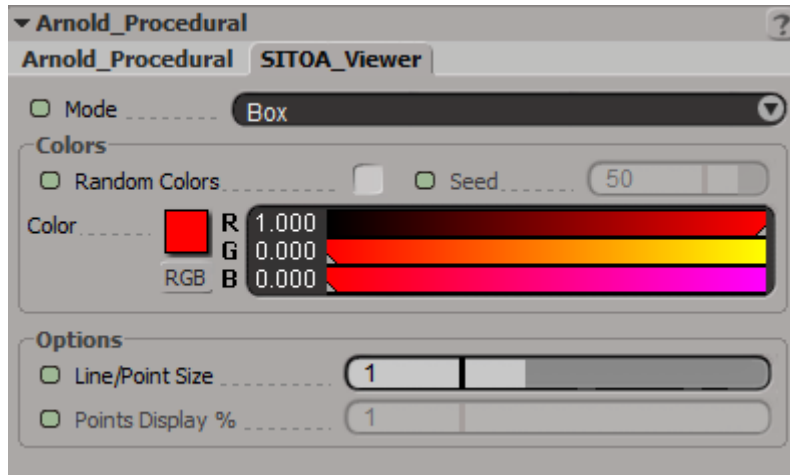


Properties

Properties (Visibility, Sidedness, User Options ...) applied to the proxy object propagate to the objects generated by the procedural.

Previewing Procedurals

The [SITOA_Viewer](#) graphic sequencer allows you to preview the content of the procedural in the viewport.



 The Utility shader's Object mode does not work with Procedurals. This is a known limitation.



Spheres



Stand-ins

The color of the two spheres changes when the Utility shader's mode is set to 'Object'. However, the Procedural's color remains the same (rollover image).

Texture Options

This property allows you to refine the options used by Arnold when looking up an image texture.

To apply it, select an image and run either

- Get->Property->Arnold Properties->Texture Options
- Arnold->Properties->Texture Options
- SITOA_AddTextureOptionsProperties (script editor).

•

The property is placed under the Custom folder of the image. Options are:

Filter

Closest, Bilinear, Bicubic, Smart Bicubic.

Mipmap Bias

An integer, defining the mipmap level bias.

Swap UV

If on, swaps U and V

U Wrap

Don't Override, Periodic, Black, Clamp, Mirror, File

V Wrap

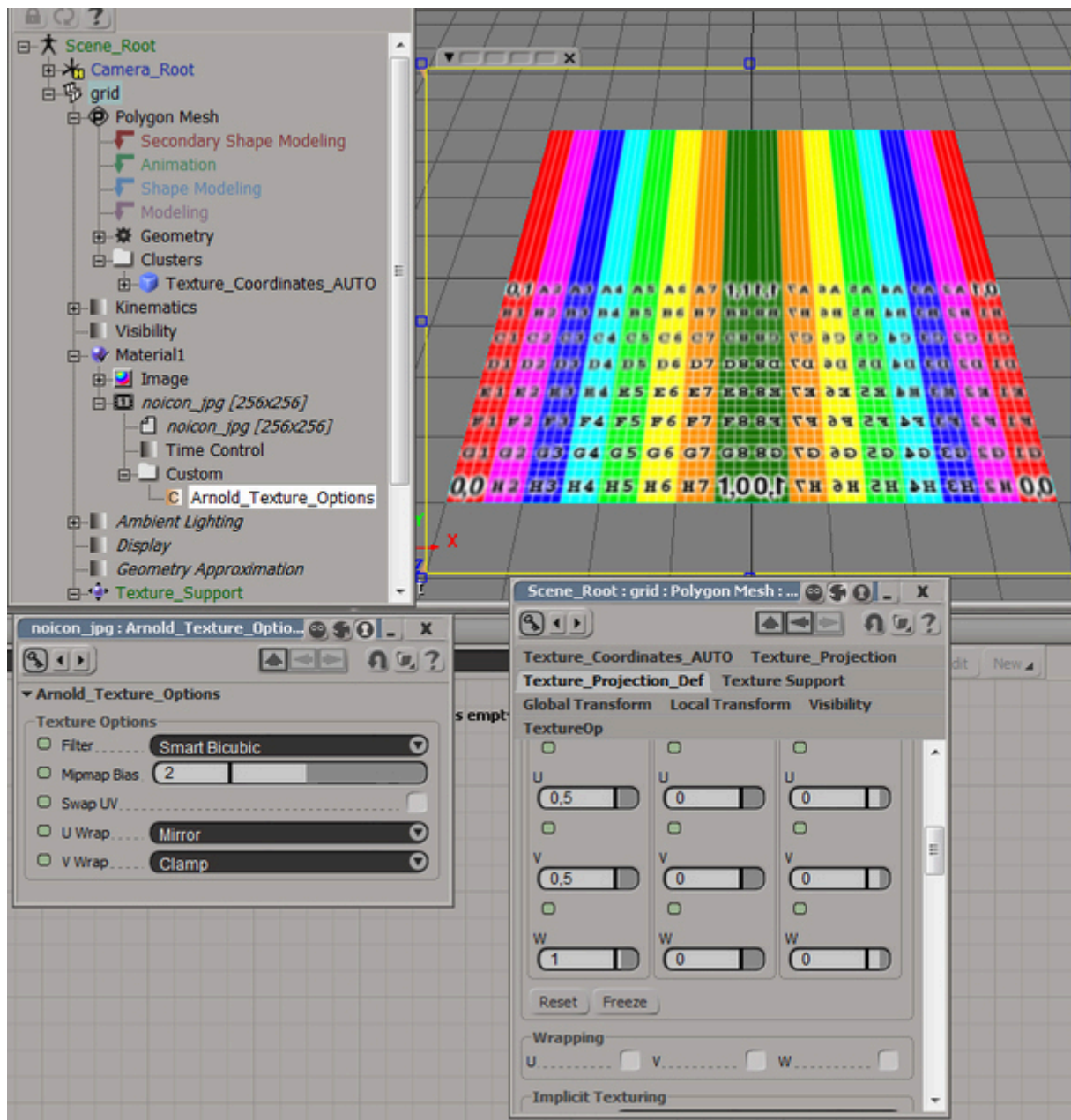
Same as above, for the V coordinate.

The wrapping options define the lookup behavior where U (and/or V) exceeds the standard 0..1 range. When set to Don't Override, the wrapping is driven by the wrapping flags of the texture projection property. Instead, if the UVs are not wrapped yet, you can decide what to do by choosing one of the five options available.

So, if you want to wrap at this later stage, remember to disable the Wrapping in the texture projection property, else the image node will get UVs in the 0..1 range, and it will not be able to apply the new wrapping modes.

In the image below, the UV range is 0..2.

We get some blur because Mipmap Bias is set to 2, Mirror wrapping along U and Clamp wrapping along the V.





User Data Blobs

Starting from 2.7, SltOA supports user data blobs. Blob is a native Softimage property, not an Arnold one.

This feature is provided for developers needing to attach some data on a given primitive (objects, cameras, lights) and to retrieve it later at render time.

The blob is usually written by some custom tool (a command or an operator) and read by a custom shader.

In Softimage, the blob buffer is an unsigned char* array, and it's exported by SltOA as an array of constant bytes for the primitive owning the blob.

It's up to the developer to perform the correct conversion between the buffers structure and the raw sequence of bytes stored in the blob, and exported by SltOA as such.

The blob is exported only if its RenderData flag is enabled.

Blobs are supported for mesh objects, cameras, lights, stand-ins, hair and ICE objects. Clusters instead are not supported. Also, the render region will not update in case your blob-writer tool modifies the buffer content, You will have to change frame, or to destroy the Arnold universe.

In the following example, we'll attach a RGB color to a cone by a cpp command. The color will then be read and returned by a custom shader.

Writing the Blob

We use a cpp command, called blob_writer, implemented by [blob_writer.cpp](#)

To run it, select an object (a cone in the example) and run:

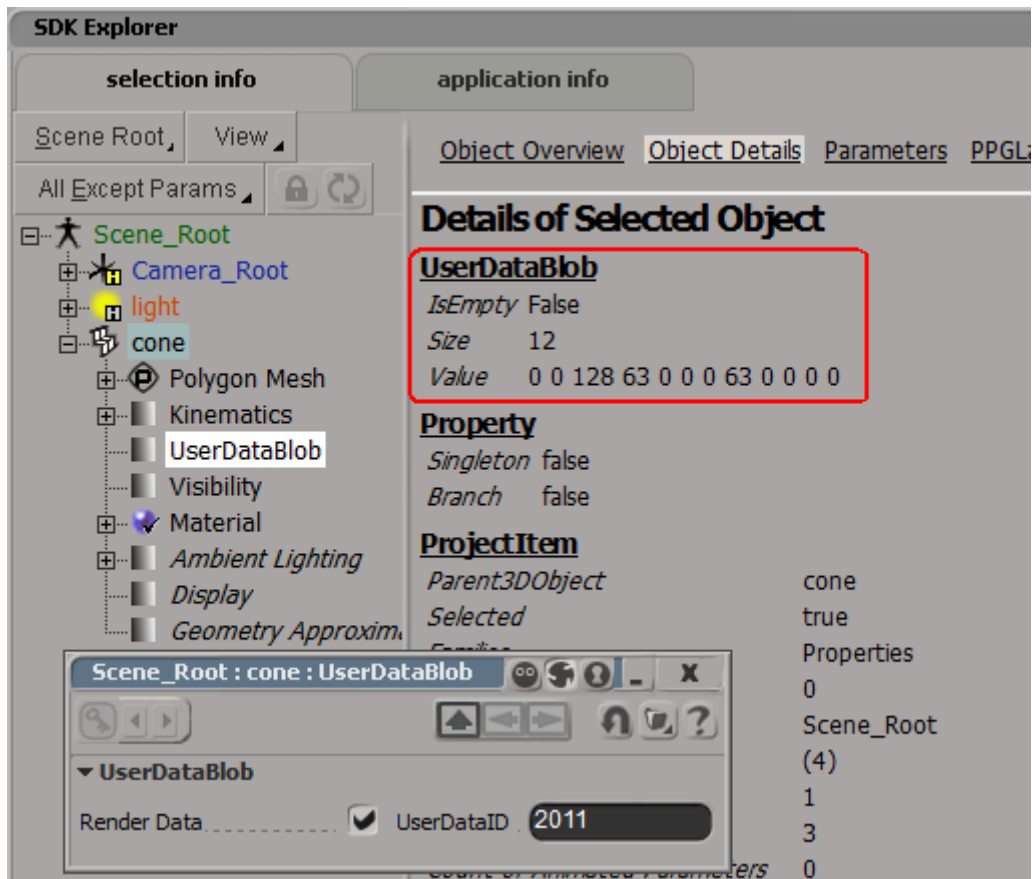
```
blob_writer("cone")
```

What the command does, is:

1. Create a user data blob property on the cone
2. Set its RenderData flag on, and its id to 2011
3. Fill a RGB structure with an orange color and push this buffer as the blob value

```
typedef struct
{
    float r, g, b;
} BlobRGB;
SICALLBACK blob_writer_Execute( CRef& in_ctxt )
{
    ...
    Primitive prim = obj.GetActivePrimitive();
    Property prop = obj.AddProperty("UserDataBlob");
    prop.PutParameterValue(L"RenderData", true);
    prop.PutParameterValue(L"UserDataID", (LONG)2011);
    UserDataBlob blob(prop);
    BlobRGB rgb;
    rgb.r = 1.0f;
    rgb.g = 0.5f;
    rgb.b = 0.0f;
    blob.PutValue((const unsigned char*)&rgb, sizeof(BlobRGB));
}
```

After running the command, the blob shows up in the SDK explorer, with the 12 bytes representing the 3 color's float values:



This is just plain Softimage. If you now export the scene to an ASCII .ass, you will see the blob section at the end of the cone polymesh block:

```
polymesh
{
  name cone.SitoA.1000
  ...
  declare UserDataBlob_ID constant UINT
  UserDataBlob_ID 2011
  declare UserDataBlob constant ARRAY BYTE
  UserDataBlob 12 1 BYTE
    0 0 128 63 0 0 0 63 0 0 0 0
}
```

Note that the user data goes by the name of the blob ("UserDataBlob" in this case, but could have been any).

Also, the user data id (2011 in this case) is exported, being the data named by the blob name plus a "_ID" suffix.



Note that, when exporting to .ass, the bytes array is dumped in ascii format. If .ass is your way to go, try to keep the blob size as small as possible. For example, if you need to pass a voxel field to your shader, you should better pass the path to the data file, instead of the full array of data itself.

Reading the Blob

To read the exported blob, we give the cone a custom shader called blob_reader, implemented by [blob_reader.cpp](#) and [blob_reader.spdl](#).

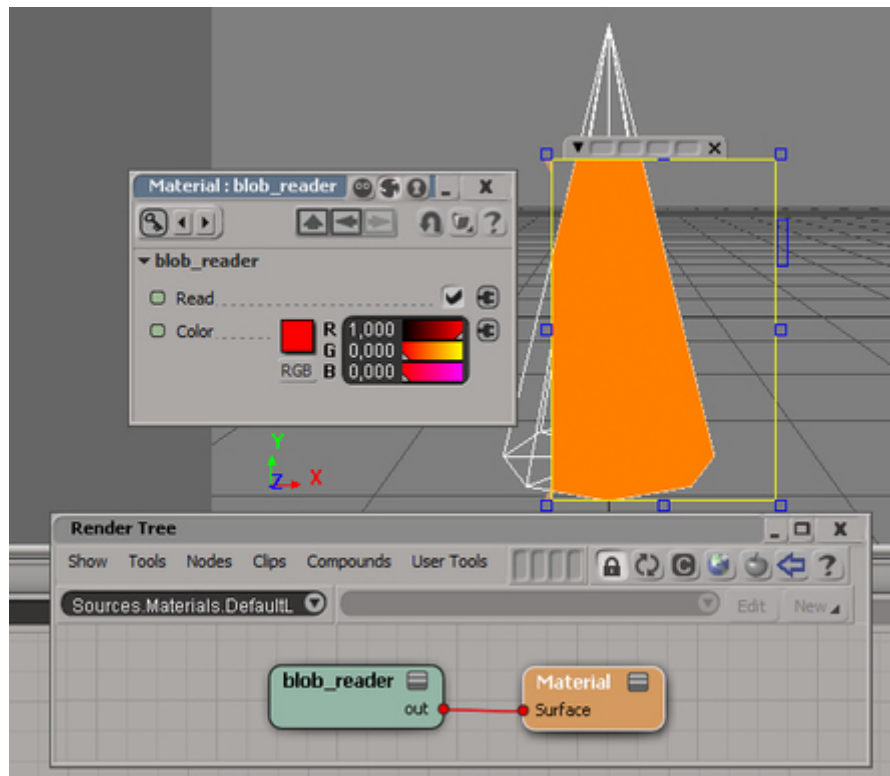
The shader looks up the data associated with the blob name and copies the 12 bytes into the output RGB values.

Note that, for documentation reasons, this code is far from being optimized. Since the data is a static array, a real production shader should convert the bytes to the expected format only once, and surely not on every evaluate cycle.

```
#define BLOB_NAME "UserDataBlob" // this must match the blob name

shader_evaluate
{
    AtRGBA color; // the result
    ...
    const AtUserParamEntry *paramEntry = AiNodeLookUpUserParameter
(sg->Op, BLOB_NAME);
    if (paramEntry)
    {
        AtArray *array = AiNodeGetArray(sg->Op, BLOB_NAME);
        AtByte bytes[4];
        for (AtUInt i=0; i<4; i++) // bytes 0..3 -> r
            bytes[i] = AiArrayGetByte(array, i);
        memcpy((void*)&color.r, (void*)bytes, 4);
        for (AtUInt i=0; i<4; i++) // bytes 4..7 -> g
            bytes[i] = AiArrayGetByte(array, 4+i);
        memcpy((void*)&color.g, (void*)bytes, 4);
        for (AtUInt i=0; i<4; i++) // bytes 8..11 -> b
            bytes[i] = AiArrayGetByte(array, 8+i);
        memcpy((void*)&color.b, (void*)bytes, 4);
        color.a = 1.0f;
        sg->out.RGBA = color;
    }
}
```

This is the result. The red color is skipped (see the full shader's code), and the orange color encoded in the blob array is returned instead.



User Options

To apply it, select one or several objects or lights and run either

- Get->Property->Arnold Properties->User Options
- Arnold->Properties->User Options
- [SITOA_AddUserOptionsProperties](#) (script editor)

This is a general purpose property, that can be used to over-ride any attribute of an Arnold core node, or to attach constant user data to it. This allows you to, for example, access and set Arnold core parameters which are not currently exposed in the SitOA user interface. The property can be applied to poly-meshes, hair, lights, and point clouds.

Mute

If on, the property is muted.

.

Resolve Tokens

If enabled, the Softimage tokens such as [Frame] are resolved before the Options string is passed to the Arnold node.

Options

The string to be set. Because the string is passed to the Arnold node directly, it is up to you to fully understand the core Arnold node type being modified and compose the string correctly. This is a powerful option and should be used with some care. To get the list of parameters and types of a given type of node, you can use `kick -info`. For example, to list all the parameters of a polymesh node, run `kick -info polymesh`. If you want to set multiple Arnold parameters in a User Options string, you can use any whitespace (spaces, tabs, newlines eg `\n`) that you would use in an actual ASS file. For example, this would set the `enable_procedural_cache` and `error_color_bad_pixel` parameters, and declare and set a user parameter named `test`:

```
enable_procedural_cache off error_color_bad_pixel 1 .25 1 declare
test constant FLOAT test 2
```

User Data

If enabled, each row of the grid can define a user data by its Name, Structure, Type and Value. You can add a row by clicking New Data, or delete an existing one by selecting it and clicking Delete Data.

Structure can be either SINGLE or ARRAY, the Type can be BOOL, INT, VECTOR, POINT, etc.

The syntax to use for the Value depends on the structure and the type.

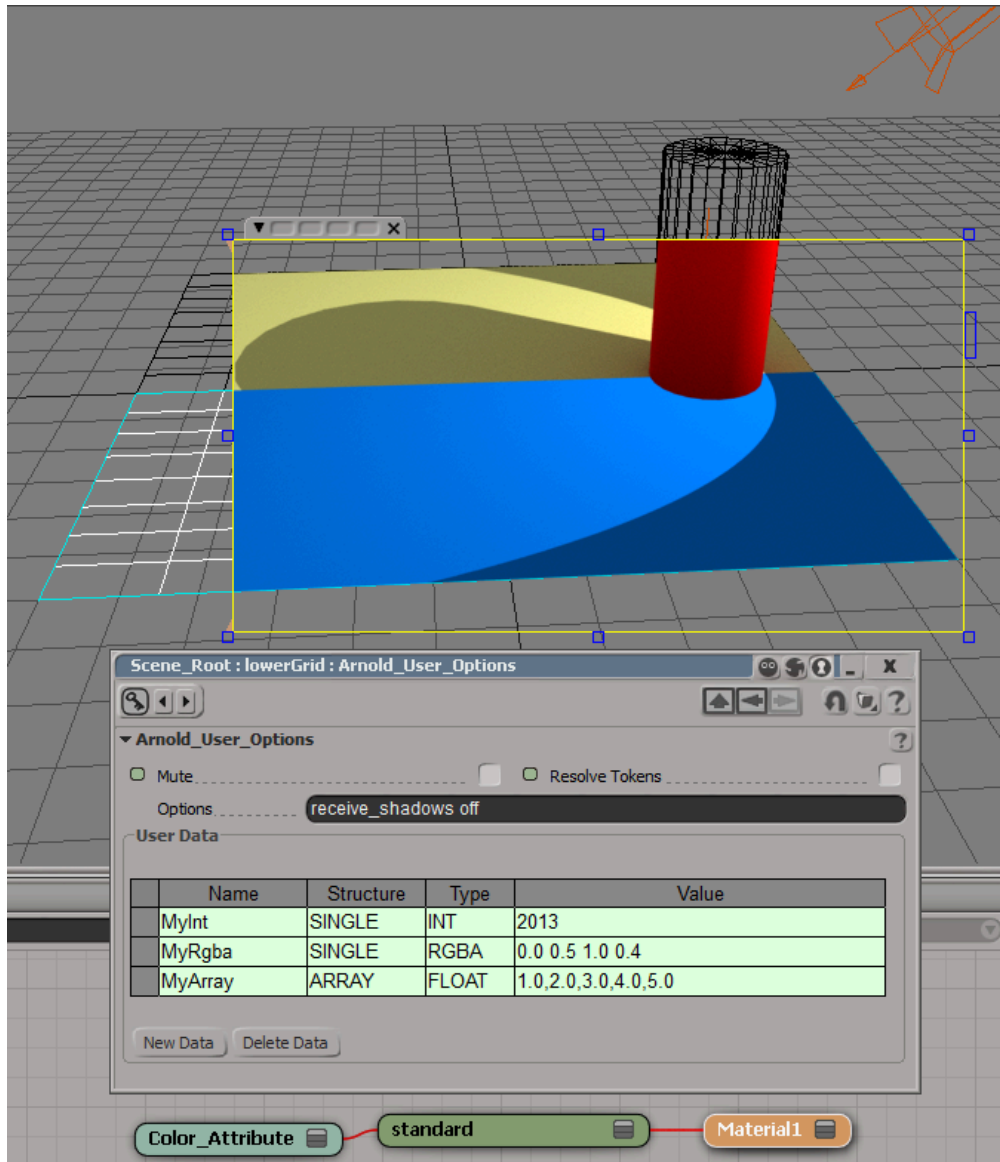
- If the structure is SINGLE, and the type is made of a single element (bool, int, float) just type the plain value. Instead, if the type is a vector, point, etc, each field must be separated by a white space.
- If the structure is ARRAY, the array members must be separated by a ',' character.

Example

In this example, the two grids share the same material, and we use both the options string and the user data for the lower grid.

The string disables the received shadows, the user data grid attaches an integer, a color and an array of float to the object.

The color user data (MyRGBA) is pulled by the attribute shader, whose default is a light yellow color. Since the attribute exists for the lower grid only, then it renders blue, while the upper grid stays yellow.



If you export the scene to .ass, you'll see the declared user data listed at the end of the grid block

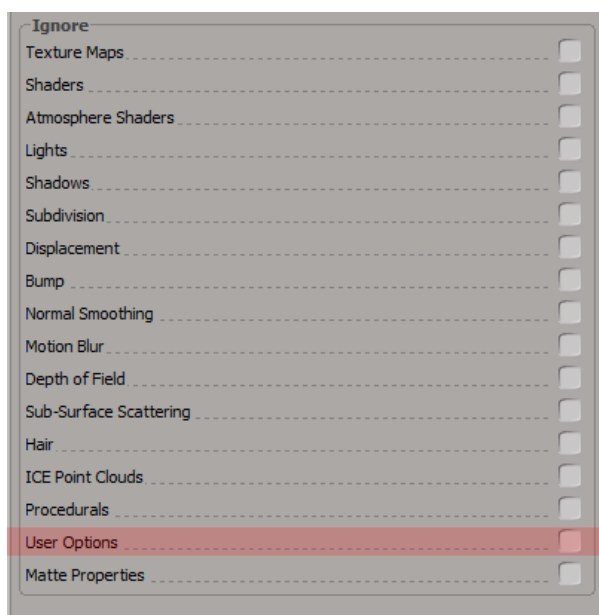
```
polymesh
{
  name lowerGrid.SItoA.48000
  ...
  declare MyInt constant INT
  MyInt 2013
  declare MyRgba constant RGBA
  MyRgba 0 0.5 1 0.400000006
  declare MyArray constant ARRAY FLOAT
  MyArray 5 1 FLOAT
  1 2 3 4 5
}
```

To access the user data in an Attribute shader, you'll need to use scripting. For example, to use MyRGBA in a Color Attribute shader, you could set the value like this:

```
# Python
Application.SetValue("Sources.Materials.DefaultLib.Material.
Color_Attribute.attribute", "MyRGBA", "")
```

Ignore User Options

All the options strings can be disabled by setting the Ignore flag for User Options in the Diagnostics tab of the rendering options:

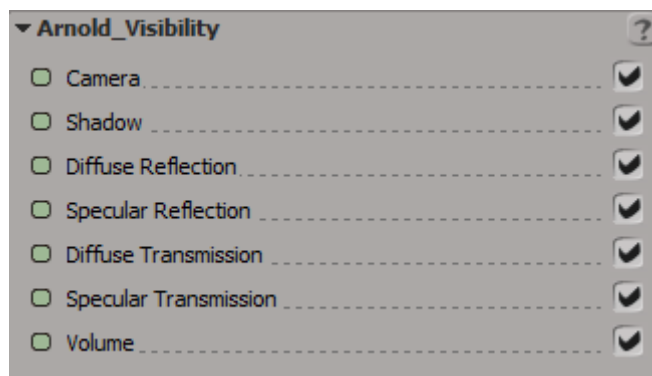


Any change in the options string forces a scene destroy, since the string is passed to Arnold as it is, and SltoA can't predict if the attribute you're setting is IPR-compatible or not.



any change in the options string forces a scene destroy, since the string is passed to Arnold as it is, and SltoA can't predict if the attribute you're setting is IPR-compatible or not.

Visibility



.

To apply it, select one or several objects and run either:

- Get->Property->Arnold Properties->Visibility
- Arnold->Properties->Visibility
- [SITOA_AddVisibilityProperties](#) (script editor).

.

You can selectively disable an object's visibility for the various types of rays in the renderer. By default, objects are visible to all types of rays. Camera - Camera (AA) rays (i.e., primary or view rays). Shadow - shadow rays fired in the direct lighting calculations. Specular - specular_reflection rays. Transmission - transmission rays. Diffuse - indirect_diffuse rays (i.e. global illumination or GI rays). Specular - indirect_specular rays (i.e. specular reflection rays).

To set the visibility, add up the values of the rays you want to "see" the object.

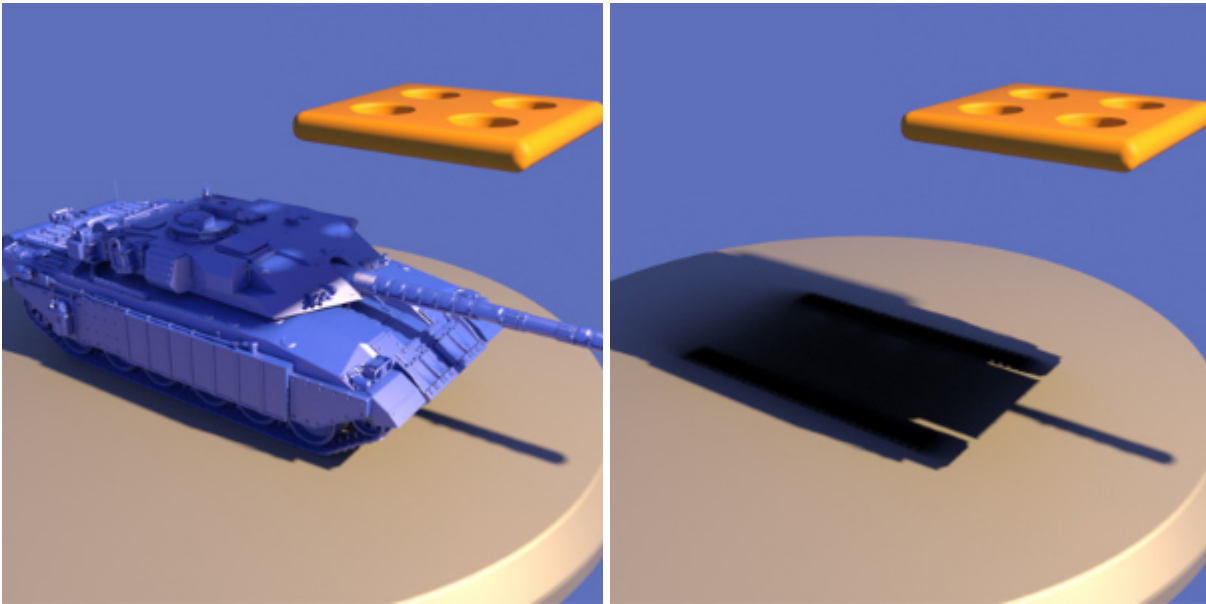
For example, if an object is visible to the camera (primary visibility), and to transmission (both diffuse and specular), then that's•visibility 13 (1 + 4 + 8)

.

Ray type	Decimal Value
Camera (Primary Visibility)	1
Shadow (Casts Shadows)	2
Diffuse transmission	4
Specular transmission	8
Volume	16
Diffuse reflection	32
Specular reflection	64
SSS (subsurface)	128

Camera

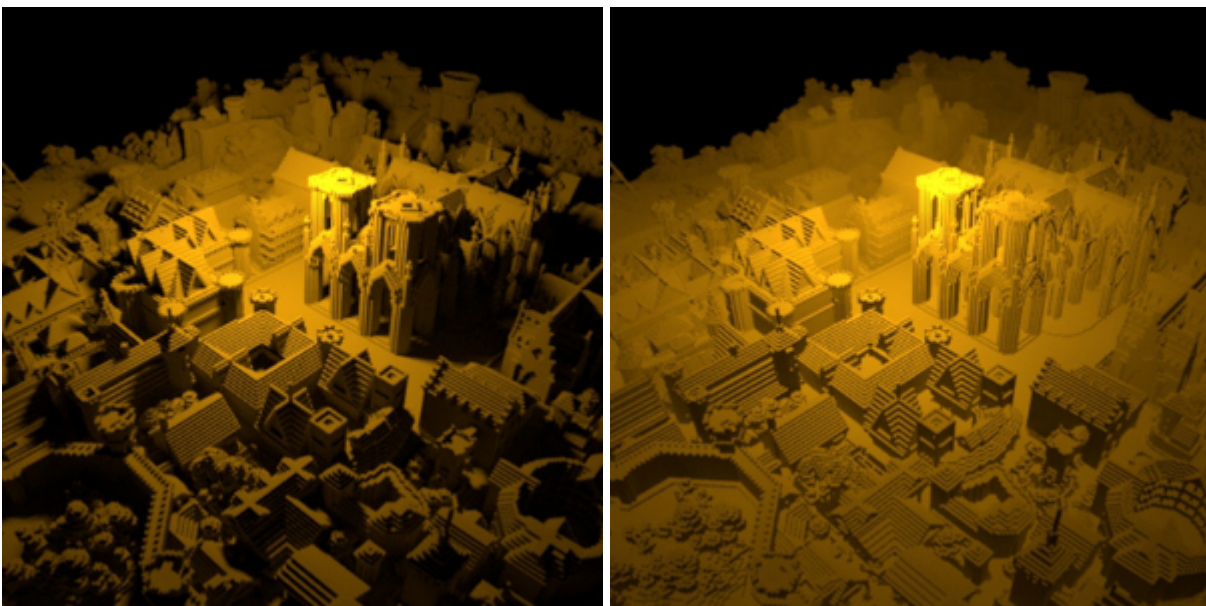
Determines whether the object is visible to camera rays.



The tank in this example is invisible to camera rays but visible to all other types of rays, including shadow rays

Shadow

Enables the computation of shadows cast from the light. •

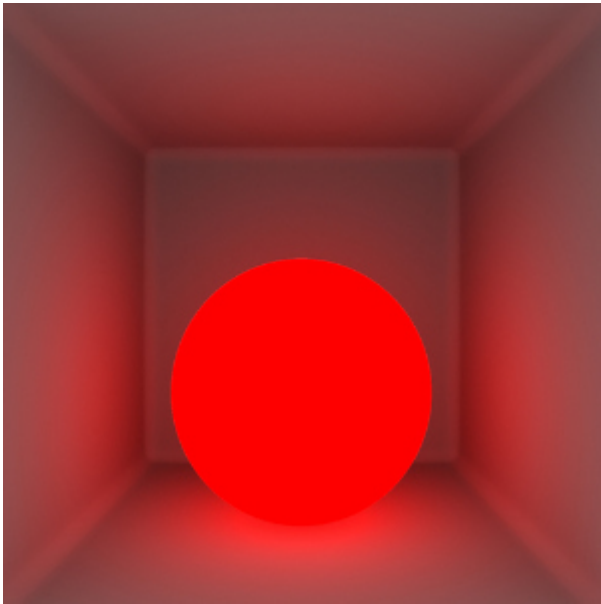


Enabled (default)

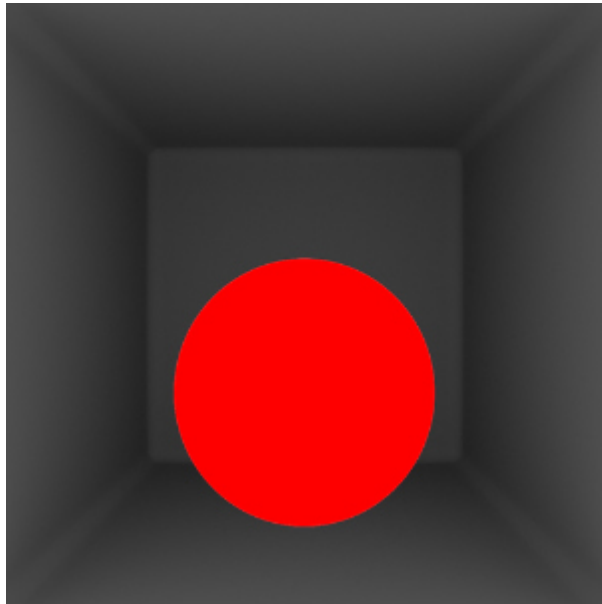
Disabled

Diffuse •Reflection

Affects indirect diffuse rays (i.e., global illumination, hemi, or GI rays).



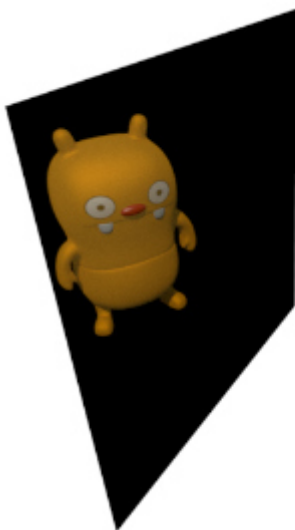
Sphere has 'Visible in Diffuse' enabled



Sphere has 'Visible in Diffuse' disabled

Specular Reflection

You can selectively disable an object's visibility for the various types of rays in the renderer. By default, objects are visible to all types of rays. Specular Reflection visibility affects indirect specular rays (i.e., specular reflection rays).



Enabled



Disabled



Additionally, you may use the built-in "Visibility" parameters that each object has by default in Softimage. When SlttoA's own visibility property is applied, it will override any Softimage visibility settings that the object itself has locally.

Diffuse Transmission

Object is visible in diffuse transmission and subsurface scattering. •

Specular Transmission

Object is visible in sharp and glossy specular transmission (refraction).

Volume



Object is visible in indirect volume rays.

-
-

Volume



OpenVDB bunny cloud volume available [here](#).

Volume file formats, such as OpenVDB, will allow storage of much larger datasets while being optimized for random access to volume data, which is more efficient for rendering.

[OpenVDB](#) is an open source C++ library comprising a novel hierarchical data structure and a suite of tools for the efficient storage and manipulation of sparse volumetric data discretized on three-dimensional grids.



Rendering of volumes should not be confused with [Atmospheric Scattering](#), which is a global effect.



A tutorial on how to write volumes to OpenVDB from Houdini can be found [here](#).



Volumes can only receive indirect lighting from other emissive volumes when the [Volume Ray Depth](#) is above 0.

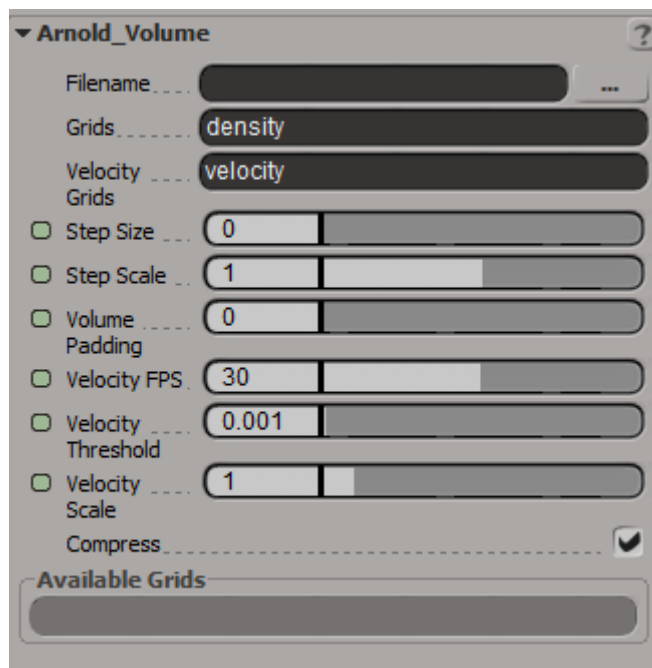


Custom AOVs are not currently supported with volumes.

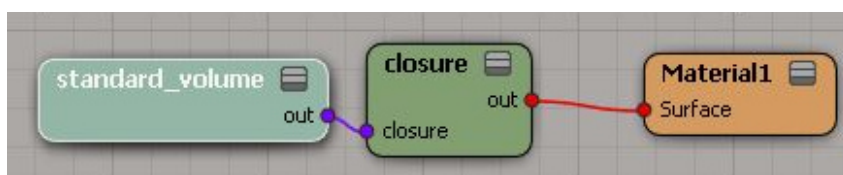
In SItOA, you can convert any shape into a volume placeholder, by applying a Volume property to it. The object is exported as an Arnold•volume•node.

To apply it, select an object and choose either:

- Get->Property->Arnold Properties->Volume
- Arnold->Properties->Volume
- SItOA_AddVolumeProperties (script editor).



i To reduce the amount of noise in a volume, you must ensure that the lights in your scene have enough 'volume samples'.



Standard Volume shader assigned to Volume (via Closure shader)

On selecting a new file, the read-only Available Grids •info field is populated with the list of available grids contained in the file. The list is also logged in the script editor, and can be queried at any time with the [SITOA_OpenVdbGrids](#) command.

Grids

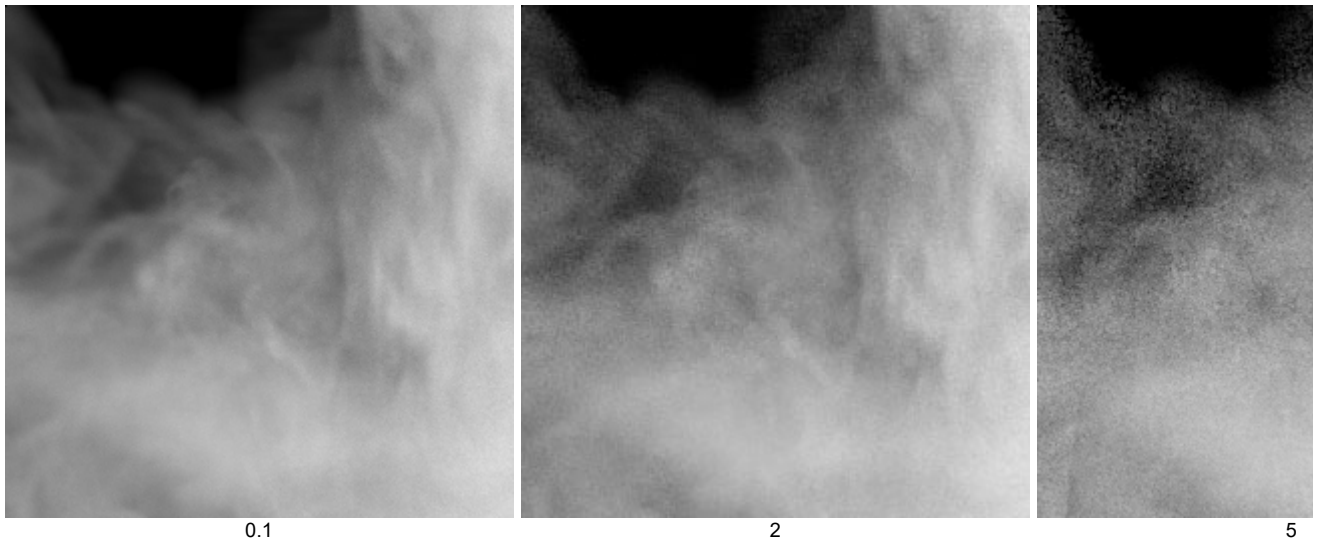
A list of OpenVDB grids to read and make available as channels in the volume shading context. Examples of Grid names are density, fuel, heat, temperature or velocity.

Velocity Grids

Either 1 vector grid (eg. •v• or •vel) or 3 float grids (eg. •vel.x., •vel.y., •vel.z) representing the velocity field, to be used for motion blur. No motion blur will occur if an invalid combination of grids or if no grids are specified. Note that all velocity grids declared here are also available as channels in the shading context.

Step Size

Sets the size for sampling inside the volume. •It should be small enough to capture the smallest feature in the volume (aka the Nyquist limit). If the size is too large, aliasing, bias or banding will appear. Smaller (and therefore more accurate) sizes will increase render times while larger sizes will render more quickly. •



 More information about volume Step Size can be found [here](#).

Step Scale

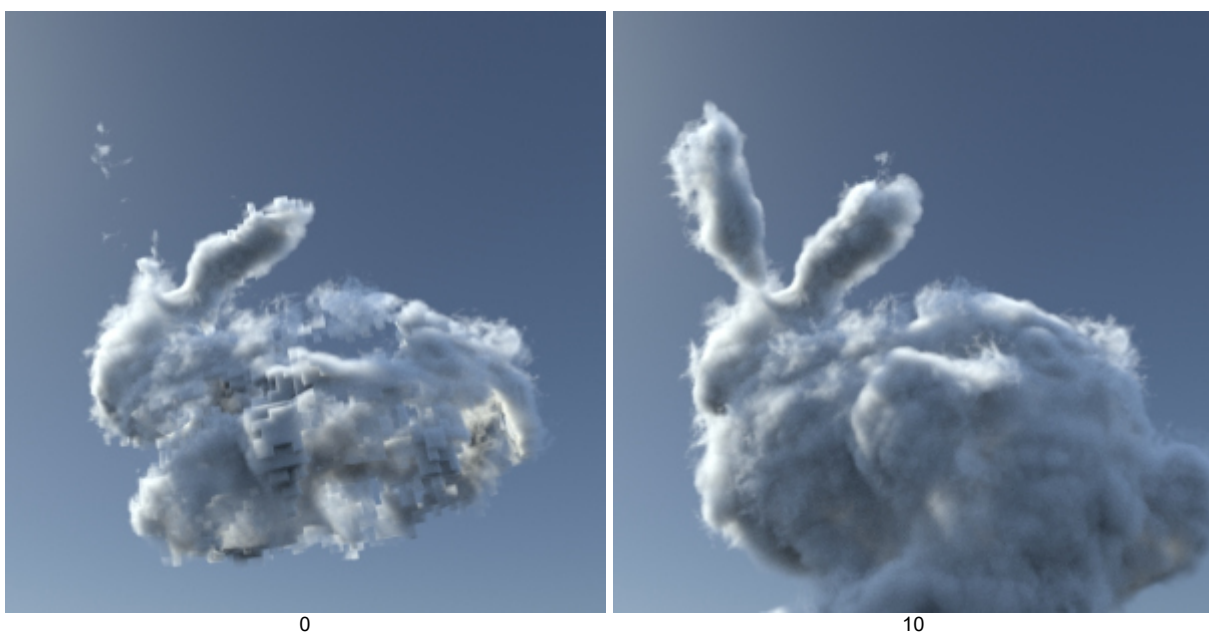
A scaling factor applied to the step size, mostly useful when the Volume Step is set to Automatic, to modulate the automatic value. For low frequency, smoke-like volumes, a value of 4 is a reasonable starting point.

Volume Padding

Enlarge the volume by `Padding`. This is useful when displacing a volume with a noise for example. Note that this slack applies not only to the bounding box but also to the voxels used for `ray extents` inside the bounding box. When velocity motion blur is enabled, some extra padding is computed, based on the maximum value of the velocity field, and the maximum of the velocity padding and user padding is applied. When `padding` exceeds the bounding box maximum dimension, the `ray extents` acceleration makes little sense and is disabled, so you should set the padding as tightly as possible.

•

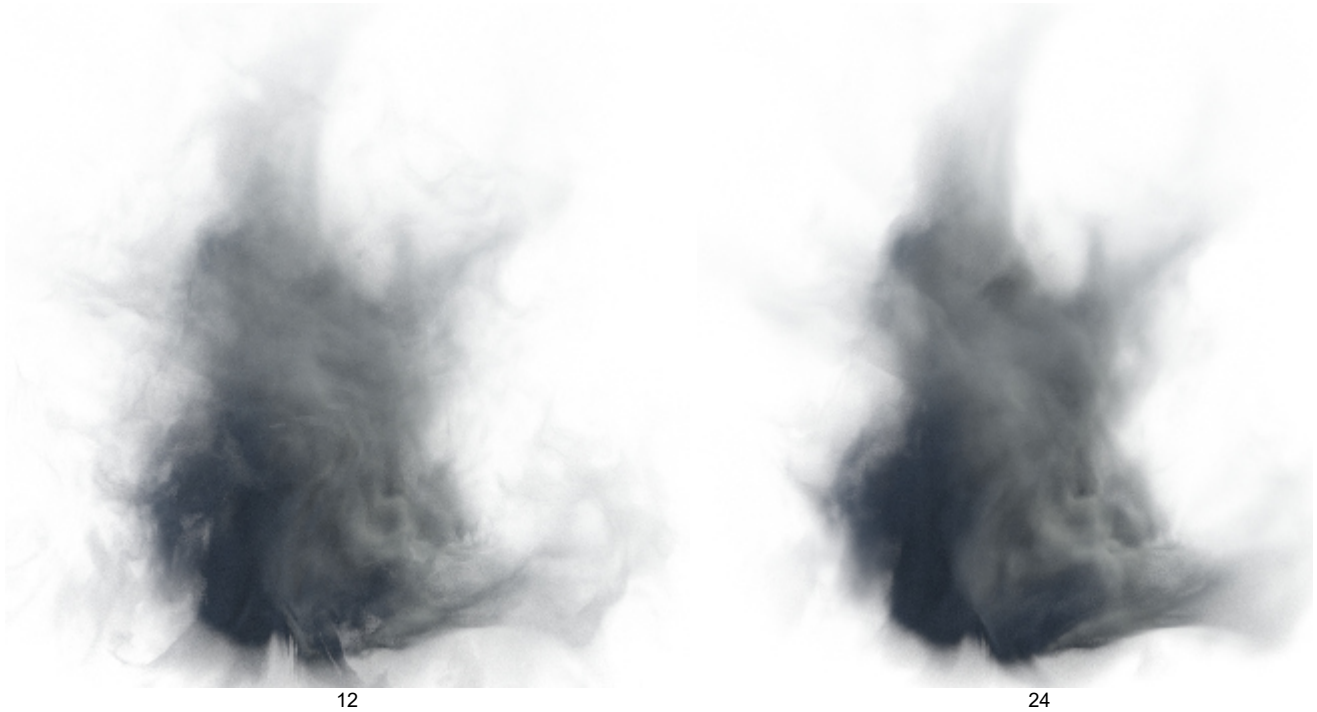
 For best performance, this should be set as low as possible, while still covering the maximum displacement amount.



The amount of padding to add to the volume bounds for displacement

Velocity• FPS

Sets the frames per second velocity value.

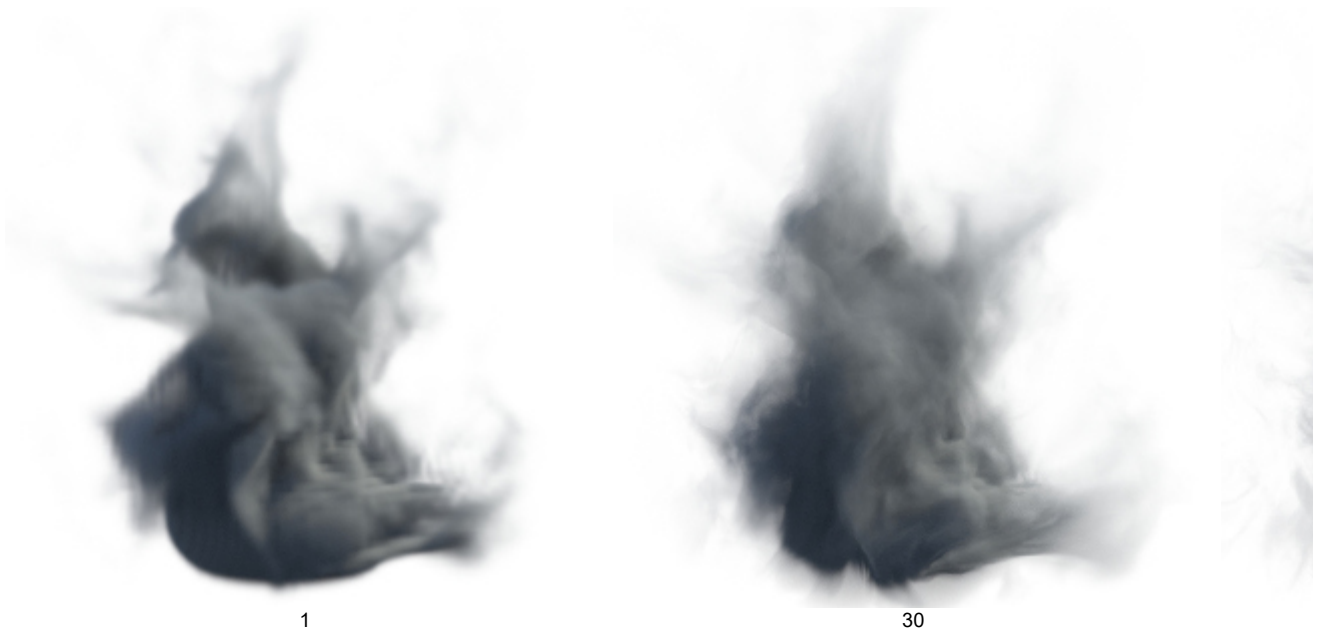


Velocity T hreshold

Controls filtering of noisy velocities resulting in the faster rendering of motion blur from physics simulations. The default value of 0.001 should have little to no visual impact. Setting it to zero disables filtering entirely.*

Velocity Scale

A scale factor for the velocity field. A value of 0 disables motion blur.



Compress

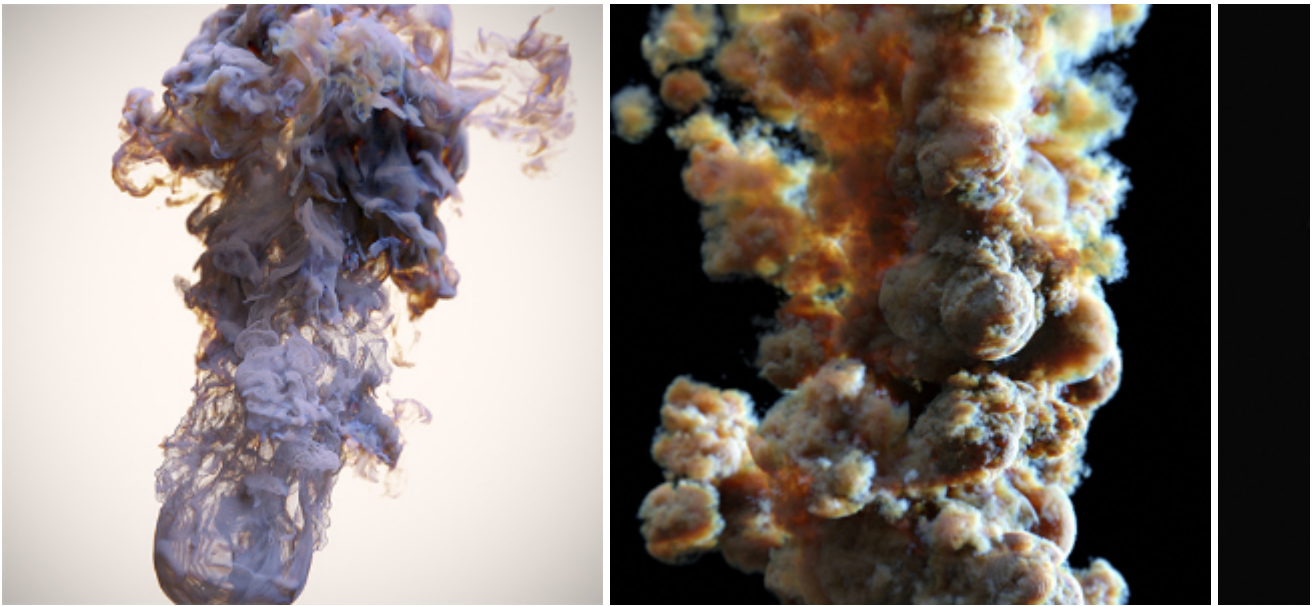
Compress grids to reduce memory usage, enabled by default. This is a lossy compression, however, in most cases renders will be almost the same. If there is a problem, however, this parameter can be used to disable lossy compression.

-
-



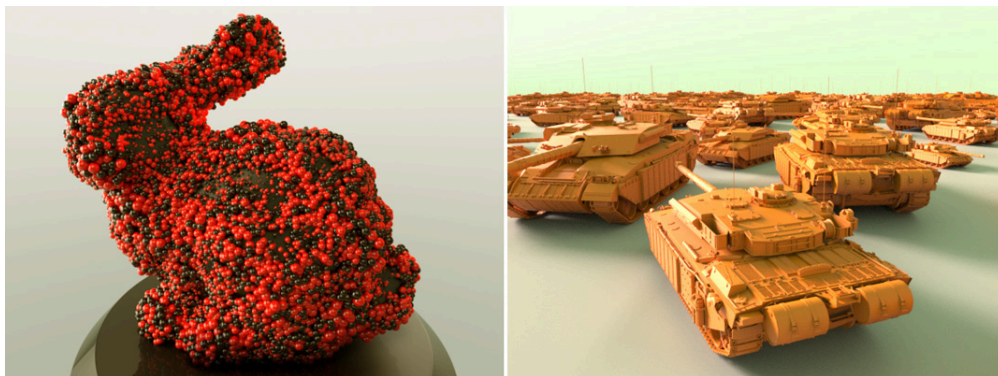
If you need to further edit the result of the reading of a channel, for instance, to color correct it, there are the [Volume Sample RGB](#) and [Volume Sample Float](#) shaders.

-
-



Further examples of volumes

ICE



.....ICE examples: a bunny model covered with particles using the sphere primitive (left) and instanced tanks (right)

•

SltoA provides support for ICE point clouds and strands. Attributes passing, motion blur and shape instancing on both points and strands are also supported.

- [Points](#)
- [Strands](#)
- [Instance Shapes](#)
- [Instancing Lights](#)
- [Instancing Stand-ins](#)
- [ICE Motion Blur](#)

Points

For shapes of type point, disc, rectangle, sphere, box, cylinder and cone, the points are exported using the respective native Arnold nodes.

For example, Arnold has a points node, whose mode can be either disk, sphere, quad. If the ICE shape is Points, we export a points node in disk mode, etc.

Any ICE attribute can be read in the rendertree with the standard attribute shaders.

The following is a table showing how each ICE shape is translated into Arnold.

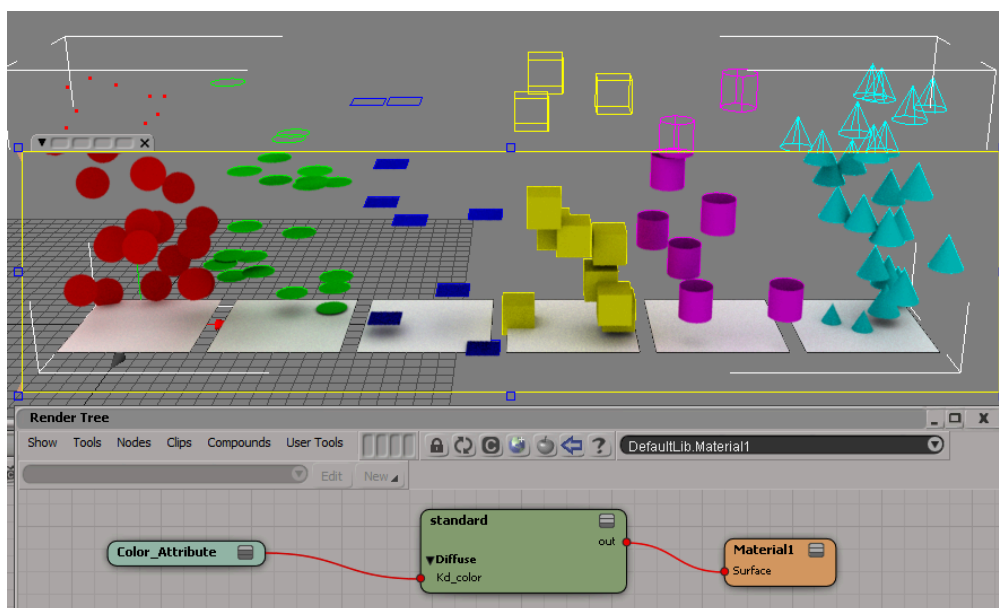
•

ICE SHAPE	ARNOLD SHAPE
Point	points (mode "disk")
Disc	disk
Rectangle	polymesh, one quad per point
Box	polymesh, 6 quad per point
Sphere	points (mode "sphere")
Cylinder	cylinder
Cone	cone
Instance Shape	ginstance

•

The Segment, Capsule, and Blob shapes are not supported. SltoA skips them and they are not translated into the Arnold scene source.

•



Strands

Simple strands are rendered using Arnold's curves primitive.

The following attributes are used, if available:

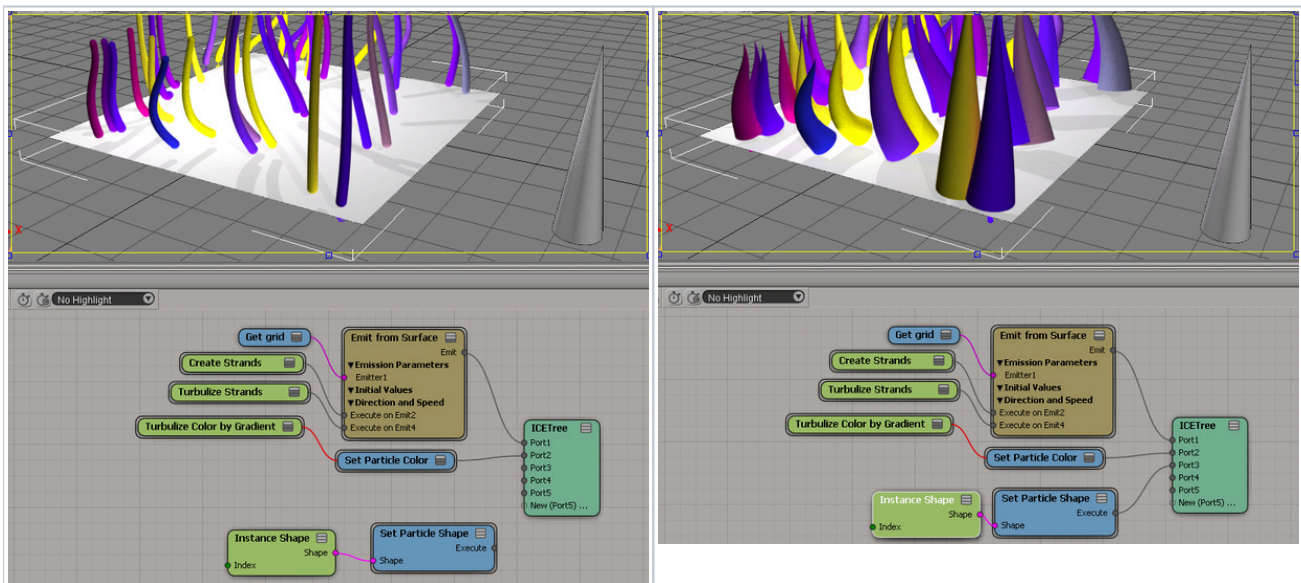
- StrandPosition.
- Size / StrandSize for constant / varying thickness.
- Orientation / StrandOrientation for constant / varying orientation.

If instead, the strand has some instance shape connected, then the shape is always bent along the strand, and a single polymesh object (or group if the instance shape is a model) is created for each strand.

You can instantiate meshes and models.

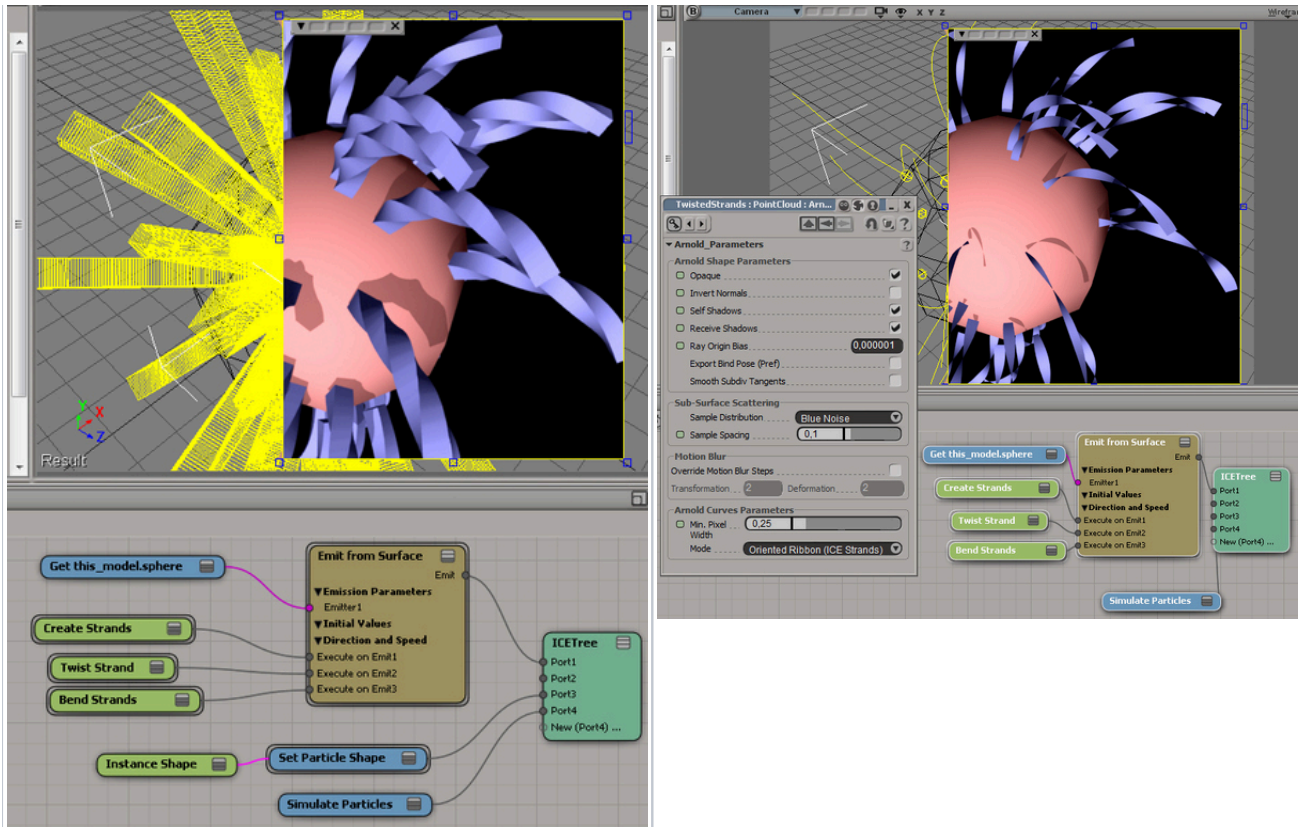
The per-point attributes (like the color in the case below) are exported as constant data for each instantiated object.

The per-strand-point attributes are exported as varying data (one value per curves point) only for plain curves nodes.



The strand orientation attribute, if available, is used:

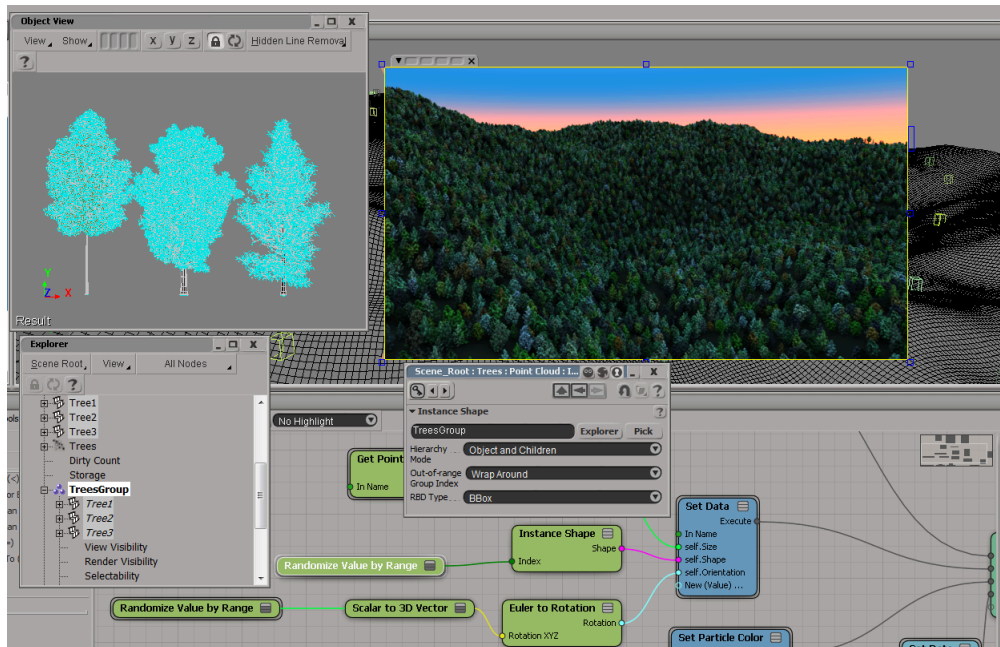
- In the case of a shape, to twirl the shape around the strand.
- In the case of a plain curves node, to set the orientation of the curve. To do so, apply an Arnold Parameters property to the point cloud, and set the Curves Mode to Oriented Ribbon (ICE Strands).



 More information can be found here: [Hair / curve-parameters](#), [Hair Tutorial](#).

Instance Shapes

Instance shapes are exported as ginstance nodes, one per point, pointing to the master object, that is exported only once. Instance animation time is also supported. Both single objects, hierarchies, and models can be shape instanced. The materials of the master shapes are used for the whole cloud.



Instancing Lights

SltoA supports ICE instanced lights. To pass data to the lights, we rely on specific attributes' names. Any attribute whose name starts with "ArnoldLight " (case insensitive) is treated as a candidate for setting a given parameter on the instanced lights.

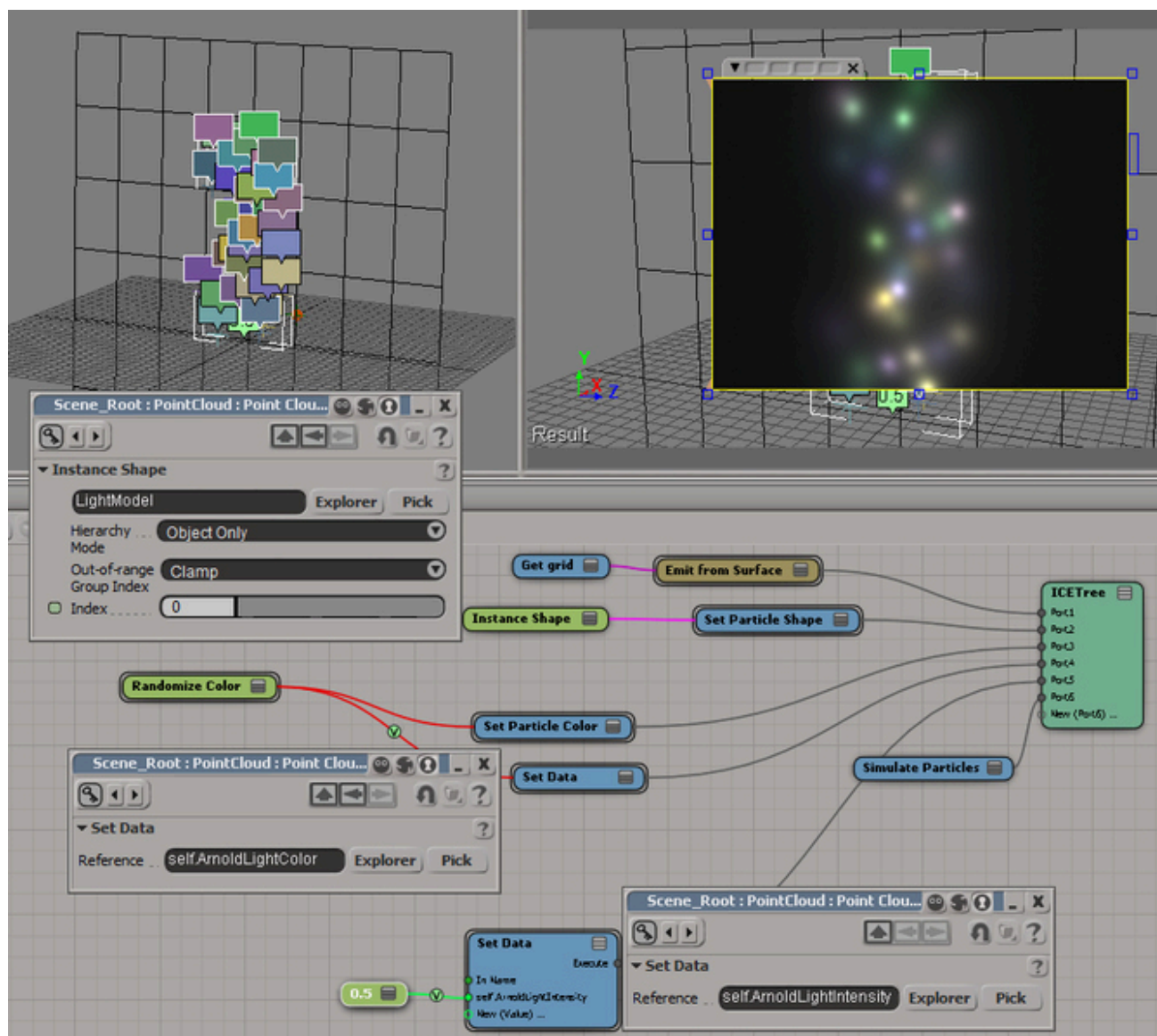
What follows "ArnoldLight" is used as the parameter name to be set. The parameter names used are not those of the light shader. The parameter names refer to the Arnold node used for that light.

For instance, all the light nodes in Arnold have parameters called "color" and "intensity".

So, you simply have to set two attributes called `ArnoldLightColor` and `ArnoldLightIntensity`, and they will be automatically pushed as the value of the corresponding parameters of the Arnold lights. There is no need to do anything in the light rendertree.

SltoA will also convert between different types. For instance, in ICE you can declare `ArnoldLightColor` as a scalar attribute, and SltoA will assign the same scalar value to the light color's R, G, B.

i Note that, not being pulled by any rendertree node, you must force ICE to export the `ArnoldLight*` attributes, for instance by displaying their values in the viewport, as showing below. Another (better) method is to make the ICE attributes pass through a Log Value. This seems to work also when the Log flag is disabled.



•

You can set light parameters of type point, vector, color, scalar, integer and boolean.

To get the full list of the parameters for a given light type, use "kick -info<node>". For instance, for a point light, "kick -info point_light" returns this:

```
Type.....Name.....Default
-----
POINT[].....position.....0,0,0
FLOAT.....radius.....0
ENUM.....decay_type.....quadratic
BOOL.....affect_volumetrics.....true
BOOL.....cast_volumetric_shadows.....true
NODE.....volume_density.....(null)
MATRIX[].....matrix
RGB.....color.....1,1,1
FLOAT.....intensity.....1
FLOAT.....exposure.....0
BOOL.....cast_shadows.....true
FLOAT.....shadow_density.....1
RGB.....shadow_color.....0,0,0
INT.....samples.....1
...
```

•

This tells you that, for instance, if you want to affect the shadow color, the correct attribute name to set in ICE will be ArnoldLightShadow_Color (again, case insensitive).

•

•

•

Instancing Stand-ins

Similarly, SItOA allows for stand-ins driven by ICE attributes.

You should proceed as usual when instancing stand-ins on a pointcloud. Then, if the standin property has the string "ArnoldProcedural" set for the path and/or the data, then it will be exported by ICE as an Arnold procedural node. In ICE, any attribute starting with "ArnoldProcedural" is a candidate for setting the corresponding procedural parameter.

In the example shown below, we set ArnoldProceduralDso to point to an .ass file (depending if the point id is even or odd) and set ArnoldProceduralData to void. "Dso" and "Data" are the names of the parameters of the native Arnold procedural node. We can then set any of the procedural parameters, as long as we use the original name of the attribute (kick -info procedural) after ArnoldProcedural. For instance, ArnoldProceduralSelf_Shadows, etc., the same way we did for lights.

If Path is a valid dll (meaning that Data must be set equal to "ArnoldProcedural" to be managed by ICE), then the dll path is used, and you can just set ArnoldProceduralData in ICE to have the same dll using different data for different points.

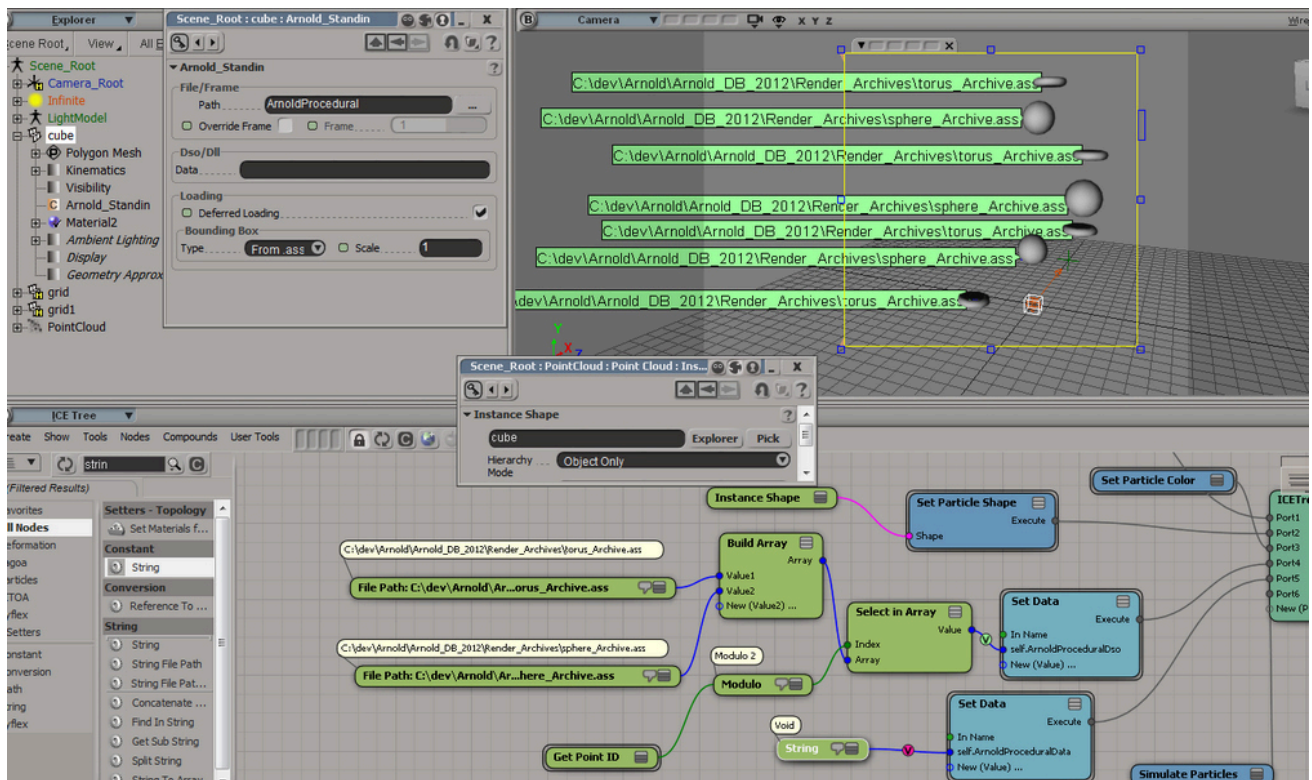
If Deferred Loading is on, then the stand-in's bounding box is used (the assoc file is not used, even though selected in the standin property). If you need to set or overwrite the min and max boundaries (for instance if using your own custom procedural), you can push two extra 3D vector attributes named ArnoldProceduralMin and ArnoldProceduralMax.



Use a Log Value node in your ICE tree to force ICE to evaluate the ArnoldProcedural attributes. If you don't force the evaluation of the attributes, the attribute values won't be available at render time.

Note that you can disable logging in the Log Value node, and ICE will still evaluate the attributes.

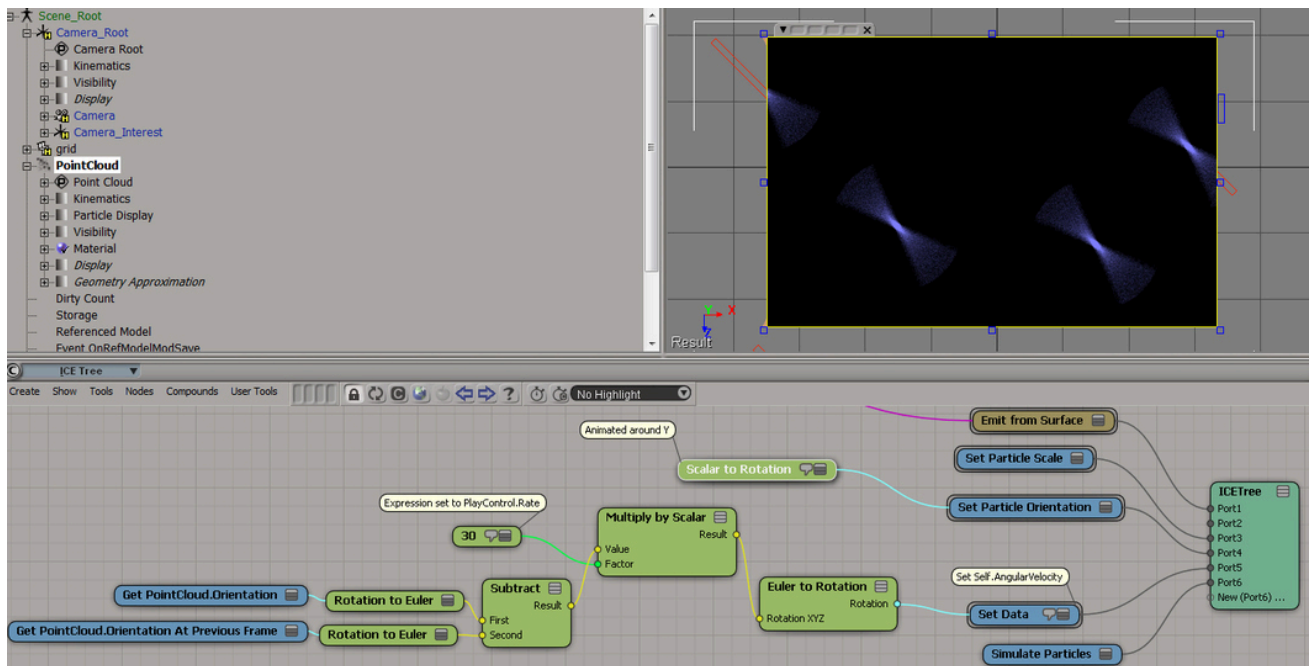
Also, while rendering, you may get some warnings from Arnold about names being reused; these warnings can be ignored.



ICE Motion Blur

To have motion blur on points or strands, SltOA uses the following attributes:

- PointVelocity for linear transformation motion blur.
- AngularVelocity for rotational motion blur. Note that this attribute does not come for free, you have to set it in the ice tree as shown in the snapshot: get the rotation, subtract it from the one at the previous frame, and multiply it by the frame rate. This way you have the angular velocity (radians per second). Finally, set this data as Self.AngularVelocity.
- StrandVelocity for linear deformation motion blur on strands.



For cases where sub-frame motion blur is needed, you must enable Exact ICE Blur in the rendering options. If enabled, SltOA does not require the velocity attributes to be defined for points and strands. Instead, the point cloud geometry is evaluated at all the motion blur times, so giving much better results. But there is a price to pay since it requires evaluating the geometry several times, which can be quite expensive. And, there are two limitations:

- The point cloud must be cached to disk with a number of subframes per frame at least equal to the number of deformation motion blur keys. See the [sub-frame motion blur](#) page for details.
- It does not work for point clouds with a varying number of points per frame (for instance, basic emissions). It will be automatically disabled, but you should do it yourself in advance to save time to the exporter.

These are the steps to follow to correctly cache a simulation on disk and read it back:

- Open Cache Manager (View->Animation>Cache Manager)
- Go to the Write Tab
- Add point cloud under Objects (use the Add Selection or Pick buttons)
- Hit the Write Cache button
- Go to the Read Tab
- Under Add to Cache List, select the cached data generated previously
- Under Apply to Objects, Add point cloud to the list
- Select the cached data under Cache List and point cloud under Apply to Objects
- Hit Apply All



See the [rendering options](#) for details on tuning the motion blur settings.



.

Troubleshooting sub-frame motion blur

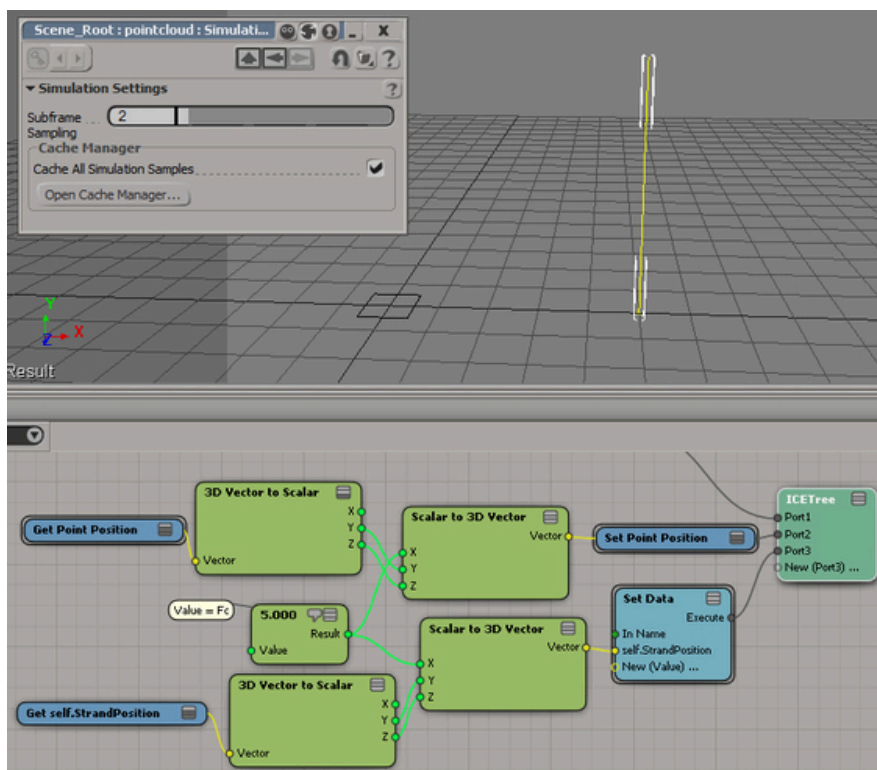
Even though you have cached the simulation as described in the previous page, you may observe Arnold rendering the motion-blurred pointcloud misplaced in time, and/or with some kind of a banding effect.

This page explains why Exact ICE Blur doesn't (always) produce the expected result, and, in any case, how to tune the caching and motion blur parameters to fix the problem.

To begin, we have to understand the relation between a cached sequence (Exact ICE Blur can work correctly only if Softimage is in cache-reading mode) and the frame time at which the geometry is pulled from inside an operator (in our case, the SltoA ICE module).

The easiest way is by an example. I have created a pointcloud made of a single, straight, strand. The strand's points have their X coordinates set equal to the current frame time by an expression ($\text{Value} = \text{Fc}$). So, for instance, at frame 5, all the strand's points have their $X=5$.

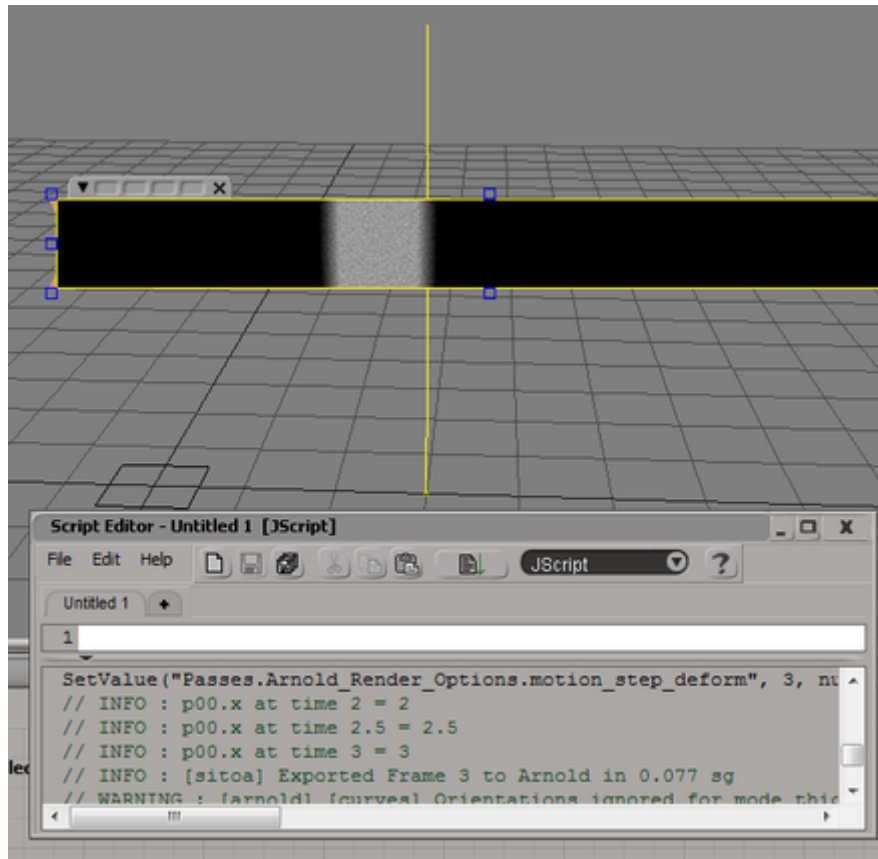
We now cache the sequence on file, with Subframe Sampling set to 2. This means that we'll have on file the strands saved at frame times 1, 1.5, 2, 2.5, etc.



In SltoA, I edited the code, so that, upon pulling the geometry, the X coordinate of the first point of the strand is logged. So now I switch in cache-reading mode, and I start playing with the motion blur settings.

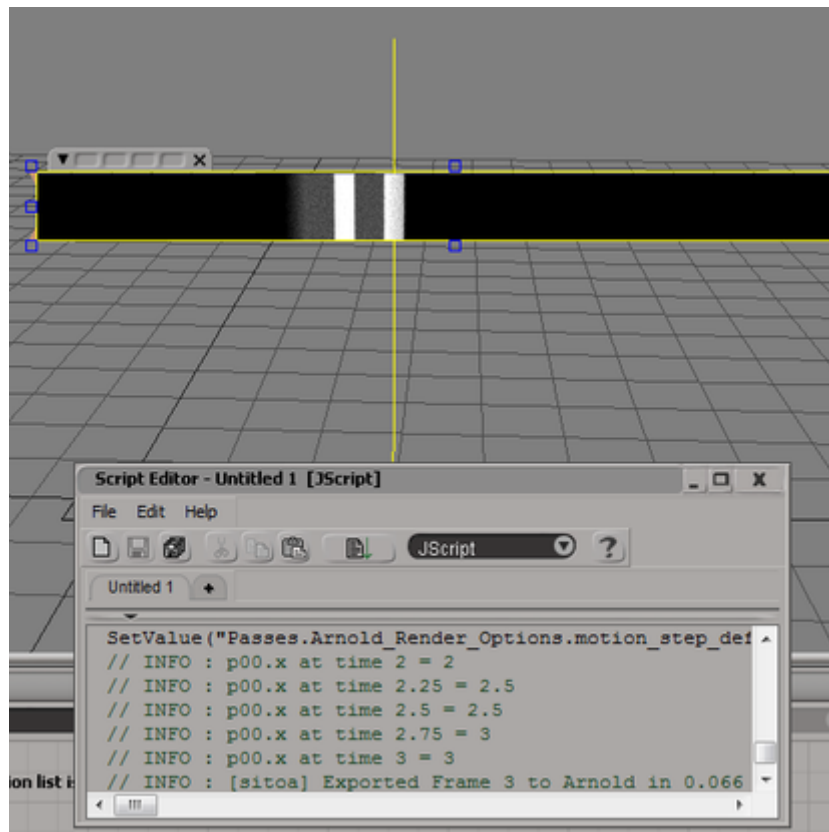
I go to frame 3 and set Lenth= 1, Position = End of Frame .

With 3 motion steps, SltoA asks the geometry at frame time 2, 2.5, 3 and logs the following:



This is a perfect match between the cached times and the motion blur times, so the render performs perfectly.

However, let's now increase the number of keys to 5.



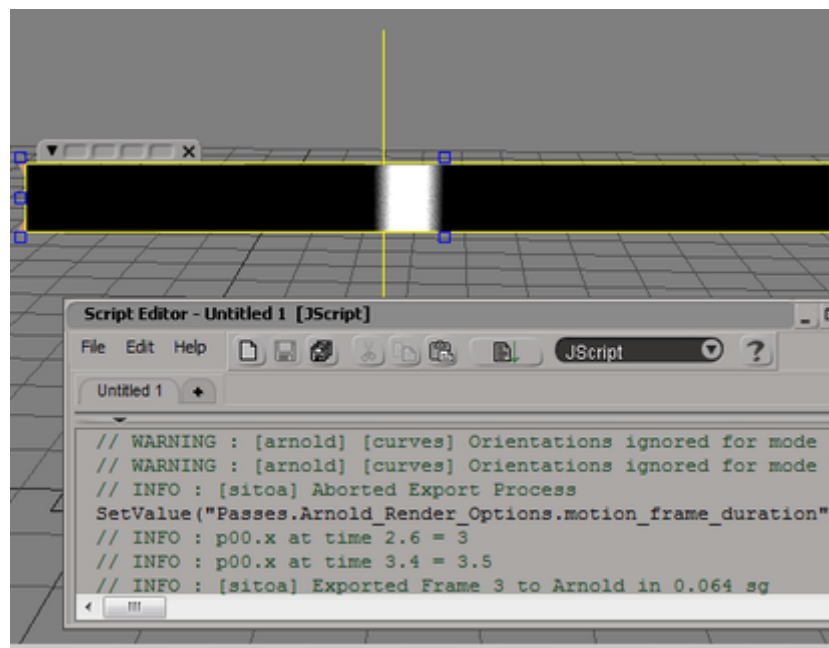
Here comes one of the problems: when SItOA asks for the geometry at a frame time other than one saved to disk, Softimage returns the geometry at the "ceiling" time.

For time=2.25, Softimage does not find on disk a cache file for it and returns the one at time=2.5 (remember that x=2.5 means "the x position at time 2.5").

The same happens for time 2.75, which gets the geometry at time 3.0.

So, you have what we can call "biased" motion blur because 2 keys out of 5 are the same, and Arnold weights the blur accordingly.

And, things can also get worst. Let's set Position = Center of Frame, Deformation Keys = 2, Length = 0.8



•

You would normally expect to have the motion blur centered around the strand, but instead, it's all shifted away (which is really ugly, if you have some skin under the hair)

The reason is the same as described above. At the time of the first key ($3 - 0.4 = 2.6$), Softimage returns the geometry at frame time 3, so the result is all shifted ahead in time.

So, how do we fix this?

1. Set the frame duration so that the motion blur times are easy to compute. Avoid 0.49 when you can use 0.5, and let this number be easily dividable by the number of keys. So, for instance, if you plan to use 7 motion blur keys, use 0.7 (or a multiple of 0.7) as your motion blur length because the key time for frame 3 will be:

- Start On Frame: ••• 3.0 3.1 3.2 3.3 3.4 3.5 3.6
- Center On Frame: • 2.7 2.8 2.9 3.0 3.1 3.2 3.3
- End On Frame: ••••• 2.4 2.5 2.6 2.7 2.8 2.9 3.0

2. Cache the simulation so that no more than one cached subframe falls into 2 or more motion blur times. So, in our example, you should cache with Subframe Sampling = 10

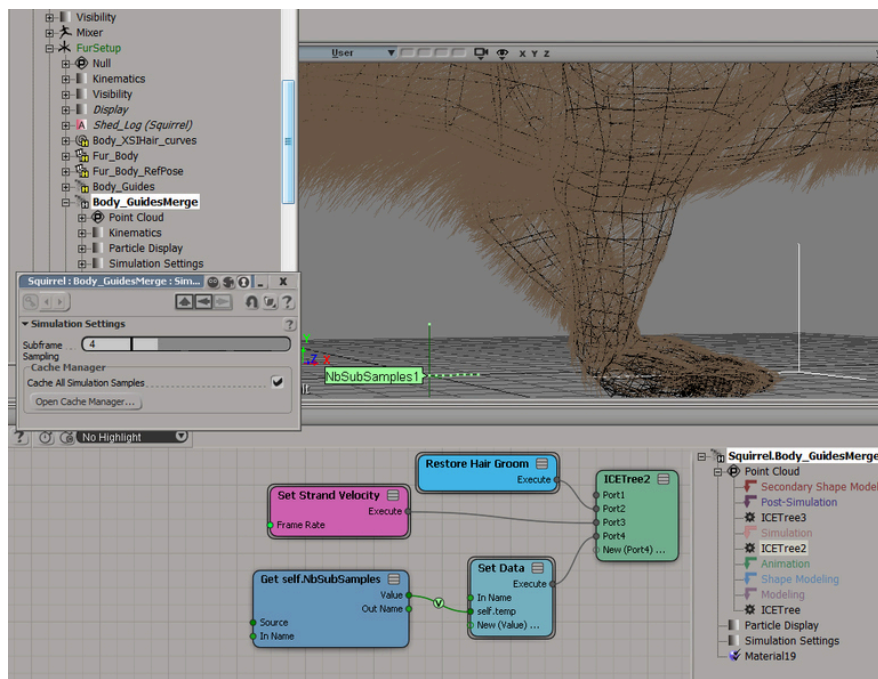
So, to recap, we can state the following:

1. Given a cached sequence, increasing the number of motion blur keys does not necessarily mean increasing the motion blur quality.
2. The number of subframe sampling steps must be equal or bigger than the number of motion blur keys
3. The subframe sampling step must be chosen so that one cached frame exists for one and only one motion blur key

•

One final note to describe another possible source of troubles. I had a scene that refused to return the correct strands position, although I had set the caching and motion blur parameters as described above. The Subframe Sampling was set to 4, but then I discovered that displaying the value in the viewport, it would show 1!

It was probably a scene initially simulated in Softimage 2012 and then imported in 2013, where the sampling controls were modified, and some changes are needed when migrating the scene, as described in the [online doc](#) under the "Updating Scenes and Compounds to Use Subframe Sampling" section. By following these steps, I had my attribute showing 4, and the cache writing/reading finally performing correctly.



•

•

Volumes



With the advent of the new volume API, this workflow is now obsolete. Use the [Arnold Volume Property](#) instead.

•

To turn an object (of polymesh or point cloud type) into a volume container, you must apply an Arnold_Parameters property to it, and set its Step Size to a positive value. If Step Size is 0, then the object is treated as a solid object and shaded with a surface shader as usual. But if Step Size is greater than 0, its actual type and geometry are ignored, and the object:

In the case of a mesh object, it's exported as an Arnold box, its extent defined by the object's bounding box multiplied by some extra scale.

In the case of an ICE point cloud, it's exported depending on the Container type:

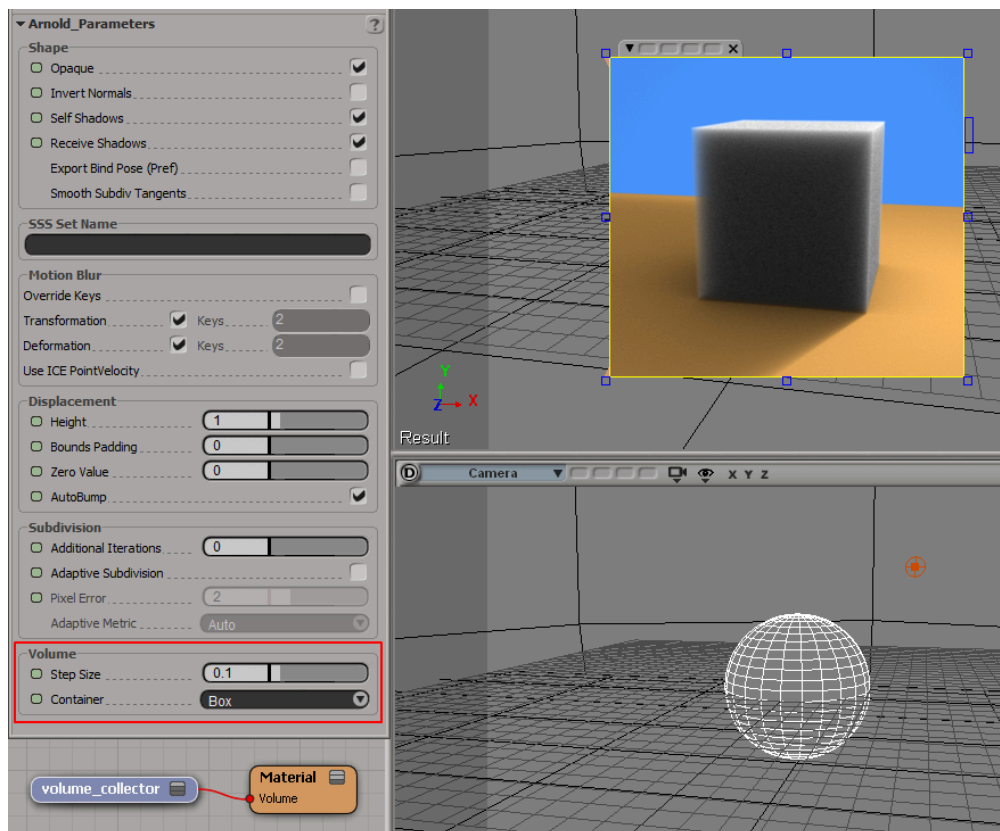
- Box : as an Arnold box as well
- Points : as an Arnold points node in sphere mode, the position, and size of each point defined by the PointPosition and Size attributes

Step Size

The integration step for the volume ray marching algorithm. Smaller values give better results with longer rendering times (big news!).

You must assign a volume shader by connecting one to the Volume port of the object's material. No surface shader is required.

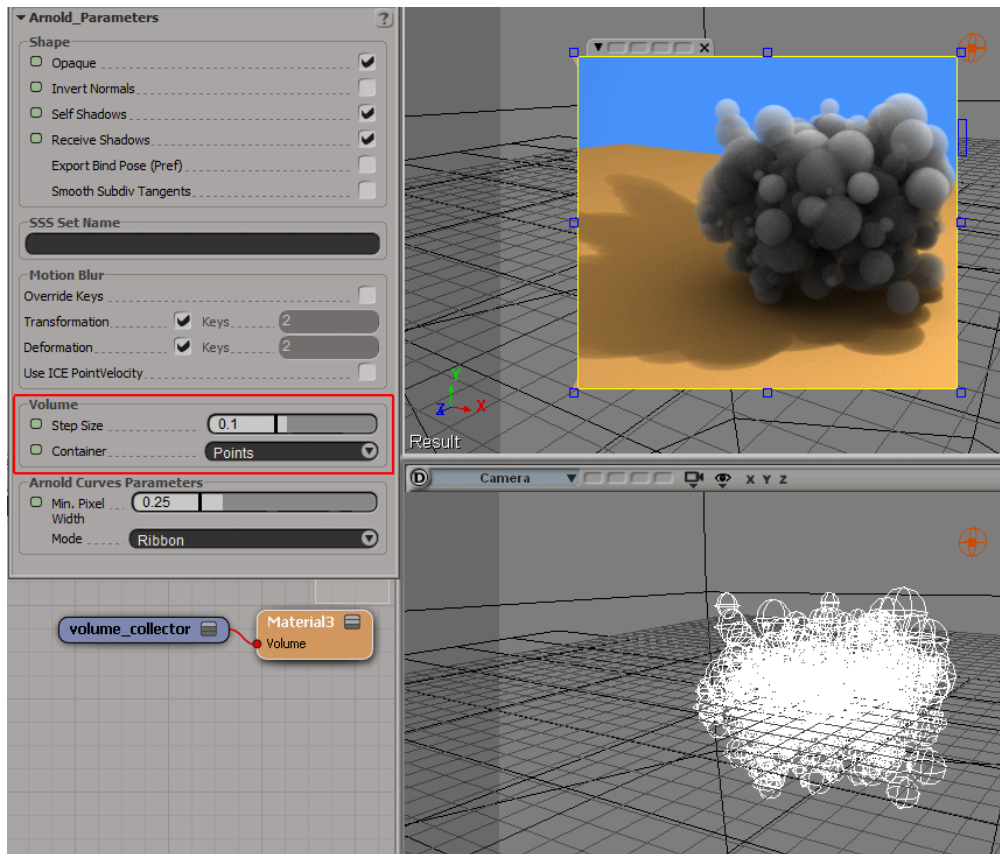
For example, if you set a positive Step Size for a sphere, and you assign a volume_collector shader to it, you get a similar result



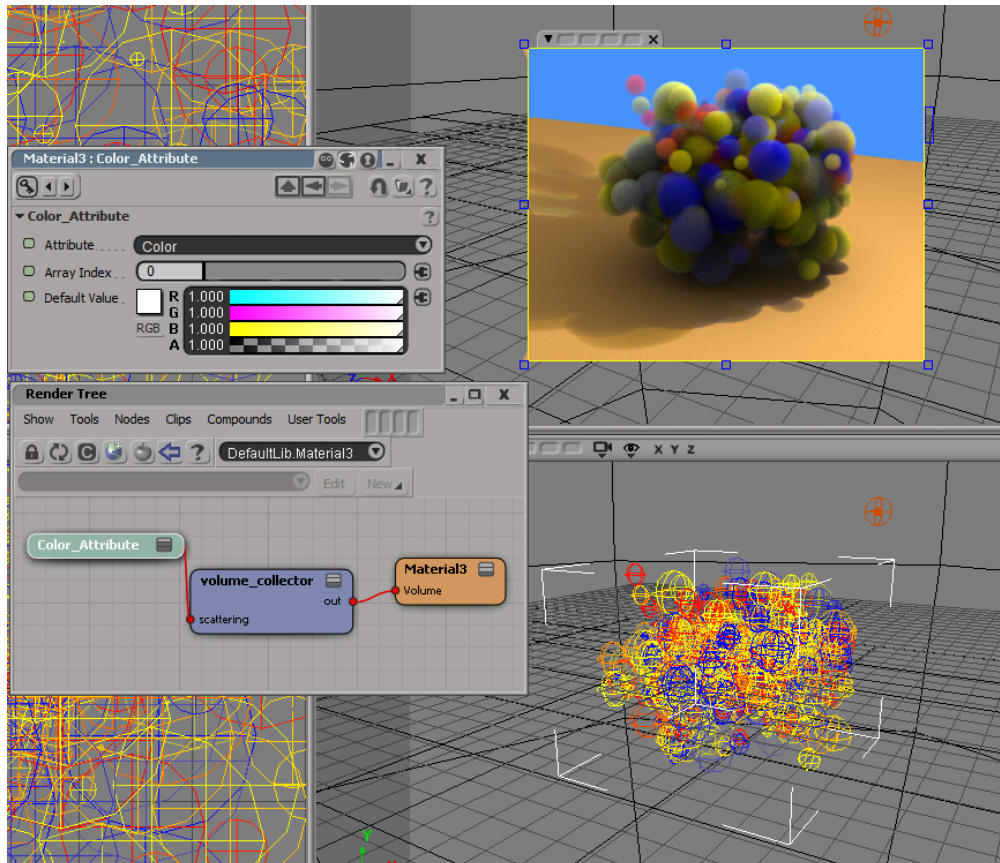
Writing out the sphere to .ass, you'll find it exported as a box object, with its min/max equal to the sphere's bounding box scaled up by 10%. The volume shader is now the box shader, and the step_size value correctly set:

```
box
{
  name sphere.SItoA.1000
  min -4.4 -4.4 -4.4
  max 4.4 4.4 4.4
  step_size 0.1
  ...
  shader "Sources.Materials.DefaultLib.Material.volume_collector.
SItoA.1000.1"
}
```

In case of an ICE pointcloud, if you let the Container type be Points, you turn each point into a volume container:



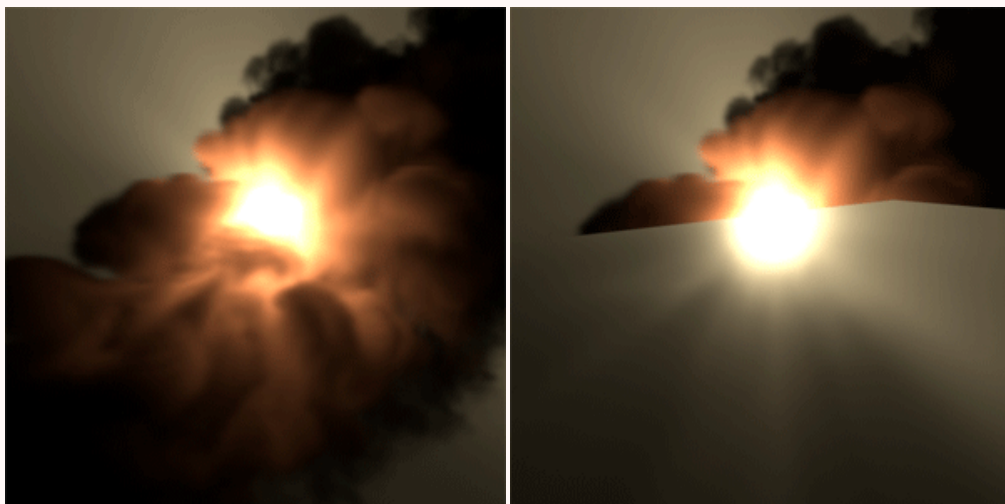
In Points mode, you can pull any ICE attribute in the volume render tree. For example, you can use the points Color to drive each point's scattering value



i To reduce the amount of noise in a volume, you must ensure that the lights in your scene have enough 'volume samples'.

! Volumes and Volume Scattering •

Currently, 'Volume Scattering' does not compose well against volumes. This is because atmosphere's return a single flat result that is opacity mapped on top of whatever is in the background of the pixel.



Left: floor plane geometry hidden. Right: floor plane geometry visible. Fluid renders incorrectly.

Click images above to view animation.



.



Writing volume shaders

This page describes the options to exchange data between a Softimage object and its volume shader. For more details on the Arnold volume rendering API, see the [Arnold wiki page](#).

A volume shader gets called as many times as required by the ray as it marches through the media, with the base marching step defined by Step Size.

Some shaders may be data independent, meaning that they compute the absorption emission and scattering values out of a given function (say a noise), and call the respective API calls.

In many case though, shaders will depend on some data set, for instance a voxel grid or a particle cloud.

For the first family of shaders, there is no need to exchange data between the containing object and the volume shader. The values to be compute will usually depend on sg->P, i.e., for volume shaders, the current raymarched point.

For data-dependent shaders, there are two ways to pass data, depending on the object type (polymesh/pointcloud).

Polymeshes

If the original volume object is a Softimage polymesh, no extra data are exported with the box.

The shader writer should provide the optional dataset either by a file (being the file path the main shader's parameter) or by a [User Data Blob](#).

As described in the blob documentation, the shader writer must provide a way to write the data into the blob, and then retrieve it in the shader, by looking up the user data attached to the shader's owner (sg->Op).

Pointclouds

For ICE pointclouds, you can go with blobs as well, but the best way is to exploit ICE attributes.

Together with the pointcloud box, SlttoA exports a sub-set of the ICE attributes as user data of the box. So, a volume shader will be able to retrieve those attributes, by looking up the user data of the shader's owner.

So, it works similarly to the blob data, but

1. The data are written automatically by SlttoA into the box user data
2. There is no data conversion needed. While the blob passes a raw array of bytes, the ICE attributes structure and data types are preserved when writing them to Arnold.

.

A shader must declare the list of attributes he wants to receive, by a special statement line in the output section of its spdl.

Some of the BA shaders (currently unsupported) coming with Softimage have such a line. An example is the following

```
PropertySet "particle_density"
{
    Parameter "out" output
    {
        guid = "{4C6879FF-7EC8-11D0-8E3B-00A0C90640EC}";
        type = color;
        flags = 1;
        ICEattribute = "PointPosition,Color,Size,ArnoldVolume0";
    }
    ...
}
```

Meaning that the shader needs PointPosition, Color, Size, ArnoldVolume0 to work properly.

The attributes that you can declare in the shaders are the following: Age, AgeLimit, Color, Id, PointVelocity, Size, StrandColor, StrandPosition, StrandSize, SizeVelocity .

Then, there are 10 extra names reserved: ArnoldVolume0, ArnoldVolume1, ... ArnoldVolume9 , that you can use to pass further attributes.

Say for example you need to pass StrandCount (the number of points per strand). What you can do is create a new ICE attribute called ArnoldVolume0, and set it equal (using a Set Data node) to the StrandCount array.

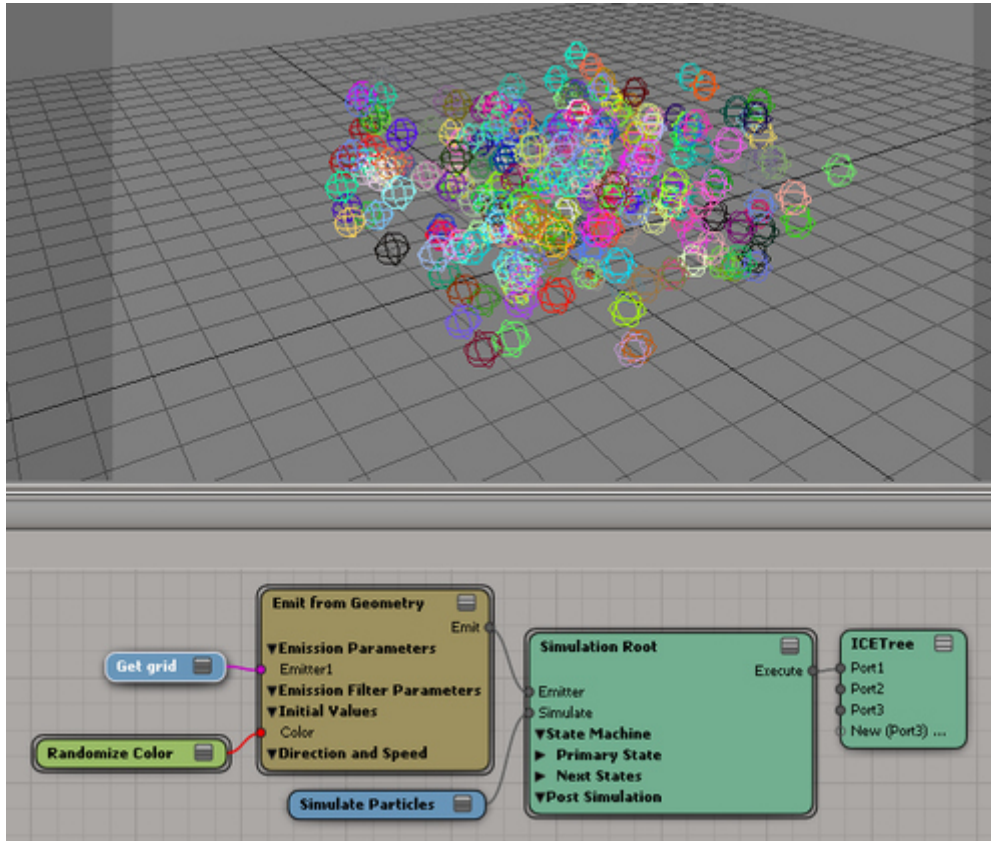
Then, if ArnoldVolume0 is declared in the spdl output, as shown above, you can retrieve the array in the shader.

The following attribute types are supported: boolean, long, float, vector, rgba, matrix4, rotation, string , and strand arrays of boolean, long, float, vector, rgba, matrix4, rotation .

Examples

Let's see an example, using a very stupid shader called `particle_density`, implemented by [particle_density.cpp](#) and [particle_density.spdl](#). This shader loops the `PointPosition` (the positions of the points). If the distance between at least one of the points and the current volume point (`sg>Ro`) is less than a given threshold (the `Max Radius` parameter), then the point's color is returned, and set as the volume scattering and attenuation value.

To start, we emit a few particles from a grid. The cloud has the needed attributes set by default (`PointPosition` and `Color`), and its `Step Size` set to 0.1.



If we want to check what happens at export time, we can dump an ascii .ass file, and under the pointcloud box we read:

```
...
declare PointPosition constant ARRAY VECTOR
PointPosition 254 1 VECTOR
  1.41966665 3.0957675 0.364348352 1.94903159 3.06433153
 -2.64184165 3.08448315 3.02823138 -1.34745932
...
declare ID constant ARRAY INT
ID 254 1 INT
  0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
 25 26 27 28 29
...
declare Color constant ARRAY RGBA
Color 254 1 RGBA
  0.410476238 0.636814773 0.896737099 1 0.00240939856 0.272344828
```



```
0.219851911 1 0.964304328 0.349832714 0.936426103 1
...
...
```

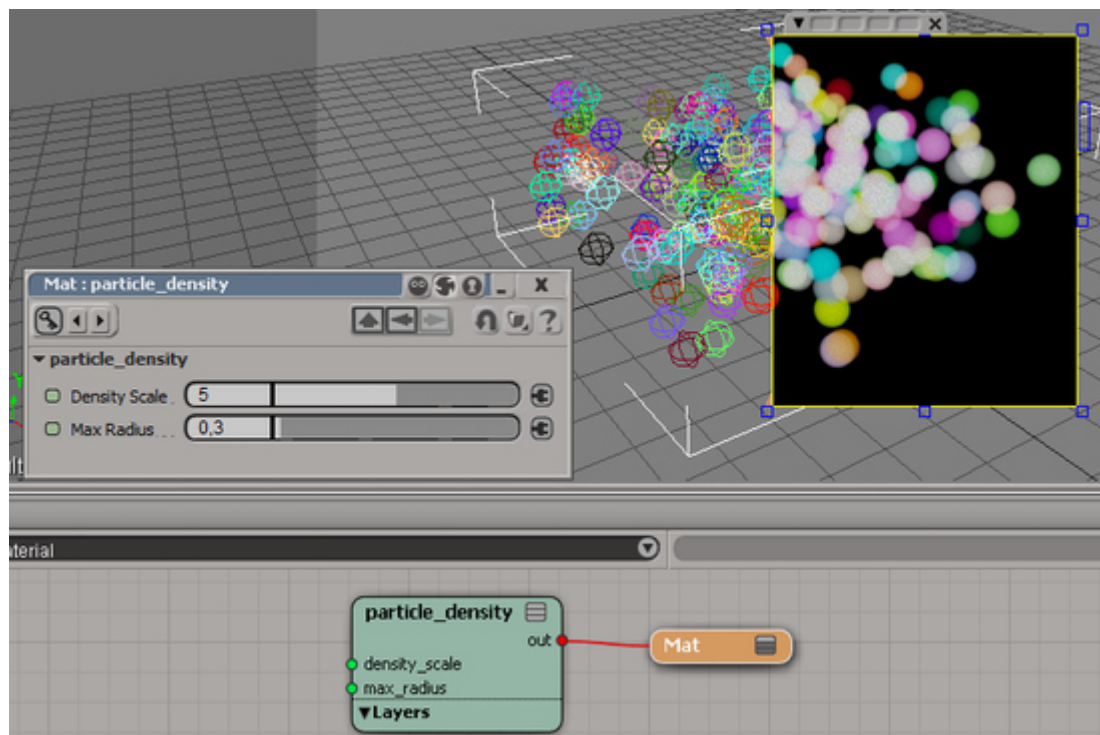
What the shader does is loading the attributes (only once) and storing them into the shader local data. This section reads `PointPosition`, and stores the points into a stl vector:

```
//PointPosition,Color
const AtUserParamEntry *paramEntry = AiNodeLookUpUserParameter
(node, "PointPosition");
if (paramEntry)
{
    AtArray *attrArray = AiNodeGetArray(node, "PointPosition");
    if (attrArray) // if the PointPosition user data exist, get the
values
    {
        m_position.resize(attrArray->nelements);
        for (unsigned int i=0; i<attrArray->nelements; i++)
        {
            AtPoint p = AiArrayGetVec(attrArray, i);
            // go to world coords, as PointPosition are in local
            AiM4PointByMatrixMult(&m_position[i], in_sg->M, &p);
        }
    }
}
```

Then, at evaluating time, the points vector is looked up. If at least one point is closer than `max_radius` to `sg->Ro`, its color is returned and used for the volume rendering calls.

```
if (data->LookupPoints(sg->Ro, AiShaderEvalParamFlt(p_max_radius),
color))
    color*= AiShaderEvalParamFlt(p_density_scale);
AiShaderGlobalsSetVolumeScattering(sg, color);
AiShaderGlobalsSetVolumeAttenuation(sg, color);
```

This is the result without shadows.



•
A further shader, `strand_density`, implemented by [strand_density.cpp](#) and [strand_density.spdl](#), does the same, but uses the strand position.

If you create some strands and export them to .ass, you'll read:

```
declare StrandPosition constant ARRAY ARRAY
  StrandPosition 8 1 ARRAY
  5 1 VECTOR 0.00297679752 7.21970749 -0.135526419 -0.0581556819
  6.89884853 -0.152862459 -0.0993935764 6.56537914 -0.157321781
  -0.127141297 6.25715494 -0.173184216 -0.141561687 6.00136375
  -0.194095552
  5 1 VECTOR -1.79802382 3.06814051 3.35425234 -1.77053165
  3.07318211 3.35395074 -1.73760593 3.07563972 3.35023117
  -1.71280038 3.08343673 3.34328961 -1.69526291 3.0968442 3.33205795
  ...
```

meaning that there are 8 strands, the first 2 made of 5 points. This shows how the strands arrays are exported as constant arrays of arrays, and each sub-array contains the data of a single strand.

The section of `strand_density` reading the points then becomes:

```
const AtUserParamEntry *paramEntry = AiNodeLookUpUserParameter
(node, "StrandPosition");
if (paramEntry) // if the StrandPosition user data exist, get the
values
{
  AtArray *attrArray = AiNodeGetArray(node, "StrandPosition");
  if (attrArray) // strandposition are stored as an array of
```



```

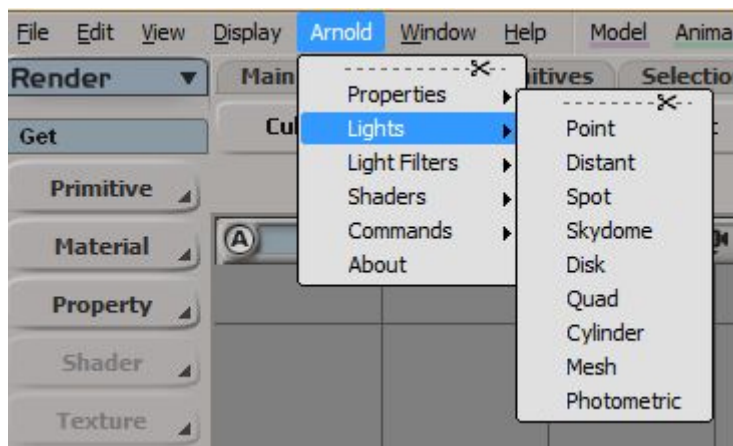
arrays
{
    m_strandPosition.resize(attrArray->nelements);
    for (unsigned int i=0; i<attrArray->nelements; i++)
    {
        AtArray *subArray = AiArrayGetArray(attrArray, i);
        m_strandPosition[i].resize(subArray->nelements);
        for (unsigned int j=0; j<subArray->nelements; j++)
        {
            AtPoint p = AiArrayGetVec(subArray, j);
            // go to world coords, as StrandPosition are in local
            AiM4PointByMatrixMult(&m_strandPosition[i][j], in_sg-
>M, &p);
        }
    }
}

```

Lights

To add an Arnold light, select one from the Get->Primitive->Light menu, or from the main Arnold->Lights menu, listing all the different types of light sources supported in Arnold.

SltoA will make use of Softimage's built-in icons used for lights, for example, to display the area-light shape/size in the viewport. However, since each Arnold light has some parameters different from the Softimage lights, each Arnold light has its own custom light shader listing its extra options.

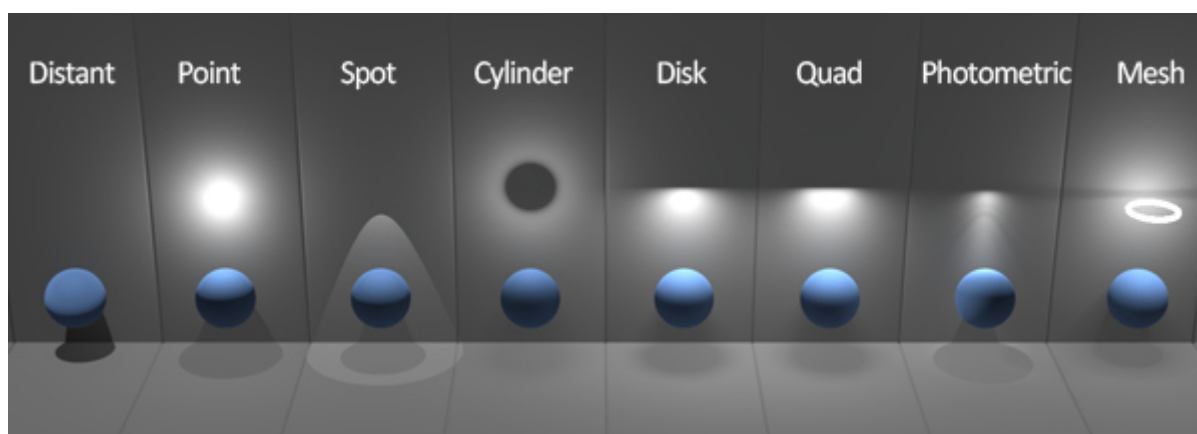


Arnold lights and light filters can be added from the Arnold top menu

In the following topics, we'll first describe the parameters that are common to all Arnold lights, then those specific to each type of light, and finally we'll describe the light filters, pluggable Arnold shaders that extend the lights with new features.

- [Cylinder Light](#)
- [Disk Light](#)
- [Distant Light](#)
- [Mesh Light](#)
- [Photometric Light](#)
- [Point Light](#)
- [Quad Light](#)
- [Skydome Light](#)
- [Spot Light](#)
- [Light Filters](#)

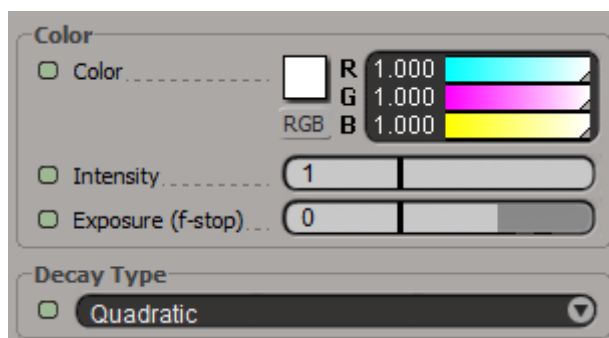
•



Common Light Parameters

The parameters described below are common to all lights in SltoA (except for distant and skydome which does not support decay).

Color and Decay



Color

The color of the light.

Sets the intensity of each RGB color channel. The slider ranges from 0 to 1, but you can enter higher values.

Intensity

Intensity controls the brightness of light emitted by the light source by multiplying the color.

Exposure

Exposure is an f-stop value that multiplies the intensity by 2 to the power of the f-stop. Increasing the exposure by 1 results in double the amount of light.

In Arnold, the total intensity of the light is computed with the following formula:

$$\text{color} * \text{intensity} * 2^{\text{exposure}}$$

You can get the same output by modifying either the intensity or the exposure. For example, intensity=1, exposure=4 is the same as intensity=16, exposure=0. **Note:** $2^0 = 1$, not 0.

$$1 * 1 * 2^4 = 16$$

$$1 * 16 * 2^0 = 16$$

The reasoning behind this apparent redundancy is that, for some people, f-stops are a much more intuitive way of describing light brightness than raw intensity values, especially when you're directly matching values to a plate. You may be asked by the director of photography (who is used to working with camera f-stop values) to increase or decrease a certain light by 'one-stop'. Other than that, this light parameter has nothing to do with a real camera's f-stop control. Also, working with exposure means you won't have to type in huge values like 10,000 in the intensity input if your lights have quadratic falloff (which they should).

If you are not used to working with exposure in the lights, you can simply leave the exposure parameter at its default value of 0 (since $2^0 = 1$, the formula then simplifies to: $\text{color} * \text{intensity} * 1$).



0 (default)



4



Light Contribution

Contribution	
☐ Diffuse	1
☐ Specular	1
☐ SSS	1
☐ Indirect	1
☐ Volume	1
☐ Max Bounces	999

Per-light scaling for Camera, Transmission, Diffuse, Specular, SSS, Indirect and Volume. Weights scaling the light contribution to each of those components independently. Should be left at 1 to produce physically accurate results.



Only Area lights and Point lights (non-0 radius) are visible to the camera. Camera and Transmission values default to 0 with Area lights.

Indirect

The relative energy loss (or gain) at each bounce. This should be left at its default value of 1.0 for physically meaningful results. Values bigger than 1 make it impossible for GI algorithms to converge to a stable solution.



1



1000



4000

Max Bounces

The maximum number of times the energy from this light will be allowed to bounce in the scene. A Max Bounces value of 0 means that the light will only be part of the direct lighting computations, effectively disabling GI for this light. Note that this value works along with the global ray depth controls, so the default value of 999 bounces per light is just a theoretical maximum; in practice, the global ray depth limits are set much lower.



999 (default)

0

Area

This group has different controls based on the light type, but all the lights share•Samples.

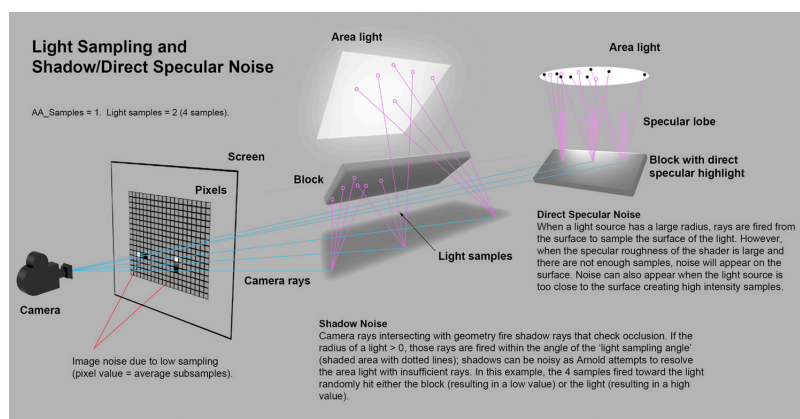
•

Sampling



Samples

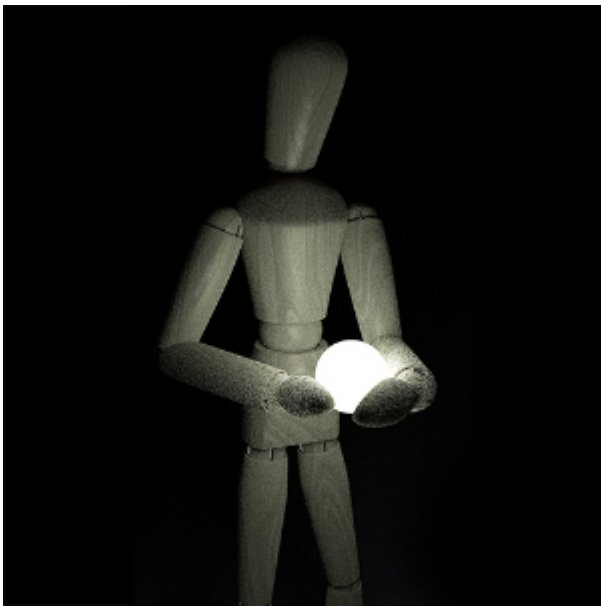
Controls the quality of the noise in the soft shadows and direct specular highlight. The higher the number of samples, the lower the noise, and the longer it takes to render. The exact number of shadow rays sent to the light is the square of this value multiplied by the AA samples.



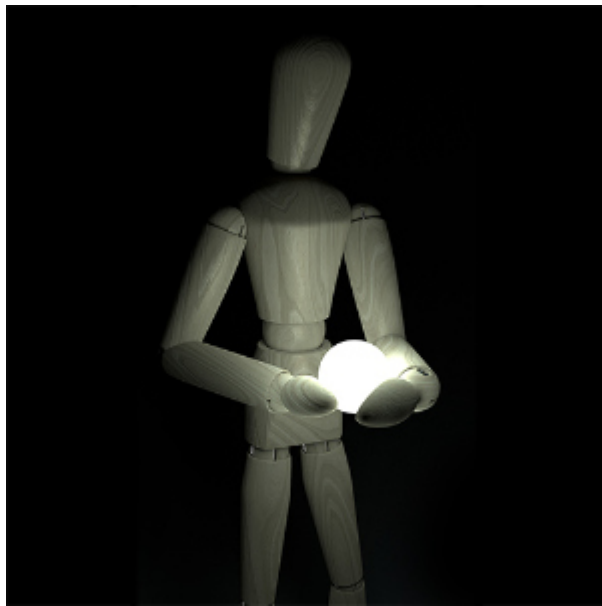
A schematic of how light noise occurs in Arnold

i Note that setting light samples to 0 disables the light.

Noise from lights can sometimes be difficult to diagnose, particularly if the light source is broad in comparison to the scene and the shadows have an extremely wide penumbra. In these cases, it can sometimes be mistaken for indirect diffuse noise. It highlights the necessity for testing noise ray type. [This diagram](#) shows how light is traced in Arnold.

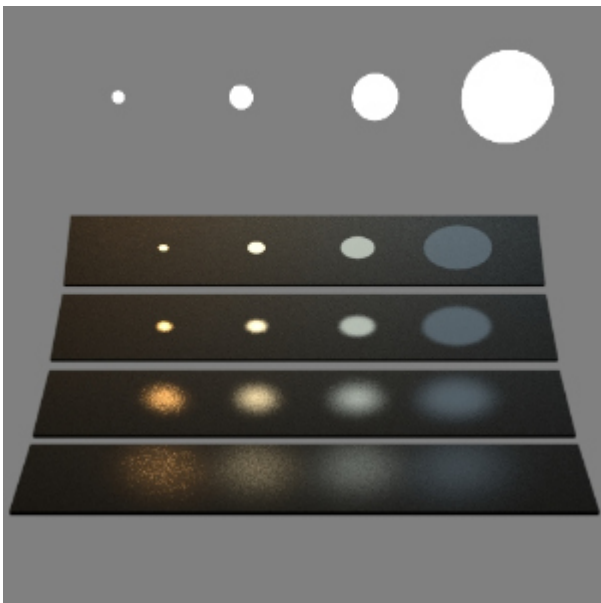


1 (default)

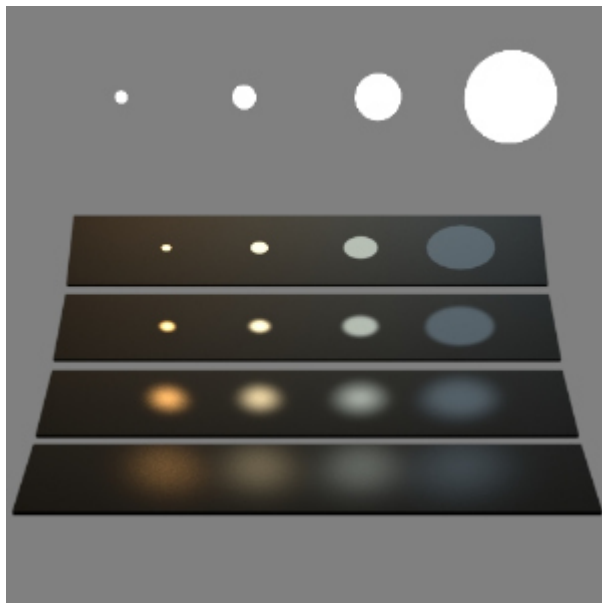


3

The example below shows specular highlights from area light sources. There are four spherical mesh light sources of varying sizes and color temperatures. Underneath are four cubes with Standard shaders assigned to them with varying degrees of specular roughness. •Note that more noise is apparent with smaller light sources. Increasing the number of light samples resolves the noise. •



1 light sample (default)



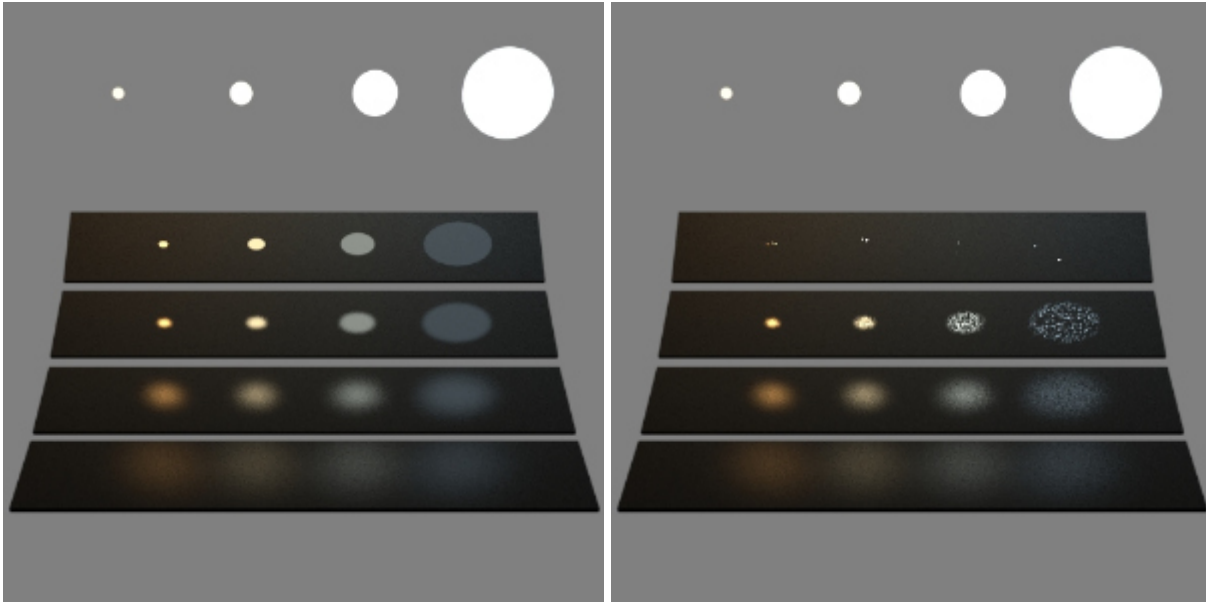
4 light samples

Specular roughness from top to bottom: 0, 0.1, 0.2, 0.3

If the issue is noise in a specular highlight, you will need to confirm that the source is the direct light and not a secondary ray type (such as specular). This is easy to achieve by turning off global illumination; set the Diffuse Depth and Specular Depth to zero (this essentially turns off all global illumination). If the noise is still there, we know it is the specular component of the illumination model. If the issue is shadow noise, then we can simply toggle ignore shadows in the Arnold render settings, and the noise will completely resolve.

The key, again, is to modify the sampling and observe the changes. Increasing the number of light samples should have an immediate and quantitative effect on the smoothness of the specular highlight and the shadow. If there is no change, light samples are not responsible for the noise.

Multiple importance sampling (MIS) is enabled by default in Arnold. The images below show the difference when rendering with and without multiple importance sampling.

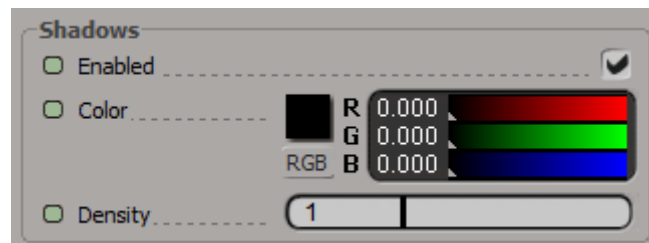


Multiple importance sampling: On

Multiple importance sampling: Off

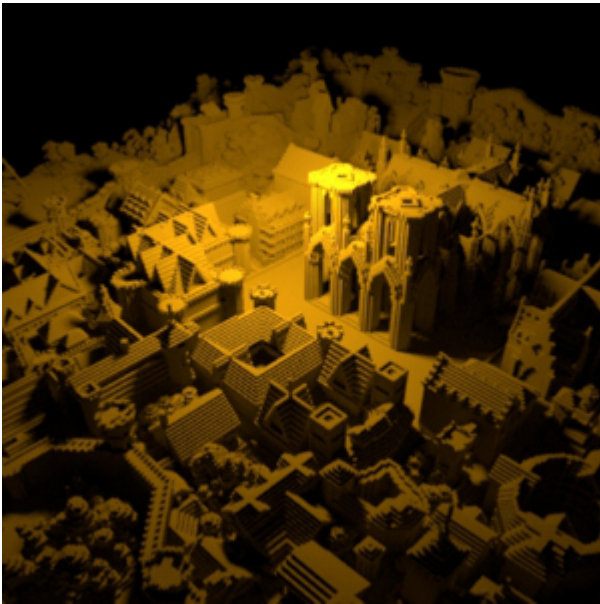
 More information about light samples and removing noise can be found [here](#).

Shadows

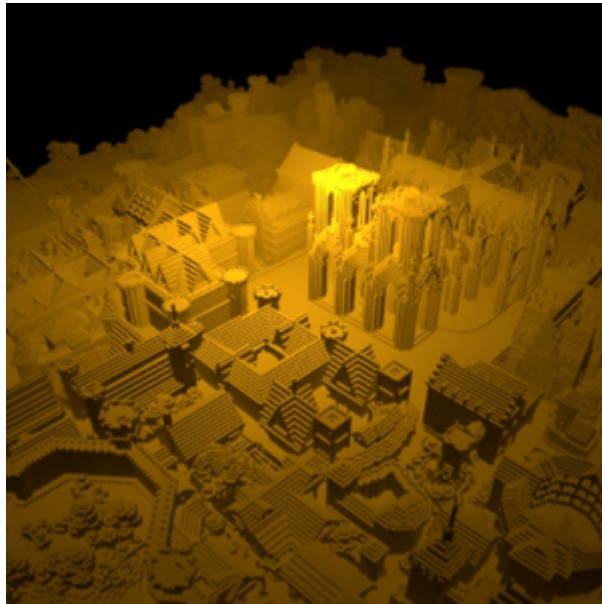


Enabled

Enables the computation of shadows cast from the light.



Enabled (default)



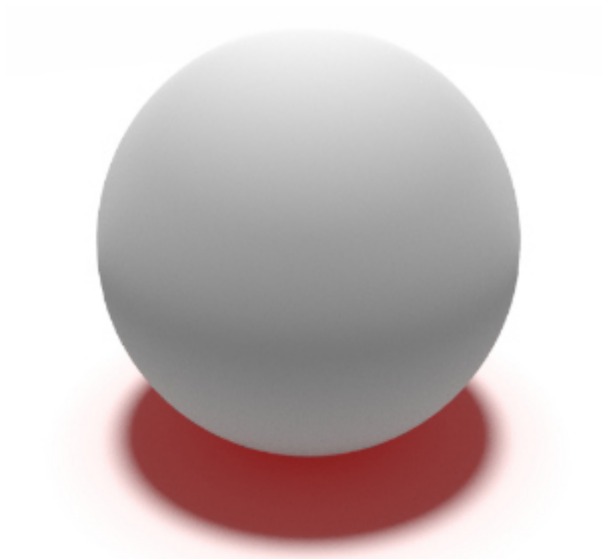
Disabled

Color

Sets the intensity of each color channel for shadows. Normally this would be black.



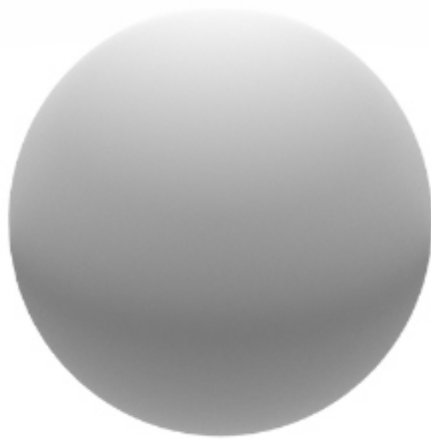
Black (default)



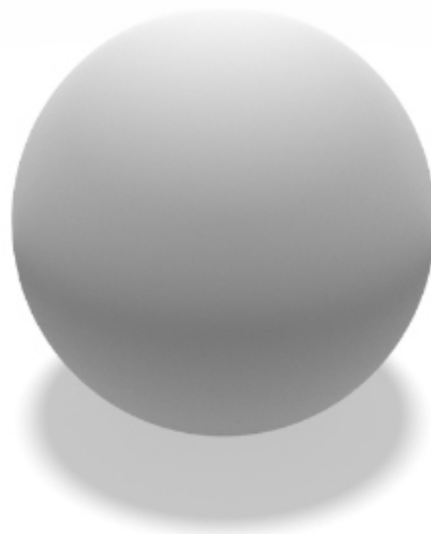
Red

Density

Sets the shadow density, or strength. This controls how the shadow blends with the material on which the shadow is cast: a value of 1.0 produces an opaque, black shadow, and a value of 0.0 gives no shadow. Normally this would be 1.0.

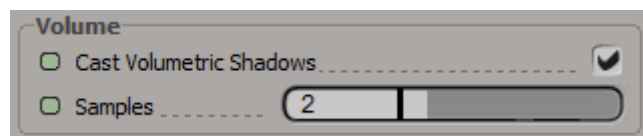


0



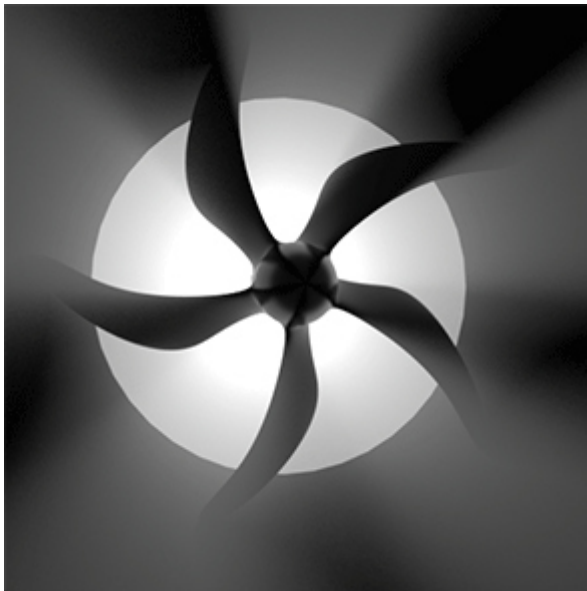
0.5

Volume



Cast Volumetric Shadows

Determines if volumetric shadows will be computed. This option is not in Distant or Skydome Lights. •



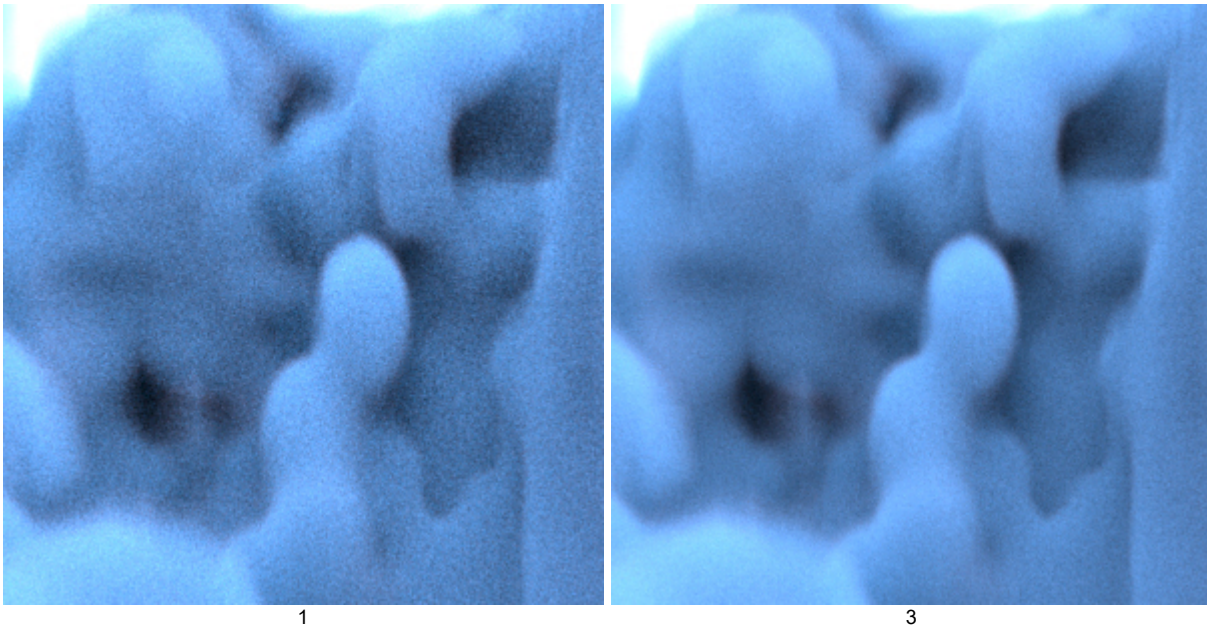
Enabled (default)



Disabled

Samples

The volume samples parameter handles the number of samples used to integrate the in-scattering from direct light. Like the "samples" parameter for surfaces, it is also a squared number.

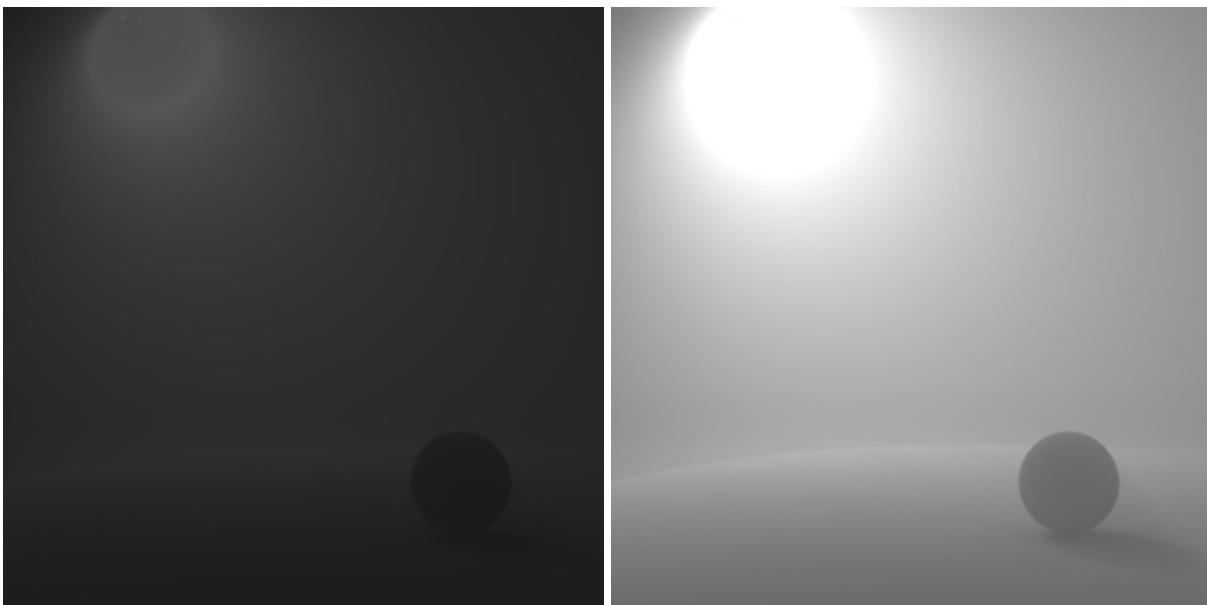


This parameter does not apply to Atmosphere Volume. To improve the quality of the Atmosphere Volume, you must increase the [Atmosphere Volume samples](#).

•

Normalize •

If enabled, you will be able to tweak the shadow softness by changing the size (i.e., radius) of the light, without affecting the amount of emitted light. This is very handy for artistic control. Otherwise, if not enabled, the amount of emitted light is proportional to the light's surface area.



Enabled (default). Radius 2.5.

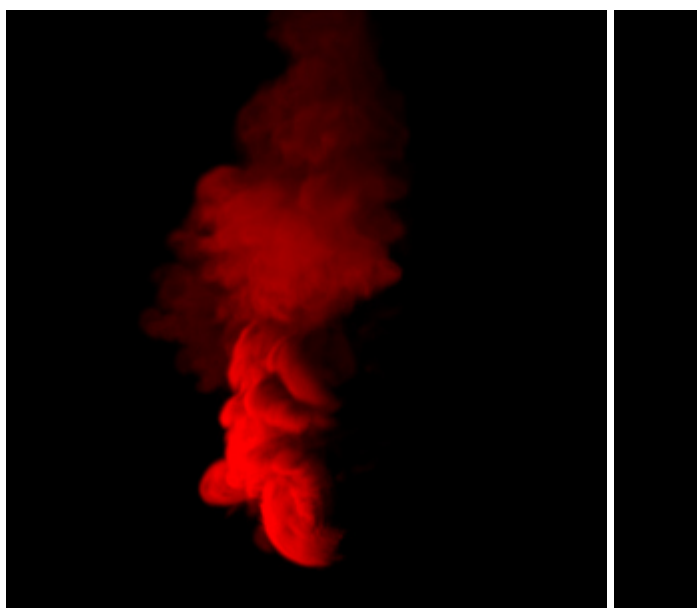
Disabled. Radius 2.5.

AOV Light Group

Per-light AOVs are available via a string parameter. Each light has an AOV parameter which writes out the light contribution to a separate AOV with a corresponding name. See also [Light Path Expression AOVs](#).



Volume Beauty AOV



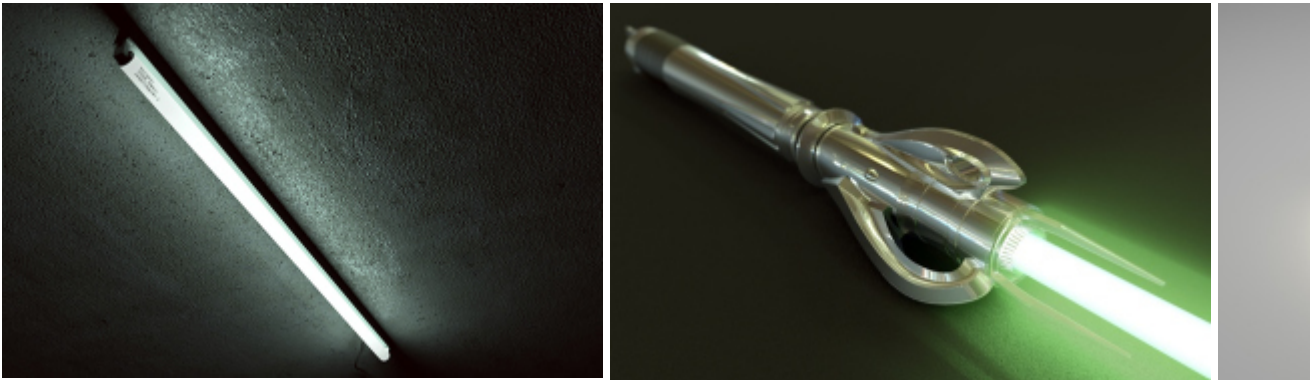
Red light volume AOV

There are two render channels that you can enable to contain the lights contribution, called `Arnold_Lights` and `Arnold_Volume_Lights`. Their format must be set manually to EXR, their data type to RGBA.

- Enter a name for the per-light AOV in the 'AOV Light Group' of the light. For example, if the light group name is "red", then the AOV name should be "volume_red".

•

Cylinder Light



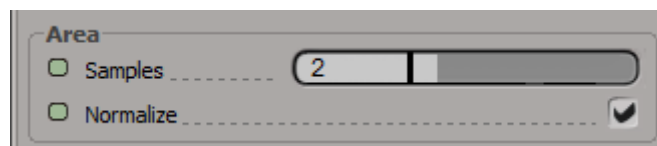
Cylinder lights are typically used to simulate fluorescent tube lighting

The cylinder light shape simulates light from a cylindrical area source (tube shape).

-

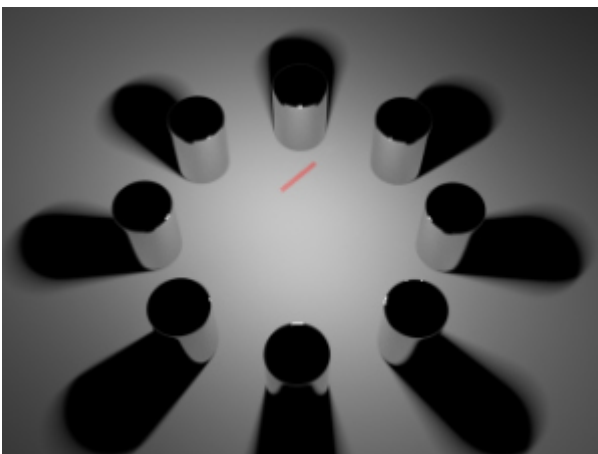
The [lights](#) page has more detail about the controls. The only additional parameter are:

-



-

Increasing the size of the cylinder light will create a larger area light size and therefore will soften the shadows.

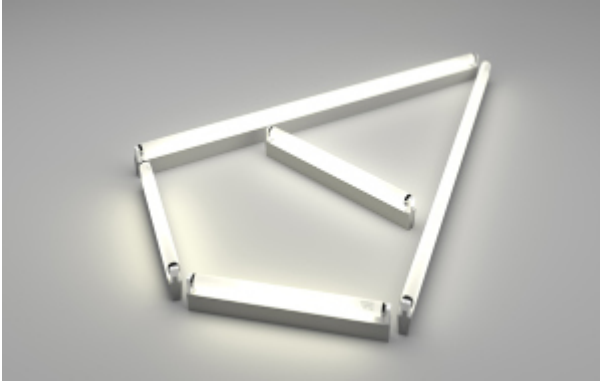


Cylinder light scaled down



Cylinder light scaled up

-
-
-



Volume Scattering enabled

Further examples of cylinder lights

Disk Light



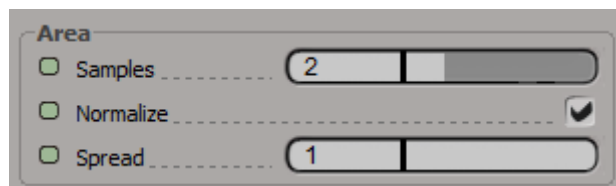
-

The disk light shape simulates light from a circular area source (flat disk).

-

The [lights](#) page has more detail about the controls. The only additional parameters are:

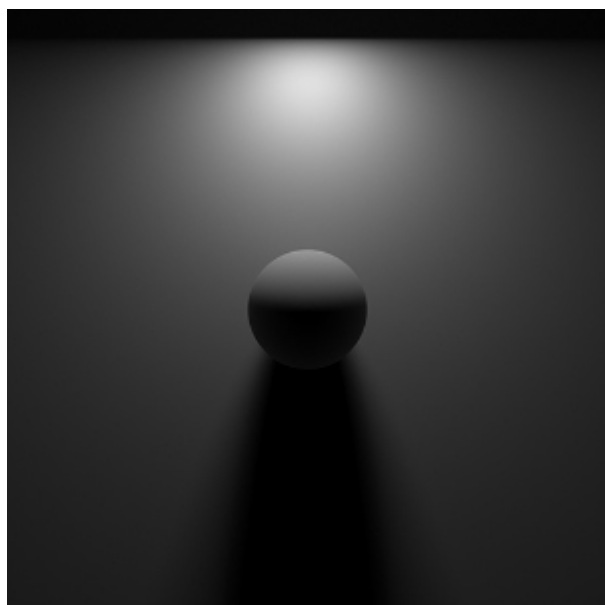
-



-

Spread

Emits light focused in the direction along the normal. The default spread value of 1 gives diffuse emission, while lower values focus the light more until it becomes almost a laser-like beam at value 0. Currently fully focused laser beams at value 0 are not supported, there is always a small minimum spread. Low spread values can be noisier than the default high spread, so be careful when using them.

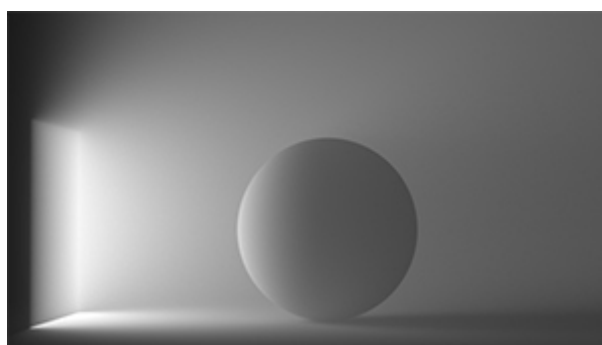


1 (default)



0.3

The animation below shows the effect when lowering the spread value.



spread value lowered from 1 (quad_light)

-
-



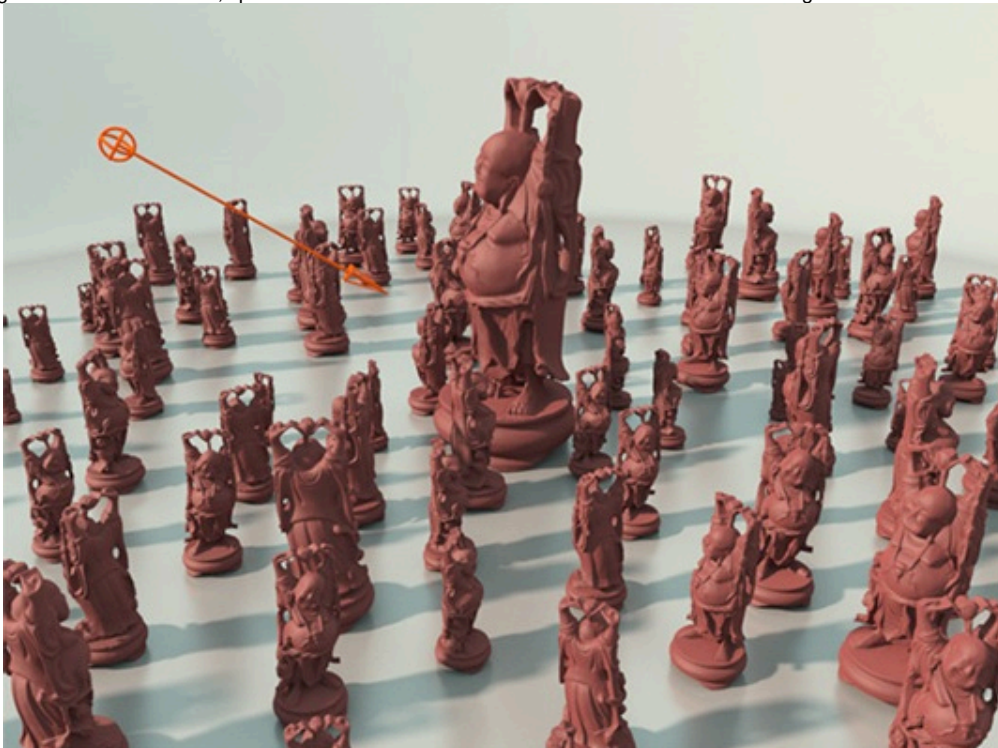
Further example of a scene lit with a disk light

Distant Light



This light can be used to simulate sunlight coming through a window

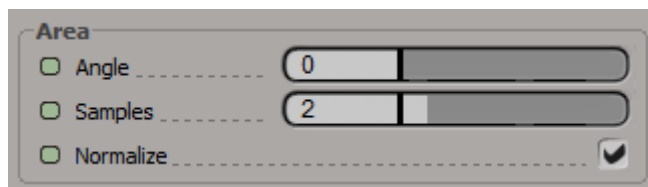
This is a parallel light from a distant source, specified in terms of a direction vector. Often used to model sunlight.



Scene illuminated with an Arnold distant light. All shadows are parallel regardless of the position of the light

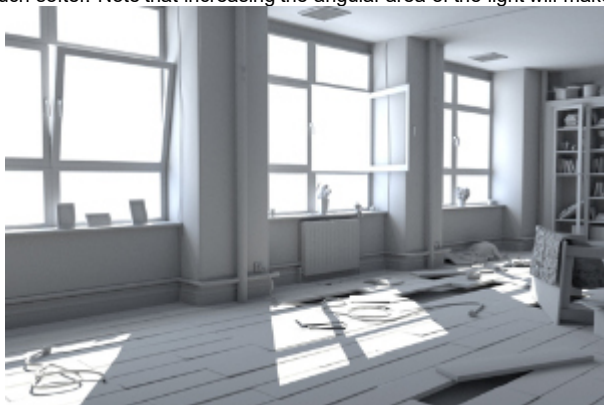
.

The [lights](#) page has more detail about the controls. The only additional parameter is:

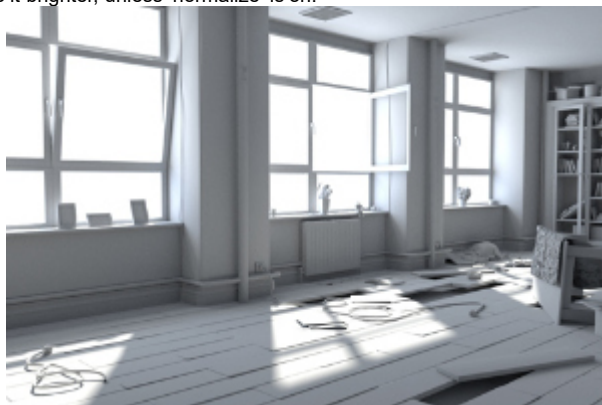


Angle

The angular size of the light, in degrees. Non-zero values produce realistic soft shadows. For example, the Sun subtends approximately 0.5 degrees as seen from the surface of the Earth. A setting of 1 or 2 will produce slightly softened shadows like hazy sunshine, larger numbers such as six will be much softer. •Note that increasing the angular area of the light will make it brighter, unless 'normalize' is on. •



0



2



Mesh Light



Torus shapes converted to mesh light. Models (left) kindly provided by [Lee Perry Smith](#) .

In situations where conventional light shapes will not suffice, mesh lights are more suitable. Mesh lights can be used to create interesting lighting effects that would not be possible any other way. For example, effects such as neon lighting or a car light motion trail effect can be achieved more easily with mesh lights.

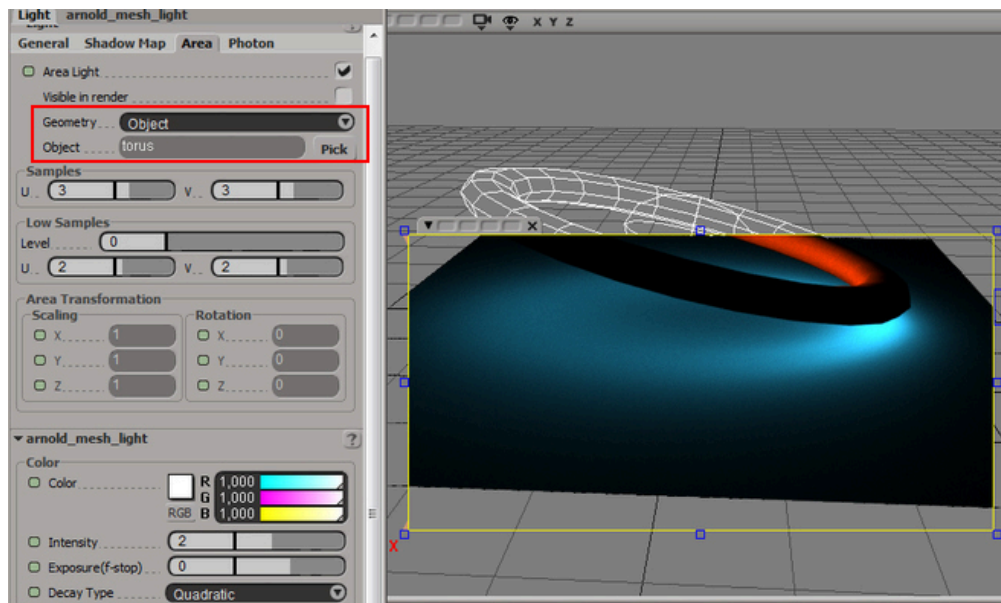


Geometry converted to mesh_light

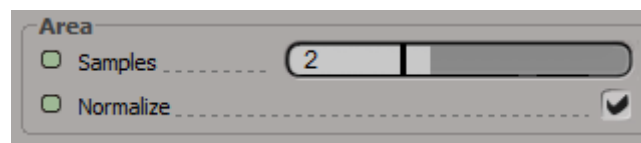
When selecting an Arnold mesh light from the light menu, you will be asked to pick a mesh object. The light is then created, and the picked object set as the light's shape in the light primitive ppg. Also, the light is pose-constrained to the mesh, so to visually match its position, scale and rotation.

The light is assigned a shader called `arnold_mesh_light`. It is this combination (the area light shape set to object and the `arnold_mesh_light` assigned as light shader) that makes SltA understand that this light is to be treated as a mesh light.

Note that the mesh, besides becoming a light source, maintains its shading properties. This means its material stays untouched and therefore reacts to the light emitted by itself as well.



The [lights](#) page has more detail about the controls. The only additional parameter are:



Normalize

If enabled, you will be able to tweak the light softness by changing the area of the mesh, without affecting the amount of emitted light. This is very handy for artistic control. Otherwise, if not enabled, the amount of emitted light is proportional to the mesh's area, as dictated by physics.

It is worth noting the the effect of the sampling control with the mesh light:



Samples 1

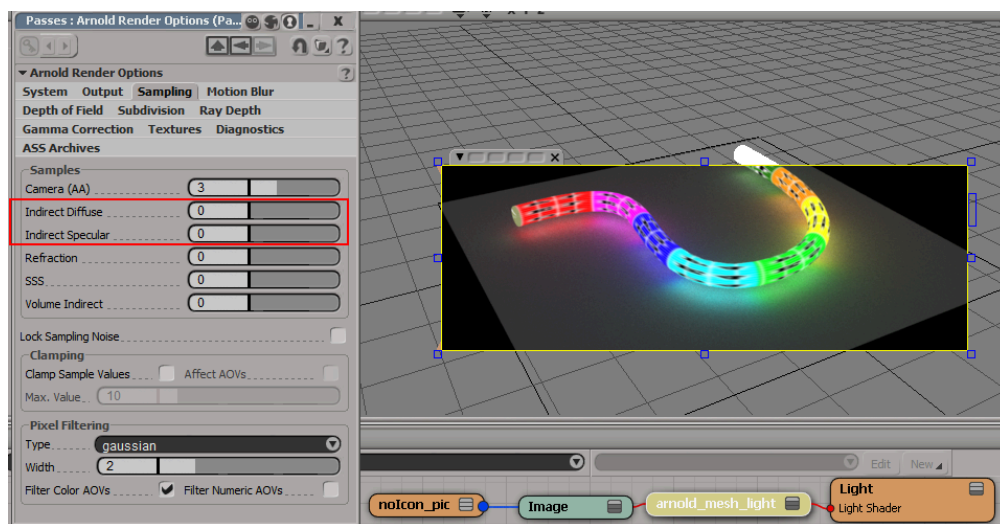


Samples 3

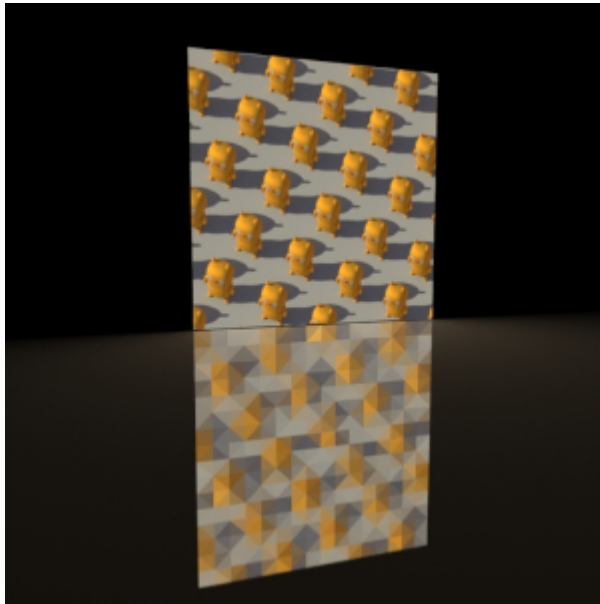


Samples 6

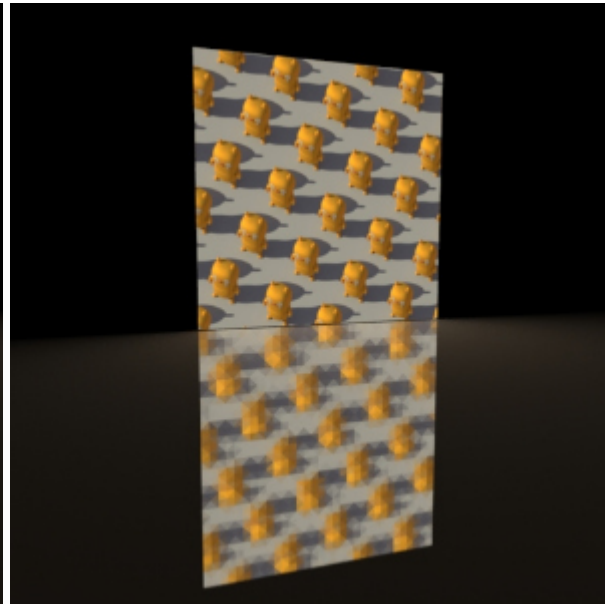
The light color can be textured. In this case, the main UV set is used to lookup the color image.



You may need to increase the number of subdivision iterations for the mesh light if the color texture is clearly visible in specular reflections. For example, this may be evident in a scene where a TV screen is reflected in a glass window.



4

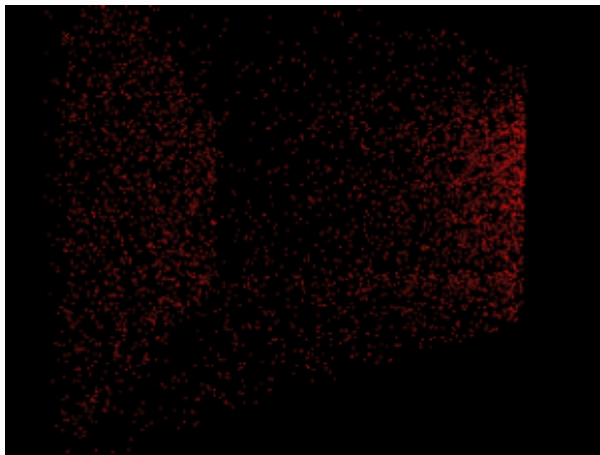


5

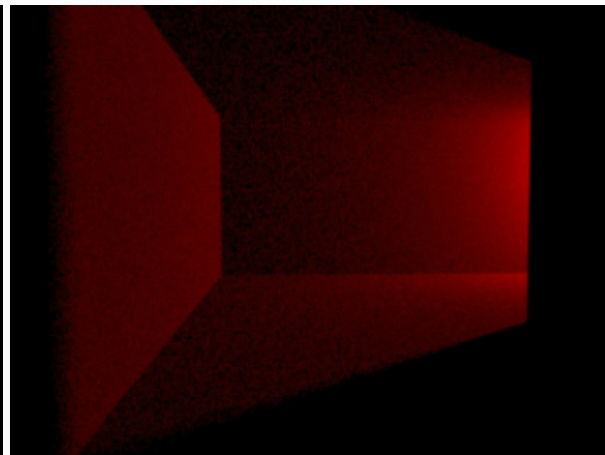
Mesh Light vs Emission

The example below is a comparison between a mesh light and a surface that has a standard_surface shader assigned to it with a high Emission value.

You can see that even with diffuse samples = 16, the emission is noisier than a mesh_light with diffuse samples = 2.

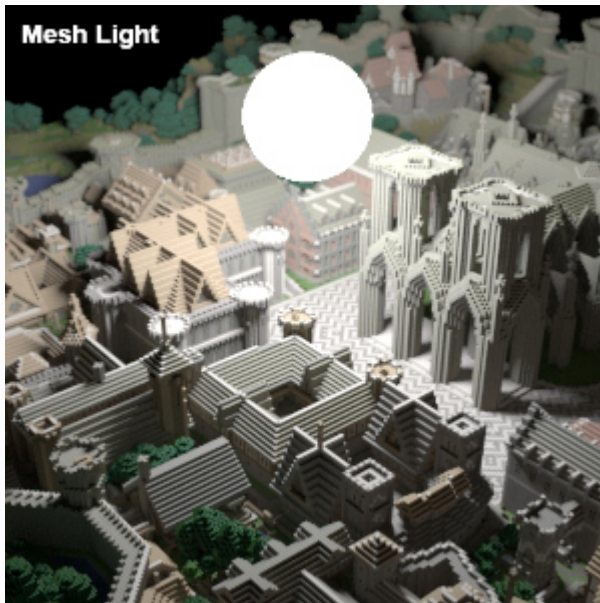


Emission, Diffuse samples = 2, Time = 0:08.44

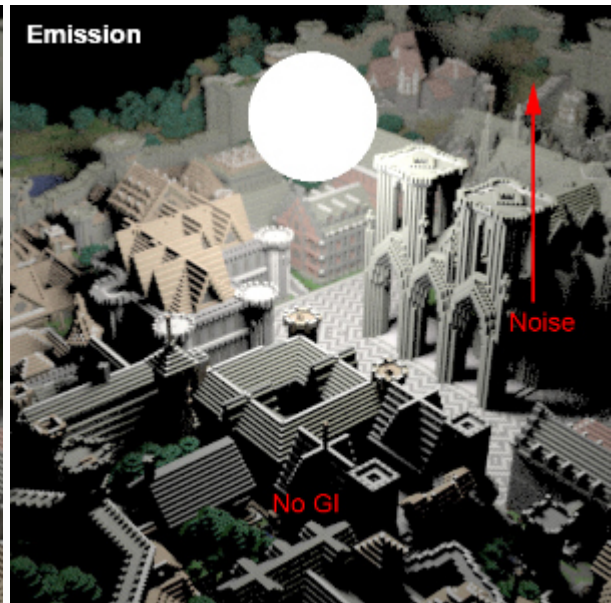


Emission, Diffuse samples = 16, Time = 5:36.86

Below is another comparison test between a mesh light (left image), and a sphere with a highly emissive standard_surface shader assigned to it (right image). As you can see in the mesh light image, light is bounced around the scene, whereas there is no bounced light in the emissive sphere render. This is because the rays from the mesh light work in conjunction with the Diffuse rays, whereas, only the primary rays are considered when using emission only. Even with 6 Diffuse samples, the render using Emission only contains far more noise.

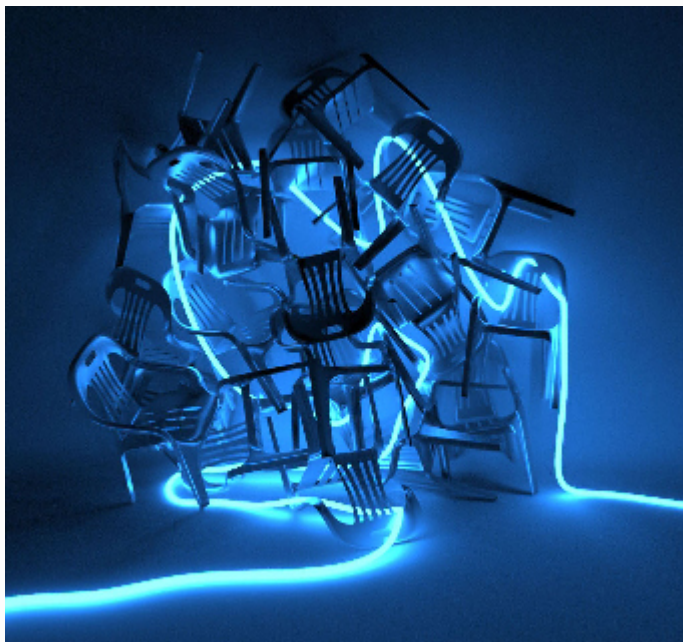


4 mesh light samples - clean render



6 Diffuse rays - noisy with no bounced light

In the example below, diffuse•samples have been increased to ten to get a clean result using only emission•in the scene. With a mesh_light , the scene only requires three•diffuse samples.



Emission. Samples 7.10.3.2 = 45mins



Mesh Light. Samples 7.3.3.2. (4 Light Samples) = 31

-
-

MultiExcerpt named 'examples ' was not found

The page: mesh_light was found, but the multiexcerpt named 'examples ' was not found. Please check/update the page name used in the 'multiexcerpt-include macro.

-

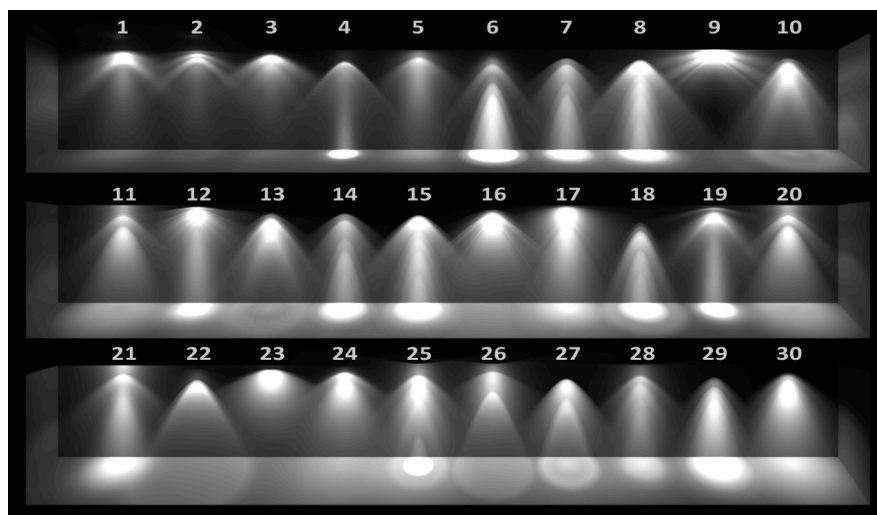
Photometric Light



Photometric lights use data measured from real-world lights, often directly from bulb and enclosure manufacturers themselves. You can import IES profiles from companies such as [Erco](#), [Lamp](#), [Osram](#), and [Philips](#); their IES files provide accurate intensity and spread data for a given light model.



Example IES profiles can be downloaded [here](#) (paths to the IES files will need to be edited).

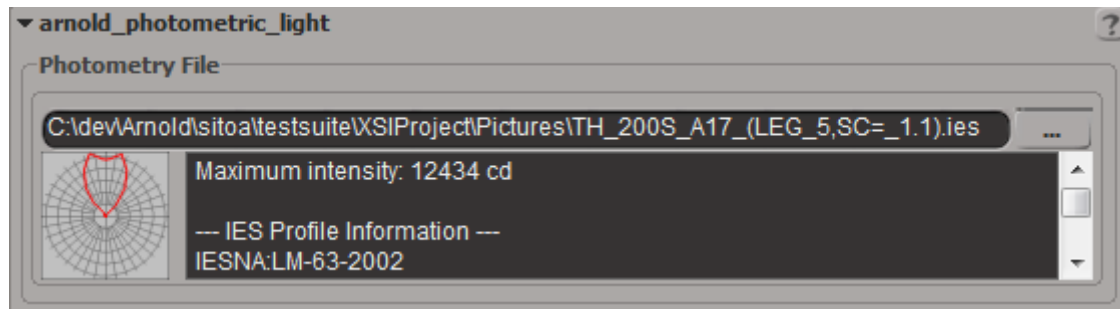


Example IES lighting profiles



Some example IES profiles can be downloaded [here](#).

The Photometric light can be found under Arnold>Lights>Photometric:



•

Photometry File

Allows you to choose an IES light profile that defines the distribution of light.

•

✓ IES file not working

Many IES files originate from various lighting manufacturers, some of whom put extra text at the beginning of their IES files. If you are experiencing difficulties rendering a particular IES file, try opening the IES file in a text editor and deleting the header text from the top of the file. The file should typically start with •TILT=NONE • and if it doesn't, that may be the culprit.

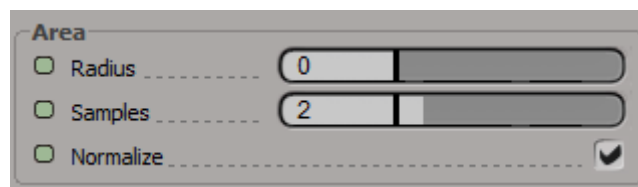
Point Light



•

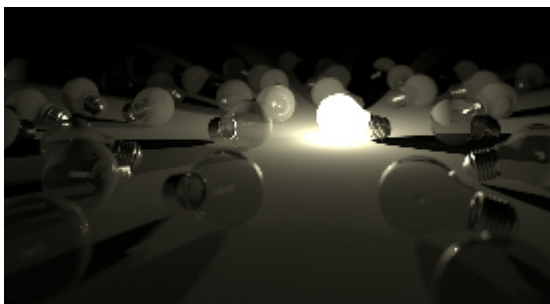
The [lights](#) page has more detail about the controls. The only additional parameter is:

•

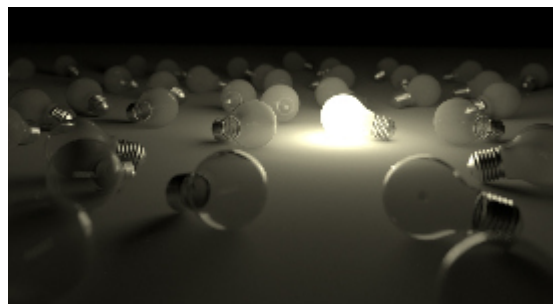


Radius

The radius of the light's spherical surface. Although the name of this light is 'point' for historical reasons, it really is an emissive sphere, unless radius is set to zero, in which case it becomes a true point light of no physical size.



0

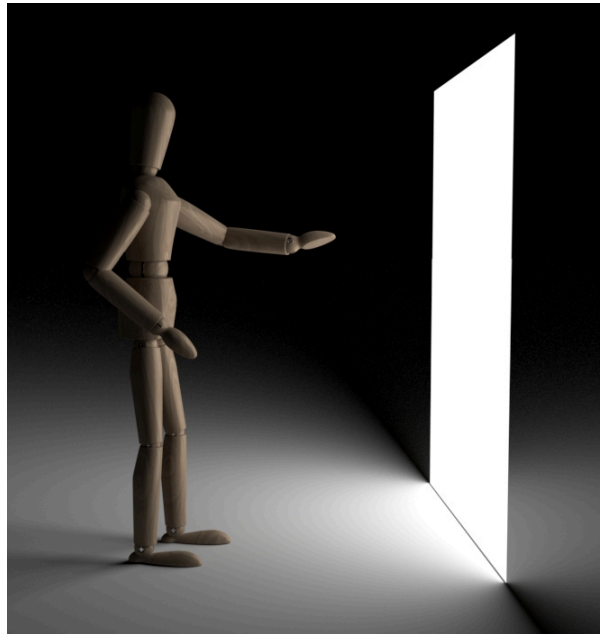


4



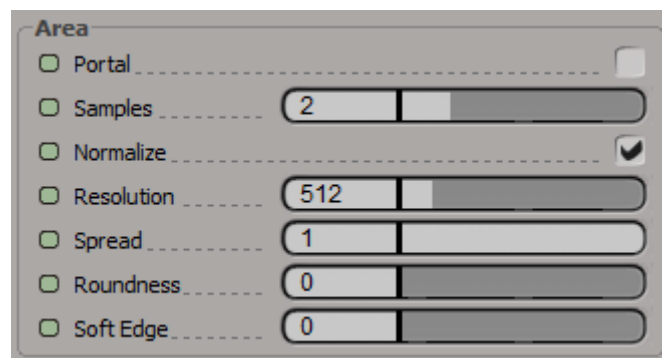
.

Quad Light



Simulates light from an area source (a quadrilateral, specified by four vertices). It can be used to model light from an extended source (fluorescent strip lights) or in some circumstances from a window.

•



The [lights](#) page has more detail about the controls. The only additional parameters are:

Portal

When enabled, this disables anything related to the illumination produced by the light object (a Portal doesn't generate light), and the object now becomes a light portal. The light portal should be oriented so that it is pointing in through the window.



Roll over image to view scene without `light_portal` (`skydome_light` only)

•

Skydome lights can use light portals to reduce noise for interior scenes, where the light comes in through relatively small openings. Instead of emitting light, they can be used to guide skydome light sampling. Light portals must be placed to cover all windows, doors, and other openings through which skydome light comes into the scene.

i **Skydome lights** also have a `portal_mode` parameter with options `off`, `interior_only` and `interior_exterior` to respectively turn off portals, block any light outside portals for interior only scenes, and let light outside portals through for mixed interior and exterior scenes. Blocking light outside portals more predictably reduces noise for interior only scenes.

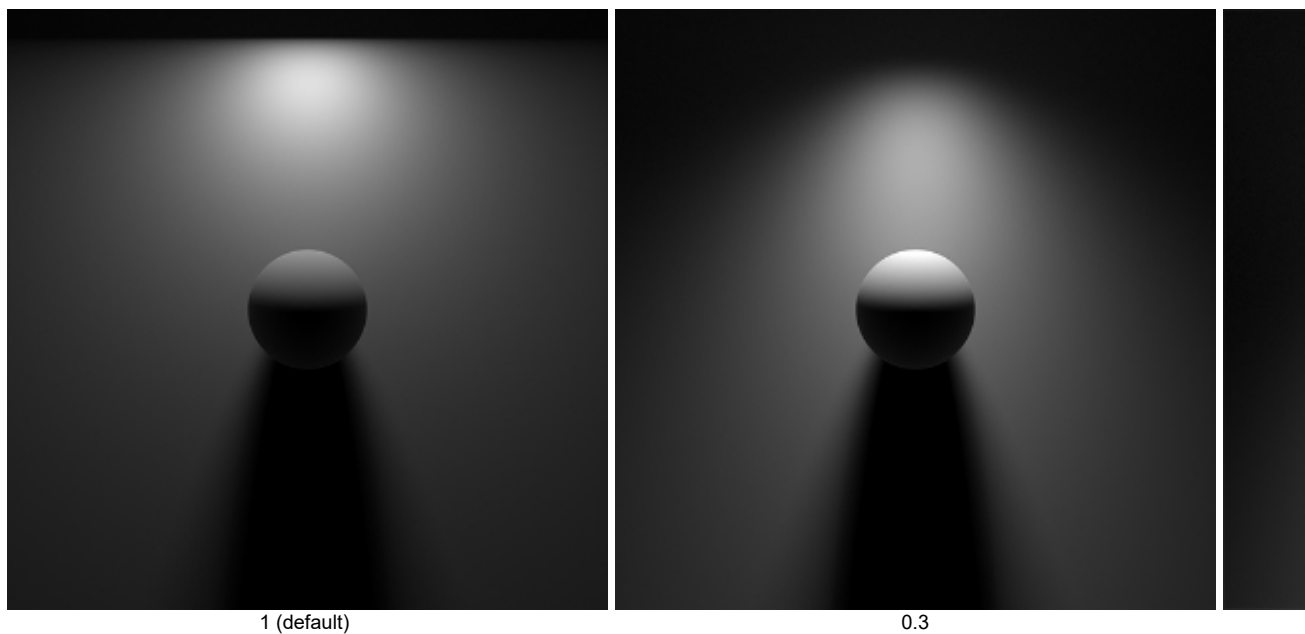
Resolution

When a shader is connected to a `quad_light`, Arnold will automatically construct importance sampling tables similar to those used in the `skydome_light`. This permits efficient sampling according to the luminance of the texture, which can greatly reduce sampling noise, especially when using HDRI textures. Like the `skydome_light`, the resolution of the table is controlled by the `resolution` parameter with a default value of 512. If you are using a color image as an input, then you will not need to set this value higher than the resolution of the image passed to the `color` parameter.

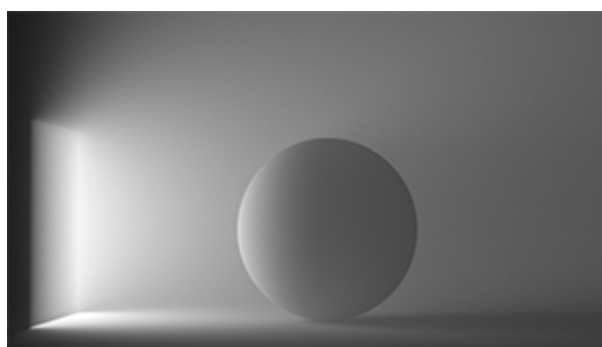
See the [Skydome light topic](#) for more hints (in particular the description of Multiple Importance Sampling the Skydome light Resolution parameter).

Spread

Emits light focused in the direction along the normal. The default spread value of 1 gives diffuse emission, while lower values focus the light more until it becomes almost a laser-like beam at value 0. Currently fully focused laser beams at value 0 are not supported, there is always a small minimum spread. Low spread values can be noisier than the default high spread, so be careful when using them.



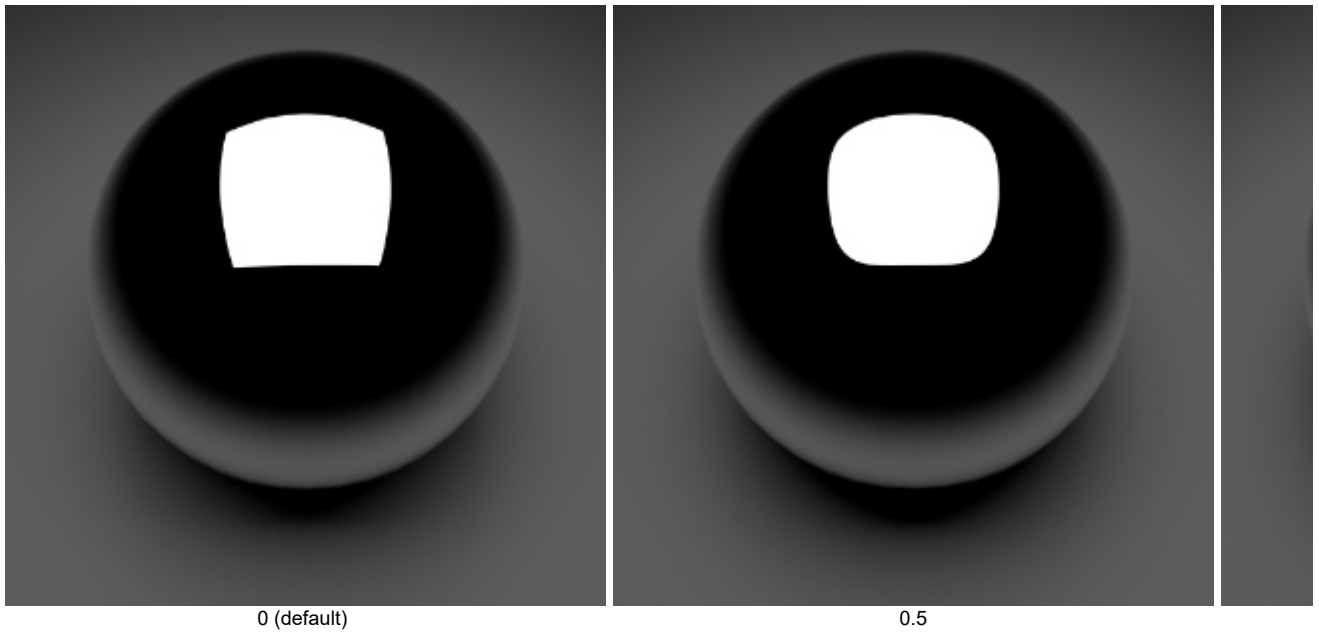
The animation below shows the effect when lowering the spread value.



spread value lowered from 1 (quad_light)

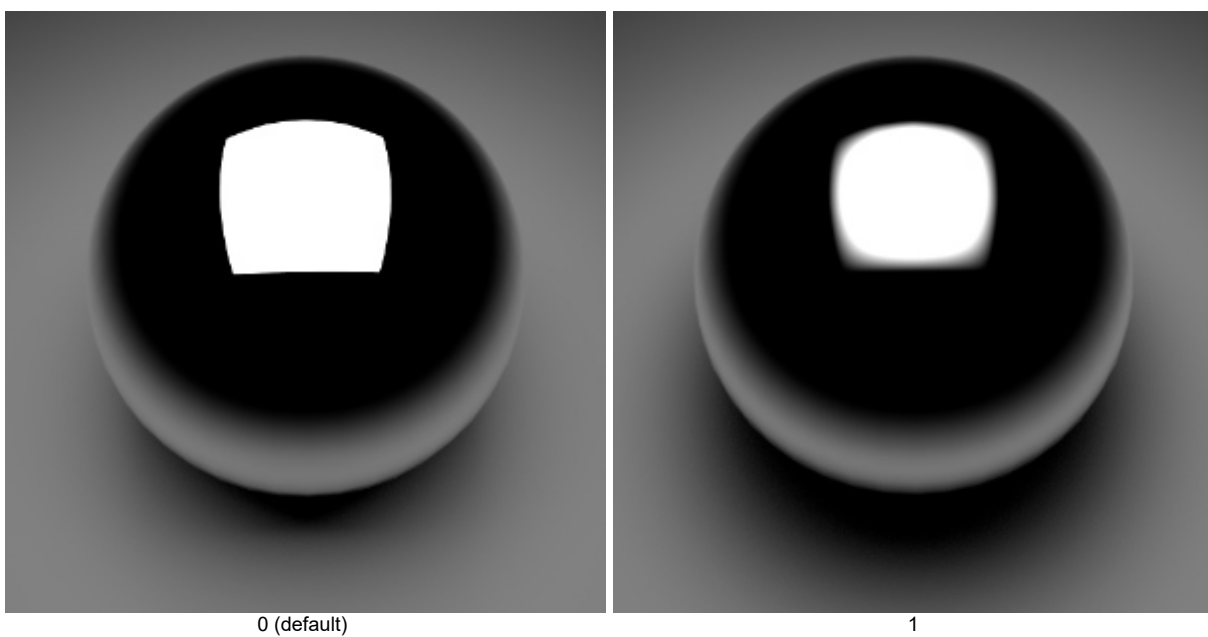
Roundness

Changes the shape of the light from a square at 0, to rounded corners, to a disk at 1.



Soft Edge

Specifies a smooth falloff for the edges of the light. The value specifies the width of the soft edge, from no soft edge at 0, to smooth falloff all the way to the light center at 1. Works similarly to the `penumbra_angle` for spot lights.

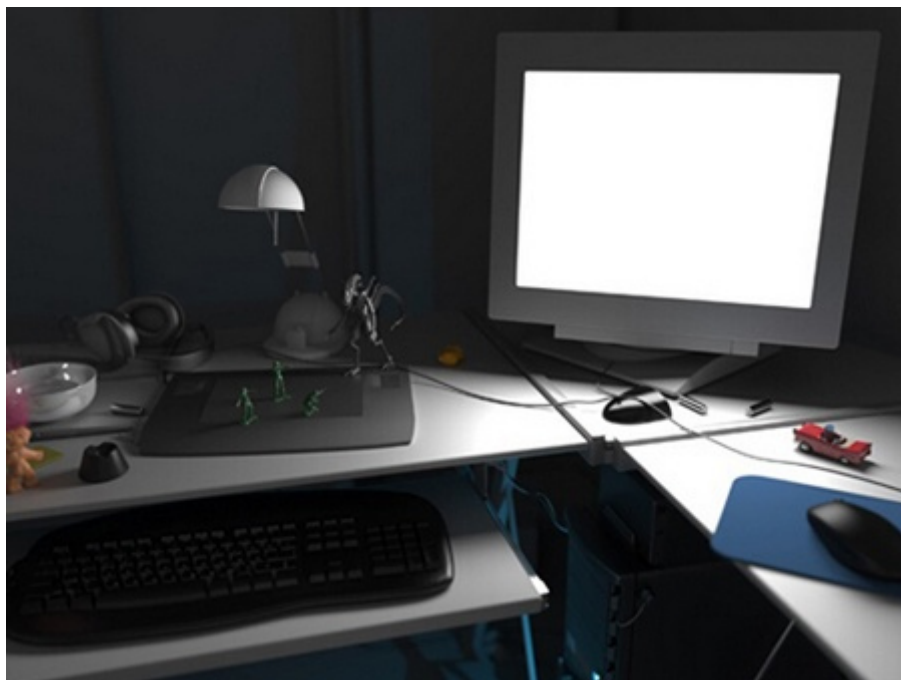


-
-
-

Further Examples



Cars lit with large 'quad' area lights simulating soft box studio lighting



Quad light positioned in front of monitor (rollover image)



.

Skydome Light

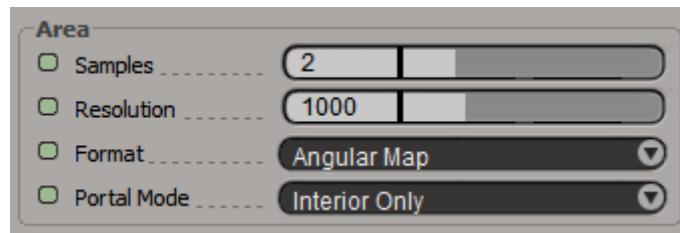


This simulates light from a sphere or dome above the scene, representing the sky. It can also be used with high dynamic range (HDR) images to perform image-based environment lighting. This is the node which is typically used for lighting exterior scenes.



This light is designed for outdoor scenes and is represented by a spherical dome in the background. Importance sampling will trace rays to specific directions of this dome. However, in an interior scene, most of these rays will hit an object, getting no contribution from the light at all and thus creating noise. In this situation, adding `light_portals` to the windows will help to reduce noise in an interior scene when using the `skydome_light`.

The [lights](#) page has more detail about the controls. The only additional parameters are:



Color

The color of the light. The Skydome light also has a texturable Color slot, so that you can map a texture that can be used for IBL.

Resolution

The resolution controls the detail of reflections of the skydome. For most accurate results the Skydome light resolution must be set to match the HDRI image resolution, however, in many cases it can be set lower without a noticeable loss of detail in reflections. By default, the parameter is set to 1000. The higher the resolution parameter, the longer the skydome_light will take to precompute the importance tables for the light, which increases scene startup time.



The resolution parameter should be used with care. The higher the resolution value, the longer the skydome_light will take to precompute the importance map for that light. For high resolution maps this can be very slow.

Format

The type of map being connected. It can be set to Lat-long (most common), Mirrored Ball or Angular.



Lat-long



Mirrored Ball



Angular

The image below is rendered with a Skydome light only. It uses 3 'Indirect Diffuse' samples and 3 light samples with a render time of six minutes. Note how all the sharp shadow details are preserved whilst the noise levels are low.

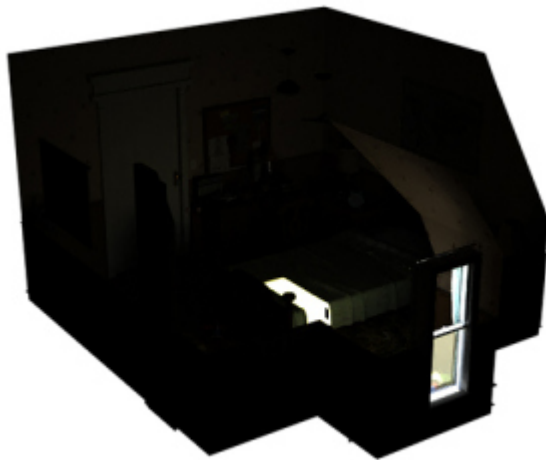


Portal Mode

Defines how the skydome light interacts with light portals.

- off : turns off portals.
- interior_only : blocks any light outside portals for interior-only scenes.
- interior_exterior : lets light outside portals through for mixed interior and exterior scenes.

Blocking light outside portals more predictably reduces noise for interior-only scenes.



interior_only (default)



interior_exterior

Contribution

Camera

By default, skydome lights are directly visible as a background. Lowering the influence of the light of camera rays makes them invisible to the camera, and makes the background transparent.

Transmission

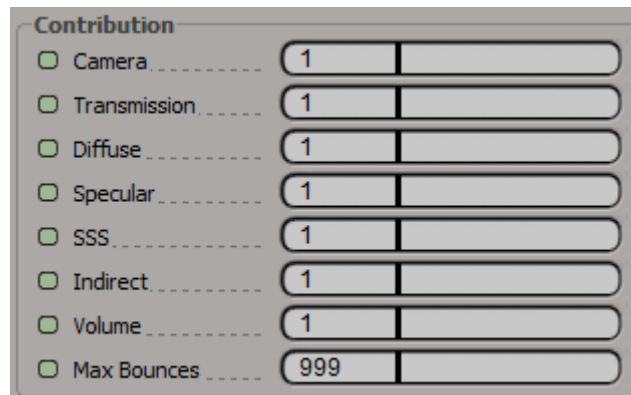
Per-light scaling for transmission. Should be left at 1 to produce physically accurate results.

Diffuse / Specular / SSS / Volume

Per-light scaling for Camera, Transmission, Diffuse, Specular, SSS, Indirect and Volume. Weights scaling the light contribution to each of those components independently. Should be left at 1 to produce physically accurate results.



Only Area lights and Point lights (non-0 radius) are visible to the camera. Camera and Transmission values default to 0 with Area lights.



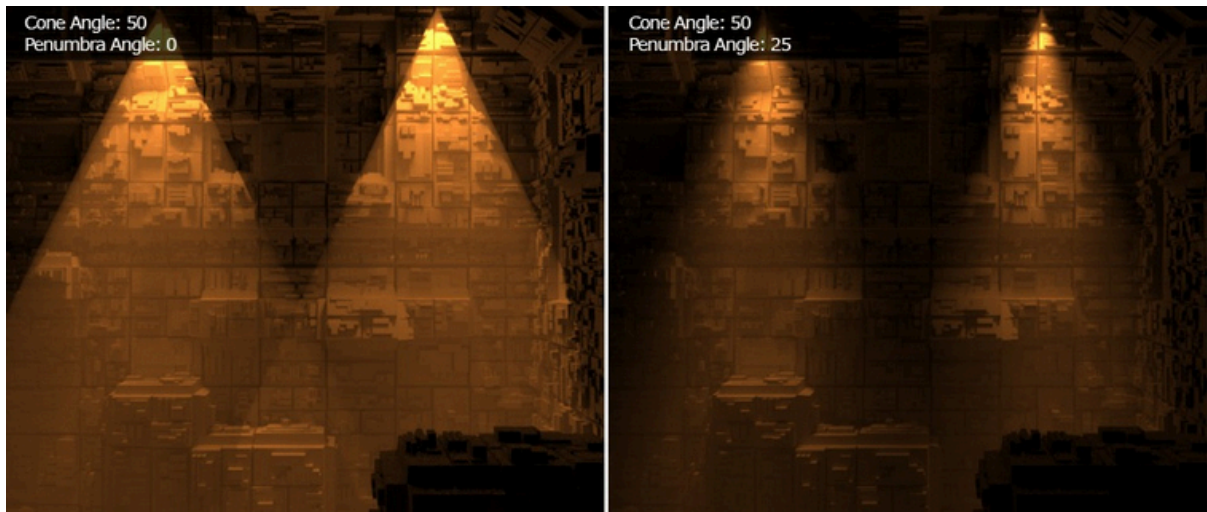
Indirect Lighting

Outputs an indirect light AOV (the skydome_light outputs to direct light AOVs by default).



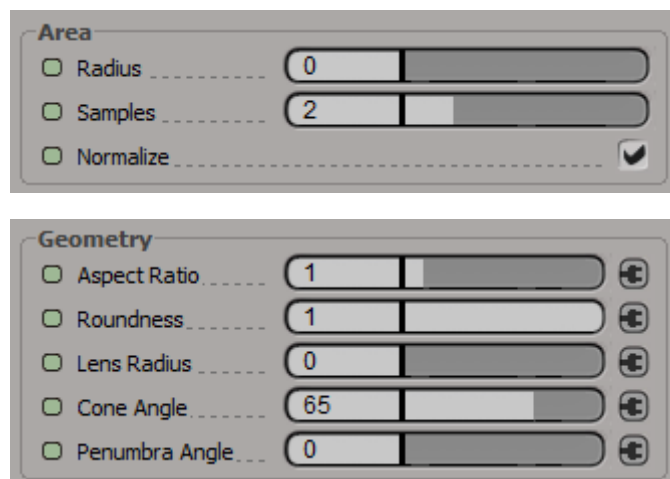
The Skydome light will not work with [Atmosphere Volume](#). You must use regular lights that have a precise location and size, and inverse-square decay.

Spot Light



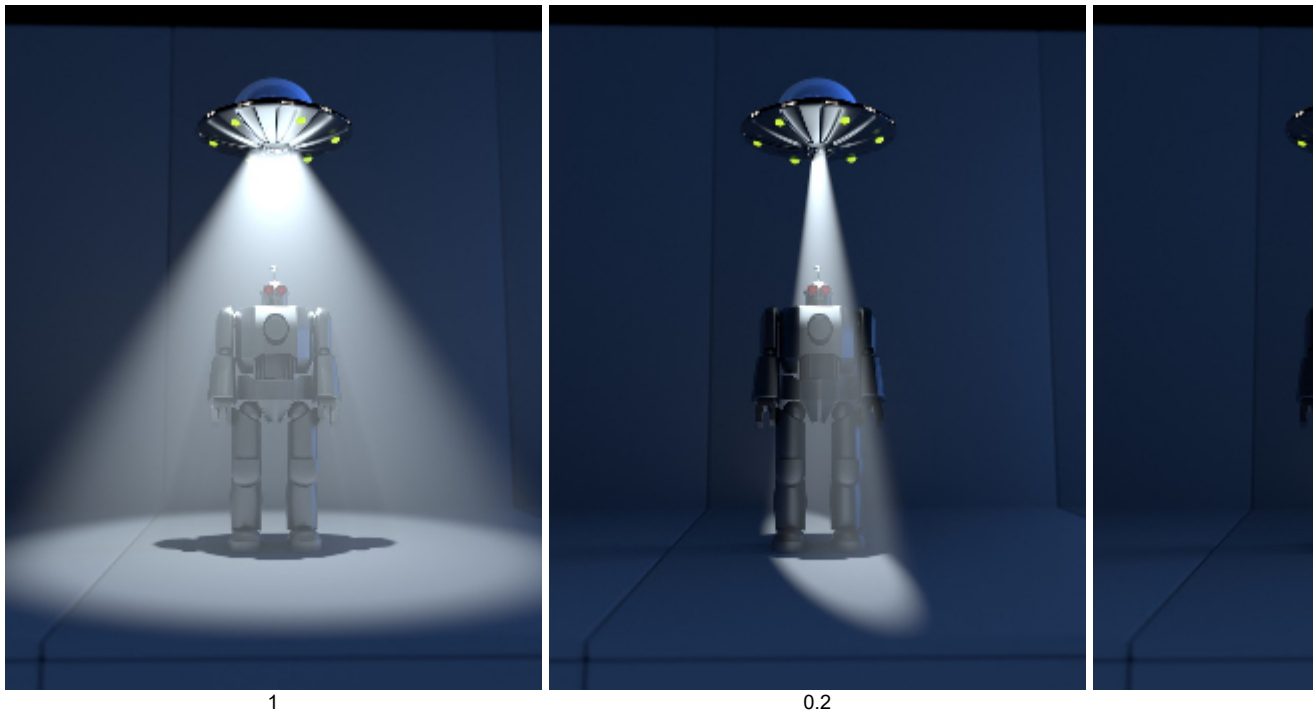
Scene illuminated with two Arnold spot lights, volume effects were enabled for illustration purposes

The [lights](#) page has more detail about the controls. The only additional parameters are:



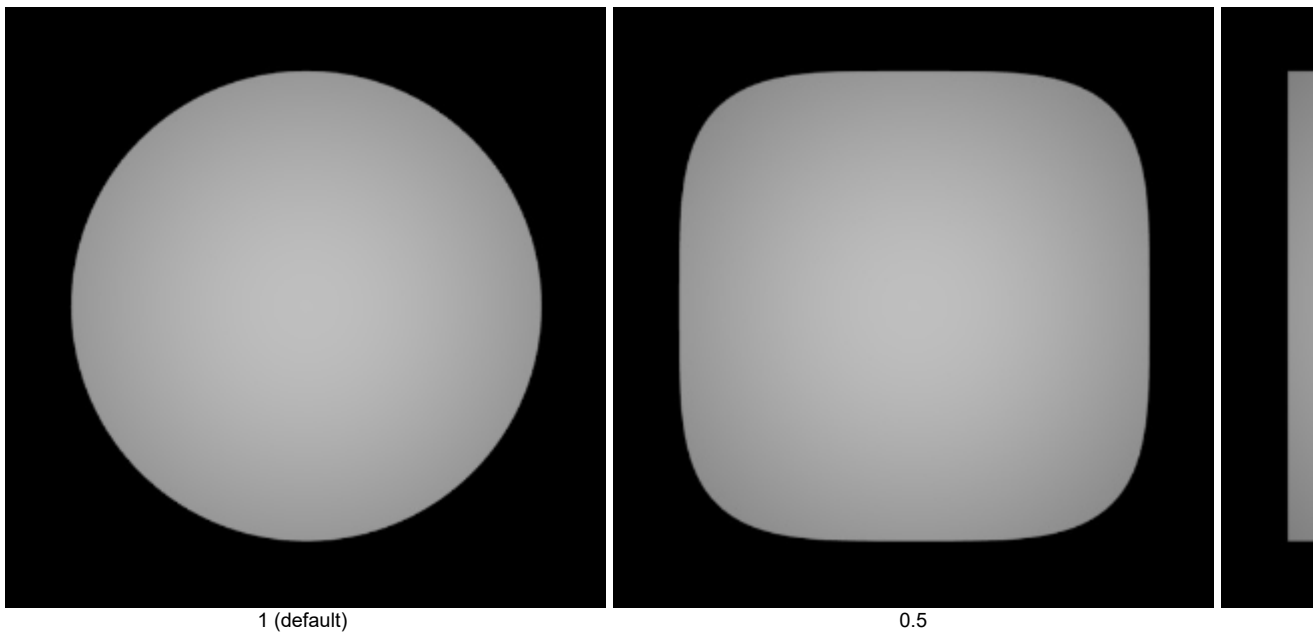
• Aspect Ratio

Set this to 1.0 (range is 0.0 to 1.0) for a cone with a circular cross-section, or to another value for an elliptical cross-section. Some theater lights, such as PAR cans, produce elliptical cross-sections.



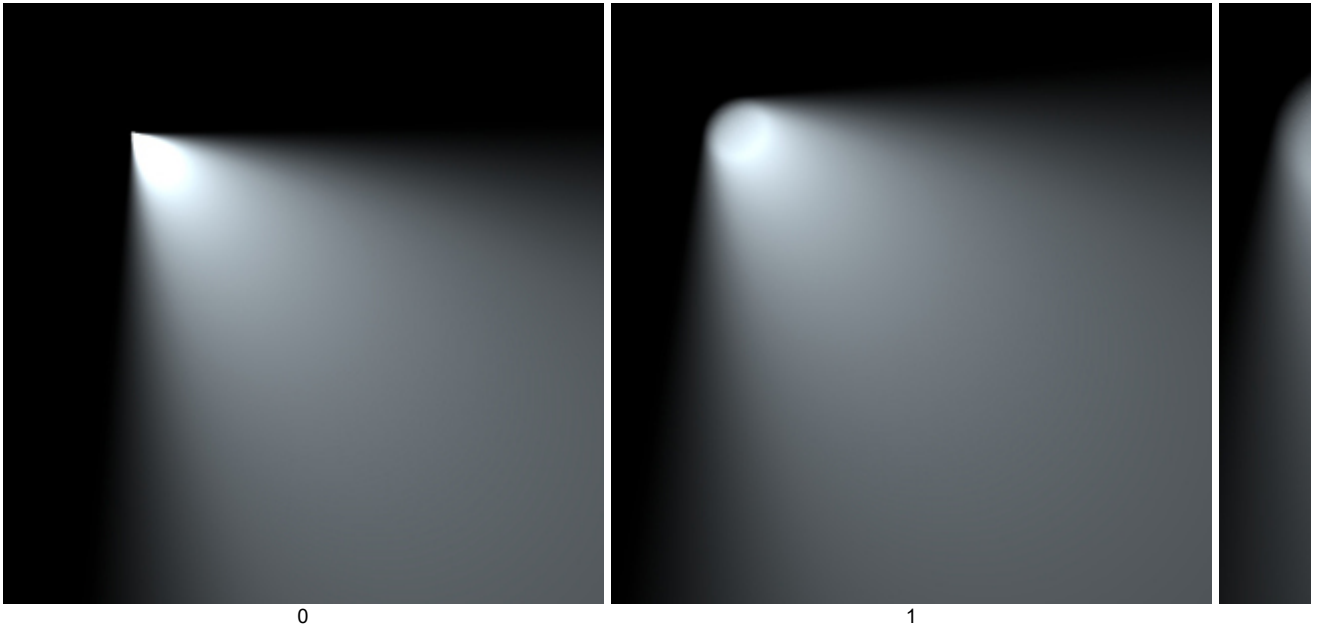
Roundness

A `roundness` parameter, going from a square shape at 0, to rounded corners, to a disk shape at 1.



Lens Radius

When set to zero, the vertex of the spot light cone coincides with the light's specified position. If non-zero, the cone becomes truncated, and the light is assumed to come from a virtual position that falls behind the light's specified position. This simulates the effect of real-world luminaires that bundle together a light bulb and a parabolic reflector inside a small, closed fixture. For example, a Hollywood-style searchlight has a big lens radius and a very small cone angle, producing an almost parallel beam of light whose intensity decreases with distance only very slowly, reaching all the way to the clouds. The effect of this parameter is more clearly seen with volumetrics.



A laser sword effect can be achieved by using a high 'lens radius' and a low cone angle (atmosphere volume enabled).

Cone Angle

The cone angle in degrees. No light will be emitted outside of this cone.

Penumbra Angle

This angle, measured in degrees from the outer edge of the cone towards the spotlight axis, specifies an area where the light's intensity smoothly falls off to zero at the cone edge.

•

Light Filters

- [Light Decay](#)
- [Gobo](#)
- [Barndoor](#)
- [Light Blocker](#)
- [passthrough_filter](#)

Arnold "light filters" are arbitrary shaders that can modify a light's output based on distance, position or other factors. This makes it easy to extend the built-in light sources with additional effects. Light filters can be added to Arnold lights from the Property/Arnold Properties/Lights/ menu.

Light Decay

This filter can be added to all the Arnold lights except skydome and distant. This is where you will specify the attenuation ranges (the type of distance fall-off used is specified on the light node itself).

Light Blocker

A light filter that when connected to a light will block out an arbitrary geometrically defined area of light. Multiple light blockers can be applied to the same light.

Gobo

This filter can only be used with spot lights. In theatrical stage lighting, a gobo (or "cookie") is a thin sheet of metal with holes used to break up the light beam into an irregular, more natural pattern such as tree leaves etc. In the computer graphics world, gobos are sometimes referred to as "slide projectors" (or sometimes "projector lights", but note that in Arnold the Gobo is not light source itself, but a filter applied to a spot light source). Any texture map or procedural shader can be projected through the light.

Barndoor

This filter can only be used with spot lights. Barndoors are opaque moving panels attached to the sides of the light's opening. They are typically used in theatrical and film lighting for additional control over the shape of the light beam. This Barndoor filter introduces four barndoor flaps. Each barndoor flap has three parameters. The first two parameters position the two ends of the flap across the face of the light. The third parameter (Edge) controls the softness of the edge. The positions are distances from the respective edge. The edge softness is a ramp width. The ramp starts at one (fully-open) at the barndoor's edge and tapers off to zero based on the edge width in the direction to the perimeter of the spotlight.

•

[passthrough_filter](#)

Sets or multiplies the light color by an input color parameter.

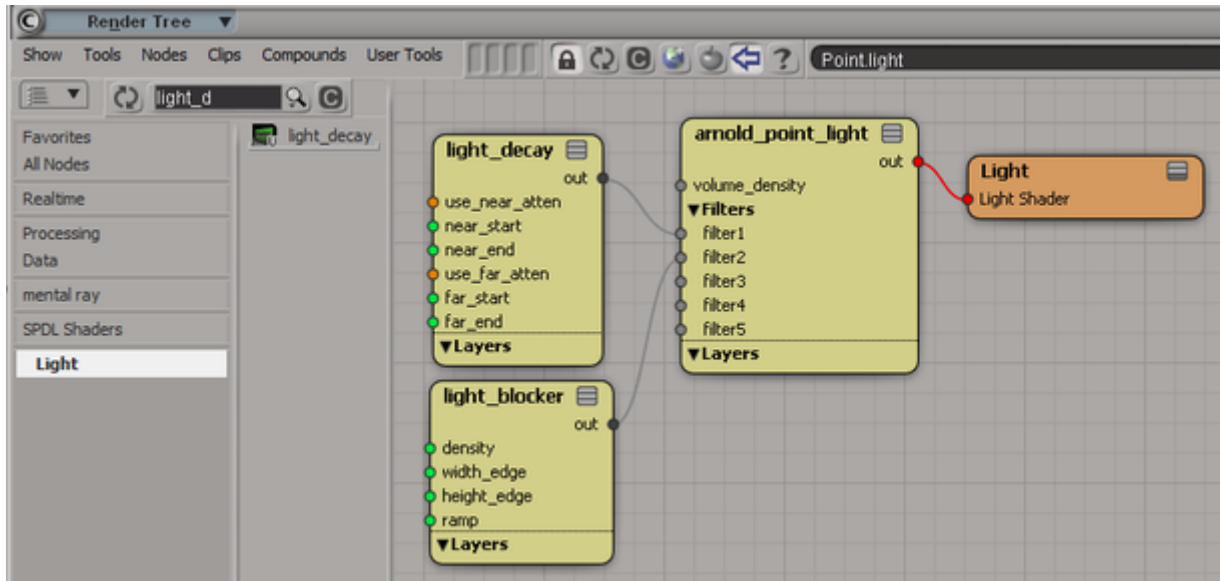
•

Light filters are shaders, categorized as light shaders. Each light has a set of ports allowing you to connect the filters. The connection can happen either by hand or by the provided menu or commands.

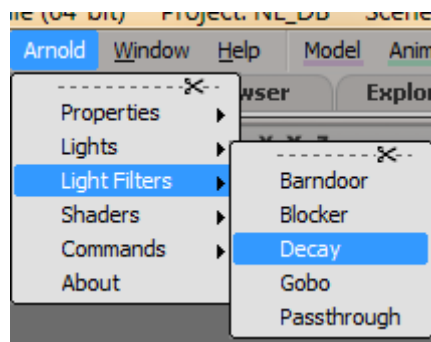
Applying filters •

You can connect a light filter either by:

1. Select the filter from the rendertree nodes, as you do with any other shader, and connect it to one of the available ports of the Arnold light shader.



2. Select one or more Arnold lights, and get the desired filter from the Arnold top menu.



3. Select one or more Arnold lights, and run one of the available scripting commands, depending on the filter type.

•

When using the menu or the commands, a check is automatically performed against the light and the filter type. For instance, an error message is fired if you try to connect a barndoor to a point light (barndoors are allowed only for spot lights). Also, the commands complain when you try to connect several barndoor or gobo filters to a given spot light.

For custom light filters, you'll have to connect it by hand, unless the developer also provides a dedicated command.

Commands

The available commands are `SITOA_AddBarndoorFilter`, `SITOA_AddGoboFilter`, `SITOA_AddLightBlockerFilter`, `SITOA_AddLightDecayFilter`, `SITOA_AddPassthroughFilter`. They have the same syntax and behavior. The syntax is (let's use the barndoor case)

```
SITOA_AddBarndoorFilter(in_collection, in_inspect)
```

where

`in_collection`: the light or collection of lights you want to apply the filter to
`in_inspect`: true if the filter shader must be inspected after being applied

The commands return the collection of applied filters, so you can further edit the filter's parameters in your script. Say for instance you have 2 Arnold lights named Point and Spot in your scene. The following script will apply a decay filter to them, enable the Near Attenuation flag, and set the Near Start/End values to 5 and 10.

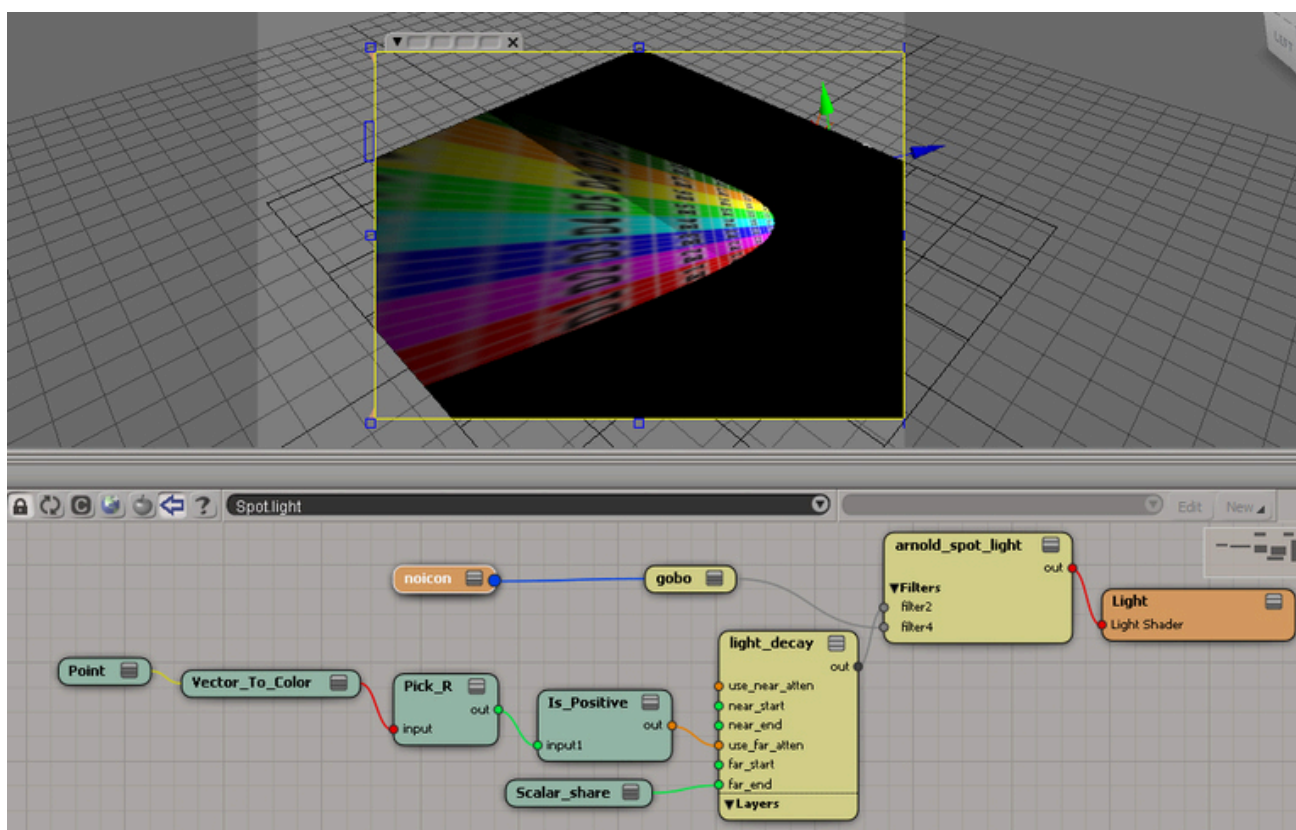
```
var filters = SITOA_AddLightDecayFilter("Point,Spot", false);
for (var i=0; i<filters.count; i++)
{
```

```
var filter = filters.Item(i);
filter.Parameters("use_near_atten").Value = true;
filter.Parameters("near_start").Value = 5;
filter.Parameters("near_end").Value = 10;
}
```

Filter texturing

Where allowed by the filter's evaluate function, the filter's parameters can be textured. In this example, the Arnold spot light has a regular gobo, and a decay filter.

The decay has far_end driven by a scalar_share cone, and the far attenuation is enabled only if the x coordinate of the lit point is positive.



Custom filters

Filters are shaders. So, if you want to write one, you should proceed exactly as when coding a shader, and you must provide a .dll/.so and a spdl file.

The following example code multiplies the light intensity by a scaling factor.

```
#include "ai.h"

enum sj_light_filterParams
{
    p_mute,
    p_scale
};

AI_SHADER_NODE_EXPORT_METHODS(sj_light_filterMethods);
```

```
node_parameters
{
    AiParameterBool("mute", false);
    AiParameterFlt ("scale", 1.0f);
}

shader_evaluate
{
    if (! AiShaderEvalParamBool(p_mute))
        sg->Liu *= AiShaderEvalParamFlt(p_scale);
}

node_initialize {}
node_update {}
node_finish {}

node_loader
{
    if (i > 0)
        return FALSE;
    node->methods      = sj_light_filterMethods;
    node->output_type  = AI_TYPE_RGB;
    node->name         = "sj_light_filter";
    node->node_type    = AI_NODE_SHADER;
    strcpy(node->version, AI_VERSION);
    return TRUE;
}
```

Although the light shaders ports accept any type of input filter, if you want to be consistent with the gray color of the ports, you should set the `spd`'s output type to `model`, and its `guid` to `"{0F15C5DD-A3CC-418B-9FF2-08C5BF5FB6C3}"` (the `model` keyword is used in mental ray for defining geometry shaders).

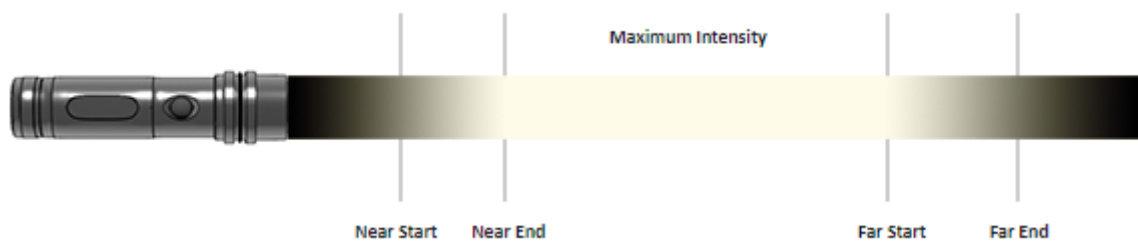
```
...
PropertySet "sj_light_filter"
{
    Parameter "out" output
    {
        guid = "{0F15C5DD-A3CC-418B-9FF2-08C5BF5FB6C3}";
        type = model;
    }
    ...
}
```

Light Decay



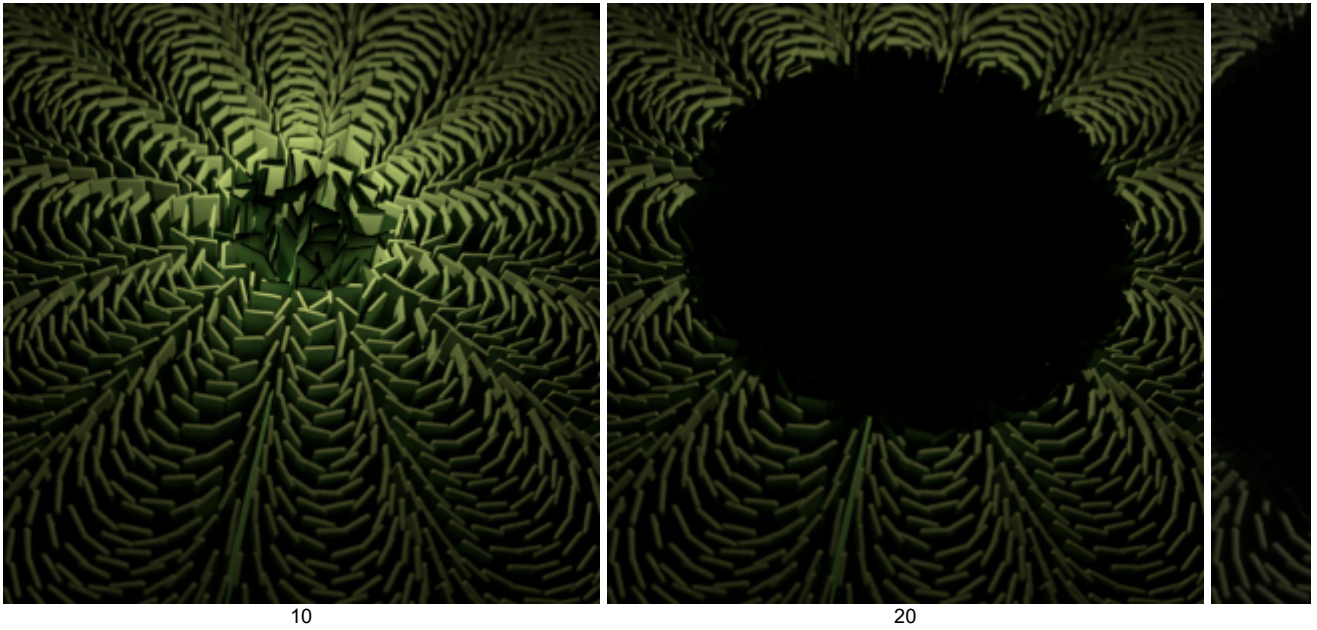
light_decay -> disk_light . far_end attenuation keyframed.

This filter can be added to all Arnold lights. This is where you will specify the attenuation ranges. All lights in Arnold use a physically-based falloff by default, but the light_decay filter can be used to adjust the falloff for artistic purposes.



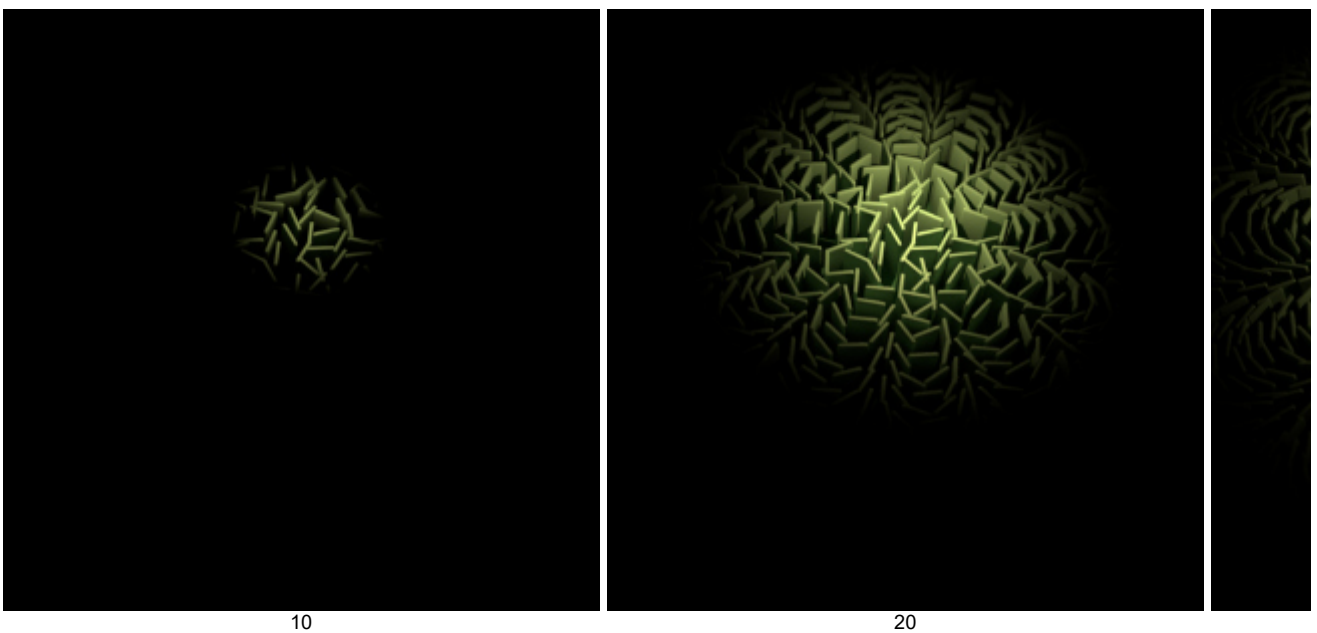
Use Near Attenuation

The Near Attenuation value sets the distance at which the light intensity "fades in". It works like a reverse decay. The light does not shine from the source to the Start range of the Near Attenuation then increases brightness until the end range of Near Attenuation.



Use Far Attenuation

The Far Attenuation value sets the distance at which the light "fades out". It works like a normal decay, but you can specify the start and end distance. The light diminishes from the Far Start range to the Far End range using a smooth ramp between.



Near Start

The distance at which the light begins to fade in. •ie, 0 = light source. •



2



4



8

Near End

The distance at which the light reaches its full value. •



5



10



20

Far Start

The distance at which the light begins to fade out.



5



10



20

Far End

The distance at which the light has faded out to 0. •



5



10



20

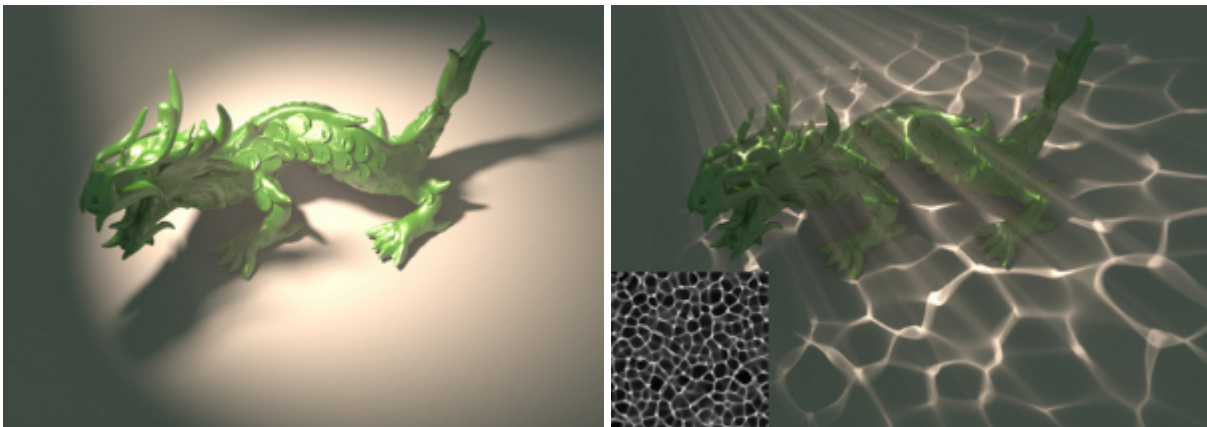
-
-
-

Gobo



Gobo filter -> spot light. Image -> slide_map.

The **gobo** filter can only be used with **spot** lights. In theatrical stage lighting, a **gobo** (or "cookie," slang for **Cucoloris**) is a thin sheet of metal with holes used to break up the light beam into an irregular, more natural pattern such as tree leaves, etc. In the computer graphics world, **gobos** are sometimes referred to as "slide projectors" (or sometimes "projector lights," but note that in Arnold the **gobo** is not a light source itself, but a filter applied to a **spot** light source). Any texture map or procedural shader can be projected through the light.

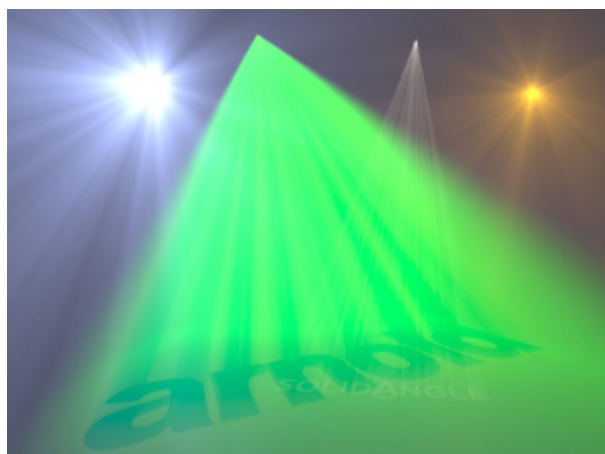


Spotlight without gobo filter

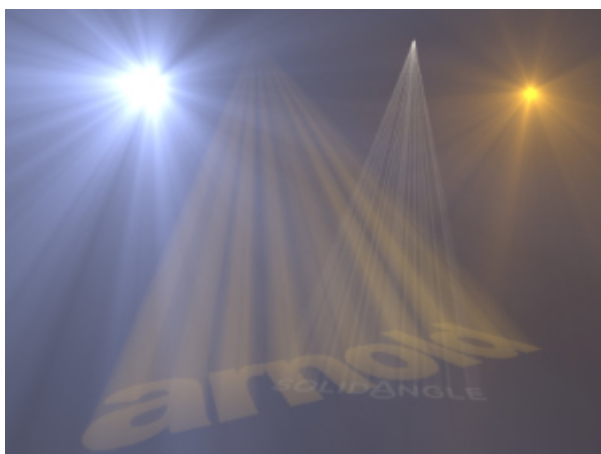
Spotlight with gobo filter

Filter Mode

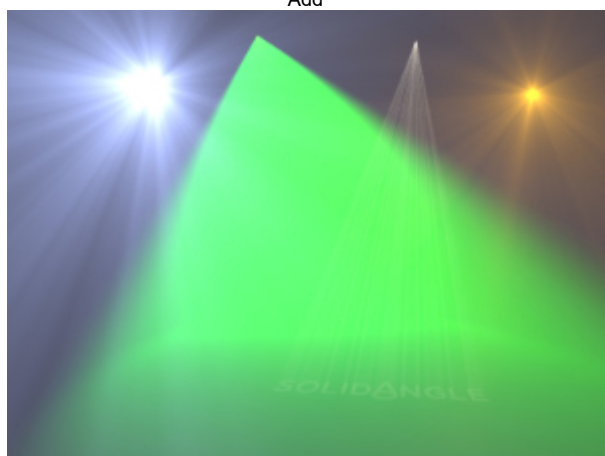
The blending equation by which the gobo's slide map is combined with the lights output. 'Blend' creates a default gobo effect. 'Replace' multiplies the slide map over the spot light. 'Add' multiplies the spot light over the gobo. 'Sub' inverts the RGB values of the slide map. 'Mix' averages the results equally between the spotlight and the slide map.



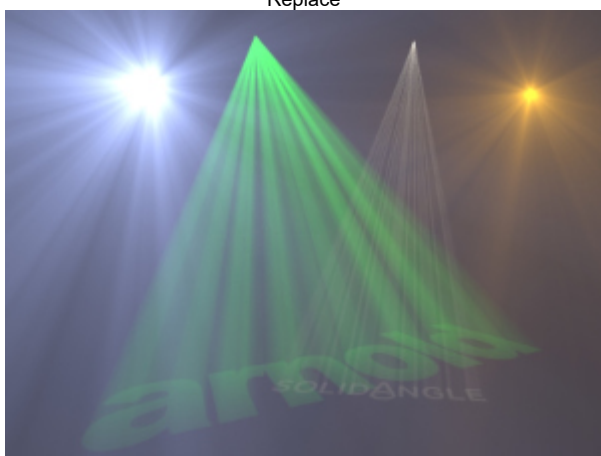
Add



Replace



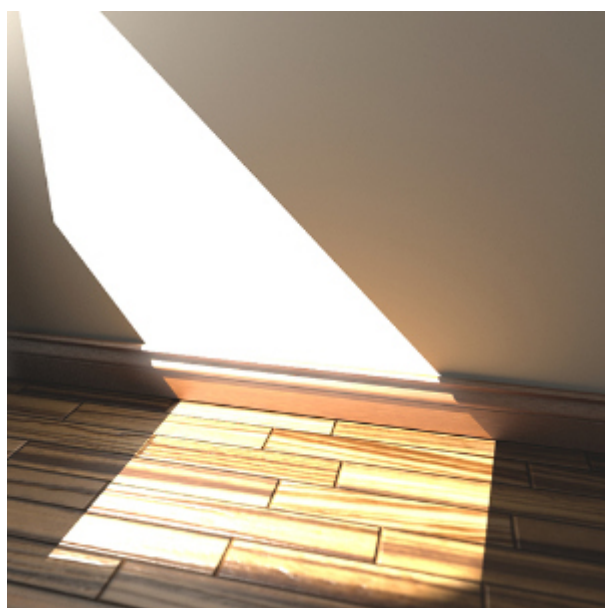
Mix



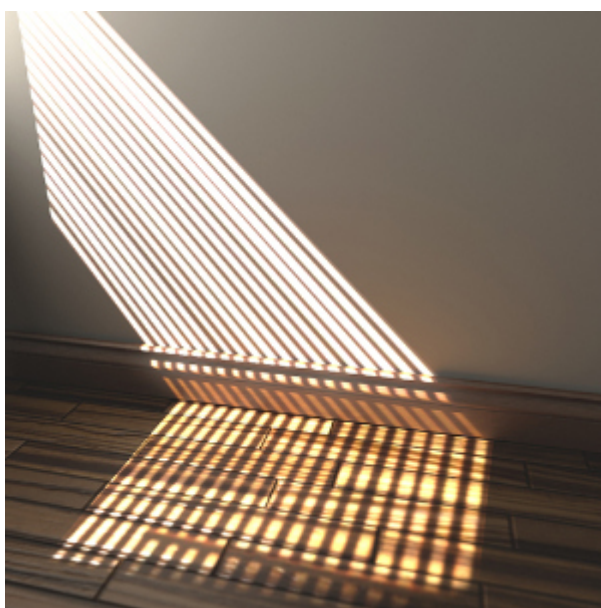
Blend (default)

Slide Map

The texture map used to create the gobo effect.



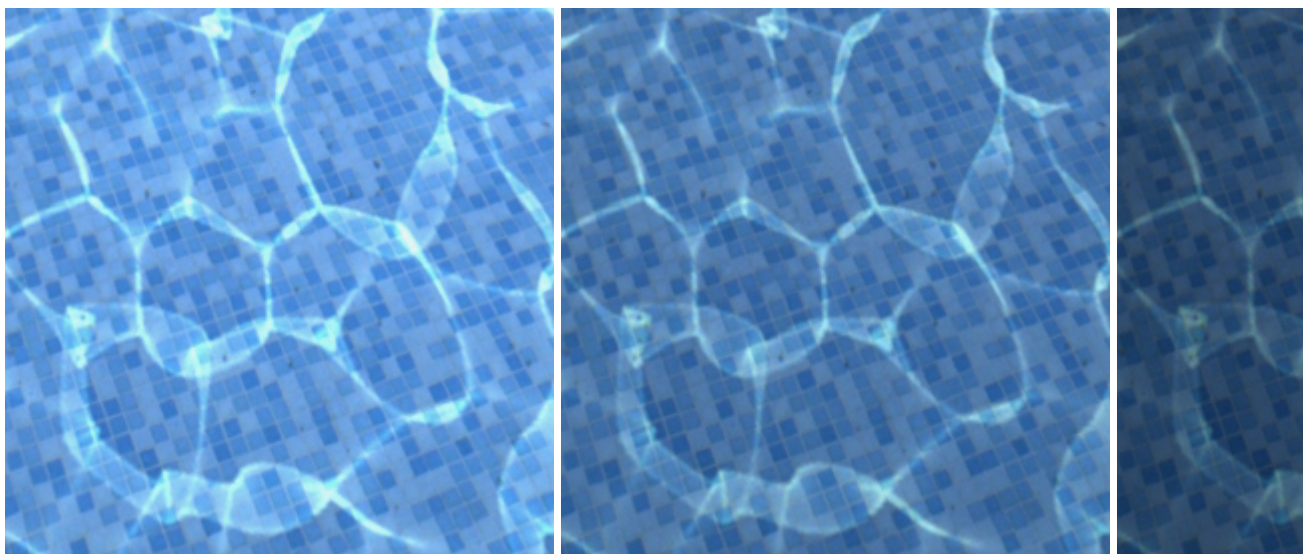
No slide_map



With slide_map

Density

Controls the density of the gobo. Higher values will make the gobo more opaque, thereby letting less light through.

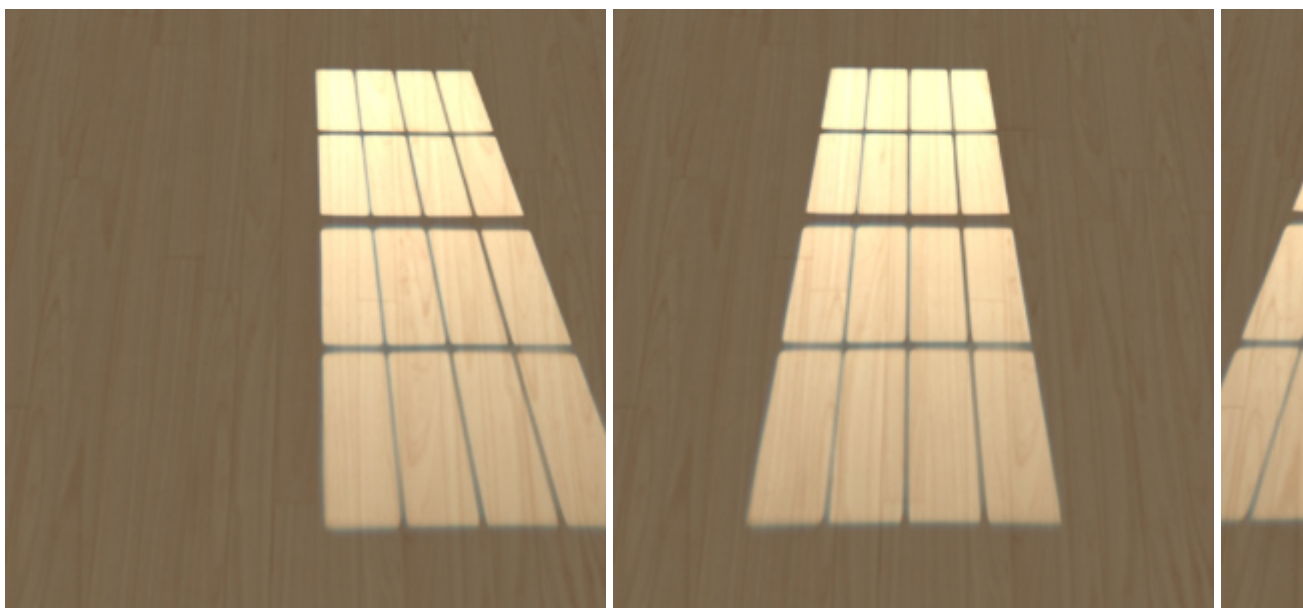


0 (default)

0.5

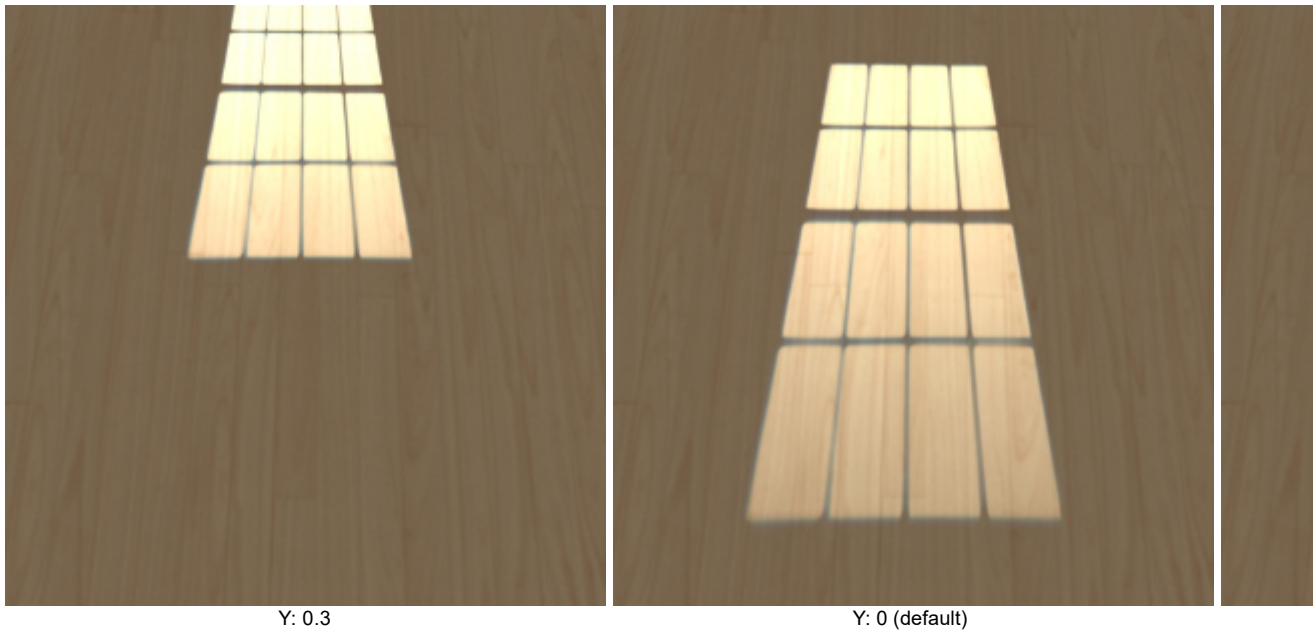
Offset

UV-coordinate values used to offset the direction of the slide map.



X: 0.3

X: 0 (default)



Y: 0.3

Y: 0 (default)

Scale S

Scales the texture used for the slide map in the S direction.

Scale T

Scales the texture used for the slide map in the T direction.

MultiExcerpt named 'scale_st_images ' was not found

The page: gobo was found, but the multiexcerpt named 'scale_st_images ' was not found. Please check/update the page name used in the 'multiexcerpt-include macro.

Wrap S

Controls how the 2d texture map is repeated on a surface in the S direction.

Wrap T

Controls how the 2d texture map is repeated on a surface in the T direction.

Periodic

MultiExcerpt named 'periodic_images ' was not found

The page: gobo was found, but the multiexcerpt named 'periodic_images ' was not found. Please check/update the page name used in the 'multiexcerpt-include macro.

Mirror

MultiExcerpt named 'mirror_images ' was not found

The page: gobo was found, but the multiexcerpt named 'mirror_images ' was not found. Please check/update the page name used in the 'multiexcerpt-include macro.

File

MultiExcerpt named 'file_images ' was not found

The page: gobo was found, but the multiexcerpt named 'file_images ' was not found. Please check/update the page name used in the 'multiexcerpt-include macro.



Clamp

MultiExcerpt named 'clamp_images ' was not found

The page: gobo was found, but the multiexcerpt named 'clamp_images ' was not found. Please check/update the page name used in the 'multiexcerpt-include macro.

Black

MultiExcerpt named 'black_images ' was not found

The page: gobo was found, but the multiexcerpt named 'black_images ' was not found. Please check/update the page name used in the 'multiexcerpt-include macro.

-

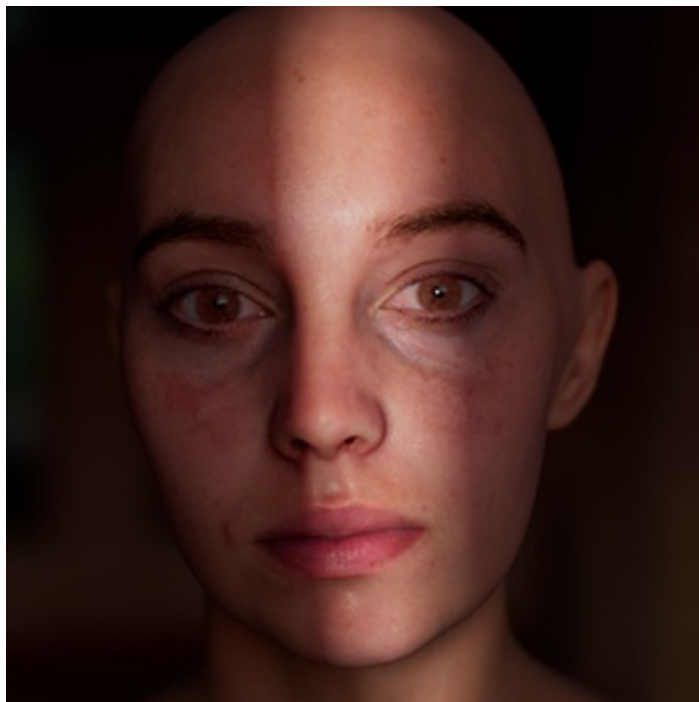
-

Example Renders

MultiExcerpt named 'example_renders ' was not found

The page: gobo was found, but the multiexcerpt named 'example_renders ' was not found. Please check/update the page name used in the 'multiexcerpt-include macro.

Barndoor



barndoor.bottom_edge : 0.1. Rollover image to view 0.3.

This light filter can only be used with spot lights. Barn doors are opaque moving panels attached to the sides of the light's opening. They are typically used in theatrical and film lighting for additional control over the shape of the light beam. This barndoor filter introduces four barn door flaps. Each barn door flap has three parameters. The first two parameters position the two ends of the flap across the face of the light. The third parameter (edge) controls the softness of the edge. The positions are distances from the respective edge. The edge softness is a ramp width. The ramp starts at one (fully-open) at the barndoor's edge and tapers off to zero based on the edge width in the direction to the perimeter of the spotlight. For example, a flap with coordinates 0.25, 0.25, and an edge of 0.125 will be perfectly "open" at 0.25 and will be perfectly "closed" at 0.375. The edge effect starts at the barndoor edge and works its way outward (*not* inward).



Without barndoor



With barndoor

*Shape of light beam controlled by a barndoor filter connected to a spotlight

MultiExcerpt named 'synopsis_images ' was not found

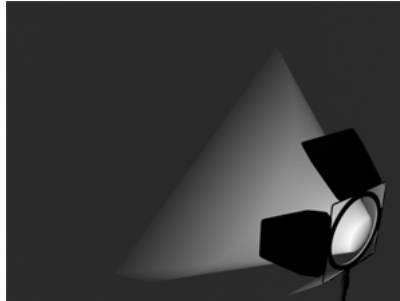
The page: barndoor was found, but the multiexcerpt named 'synopsis_images ' was not found. Please check/update the page name used in the 'multiexcerpt-include macro.

•

The images below show the various attributes of the aiBarndoor filter (the spotlight model is purely for illustrative purposes).

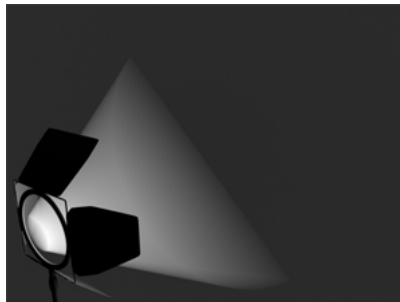
Top Left

Moves the left corner of the top flap across the face of the light.



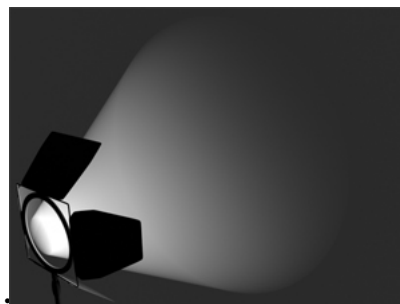
Top Right ••

Moves the right corner of the top flap across the face of the light.



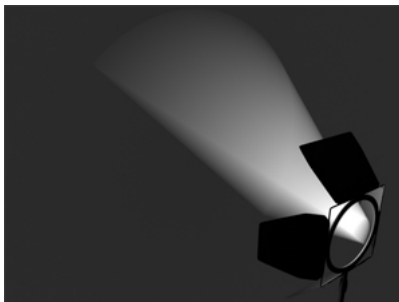
Top Edge

Controls the edge softness of the top flap.



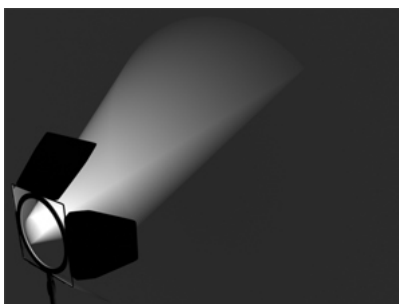
Bottom Left

Moves the left corner of the bottom flap across the face of the light.



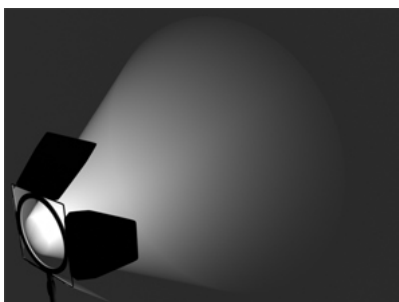
Bottom Right

Moves the right corner of the bottom flap across the face of the light.



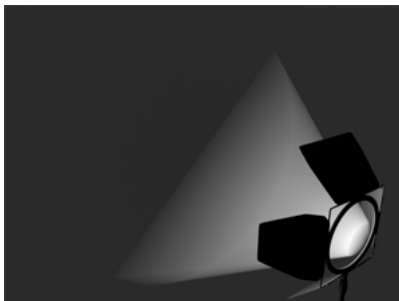
Bottom Edge

Controls the edge softness of the bottom flap.



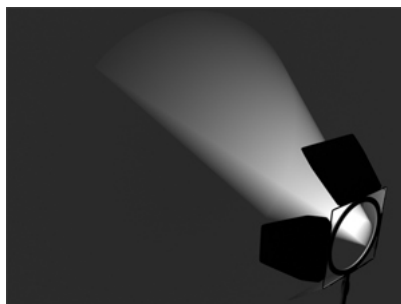
Left Top

Moves the top corner of the left flap across the face of the light.



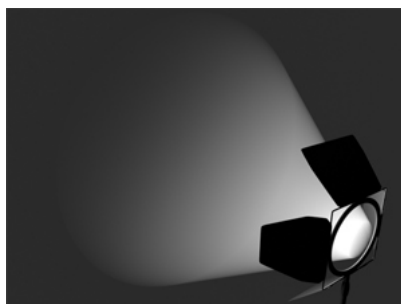
Left Bottom

Moves the bottom corner of the left flap across the face of the light.



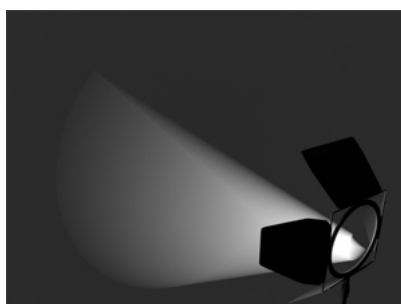
Left Edge

Controls the edge softness of the left flap.



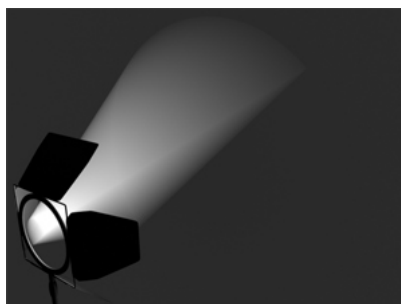
Right Top

Moves the top corner of the right flap across the face of the light.



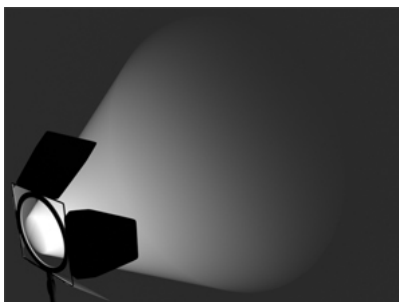
Right Bottom

Moves the bottom corner of the right flap across the face of the light.



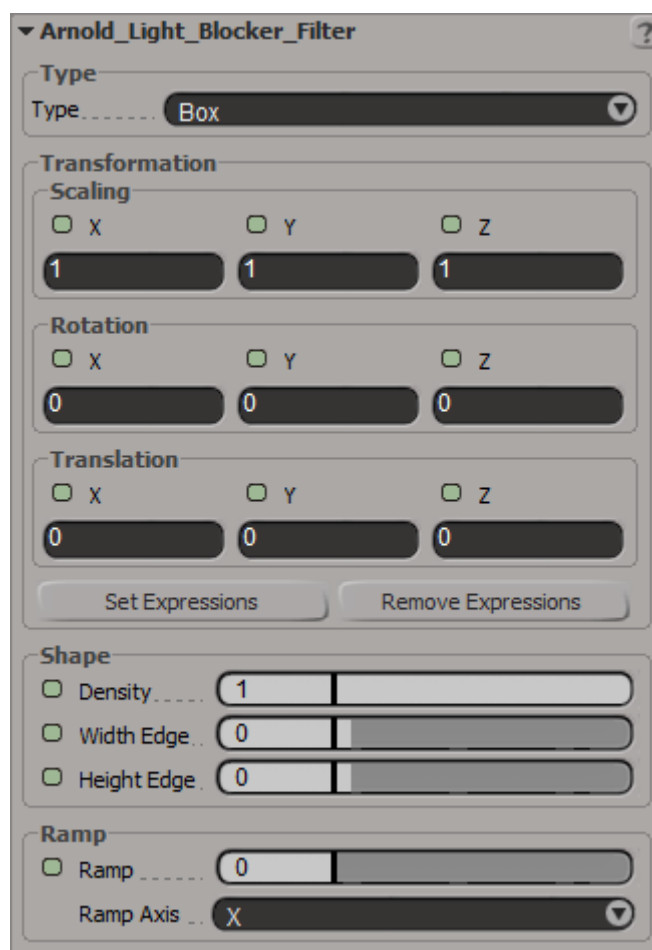
Right Edge

Controls the edge softness of the right flap.



.

Light Blocker



Light Blocker attributes



Rollover image to view light_blocker

Light blockers give flexibility to the lighting TD. They can be used as an artificial method of masking light in a scene without the overhead of adding additional geometry. Used carefully, they provide a degree of artistic freedom, allowing you to define the light boundaries in non-physical ways.

The light_blocker is a light filter that specifies a primitive volume defined in the Geometry Type (box, sphere, cylinder, or plane) which when connected to a light node will block or modify the light as it passes through that volume. The blocker will only influence a light it is connected to, just like other filters (i.e. the geometry affects nothing else, including GI, and is invisible). One, simplistic use would be to mimic the effect of a complex lamp housing, (possibly in combination with the barndoor filter) without modeling the lamp geometry. Another use would be to create shadows in your scene that look different from how they normally would.



You can also have multiple light blockers on the same light.



A video showing the LightBlocker in use can be found [here](#).

In the following examples an LightBlocker is connected to an AreaLight.

The translation, rotation and scale of the light blocker are determined by a simple wireframe shape node. By default, the position of the light blocker will appear at 0,0,0.

Geometry Type

Determines the shape of the blocked light. A light blocker can be a box, cylinder, sphere, or plane.



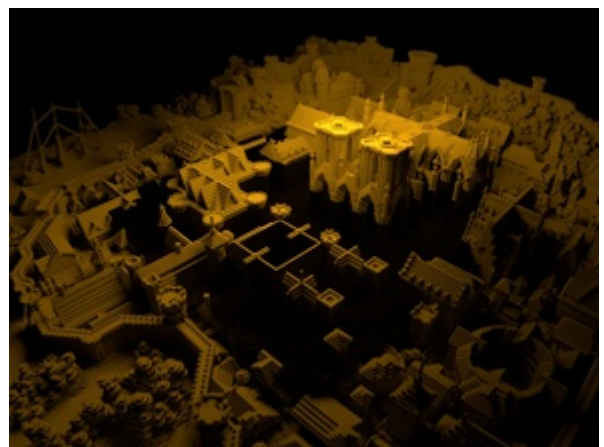
Box . Rollover image for GUI (Maya).



Cylinder . Rollover image for GUI (Maya).



Sphere . Rollover image for GUI (Maya).



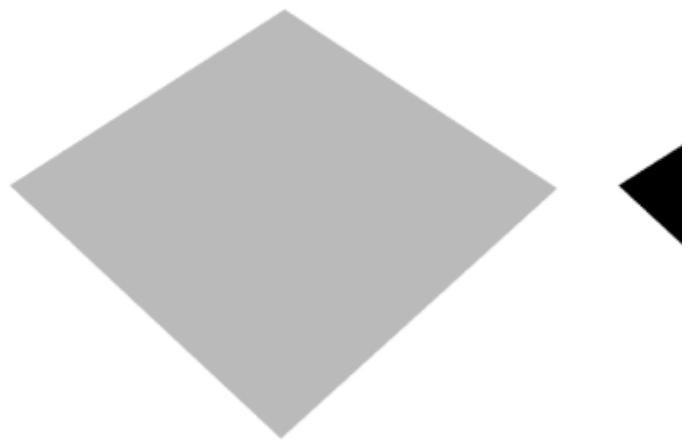
Plane . Rollover image for GUI (Maya).

Density

This value is the strength of the light_blocker effect. The light_blocker will not be apparent unless the density value is above 0. •



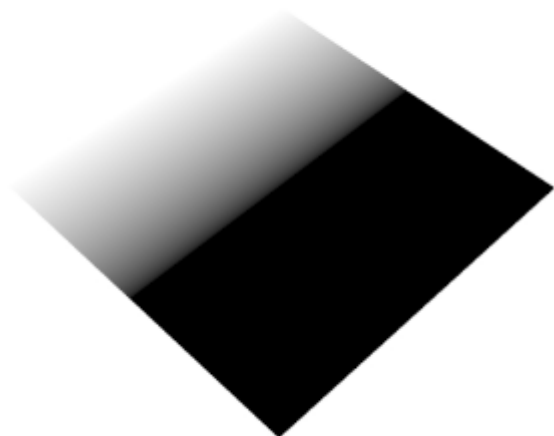
0.2



0.5

Axis

Attenuates the ramp based on the direction set. ••



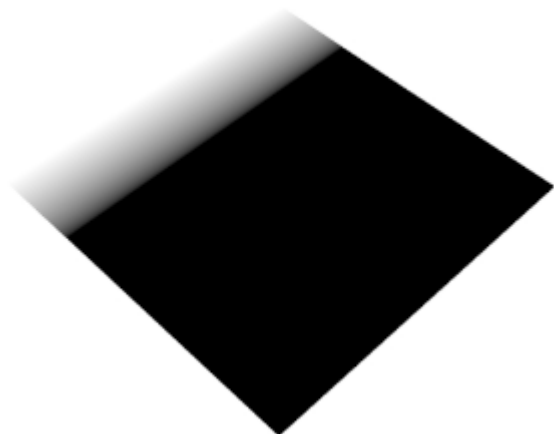
x



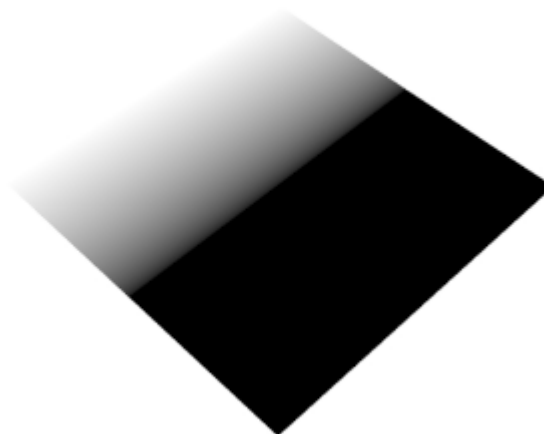
y

Ramp

This is the magnitude of the ramp multiplier, applying along the Ramp Axis direction. •Negative values flip the Ramp Axis direction.



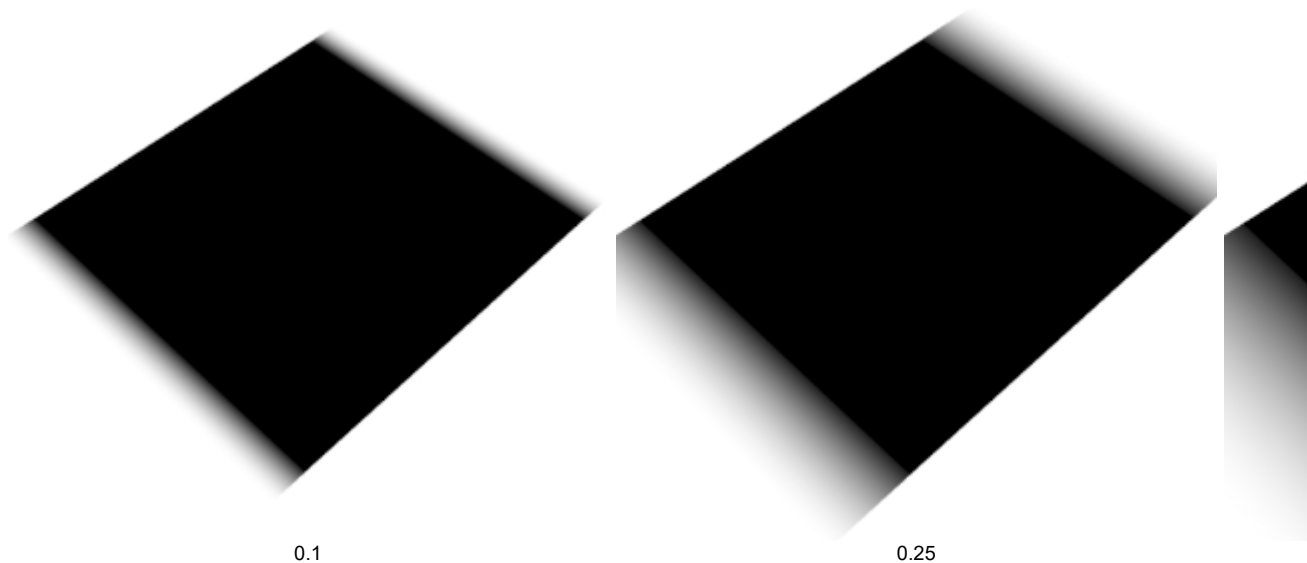
0.25



0.5

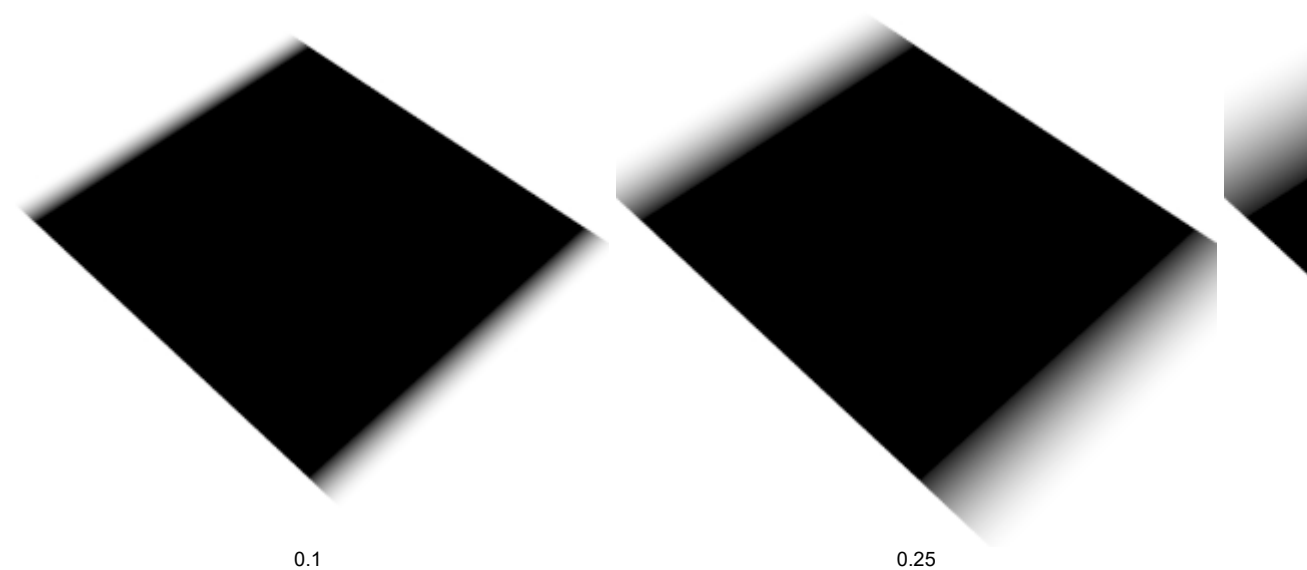
Height Edge

Attenuates the edge of the height of the light_blocker. •



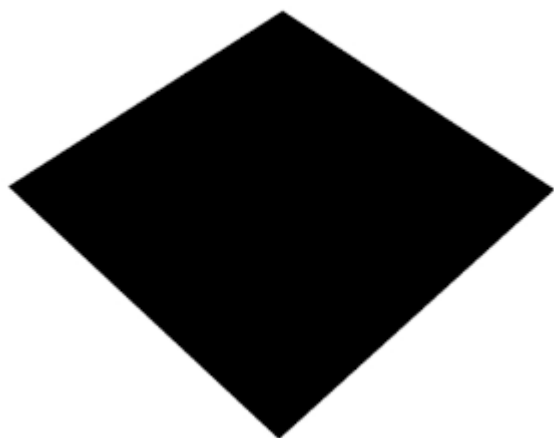
Width Edge

Attenuates the edge of the width of the light_blocker. •

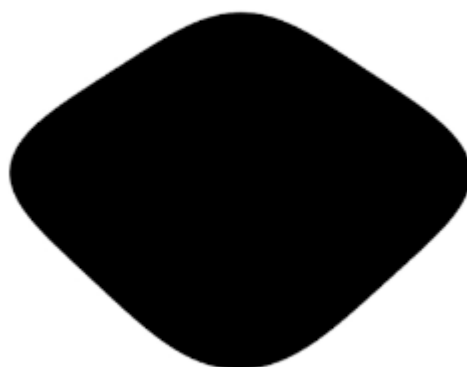


Roundness

Increases the circular shape of plane light blockers.



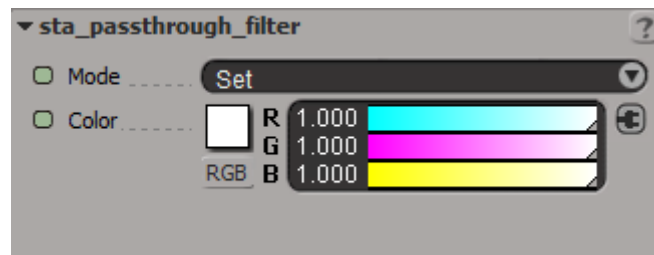
0



0.5

-
-

passthrough_filter



•

This filter overrides or multiplies a light's color by an input color.

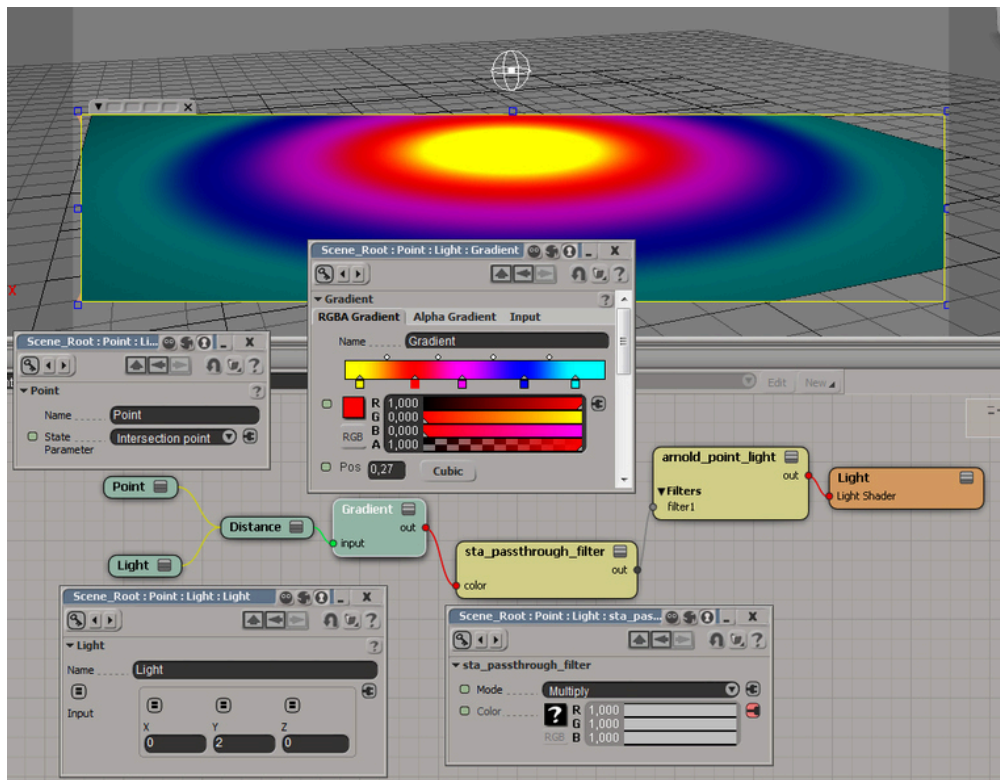
It allows plugging any shader as a filter, making the light color fully texturable (ie networkable), which right now is not allowed by Arnold.

The Mode parameter can either be:

- Mute: do nothing
- Set: overrides the light color with Color
- Multiply : multiplies the light color by Color

•

In this example, Color is a gradient, driven by the distance between the light (set as an expression) and the lit point (a vector state shader set to Point).



•

•

Camera

The Arnold core renderer provides several separate camera nodes: perspective, orthographic, fisheye, cylindrical and spherical.

For a standard perspective camera, you can simply rely on the Softimage perspective camera. Stereo cameras, belonging to a group that is assigned as the pass camera, are also supported when rendering interactively or from xsibatch.

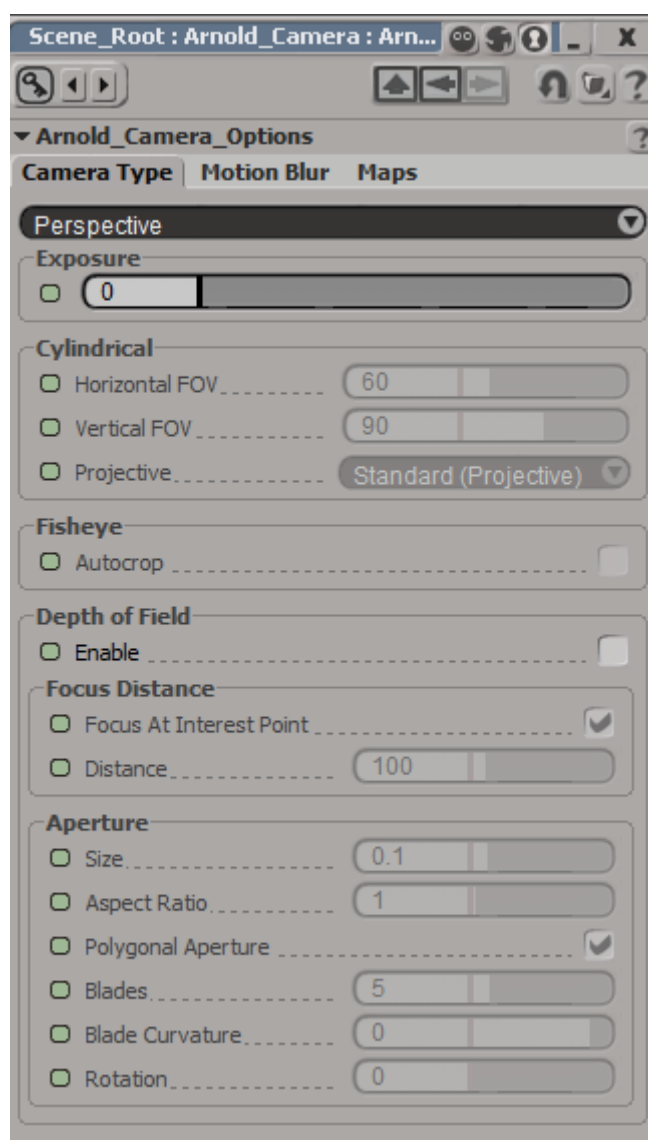
For an orthographic camera, simply turn the projection type of the camera to Orthographic.

If you need a different type of camera, or to access the camera-specific parameters, you can either:

- Select a camera and apply the [Camera Options](#) property.
- Get a new camera from Arnold->Cameras or Get->Primitive->Camera

The first method is recommended if you're fine with the current Softimage camera, and you just want to tune attributes specific to Arnold, like the DOF and motion blur settings.

The latter, will create a Softimage perspective camera, and apply the Camera Options property already set, based on the camera type you selected.



Camera Type

The supported types are:

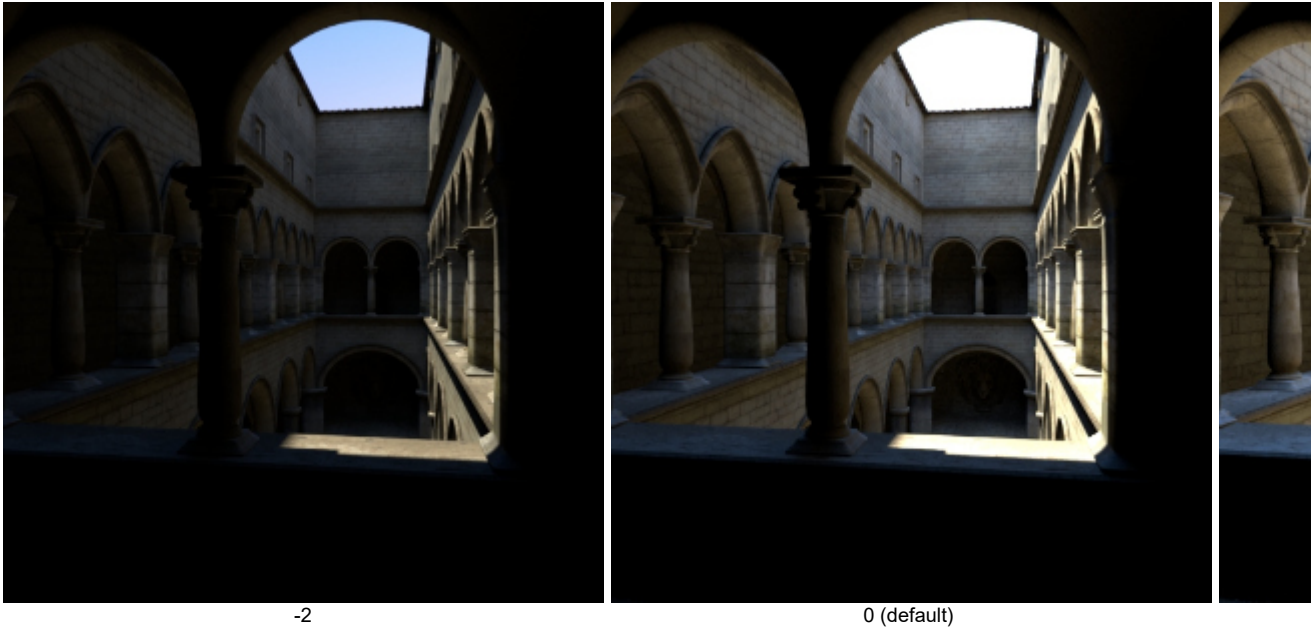
- perspective

- cylindrical•
- fisheye•
- spherical

You can also use custom camera nodes: set the camera type to Custom (lens shader) and connect your node at the top of the lens shaders stack of the camera.

Exposure

Simulates the effect of camera exposure (in a non-physical way). Increasing this parameter by a value of one gives you one stop up (doubles the brightness).



Depth of Field

This group controls the Depth of Field settings. Further examples can be found [here](#).

Enable

Enables depth of field effects.



Focal Distance

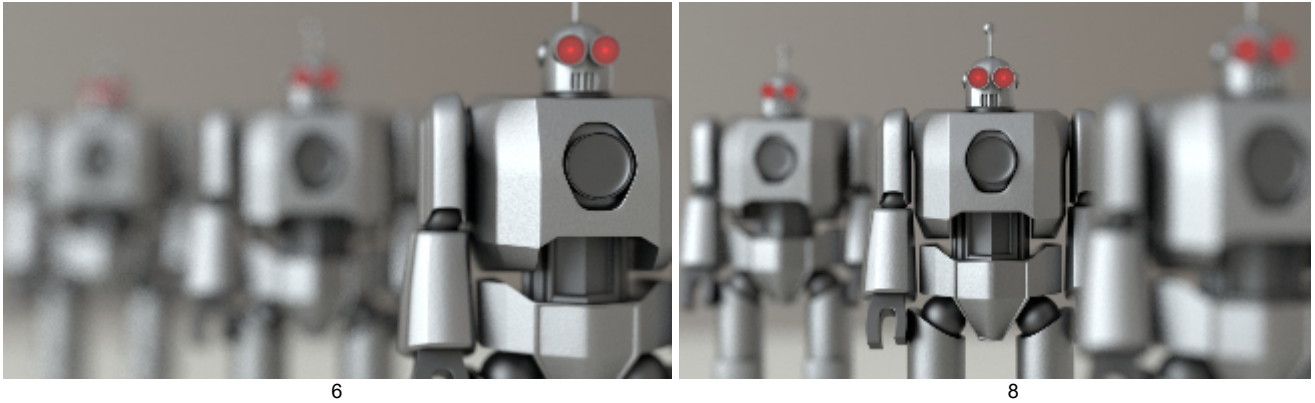
Focus at Interest Point

If enabled, the focal distance will be automatically set to the distance from the camera to the camera interest point.

Distance

The distance at which objects appear in perfect focus, for a non-zero aperture value, in case **Focus at Interest Point** is disabled.

This is the distance at which objects appear in perfect focus, for a non-zero aperture value.



Aperture

These settings define the geometry of the camera's aperture and its bokeh effects.

Size

The radius of the aperture in world units. The smaller the aperture, the sharper the images (shallower depth of field). In the limit, a size of zero produces no depth of field blurring.

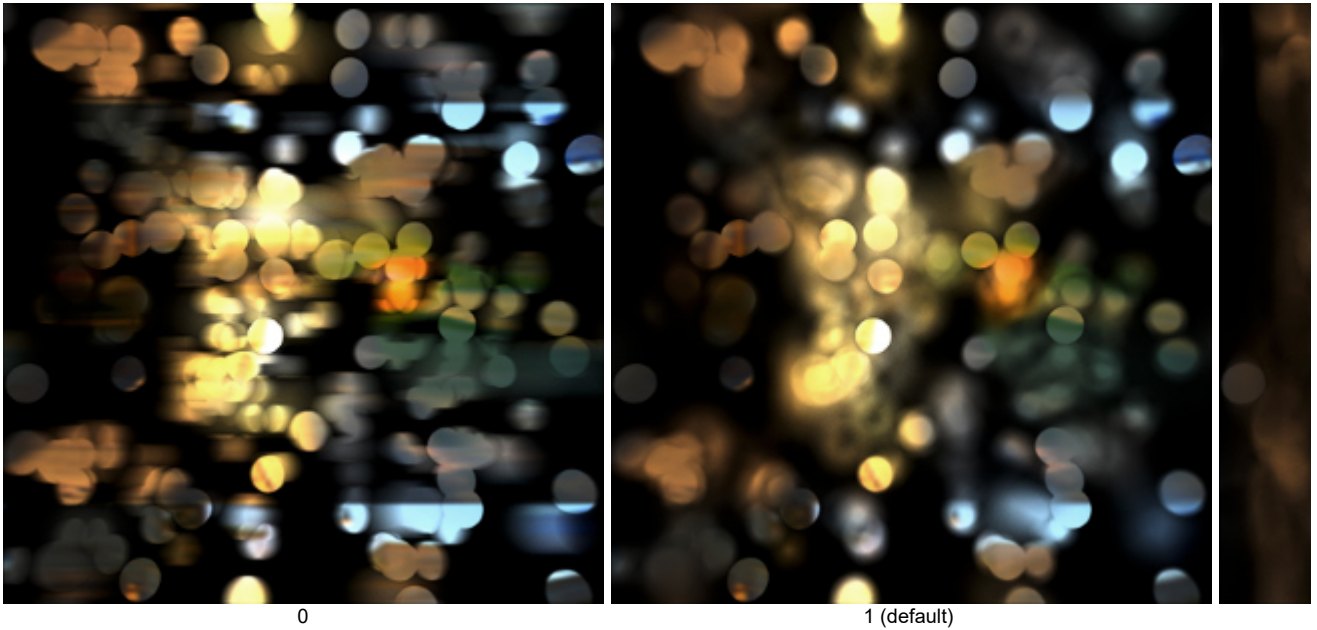
The radius of the aperture in world units. The smaller the aperture, the sharper the images (shallower depth of field). In the limit, a size of zero produces no depth of field blurring.



Aspect Ratio

Used to stretch the aperture vertically. A value of 2 produces an elongated defocusing effect reminiscent of an anamorphic lens, while a value less than one will squash it.

Values bigger than one produce an elongated defocusing effect, reminiscent of an anamorphic lens, while a value less than one will squash it.



•

Polygonal Aperture

Determines whether the aperture will behave as a regular polygon of N-sides or as a perfect disk.

Blades

A number of blades (or polygon sides) of the polygonal aperture. 0 is considered a circle aperture



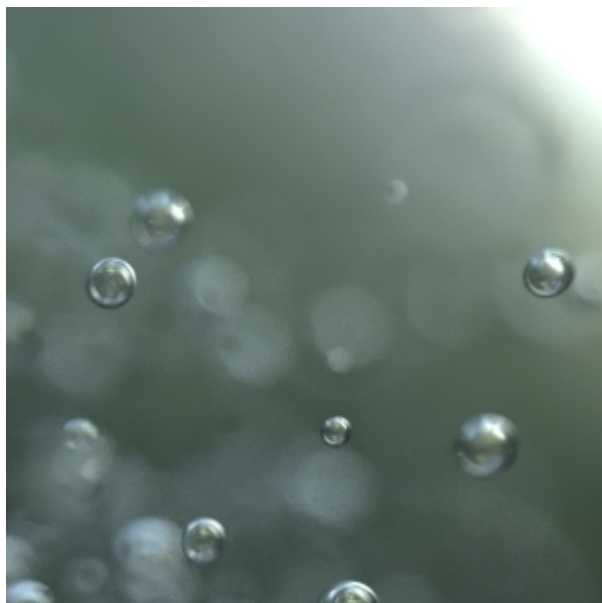
Blade Curvature

The curvature of the polygonal aperture sides. A value of 0 means hard straight sides. Increasing this value results in progressively more curved edges, all the way to 1.0 which produces a perfect disk. Negative values produce a "pinched" or star-shaped aperture.

The curvature of the polygonal aperture sides. A value of 0 means hard straight sides. Increasing this value results in progressively more curved edges, all the way to 1.0 which produces a perfect disk. Negative values produce a "pinched" or star-shaped aperture.



0

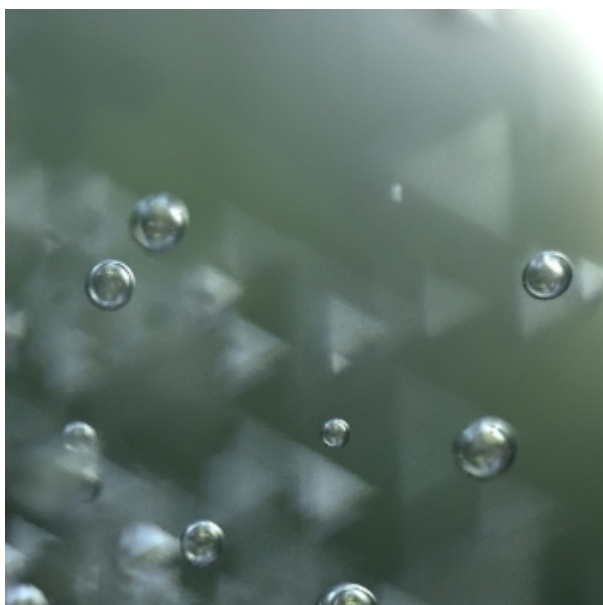


1

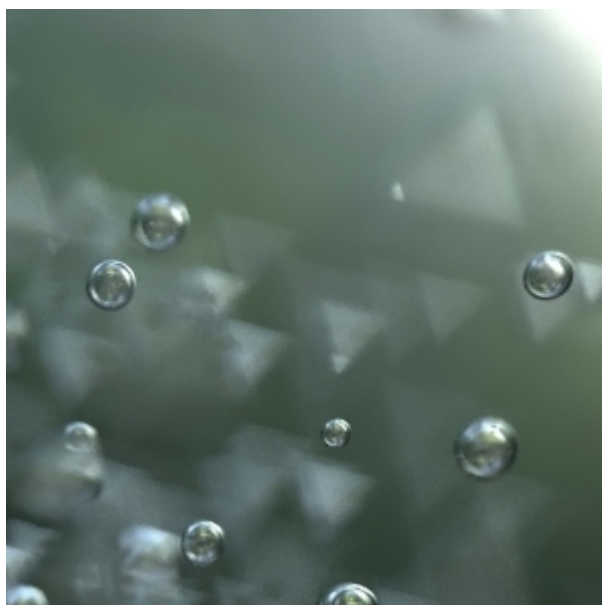
Rotation

Rotation angle (in degrees) of the aperture.

Rotates the aperture by the specified number of degrees. •



0 (default)



15

Perspective

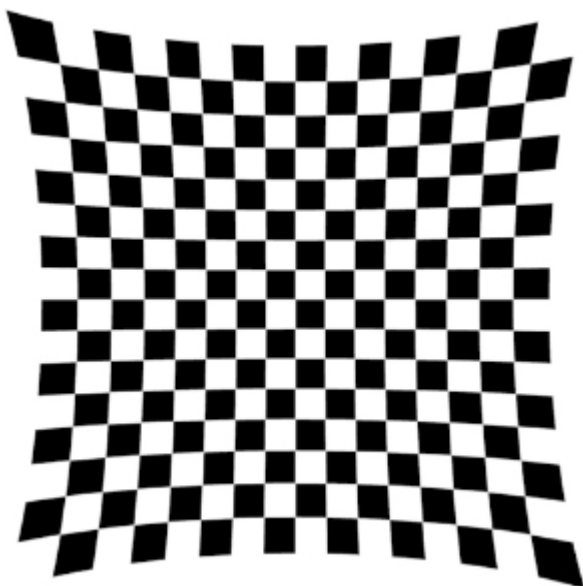
This camera node provides a perspective view, like a regular camera. This is the 'standard' camera type that you will probably use most often. In addition to a normal perspective, this camera type also provides controls for the depth of field, and the focal length of the camera, as well as controls, to do with the aperture of the camera. This allows various camera lens effects, including bokeh, to be simulated.

The [cameras](#) page has more details about the controls.

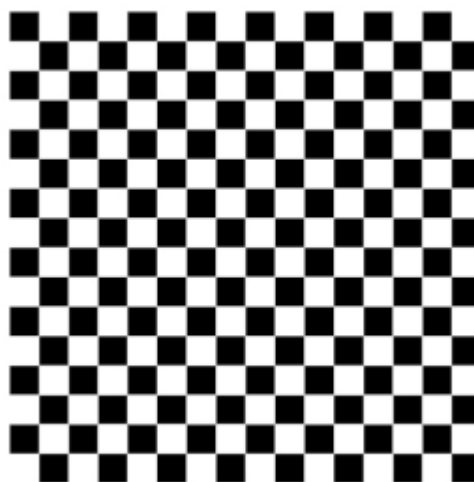
•

Radial Distortion

Radial distortion, with negative and positive values resulting in pincushion and barrel distortion respectively. Real-world lenses can show such distortion when using, for example, a very short focal length or a high zoom.



-0.1 (pillow distortion)



0 (no distortion)



Radial Distortion (-1.3 to 1)

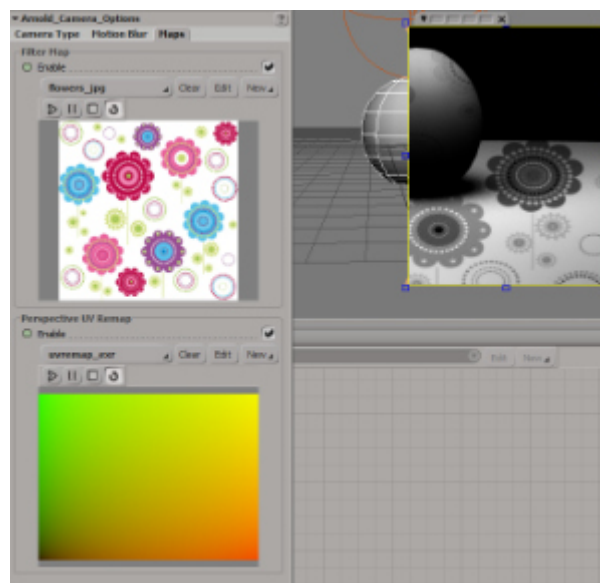
•

A 'barrel distortion' camera lens effect has been reproduced in the interior scene below, by simply increasing the Radial Distortion to 0.2 (rollover image).



Rollover image

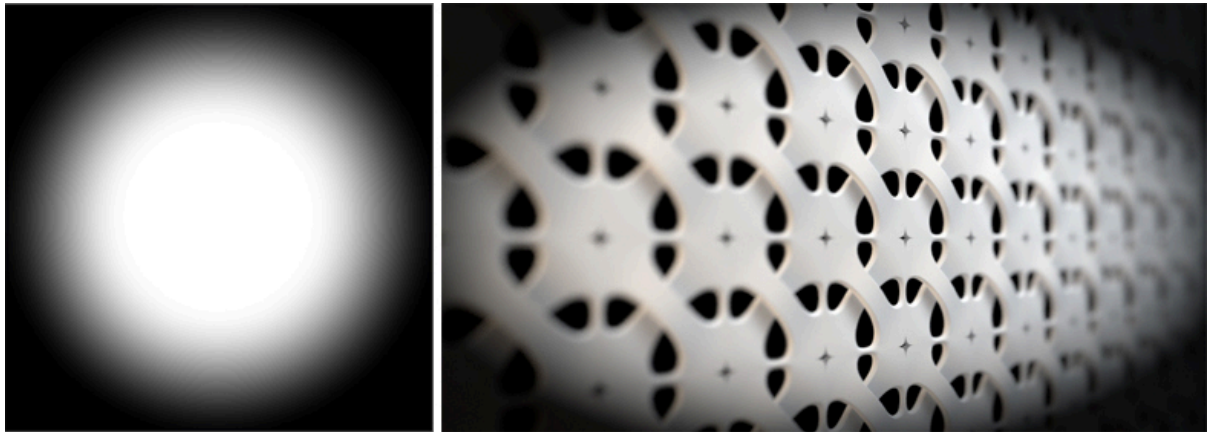
Camera Maps



Filter Map

Weights the camera sample by a scalar amount defined by the shader linked to the filtermap. This shader will use as an input, u,v coordinates in image-space coords $[0,1)$ and x,y in pixel coordinates. This allows you to darken certain regions of the image, perfect to simulate [vignetting](#) effects.

There is an optimization in place where if the filter returns pure black then the camera ray is not fired. This can help in cases such as when rendering with the fisheye camera where, depending on its autocrop setting, parts of the frame trace no rays at all.



Circular ramp mapped to the camera's `filtermap` to create a vignette effect

Perspective•UV Remap

This parameter takes a 2D displacement image and uses it to distort the rendered output of the camera's lens. The left image below has been rendered using a colored ramp connected to the UV Remap attribute. In the right image, the same ramp texture has been distorted in a 2d image post-processing software package.



UV Remap



Remap image has been distorted



When using `uv_remap` you should use the image shader's '`image.swap clamp`' and '`image.twrap clamp`' attributes to minimize edge effects due to filtering.

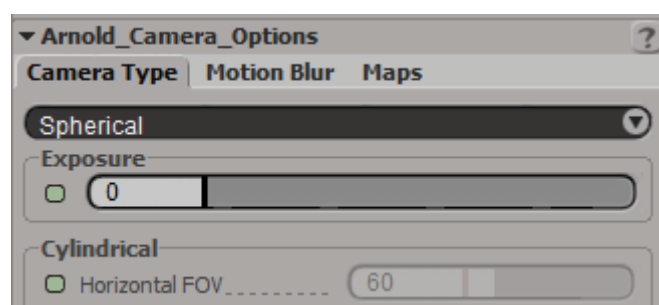
Spherical Camera



Scene rendered with `spherical_camera` lens

This is a spherical camera which means that the camera lens has a spherical appearance. A common use of this camera projection is to allow the creation of environment maps (in spherical map format) for later use as reflection maps or for environment lighting. To get the full spherical range, the camera's screen window must be set to $[-1, -1]$ to $[1, 1]$. Note that the same mapping could be achieved in the cylindrical camera with careful setting of the Horizontal FOV, Vertical FOV, and Projective parameters but the spherical camera is provided for convenience.

 Depth of Field is not available for this camera type.



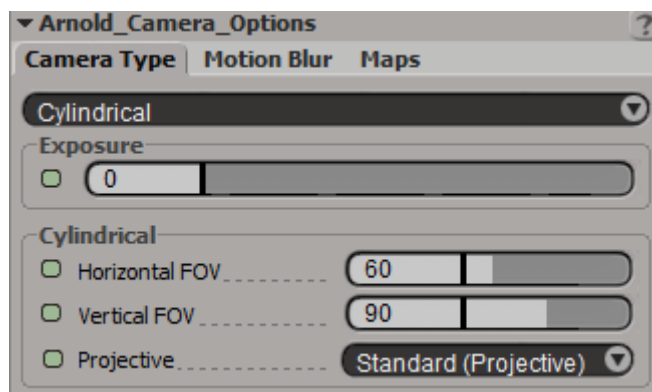
This is a spherical camera which means that the camera lens has a spherical appearance. It has no parameters.

Cylindrical Camera

This camera node provides a cylindrical view, useful for creating panoramic views. A common use of this camera projection is to allow the creation of environment maps (in 'lat-long' or latitude-longitude format) for later use as reflection maps or for environment lighting.

 Depth of Field is not available for this camera type.

:

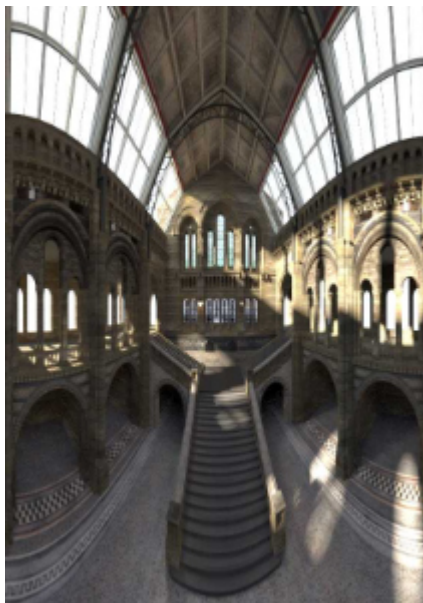


Horizontal Fov•

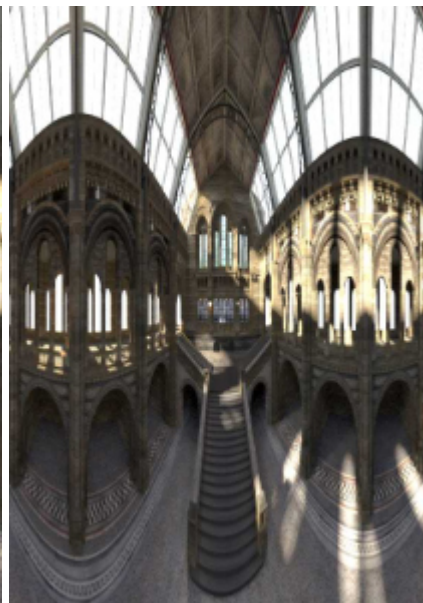
Controls the field of view in the horizontal axis. •



60 (default)



180



270

Vertical Fov

Controls the field of view in the vertical axis (see 'projective'). •



30



90 (default)



270

Projective

The Arnold cylindrical camera can operate in two modes. With 'projective' set to off, all rays have a common origin - they are cast from the center of the cylinder. In the vertical direction the camera acts as a pinhole camera, and in the horizontal direction, it acts as a spherical camera. With 'projective' set to on, the camera is more of a true cylindrical camera; in the vertical direction the camera acts as an orthographic view, and in the horizontal direction it acts as a spherical camera.

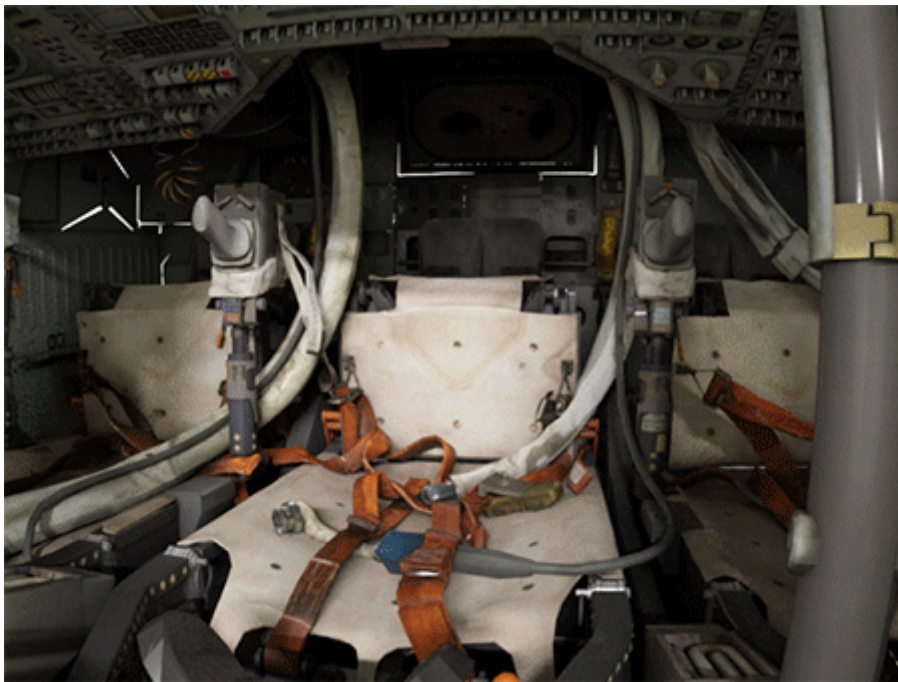
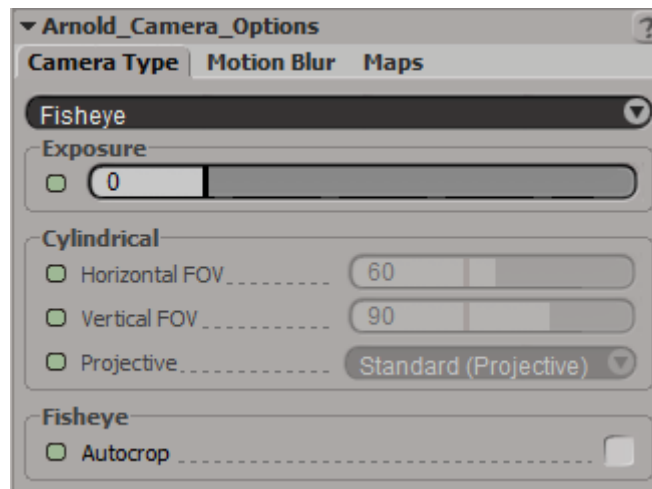


Enabled (default)



Disabled

Fisheye Camera



Fov: 90 to 180

This camera node provides a fish-eye lens type view. The depth of field and aperture controls are similar to the standard perspective camera, but the fisheye camera also provides control over the field of view.

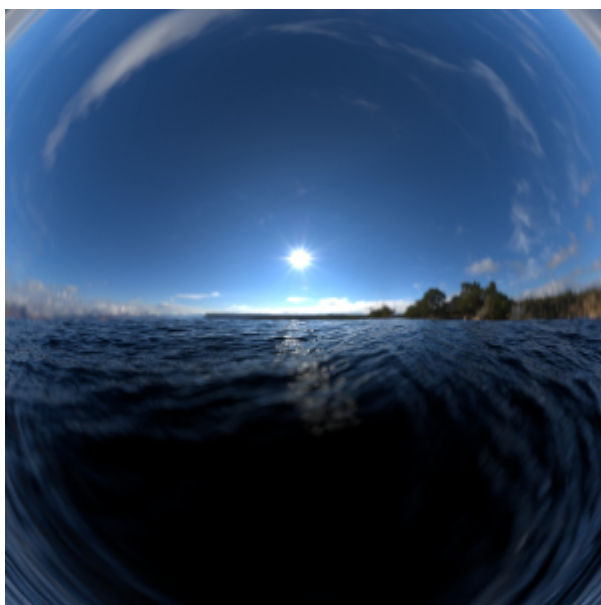
Depth of field and aperture are controlled by the Softimage camera settings, but the Arnold fisheye camera lens shader also provides an Autocrop control.

Auto Crop •

Crops a full-screen image from the center of the fisheye image to remove any white border.

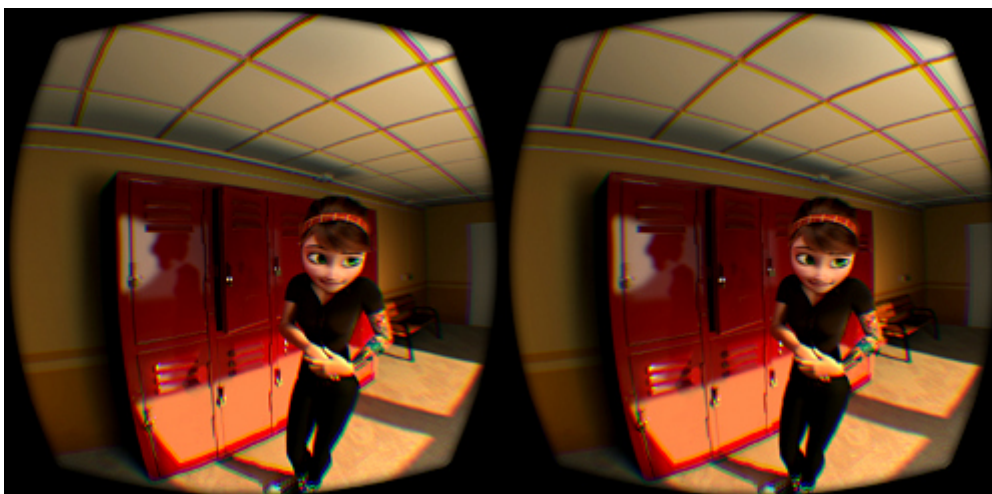


Disabled (default)



Enabled

VR Camera



This is a generic VR camera implementation. It features two cameras offset by a distance with a specific 360° projection. Its main advantages are:

- It works with any scene component that can be rendered in an offline renderer: meshes, hair, particles, volumetrics and complex shading networks.
- An easy learning curve to begin to create VR content. Just add the VR camera to your existing project.
- Modest hardware requisites to reproduce the content. Any platform that can reproduce a video with the required projection is fine to experience the generated content.
- Content that is easy to distribute either as a video file or via video streaming. It can be reproduced using dedicated software or an app, or with web standards like WebGL, WebVR. It also works with Google and Facebook 360 3D videos.

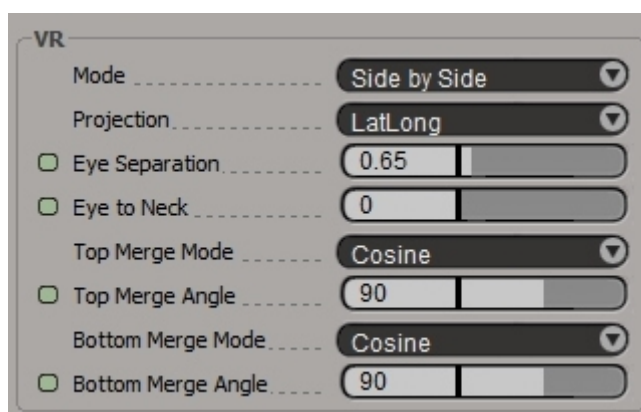
Limitations

- Poles: By default, poles will show very evident artifacts. This requires that you adjust the stereoscopic effect for each scene and smooth it near the poles, thus diminishing the stereoscopic effect.
- Tilt: Due to the way the stereoscopic effect is done, tilting your head will destroy the stereoscopic perception.
- Parallax: When you move your head along any axis, there is a change in the viewpoint that the offline VR scene can't take into account. This can diminish the immersion of the experience since we can only react to head rotations.



It is recommended that Lock Sampling Pattern is enabled in the Arnold Render Settings when using the `vr_camera`. This will avoid the sample pattern changing from one frame to the other, which can be distracting in VR.

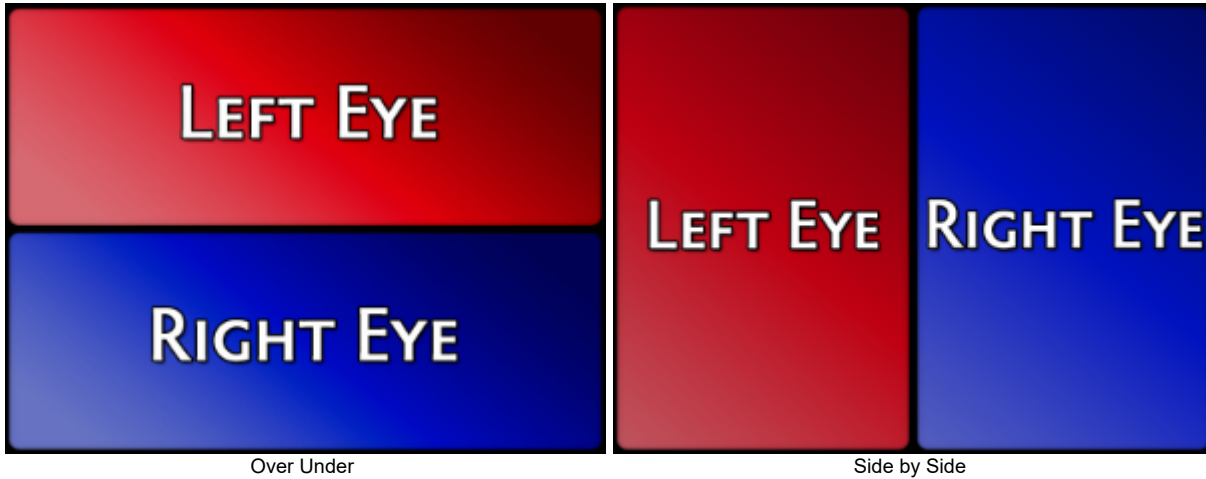
The [cameras](#) page has more details about the common camera parameters. The additional attributes are shown below:



Mode

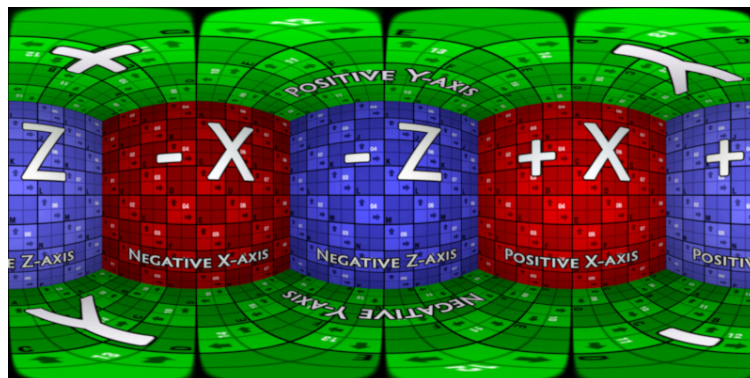
There are four mode options available so that you can get the result that better adapts to your pipeline. They are as follows:

1. Side by Side
2. Over Under
3. Left Eye
4. Right Eye



Projection

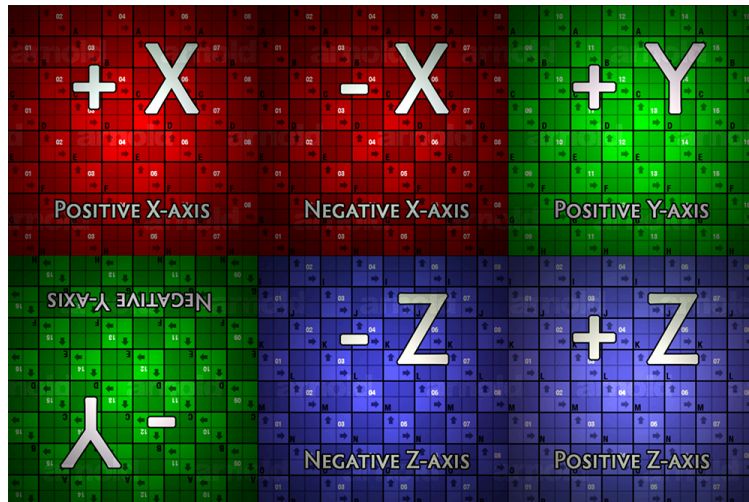
Depending on the selected projection, and options, each sample will correspond to a ray direction so that all of the space around the camera is completely covered. Choose between lat-long, cube map (6x1), cube map (3x2).



Latlong Projection

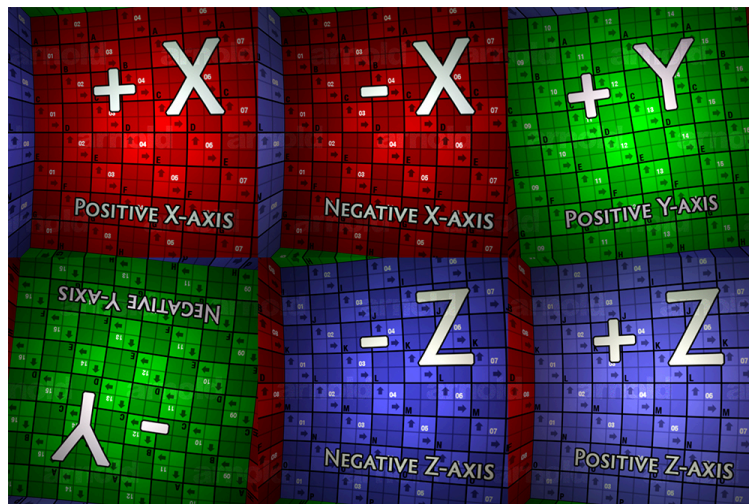


Cubemap (6x1)



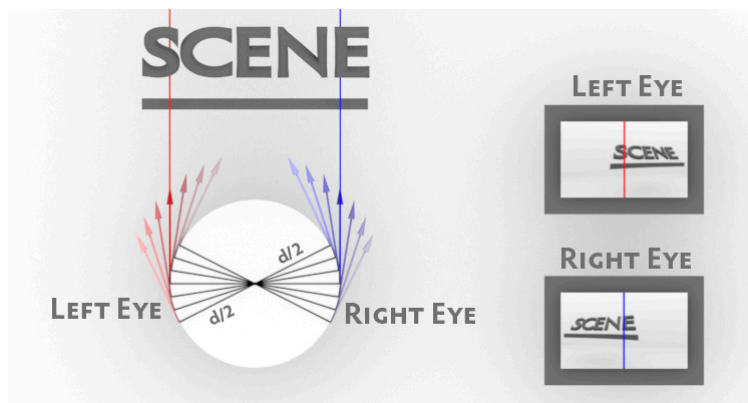
Cubemap (3x2)

The 3x2Cube Map has the advantage of better aspect ratio images.



Eye Separation

Defines the separation between the right and the left camera, required to achieve the stereoscopic effect. The camera origin position is updated for each sample and displaced from the center perpendicularly to the ray direction. Doing this per sample level and not per pixel creates a better result than using two physical cameras. Here is a picture explaining this:



Eye to Neck

The horizontal distance from the neck to the eye.

Top Merge Mode

These parameters define the merging function of the sky. Usually, a Cosine function (Cos) will be smoother and less prone to artifacts. Choose between None, Cosine or Shader.

Top Merge Angle

Defines the angle in degrees from where the merge starts to take effect in the sky. The nearer the angle to the pole (0° top or 180° bottom), the bigger stereoscopic effect you will see below it, but the most probable artifacts will appear at the poles.

Below you can see the difference between a start top angle from 0 to 80° using a cos merging function:



Bottom Merge Mode

These parameters define the merging function of the floor. Usually, a Cosine function (Cos) will be smoother and less prone to artifacts. Choose between None, Cosine or Shader.

Bottom Merge Angle

Defines the angle in degrees from where the merge starts to take effect on the floor. The nearer the angle to the pole (0° at the bottom, 180° at the top), the bigger stereoscopic effect you will see below it, but the most probable artifacts will appear at the poles.

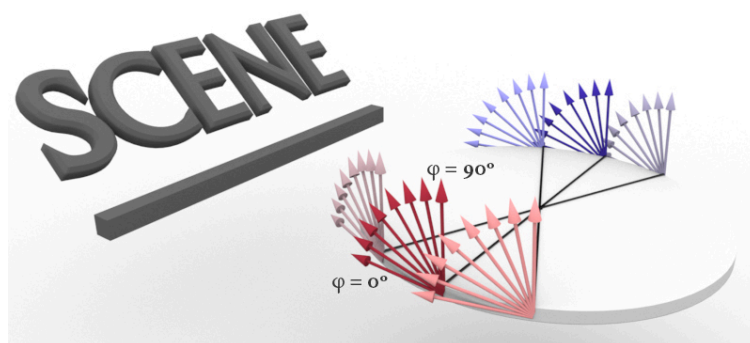


If the bottom_merge_angle is above the top_merge_angle, it will be clamped to the top_merge_angle.

Workflow

Pole Merging

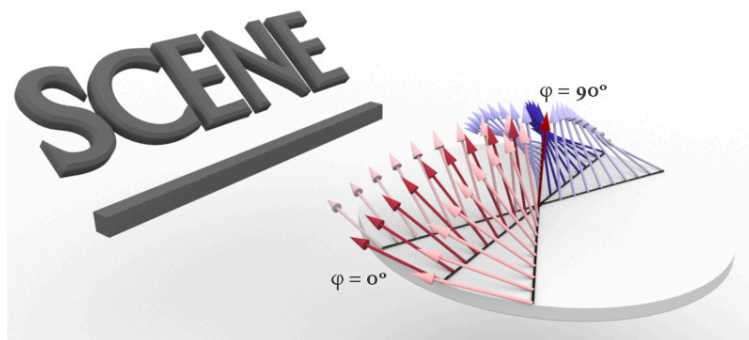
While this method creates a very nice 3D effect for objects around the viewer, some nasty artifacts will appear at the top and bottom poles. This is because the camera offset makes it impossible for the rays to reach the top and bottom positions as you can see here:



Here you can see an example of how poles look like in this case:



We can fix these artifacts by smoothing the camera displacement when rays point upwards or downwards. This solution will remove the stereoscopic effect in the poles, but in practice, it is not very noticeable. In this case, the camera rays will look like this:



The pole merging will generate a result like this:



Different settings are provided to control this pole merging so it can be adjusted depending on the scene. First, top and bottom poles parameters are independent as they could have different requisites in your scene. For example, the pole merging at the floor might need to be very smooth if it has a lot of detail, but top merging can be more aggressive if it has a flat color, giving the upper hemisphere a more relevant stereoscopic effect. An aggressive merging at the top will look like this:



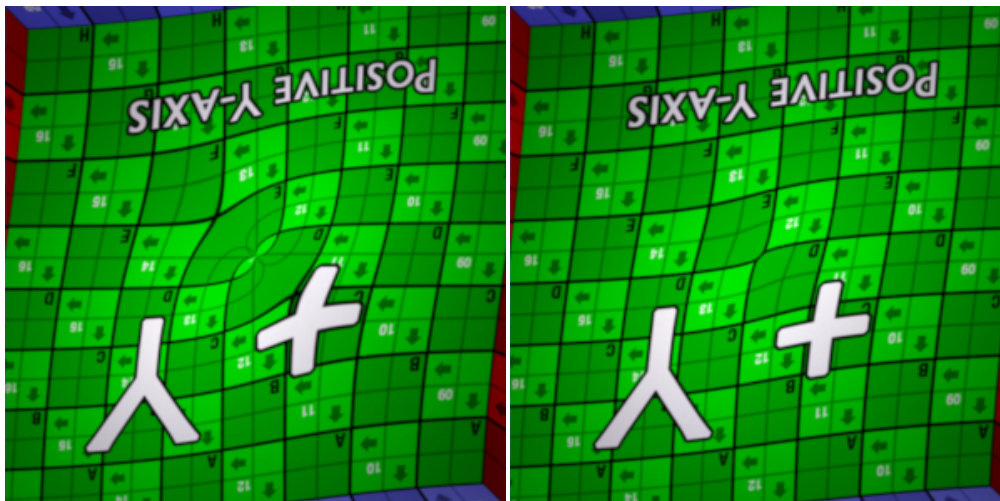
Pole Merging Workflow

To maximize the stereoscopic effect in the scene while avoiding poles artifacts, the top and bottom poles can be adjusted independently to better suit the specifics of a scene.

The artifacts at the poles can be of two different kinds:

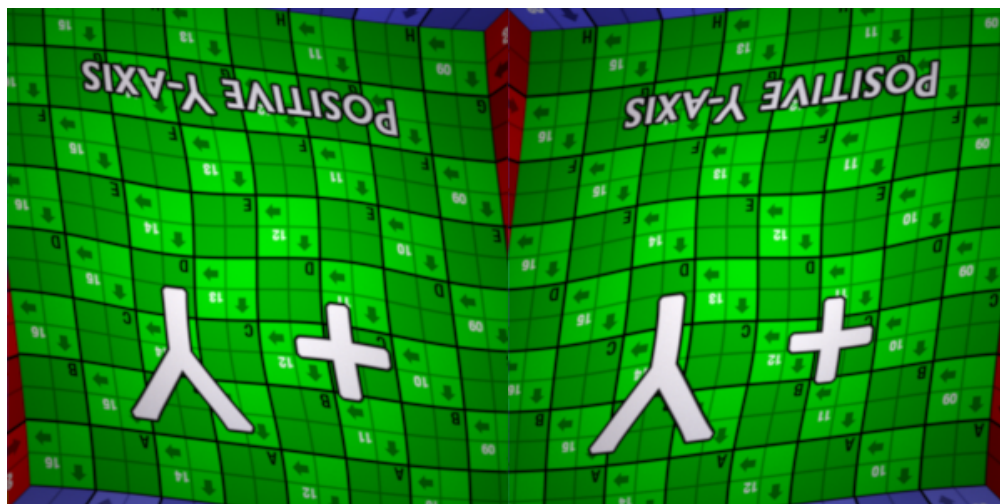
Distortion Artifacts:

The single image you see from an eye has some deformations at the poles. Perpendicular lines do not meet at 90 degrees. These artifacts tend to appear as you increase the Merge Angle. Some examples:



Stereoscopic Artifacts:

When you take a look at a single eye image, you might notice a different distortion, but when you see both right and left images, you will notice a circular wave on the pole. Here is an example:



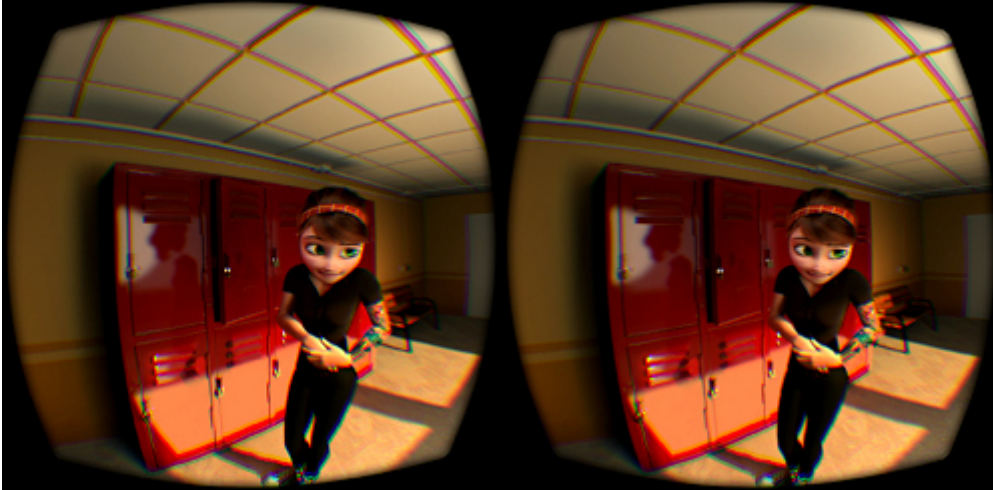
Right eye/Left eye

-
-

Example Workflow:•Mery Video

Making [this video](#) was quite simple. The following steps were taken to create it:

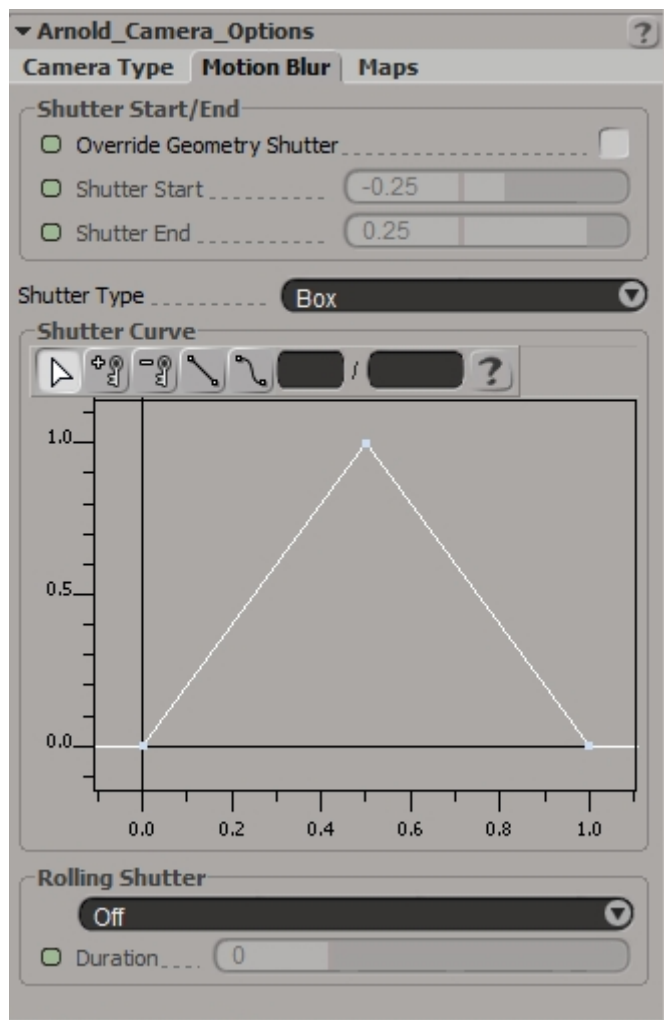
- Add a generic VR camera and adjust it with the correct parameters.
- Fix the scene so that you can look in any direction. Behind the windows, some images were added to simulate the outdoor as well as a couple of columns to hide some lighting artifacts that didn't appear in the original scene camera.
- A single frame with the credits was rendered and added back to the final video inside Nuke.



Camera Motion Blur

These are the motion blur parameters that belong to the camera, as opposed to the ones in the rendering options, that define instead the times at which the scene is evaluated.

By default, the shutter start/end times are taken from the rendering options. You can, however, override them by enabling Override Geometry Shutter .

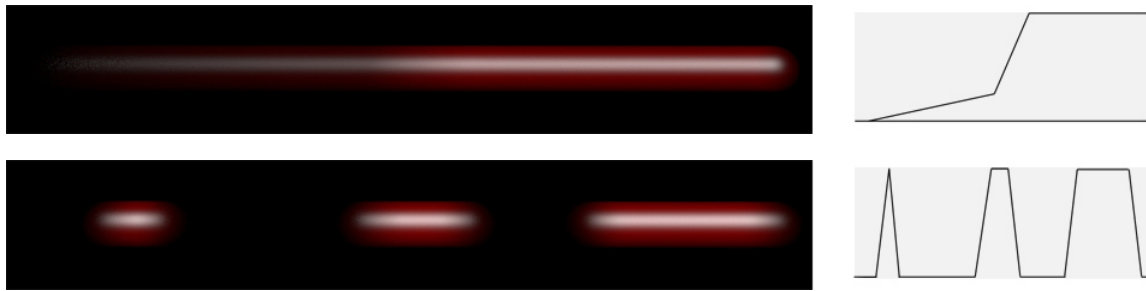


Shutter Type

The filtering applied to time samples. By default, this is a box filter, with all time samples having the same weight. A triangle (or "tent") filter is also available which produces smoother trails.

Arnold supports custom shutter shapes with the shutter curve camera parameter. You can define as many points as required. Coordinates increase from 0 (corresponding to the shutter_start) to 1 (corresponding to the shutter_end). Values in the vertical axis must be non-negative, and it is not recommended to enter values above 1. The values are linearly interpolated between each point. In the examples below, you can see the effect different curve shapes have on the motion blur trail of a sphere that has been key-framed moving from left to right.



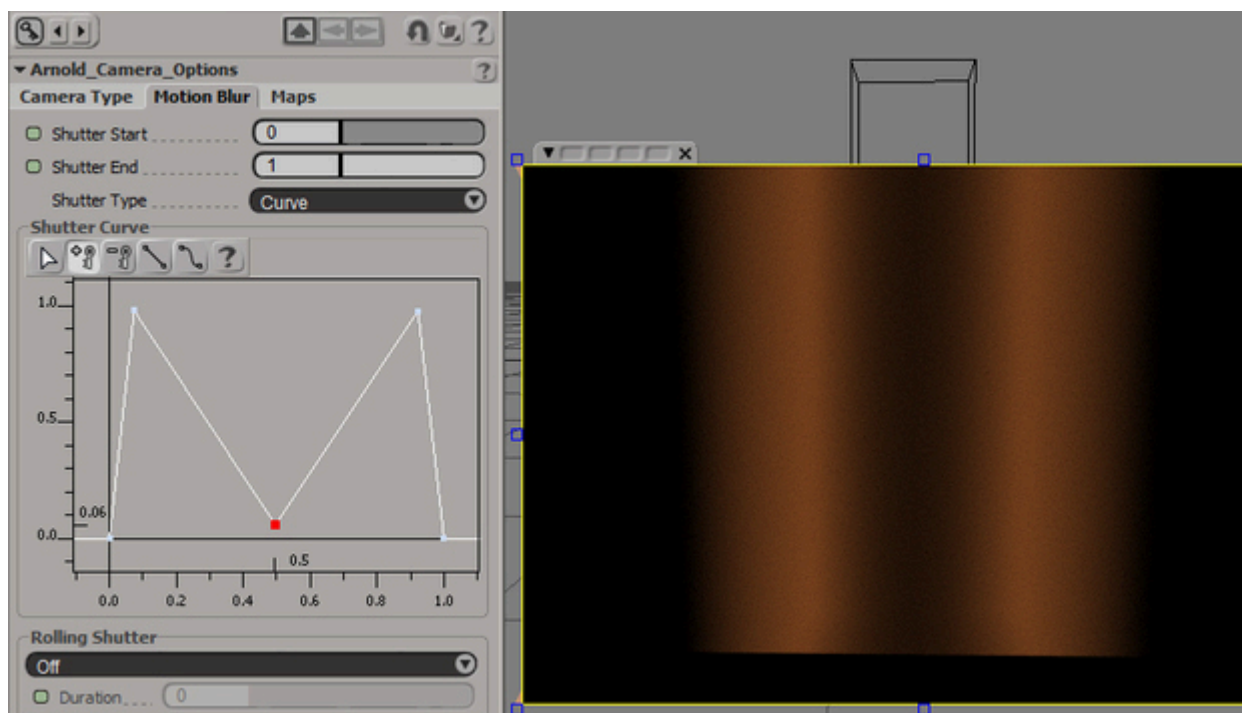


Various custom camera shutter curve shapes

•

The example below shows a box moving from left to right with shutter curve enabled:

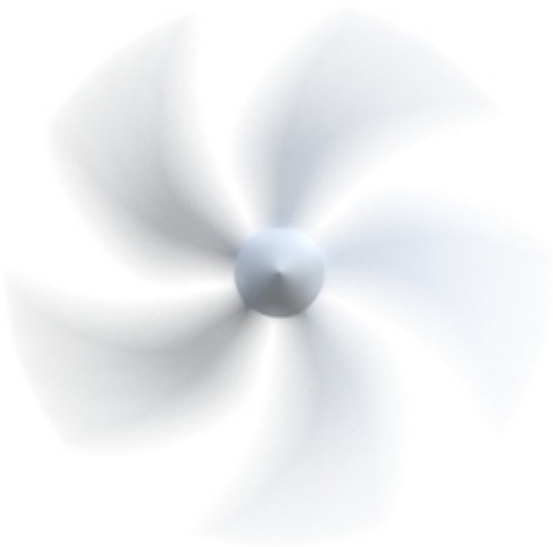
•



•

Rolling Shutter

Rolling Shutter is used to simulate the type of rolling shutter effect seen in footage shot with digital cameras that use CMOS-based sensors such as Blackmagnics, Alexas, REDs, and even iPhones. This method is implemented by rolling (moving) the shutter across the camera area instead of the entire image area all at the same time. •



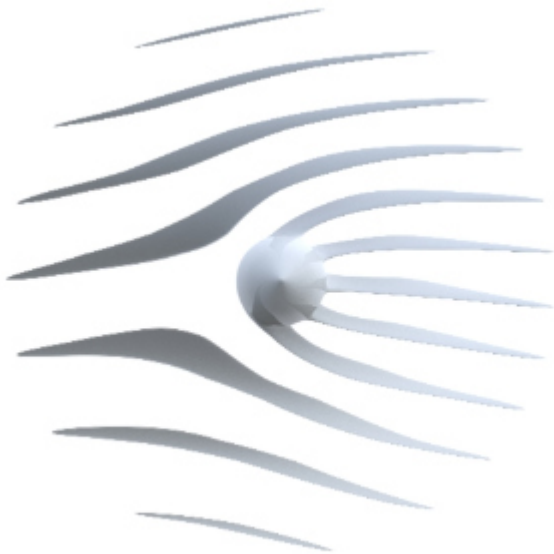
Without rolling shutter (rollover image).



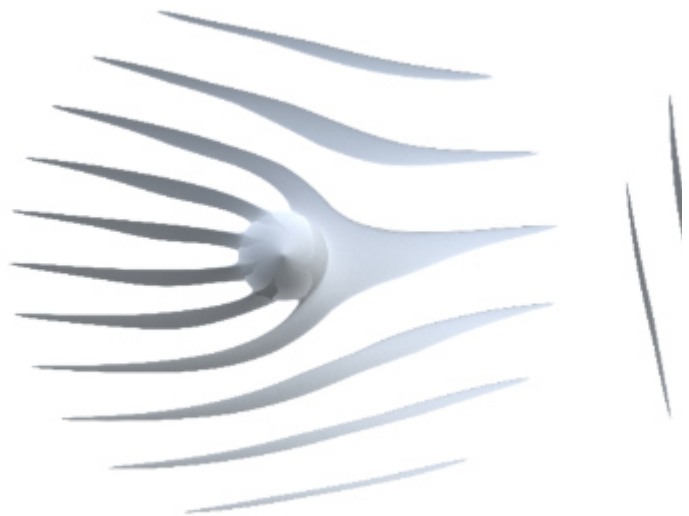
With rolling shutter (rollover image).

•

The Rolling Shutter direction specifies the direction that the rolling shutter takes place. The default is 'off' and can be set to 'top' (top to bottom being the most common scanning direction), 'bottom', 'left' or 'right'.



Top



Bottom

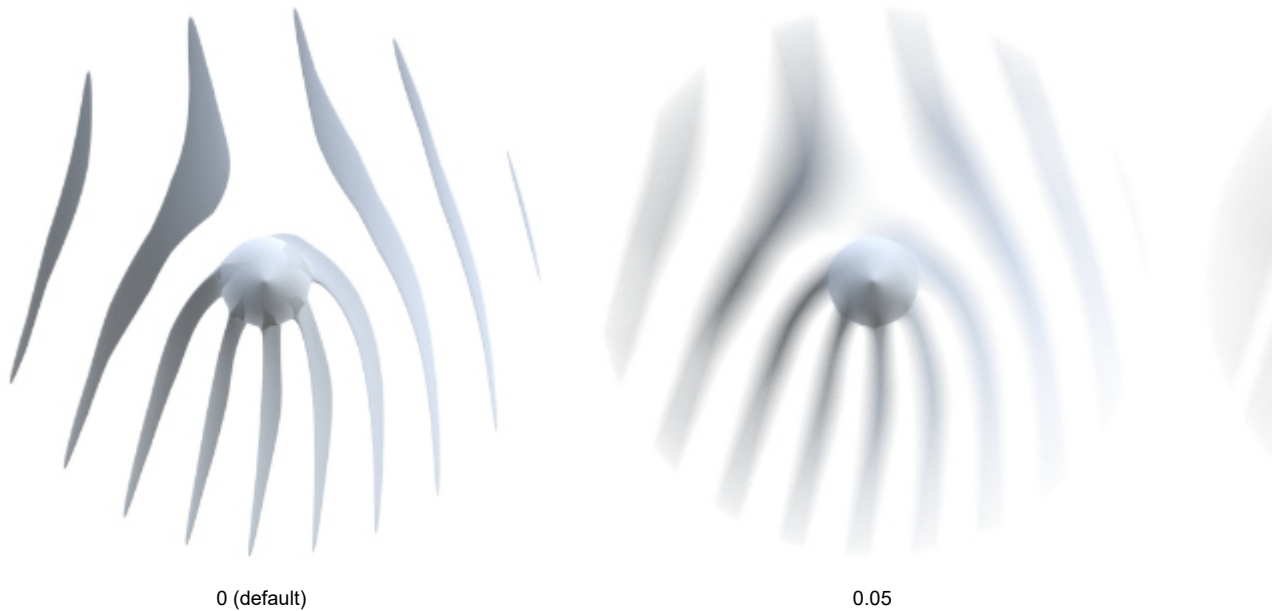
Interesting effects can be achieved when combining motion blur 'length' with rolling shutter:



Motion blur 'length' from 0 to 2

Rolling Shutter•Duration

With this parameter, it is possible to control the duration of exposure of the scanlines in a rolling shutter camera. Valid values for this parameter are in the 0 to 1 range, where a value of 0 gives you an instantaneous exposure of each scanline (the default value and the rolling shutter's previous behavior), and a value of 1 exposes every scanline for the entirety of the camera's shutter interval (the same result that a camera without rolling shutter would give).



0 (default)

0.05

-
-



Shaders

SitoA exposes all the Arnold core shaders and supports some of the legacy Softimage shaders. Also, SitoA is able to auto-define a basic UI for custom shaders that just provide their dll/so library file.

•

- [Closure Connector](#)
- [Displacement](#)
- [Arnold shaders](#)
- [Softimage shaders](#)
- [Dll-so shaders](#)
- [Third Party Shaders](#)
- [Legacy Shaders](#)

Closure Connector

Arnold 5 introduced a new closure parameter type, which shaders can output instead of final colors. There are BSDF, BSSRDF, emission, matte, transparency and volume closures. SltoA exposes the closure typed shaders and parameters as a new type, showing in purple in the rendertree.

The main goal of adding a new type, instead of simply recycling the Softimage color type, is not letting the user connect closures to color port, and vice-versa.

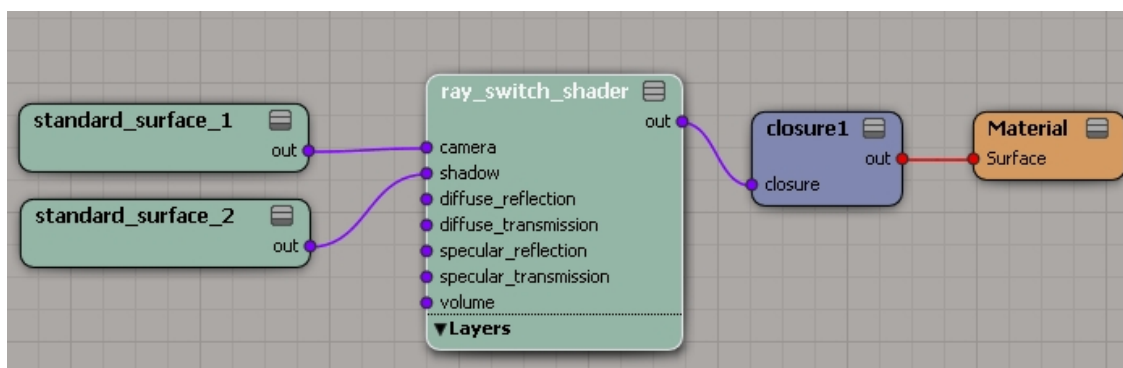
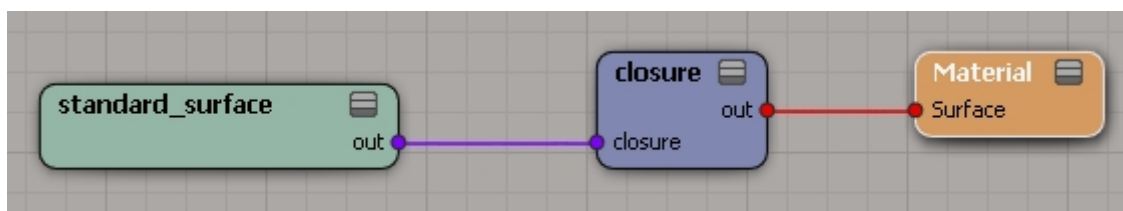
However, since the surface part of the Softimage materials is a color, we provide a special node, called "closure" , that must be used to connect shaders such as the standard series into the surface slot.



The connector is automatically added between the Arnold shader and Surface when you create any of the standard shaders from the Arnold->Shaders top menu.

Please be sure to use closure only to connect the render tree into the Surface slot, and not in any other part of the tree.

Examples



Displacement



Displacement texture map from [JSplacement](#) (left). Roll over image to view without displacement (right).

Displacement maps can be an excellent tool for adding surface detail that would take far too long using regular modeling methods. Displacement mapping differs from bump mapping in that it alters the geometry, and therefore will have a correct silhouette, and self-shadowing effects. Depending on the type of input, the displacement can occur in two ways: Float, RGB & RGBA inputs will displace along the normal while a vector input will displace along the vector.

The example above shows how a simple plane, with the addition of a displacement map, can produce an interesting looking simple scene.

You should ensure that your base mesh geometry has a sufficient number of polygons otherwise subtle differences can occur between the displaced low-resolution geometry and the high-resolution mesh from which it was generated.

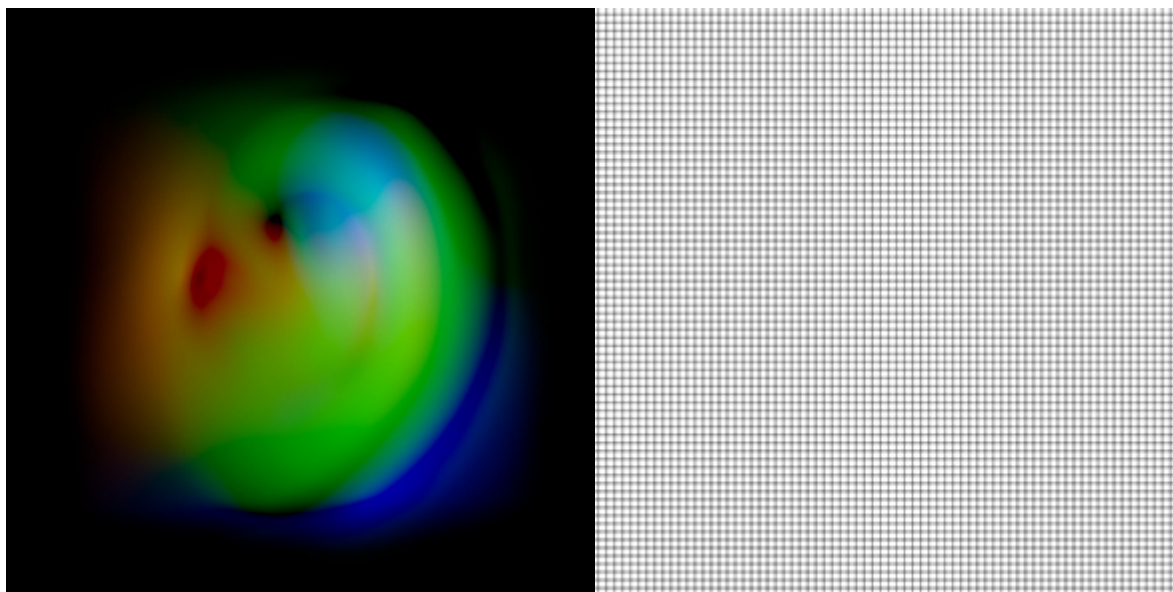


Make sure that you use a 32-bit or 16-bit floating-point format to store your image, and not an integer format. An integer format will not work correctly. This is because integer formats do not support negative pixel values, which are used by floating-point displacement maps.

•
SlttoA follows the standard Softimage workflow. You can connect any shader to the Displacement slot of the material.

The settings are driven by the [Arnold Parameters](#) property, so they belong to the mesh owning the property, and not to the displacement shader (which could be shared between several objects).

Vector Displacement

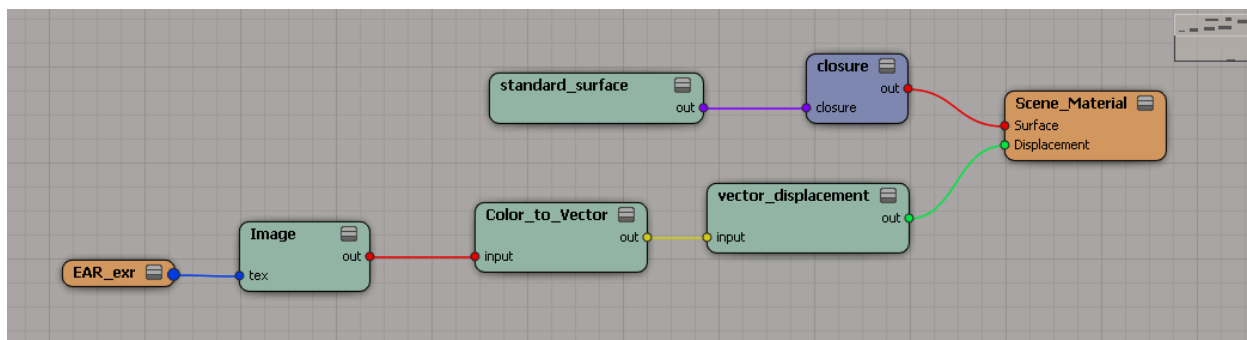


Ear tangent space vector displacement map from Mudbox (left). Polygon plane displaced with ear vector displacement map (right).

Traditional displacement maps are not used for any surface change that is not perpendicular to the base mesh's polygons. Vector displacement maps can displace in directions other than the face normal, which is much more flexible. Vector displacement uses the color channels that specify a vector in a certain space to displace the vertices of the geometry in that direction and magnitude.

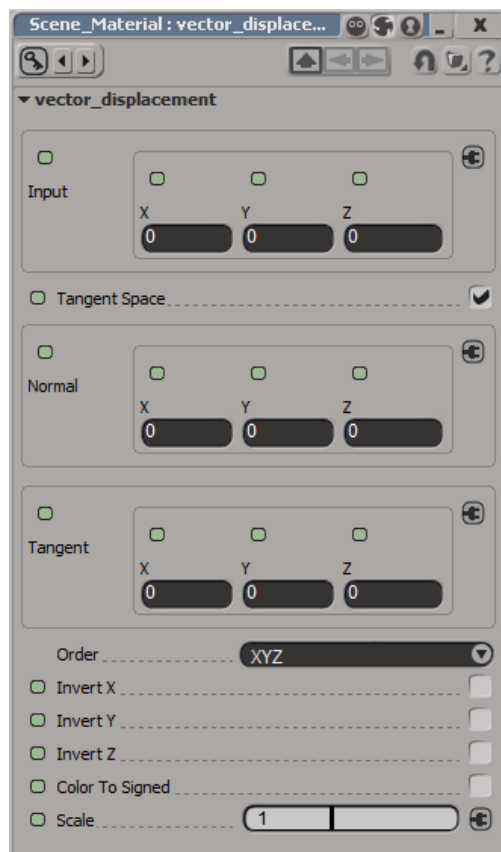
The ear vector displacement map can be downloaded [here](#). The scene file can be found [here](#).

-
-



Ear.exr vector displacement map connected to vector_displacement shader

-
-



•

Input

Connect the vector displacement map to this input.

Tangent Space

Specifies if the input is in world space or tangent space.

Normal

The normal and tangent parameters can be optionally linked to define a custom tangent coordinate system that the input is transformed from. If the normal is not linked, it will use the default surface normal.

Tangent

The tangent map. Together with the shading normal, it defines the tangent coordinate system that the input vector applies to. If available from your sculpting tool, you should connect here the tangent map that the normal map relies on. If 0, the shader attempts the following actions to build the frame:

1. Look for vector user data named "tangent" and "bitangent".
2. Use the UV derivatives.
3. Build its own local frame.



The shader works in tangent space only. If your tangent map was exported in either world or object space, you can instead use the more generic space_transform shader.

Order

Lets you shuffle the input channels order.

Invert X/Y/Z

If enabled, inverts (1-channel) the selected input channel.

Color to Signed

Controls the amount of displacement. Displacement_height can have either positive or negative values. This attribute only applies to normal displacement. •You can use this value to compensate for any inconsistencies between the exported displacement map and the low-resolution geometry.



0



0.2

Scale

Controls the amount of displacement. Displacement_height can have either positive or negative values. This attribute only applies to normal displacement. •You can use this value to compensate for any inconsistencies between the exported displacement map and the low-resolution geometry.

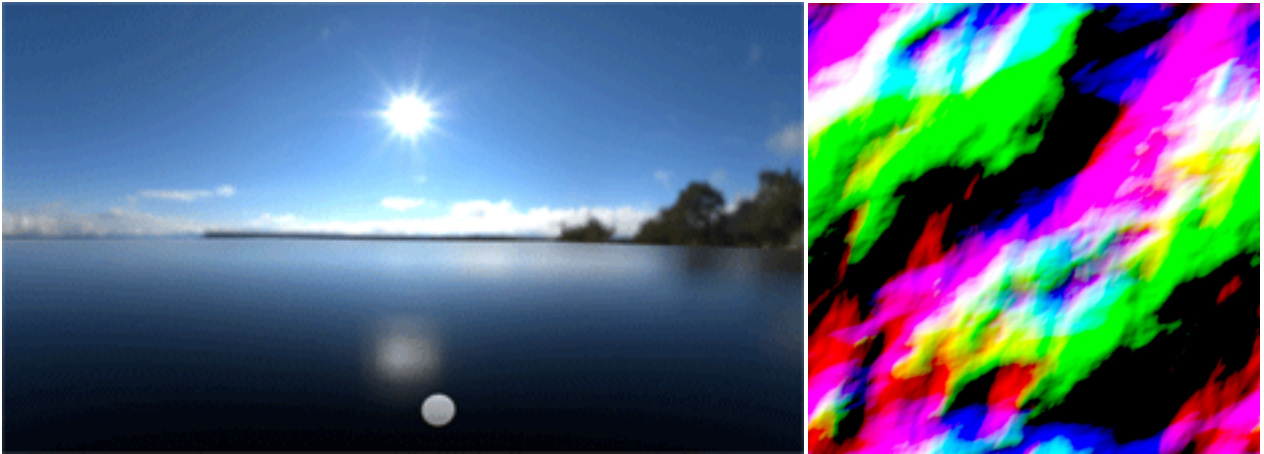


0



0.2

-
-
-



Further example: Sea waves are displaced using a vector map (right)

Arnold shaders

These are the Arnold core's shaders that are exposed by SlttoA.

- [AOV Shaders](#)
- [Atmosphere](#)
- [Bump](#)
- [Closure](#)
- [Color Shaders](#)
- [Conversion](#)
- [Environment Shaders](#)
- [Math Shaders](#)
- [Shading State](#)
- [Surface](#)
- [Texture](#)
- [Utility Shaders](#)
- [Volume Shaders](#)
- [User Data Shaders](#)

-
-
-
-
-
-

AOV Shaders



•

The `aov_write` shaders allow you to write float, int or color data to custom AOVs by your needs.

Passthrough

Allows parallel evaluation in a shader network.

AOV Input

Input float value to write to the AOV.

AOV Name

Name of the AOV.

Blend Opacity

For transparent surfaces, this controls if only the first surface writes to the AOV, or if the value from all transparent surfaces is blended together.

Atmosphere

Below is a list of atmosphere shaders available in SlttoA:

- [Fog](#)
- [Atmosphere Volume](#)

Fog



Rollover image

This atmospheric shader node simulates the effect of light scattering, which causes more distant objects to appear lower in contrast, especially in outdoor environments.

Color

The color of the fog. Unsaturated values work best.



0.2 0.2 0.2



1 1 1

Distance

Fog distance controls the fog density. Fog density is modeled with an exponential distribution. Smaller values make the fog appear less dense, while larger values make it appear denser.



5 (Z-Axis)



10 (Z-Axis)



100 (Z-Axis)



400 (Z-Axis)



Height

This value changes the rate of the exponential decay along the direction axis. •



1 (Y-axis)



3 (Y-axis)



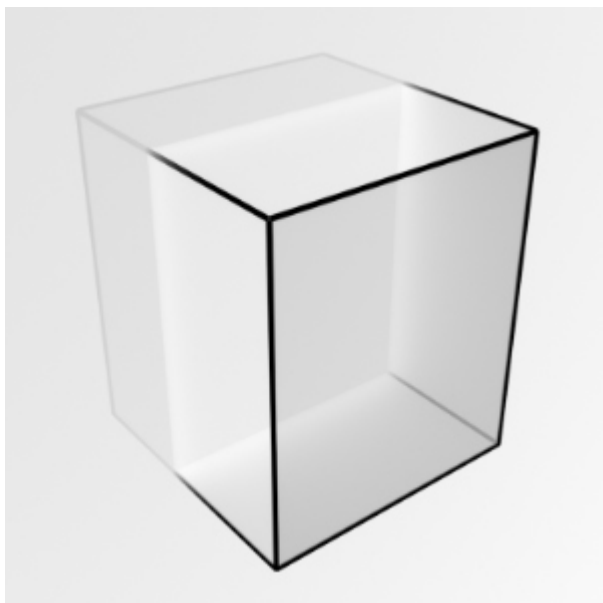
1 (Y-axis)



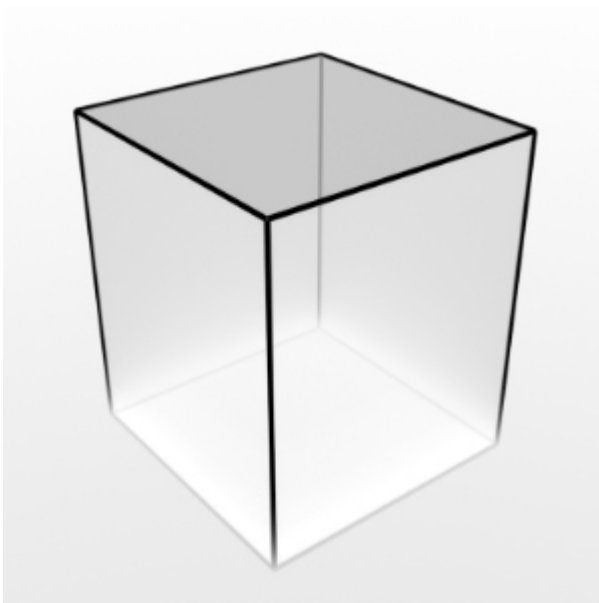
2 (Y-axis)

Ground Normal

Determines the axis for the direction of fog. •



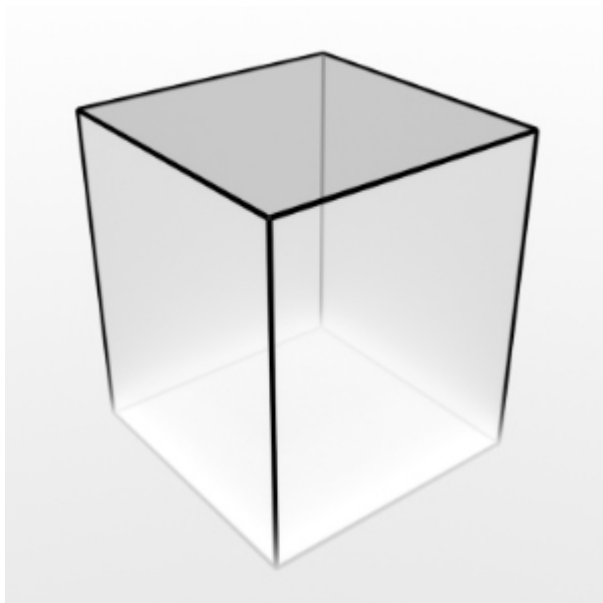
X: 1



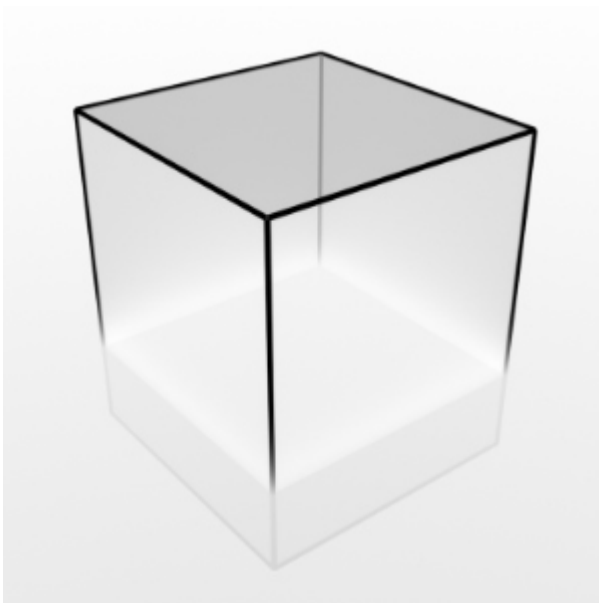
Y: 1

Ground Point

Determines the starting point for the fog along the axis set by the direction.*



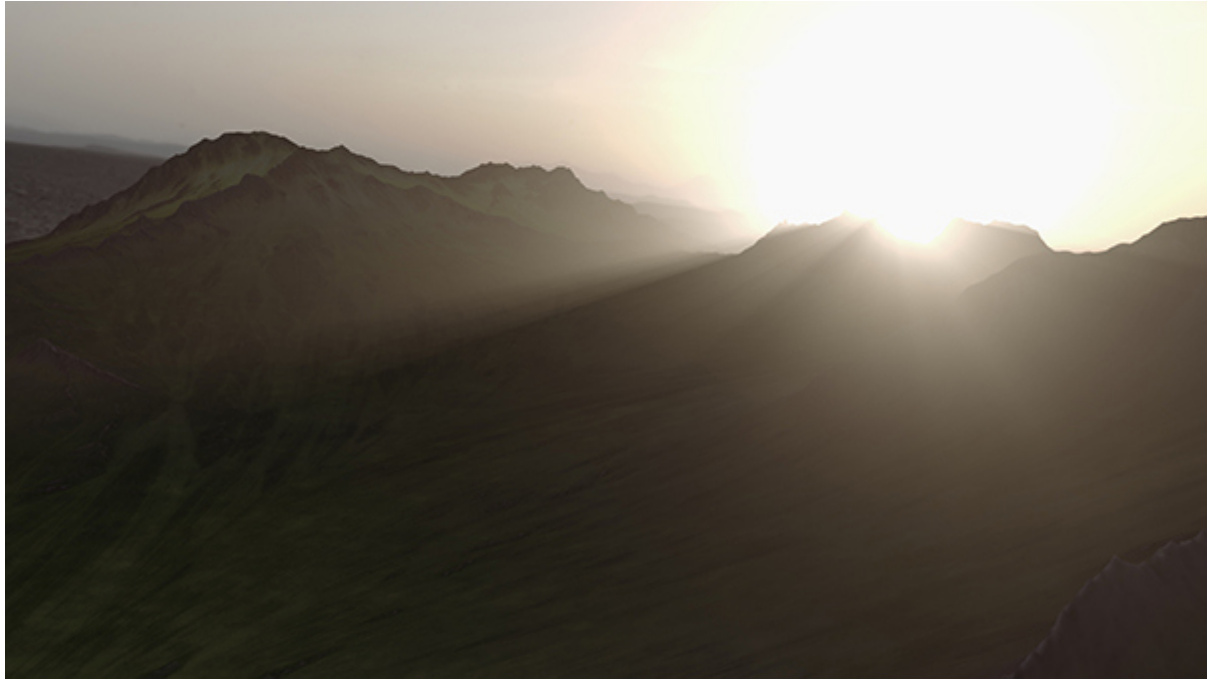
0 (Y-Axis)



25 (Y-Axis)

•

Atmosphere Volume

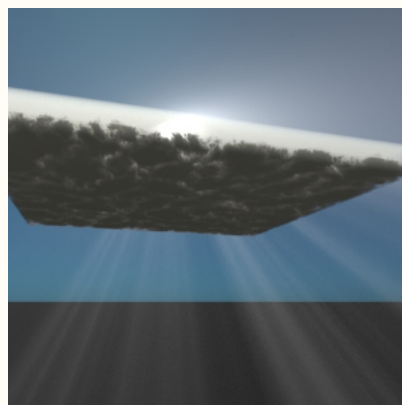


Rollover image

This shader simulates light scattered by a thin, uniform atmosphere. It produces shafts of light and volumetric shadows cast from geometric objects. It works with point, spot, and area lights, but not with distant or skylights. This is a scene-wide volume shader (or an atmosphere shader in Arnold's terms).



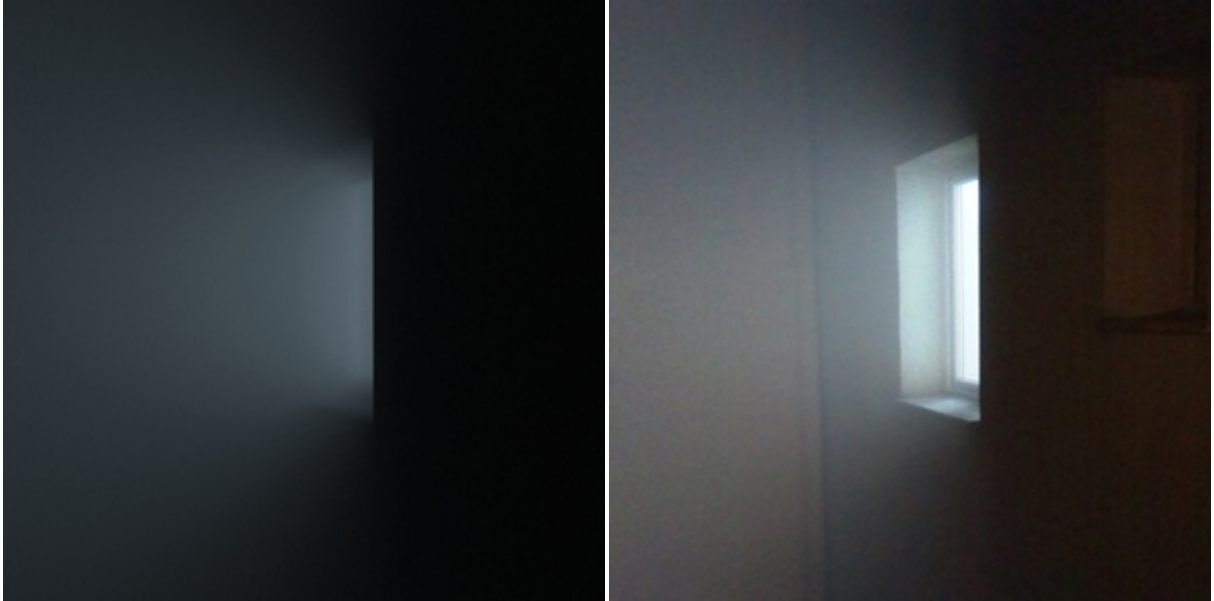
- `atmosphere_volume` used to be called `volumetric_scattering` and should not be confused with volume rendering of fluid type objects.
- `atmosphere_volume` only works with 'local' lights that have a precise location and size and inverse-square decay. It does not support lights at an infinite distance, such as the Skydome light or directional light.
- Currently, `atmosphere_volume` does not compose well against volumes. This is because atmospheres return a single flat result that is opacity mapped on top of whatever is in the background of the pixel.



`atmosphere_volume` cannot 'penetrate' through the cloud volume

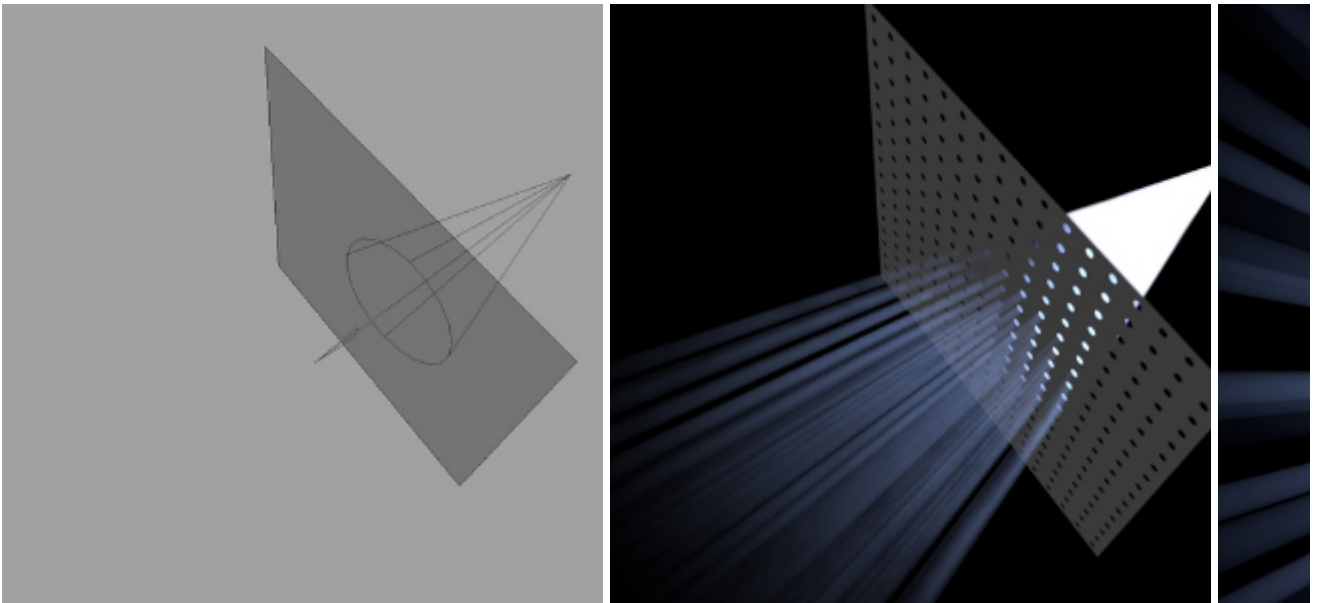


`atmosphere_volume` should be composited using an 'additive' mode such as 'screen' because volumetric scattering is the light that cannot be represented in the alpha channel.



enable_matte enabled for standard_surface shader assigned to wall geometry (left). atmosphere_volume render composited using 'screen' (right). Rollover images.

The example below demonstrates the effect of `atmosphere_volume` through a medium. It consists of a polygon plane with a circular ramp texture connected to the `opacity` of a `standard_surface` shader. The spotlight is pointing at the plane and `atmosphere_volume` is enabled.

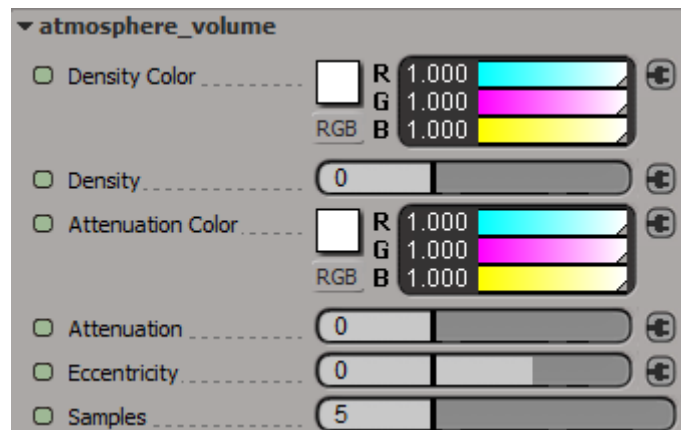


*Polygon plane with circular ramp texture -> opacity of a standard_surface shader

MultiExcerpt named 'atmosphere_volume ' was not found

The page: Atmosphere Volume was found, but the multiexcerpt named 'atmosphere_volume ' was not found. Please check/update the page name used in the 'multiexcerpt-include macro.

Volume Attributes



These parameters control the scattering of light by the participating medium (fog, smoke, etc.).

Color

The density control is multiplied by this RGB value (so a blue value here means the blue light is scattered).



White (default)



Blue

It is also possible to map textures to this parameter to get interesting volumetric effects. The example below uses noise texture to simulate a cloudy fog effect.

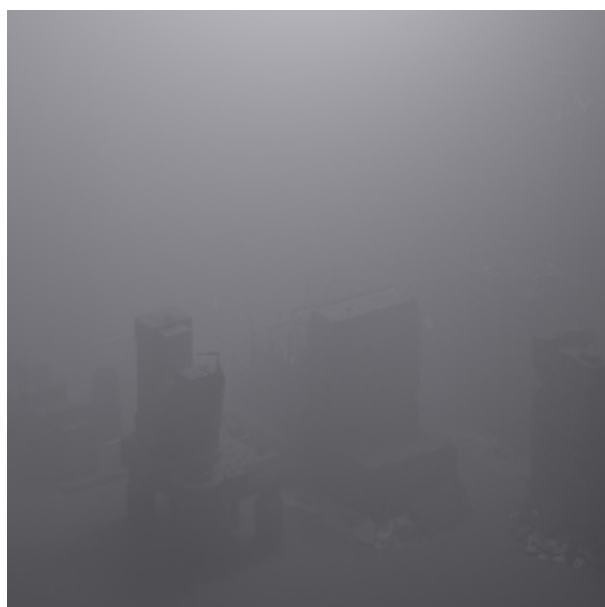


noise -> color

Density

The following images show the effect of altering the density setting.

Increases/decreases the atmospheric volume density. • The following images show the effect of altering density.



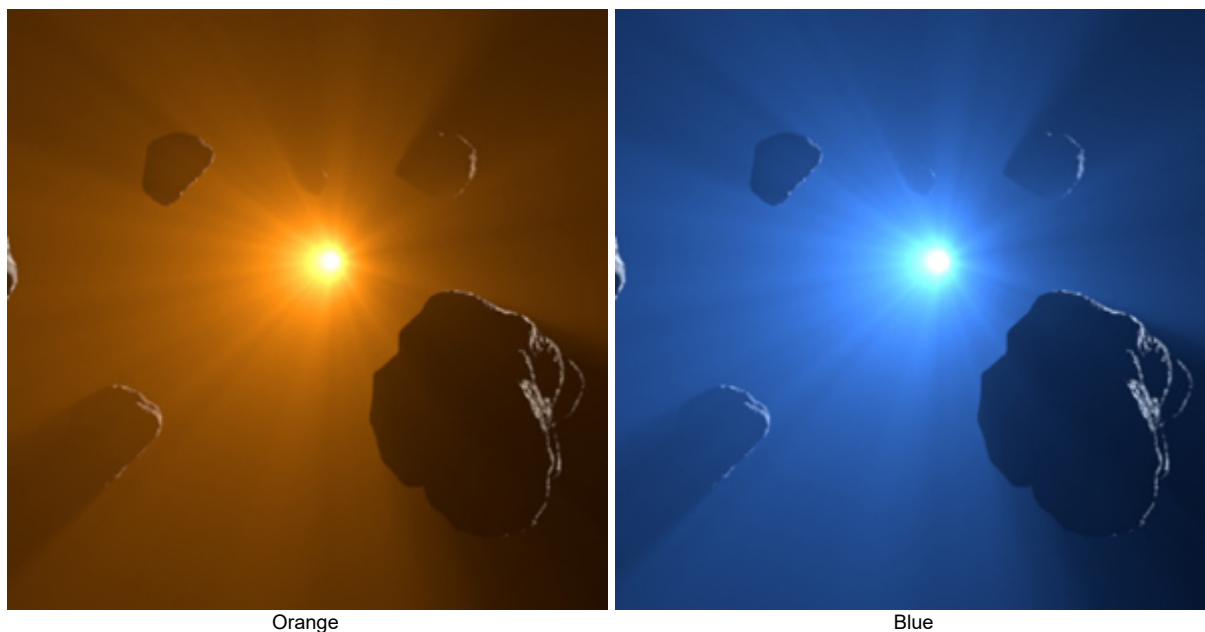
0.1



0.5

Attenuation Color

The attenuation control is multiplied by this RGB value (so a red value here means red light is attenuated).

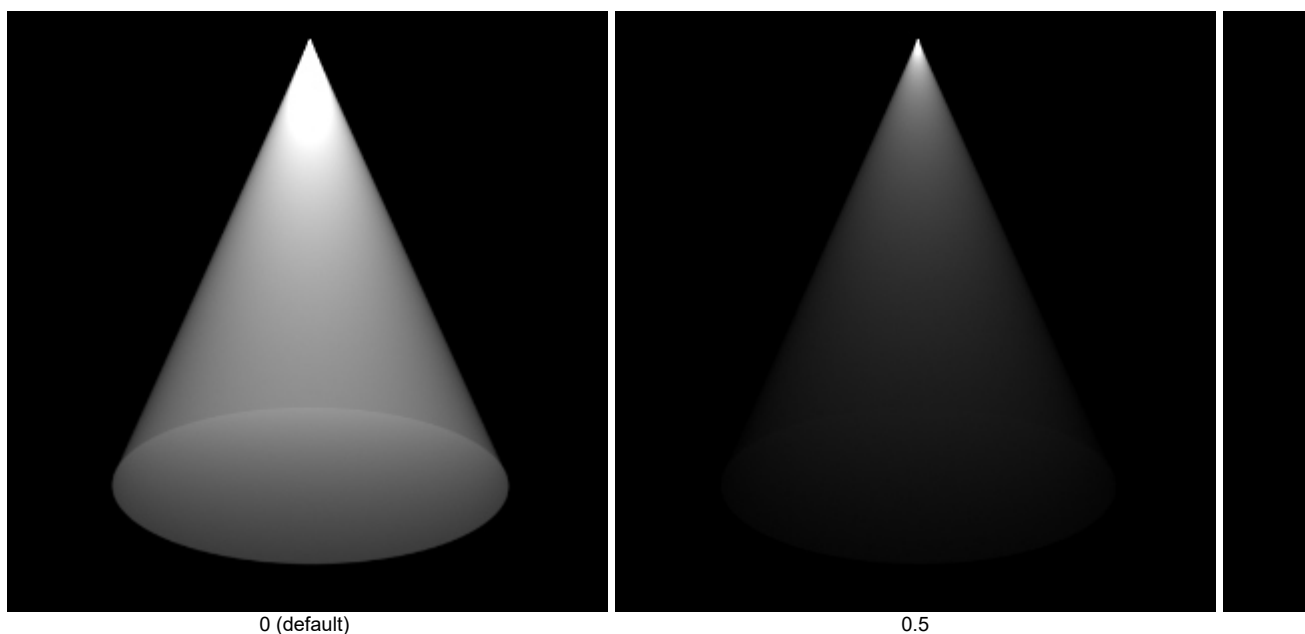


Orange

Blue

Attenuation

The attenuation parameter sets the rate at which the rays of light traveling through the scattering medium are extinguished and how much light coming from the background is blocked. A high value means that light will only travel a short distance through the volume, while a low value means that light will travel a long distance through the volume.



0 (default)

0.5

Eccentricity

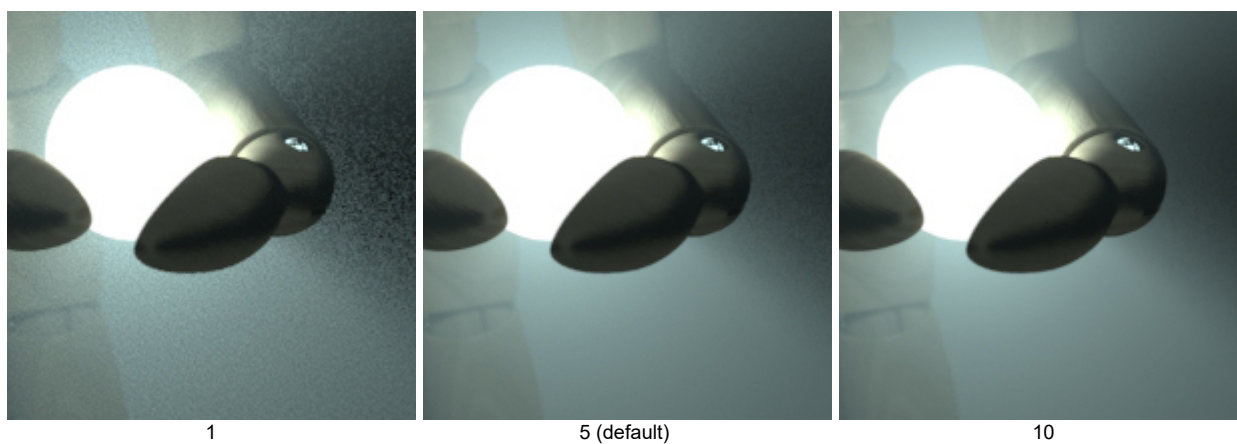
Henyey-Greenstein Anisotropy coefficient between -1 (full back-scatter) and 1 (full forward-scatter). The default is 0 for an isotropic medium, which scatters the light evenly in all directions, giving a uniform effect. Positive values bias the scattering effect forwards, in the direction of the light, while negative values bias the scattering backward, toward the light. Changing the eccentricity, therefore, means that you will get a different effect depending on whether the camera is looking toward the light or away from the light. •

Note that values very close to 1.0 (above 0.95) or -1.0 (below -0.95) will produce scattering that is so directional that it will not be very visible from most angles (and what scattering you do see may be noisy), so such values are not recommended.

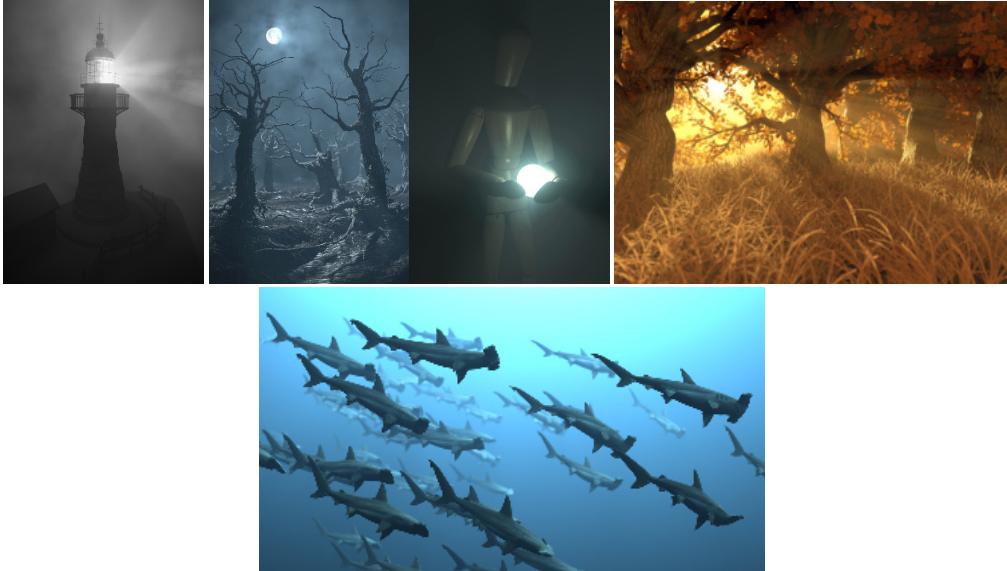


Samples

The samples are distributed according to the volume density. More samples will refine the quality of the solution.



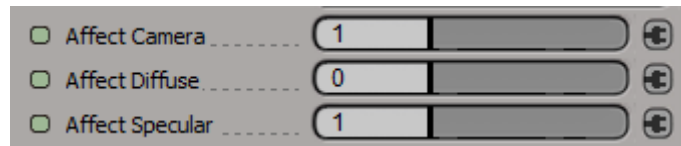
-
-



Further examples

-
-

Contribution Attributes



The following attributes allow you to control the contribution that the atmosphere makes to different types of rays in Arnold - camera rays, diffuse GI rays, and specular rays.

Camera

Controls the degree to which the atmosphere_volume affects camera rays.



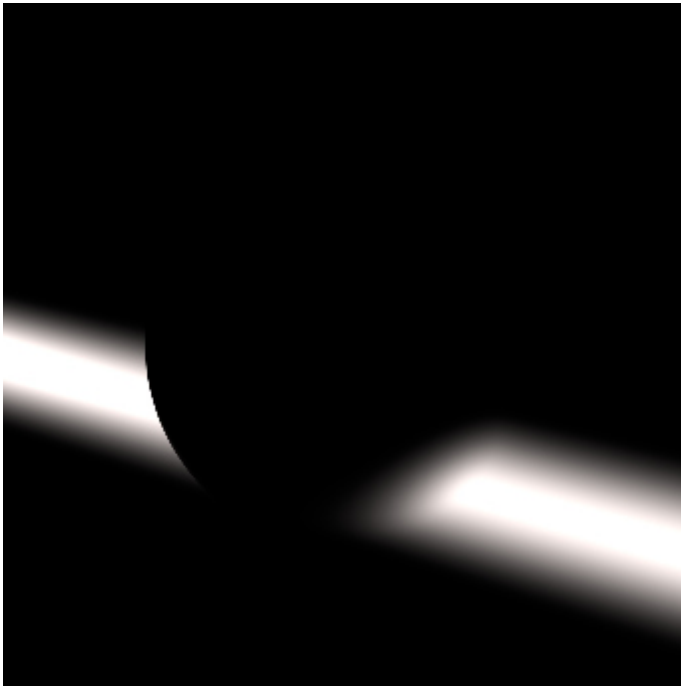
0.1



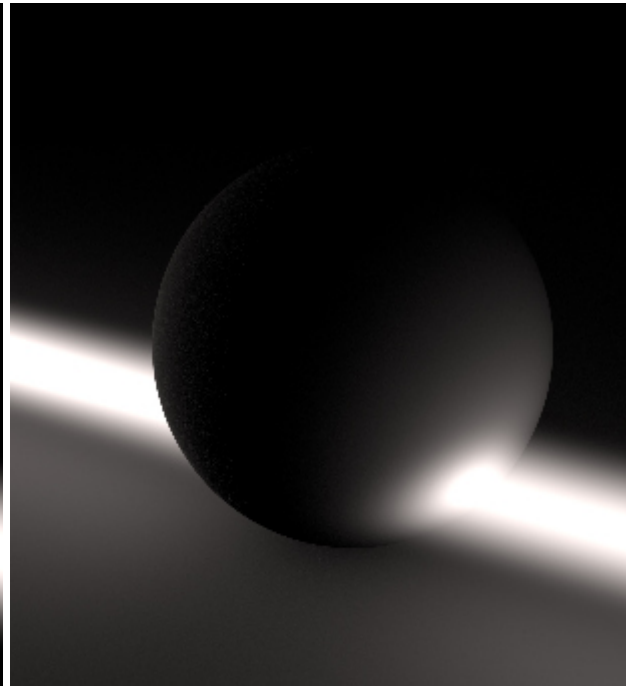
1 (default)

Diffuse

Controls the degree to which atmosphere_volume affects GI_diffuse_rays.



0 (default)



1

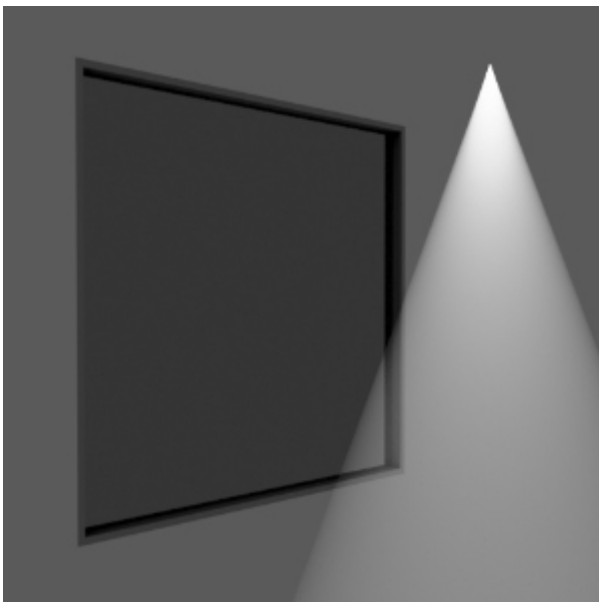
Thin volumetric spotlight with a small cone angle



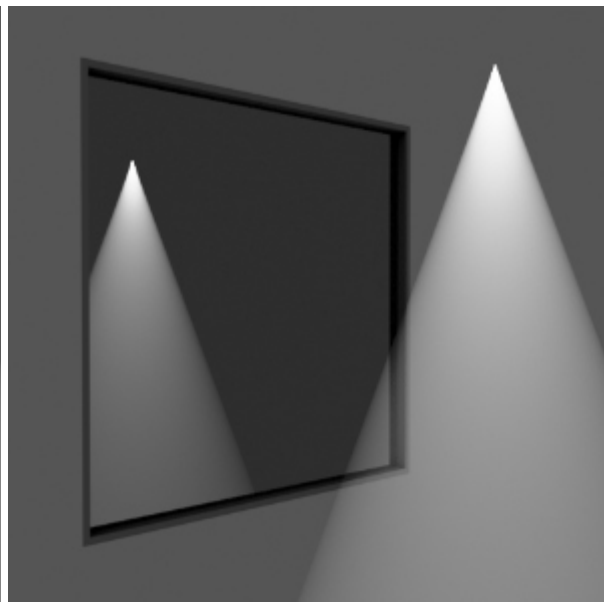
Care should be used when enabling these settings. Non zero values means slower renders because the GI rays now need to also do volume calculations, and most of the time this effect is subtle and not worth the extra effort.

Specular

Controls the degree to which atmosphere volume affects specular rays.



0



1 (default)

-
-

Bump

The following bump shaders are available in SlttoA:

-

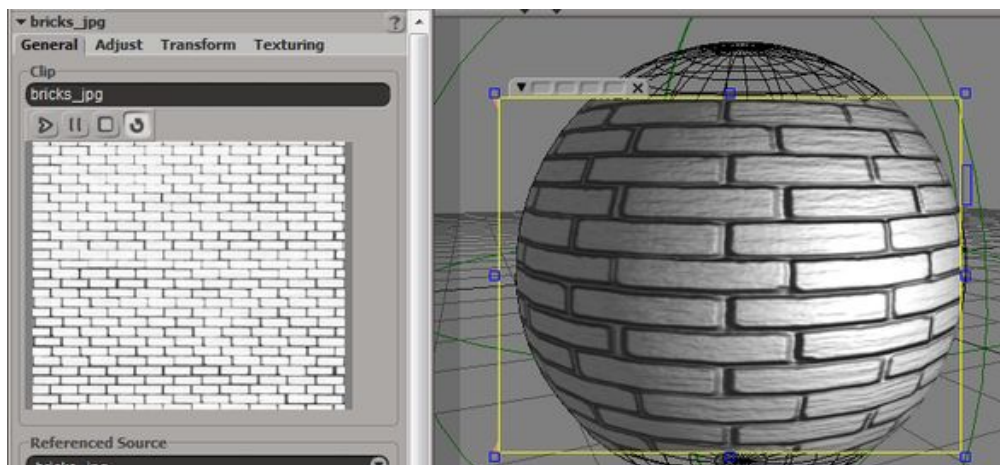
- [Bump2d](#)
- [Bump3d](#)
- [Normal Map](#)

Bump2d

Provides bump mapping based on a 2d texture map. Note that bump mapping does not alter the geometry of the object, it only changes shading normals so that things won't look right in silhouette; however, very often this does not matter. The bump2d node evaluates the bump function at three locations with an offset in u and v from the shading point, via a "forward differencing" algorithm.



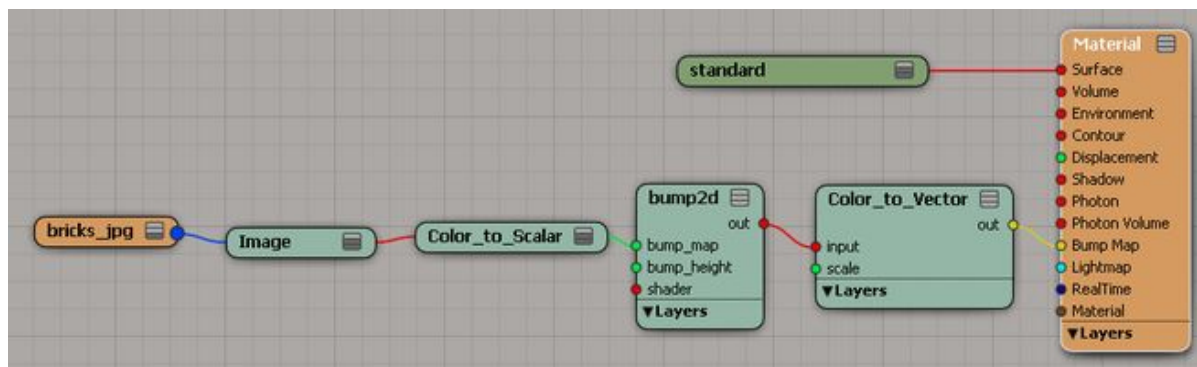
Keep in mind that the object will need UV coordinates that will be used to derive the perturbation.



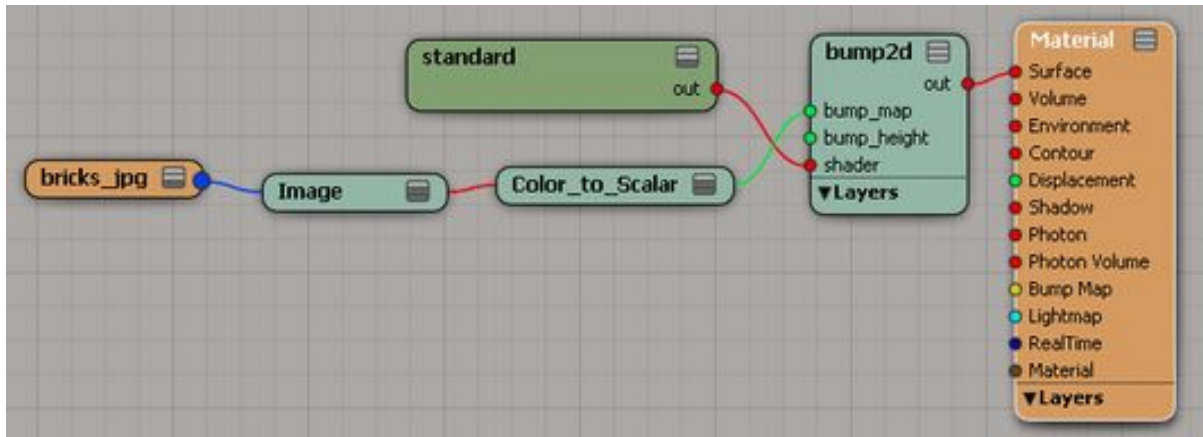
A sphere using bump2d with a heightmap texture.

There are two ways to apply a bump2d shader. Both methods will give the same result. Keep in mind that the object will need UV coordinates that will be used to derive the perturbation.

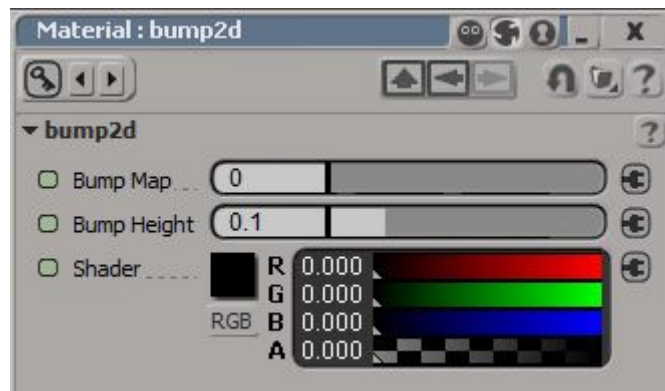
The first method is the common workflow in Softimage, using the Bump Map slot of the material:



The second method is to connect bump2d to the surface slot of the material •and connect the main surface shader to its shader slot:



Settings



Bump•Map

The input for the bump mapping.

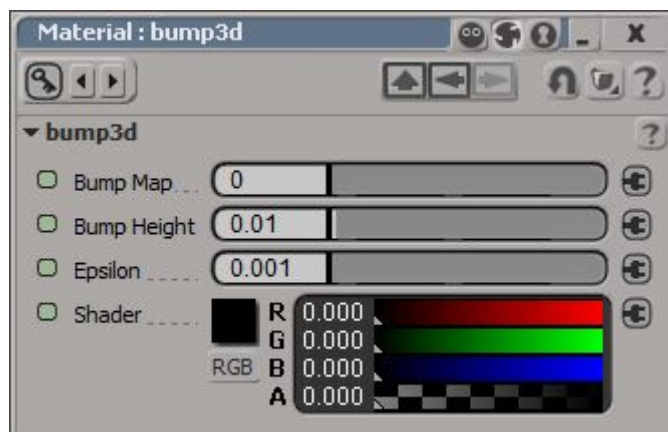
Bump Height

This allows you to scale the height of the bump map output.

Normal

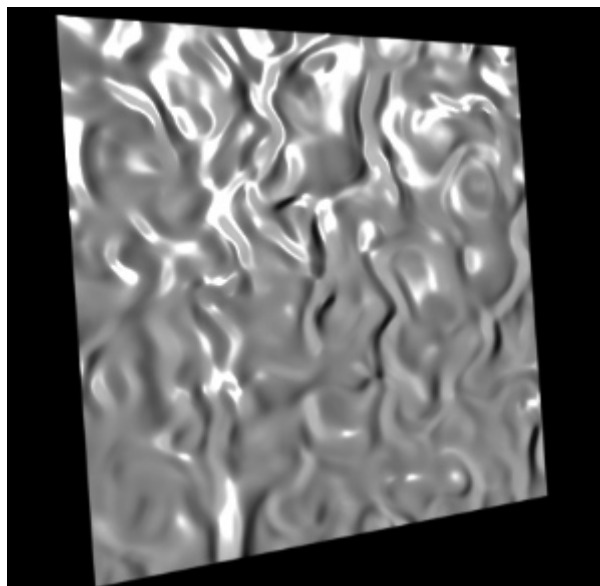
When linked, this attribute will be used as the normal that will get bumped. It can be used to chain multiple normal adjustments (normal maps, bumps, etc).

Bump3d



Provides bump mapping based on a 3d input. Bump3D works by evaluating the bump input at different points ($P + \epsilon$, $P + \epsilon$, $P + \epsilon$). Since the only thing displaced is the point, the UV coordinates will be the same in the different lookups. These will give the same texel in the image and result in no perturbation of the normal. If you are using texture file images, you should use bump2D for images.

Keep in mind that each `bump3d` node needs to evaluate the texture map four times (to get the partial derivatives in X, Y, Z).



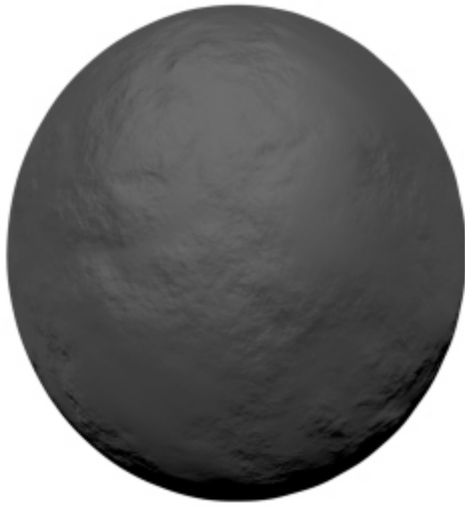
Grid using bump3d with a noise shader attached to the Bump parameter

Bump•Map

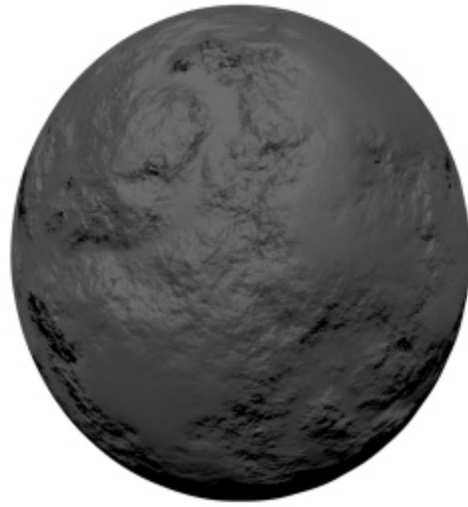
The input used to compute the normal perturbation, provided as a float value representing the height, or directly as an RGB texture map, in which case the first component (R) will be used.

Bump Height

Scales the height of the bump map output.



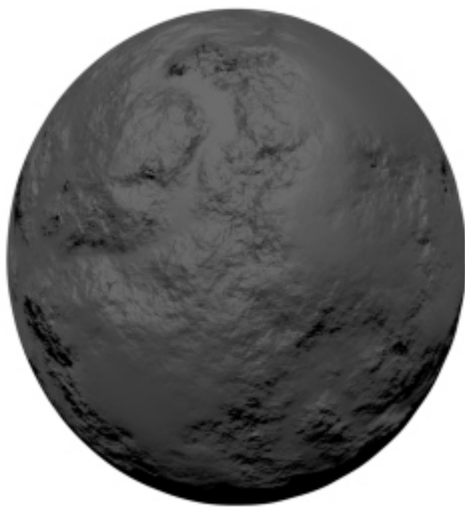
0.02



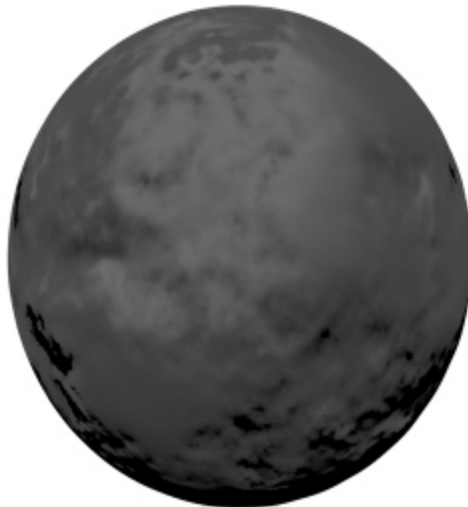
0.06

Epsilon

Epsilon is used to remove noisy shading artifacts caused by insufficient precision in the small floating point differences between P and $P + \epsilon$. Ideally, it should be left at its default value of $1e-5$, but in problematic cases, it may be preferable to increase the epsilon, which may solve the numerical artifacts trading them for a blurrier bump.



0



0.05

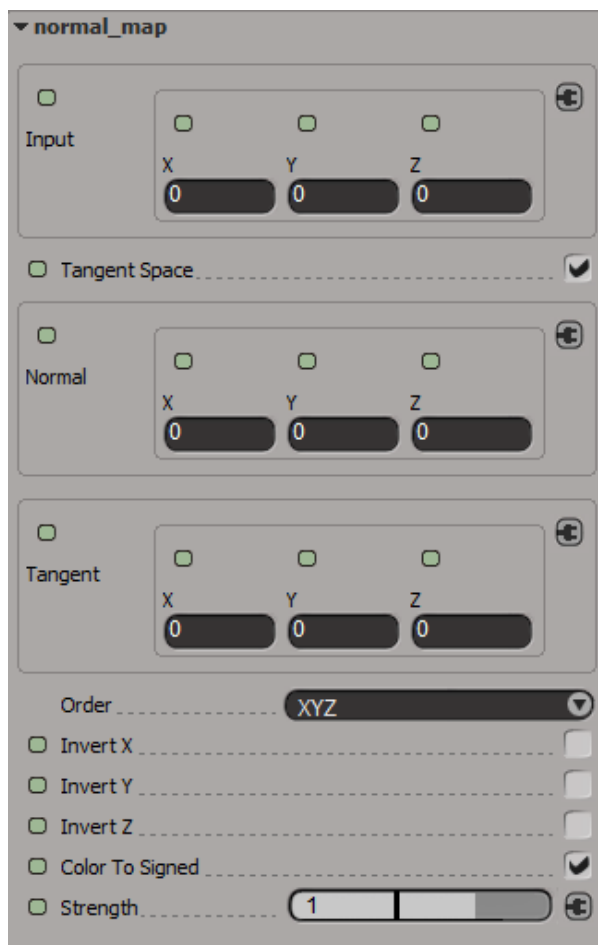
•

Normal

Outputs a normal vector that can be linked to normal parameters in `ambocc`, `lambert`, `standard_surface` and `utility` shaders.

•
•

Normal Map



Normal mapping works by replacing the interpolated surface normal by the one evaluated from an RGB texture, where each channel (Red, Green, Blue) corresponds to the X, Y and Z coordinates of the surface normal. It can be faster than bump mapping since bump mapping requires evaluating the shader underneath at least three times.

Input

The map, usually exported from Mudbox or ZBrush.

Tangent Space

Specifies if the input is in world space or tangent space.

Normal

The normal and tangent parameters can be optionally linked to define a custom tangent coordinate system that the input is transformed from. If the normal is not linked, it will use the default surface normal.

Tangent

The tangent map. Together with the shading normal, it defines the tangent coordinate system that the input vector applies to. If available from your sculpting tool, you should connect here the tangent map that the normal map relies on. If 0, the shader attempts the following actions to build the frame:

1. Look for vector user data named "tangent" and "bitangent".
2. Use the UV derivatives.
3. Build its own local frame.



The shader works in tangent space only. If your tangent map was exported in either world or object space, you could instead use the more generic space_transform shader.

Order

Lets you shuffle the input channels order.

Invert X

If enabled, inverts (1-channel) the x input channel.

Invert Y

If enabled, inverts (1-channel) the y input channel.

Invert Z

If enabled, inverts (1-channel) the z input channel.

Color To Signed

For 8-bit maps. If enabled, the input is remapped to the $[-1, 1]$ range.



In general, if you want to use a 16/32 format that can contain negative values, you should avoid the $[-1..1] \rightarrow [0..1]$ conversion, and so that map would contain already the normal value, in which case color-to-signing back should be avoided in the shader.



The default values of these parameters (order, invert, color_to_signed) let you correctly import a map generated in Mudbox in Tangent coordinate space.

Strength

.

Closure

This shader parameter may be used to link **closure shaders** to control color and transparency. The output in Softimage is a color, but internally it's a closure. A Closure shader is created on the fly whenever a Standard shader is created from the Arnold top menu.



Arnold 5 differentiates between color and closure type outputs. These are BSDF, BSSRDF, emission, matte, transparency and volume closures. A closure can be connected only to a closure type input. You can only link a shader to these closure type parameters.



Closure shaders can only be connected to closure ports.

Ray Switch Shader connected to Closure shader (closure types are colored purple)

-
-

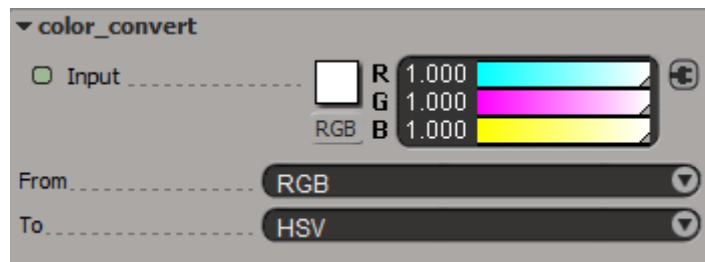
Color Shaders

Below is a list of color correction shaders available in SltoA:

-

- [Color Convert](#)
- [Color Correct](#)
- [Color Jitter](#)
- [Composite](#)

Color Convert

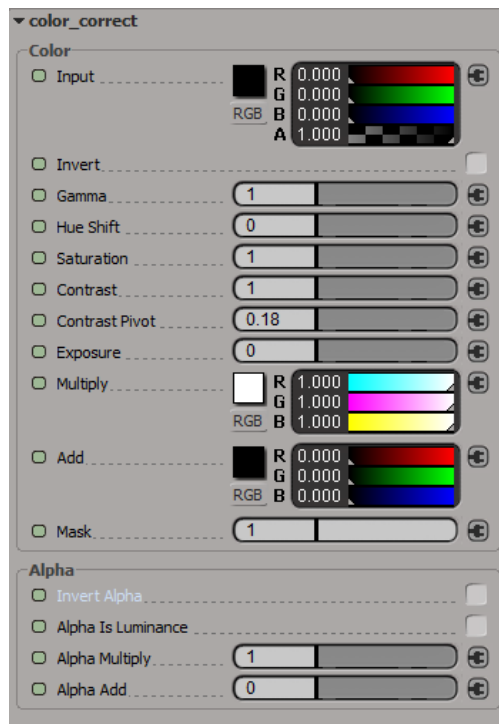


.

The color_convert node allows you to convert input color from one of the following color spaces to another:

- **RGB**: Linear RGB
- **HSV**: Hue, Saturation, Value

Color Correct



.

The color_correct node allows you to adjust the gamma, hue, saturation, contrast, and exposure of an image. Alters the **Input** color with the following operator, applied in the same order as the parameters.

Input

Input color to apply the color correction to.

Invert

Invert the input RGB color.

Gamma

Applies a gamma correction to the color.



1 (default)



2

Hue Shift

Rotates the color hue, a value of 1 means a full rotation. •



0 (default)



0.1

Saturation

Scale the saturation by this factor.



1 (default)



2

Contrast

Scales values around the Contrast Pivot.



0.1



0.2

Contrast Pivot

The origin of the contrast scaling. The default is 0.18 which is the average perceptual mid-gray.



0.1



0.2

Exposure

Multiplies the color using a photo f-stop scale, where an increment of 1 doubles the luminosity.



0 (default)



1

Multiply

Multiply the color by this factor.



1 (default)



2

Add

Adds this value to the color.



0 (default)



0.2

Mask

Blends between the **Input** color and the color corrected one.



0



1 (default)

Alpha

Invert Alpha

Invert the input alpha.

Alpha is Luminance

Ignore the input alpha, setting the output alpha to the luminance of the RGB color.

Alpha Multiply

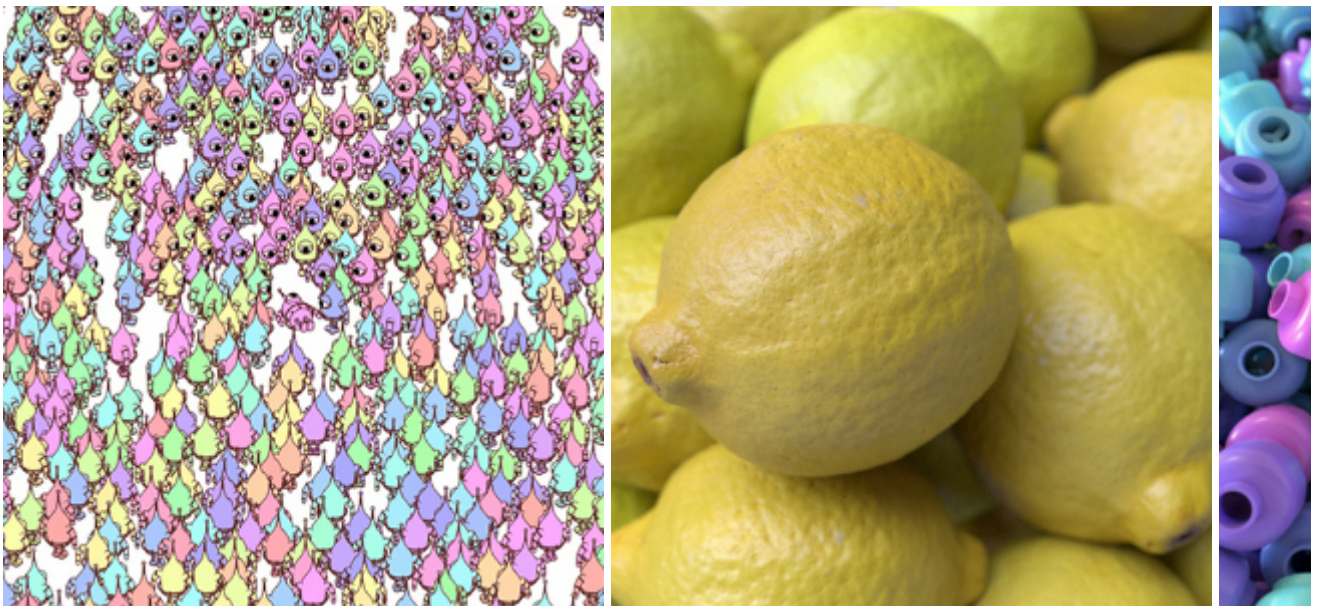
Multiply the output alpha with a factor.

Alpha Add

Add a value to the output alpha.

•

Color Jitter



The `color_jitter` shader enables you to alter the input color by applying a random color variation. For each of the following parameters, you can specify the range of hue, saturation, and gain (HSV) for the random colors. The seed is used to get a different random variation.

color_jitter

Input

☐ Input

Object

☐ Gain Min 0

☐ Gain Max 0

☐ Hue Min 0

☐ Hue Max 0

☐ Saturation Min 0

☐ Saturation Max 0

☐ Seed 0

Face

☐ Gain Min 0

☐ Gain Max 0

☐ Hue Min 0

☐ Hue Max 0

☐ Saturation Min 0

☐ Saturation Max 0

☐ Seed 0

User Data

☐ Data 0

☐ Gain Min 0

☐ Gain Max 0

☐ Hue Min 0

☐ Hue Max 0

☐ Saturation Min 0

☐ Saturation Max 0

☐ Seed 0

Procedural

☐ Gain Min 0

☐ Gain Max 0

☐ Hue Min 0

☐ Hue Max 0

☐ Saturation Min 0

☐ Saturation Max 0

☐ Seed 0

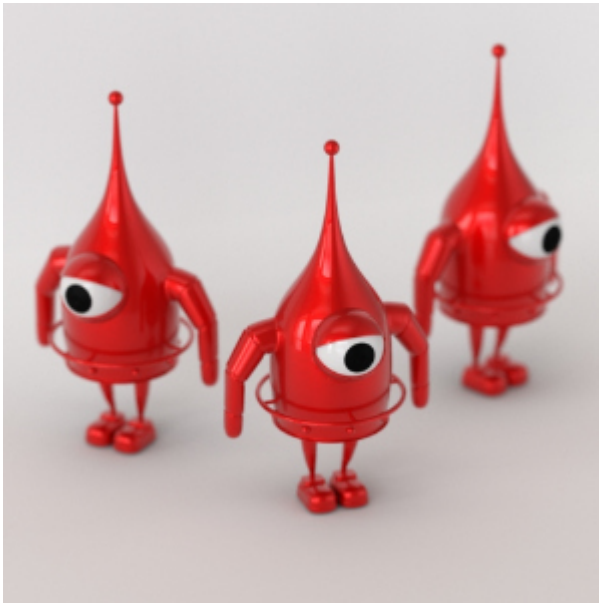
User Data

Gain Min/Max

Specify the range of a random value to be added to gain. The random value is generated based on "data_input."



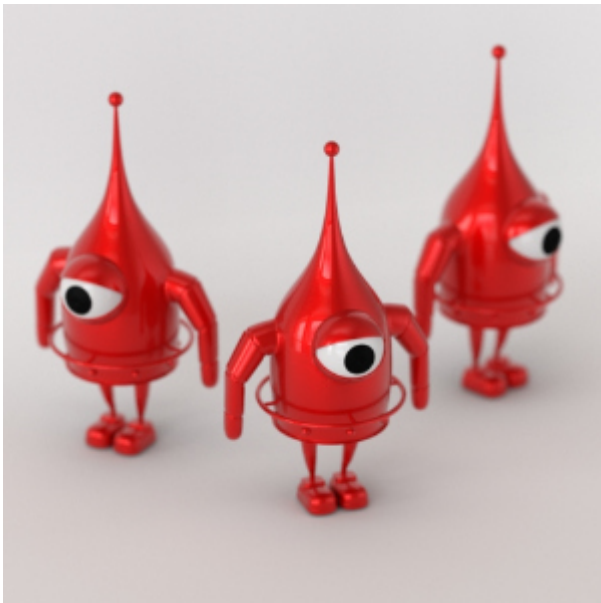
Min: 0. Max: 1



Min: 1. Max: 1

Hue Min/Max

Specify the range of a random value to be added to hue. The random value is generated based on "data_input."



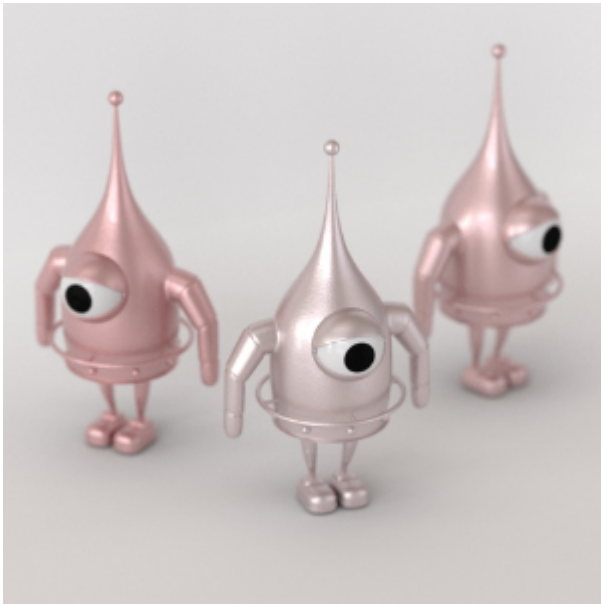
Min: 0. Max: 0



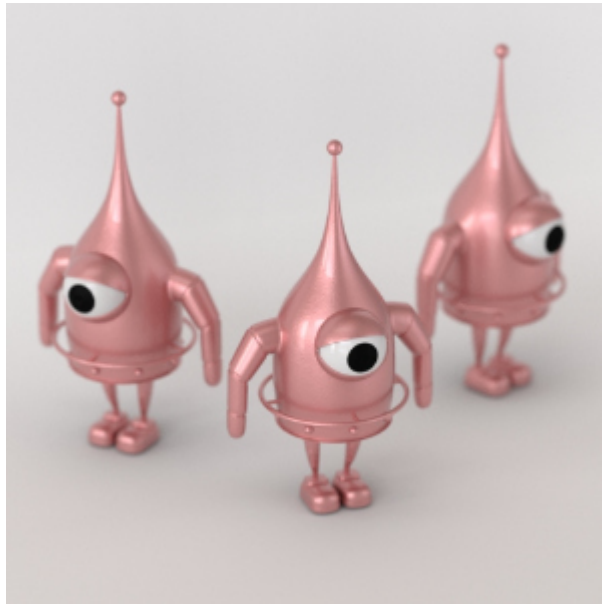
Min: 0. Max: 0.1

Saturation Min/Max

Specify the range of a random value to be added to saturation. The random value is generated based on "data_input."



Min: 0. Max: 0.5



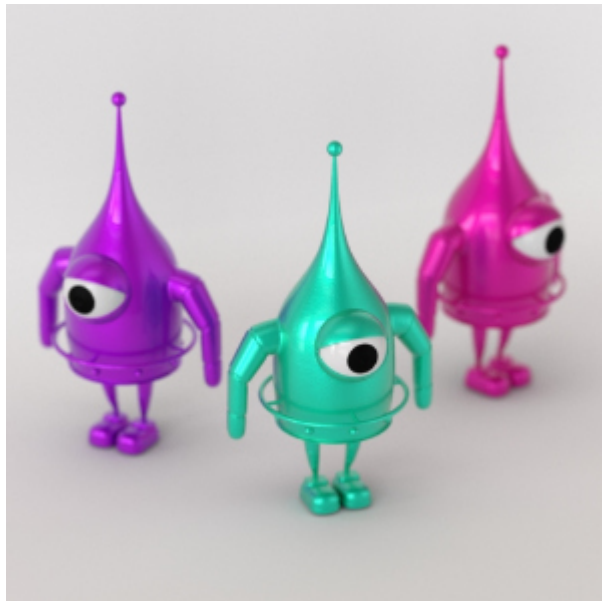
Min: 0.5. Max: 0.5

Seed

Gives a different color variation. This only affects the randomization due to "data_input."



Seed: 0



Seed: 1

Procedural

Gain Min/Max

Specify the range of a random value to be added to gain. The random value is generated based on the name of a procedural object.

Hue Min/Max

Specify the range of a random value to be added to hue. The random value is generated based on the name of a procedural object.

Saturation Min/Max

Specify the range of a random value to be added to saturation. The random value is generated based on the name of a procedural object.

Seed

Gives a different color variation. This only affects the randomization due to the name of a procedural object.

Object

Gain Min/Max

Specify the range of a random value to be added to gain. The random value is generated based on the name of an object.



Min: 0. Max: -1



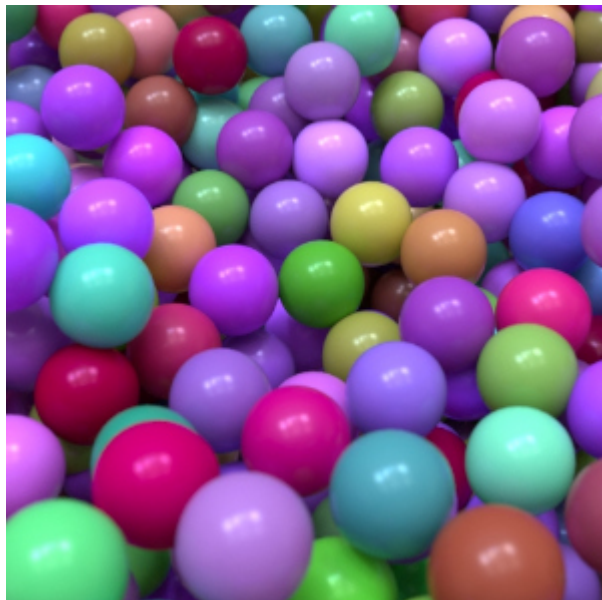
Min: 0. Max: 0

Hue Min/Max

Specify the range of a random value to be added to hue. The random value is generated based on the name of an object.



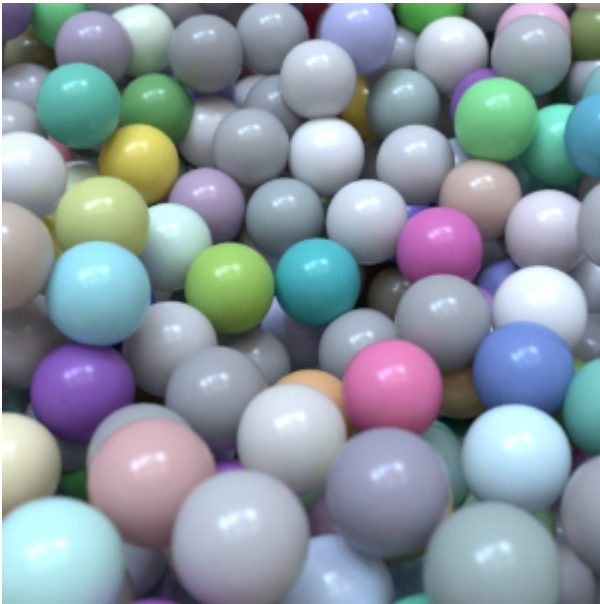
Min: 0. Max: 0



Min: 0. Max: -1

Saturation Min/Max

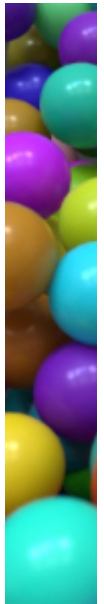
Specify the range of a random value to be added to saturation. The random value is generated based on the name of an object.



Min: 0. Max: -1

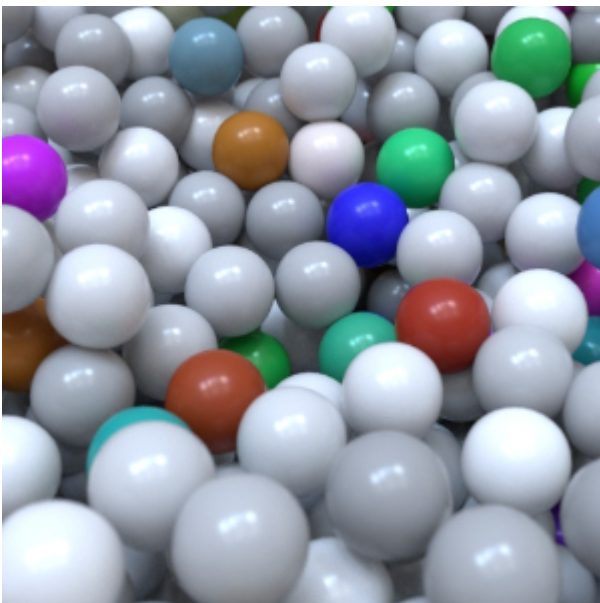


Min: 0. Max: 0



Seed

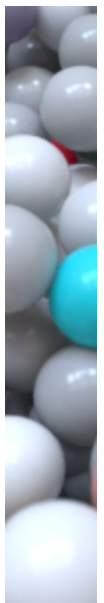
Gives a different color variation. This only affects the randomization due to the name of an object.



0



1

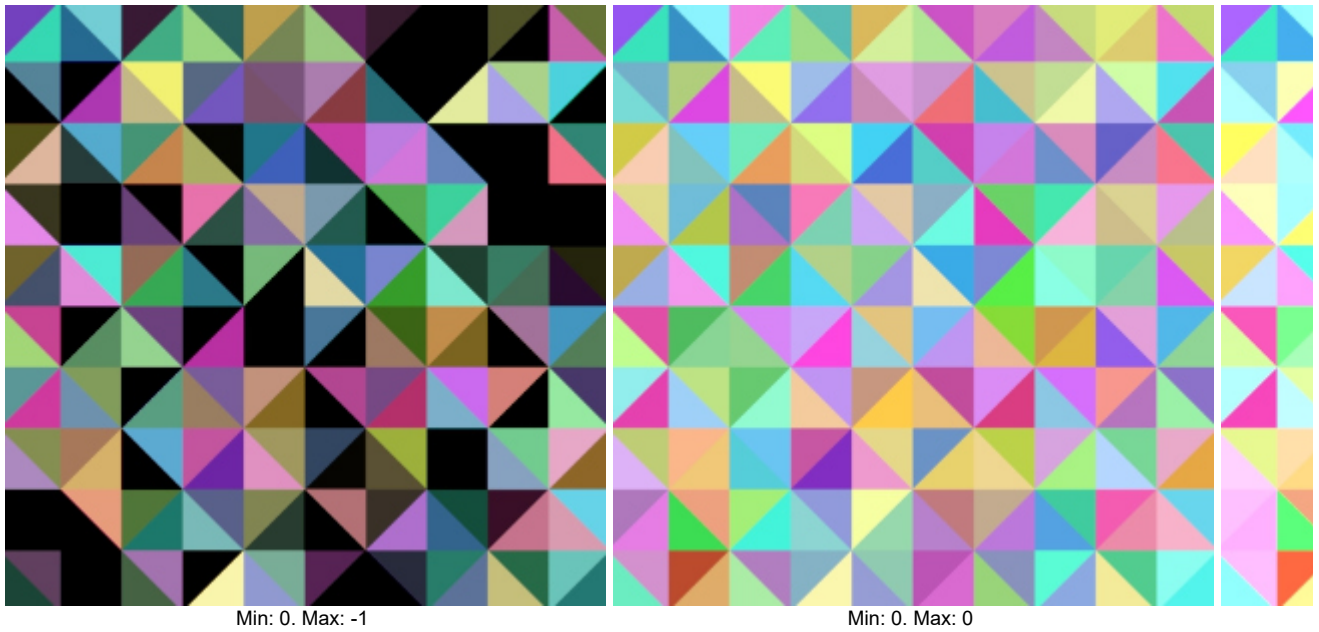


Saturation Min (-10). Saturation Max (2).

Face

Gain Min/Max

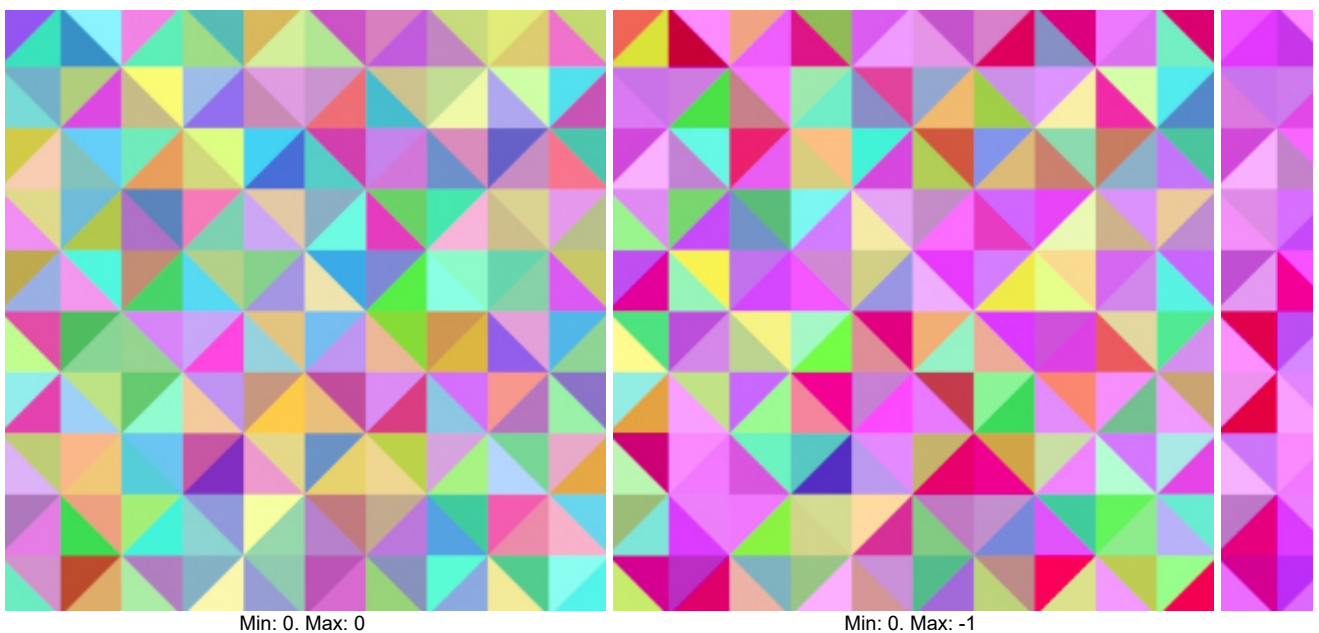
Specify the range of a random value to be added to gain. The random value is generated based on a primitive id.



Care should be taken when combining face variations with certain sorts of adaptive subdivision. The variations may cause flickering when animated.

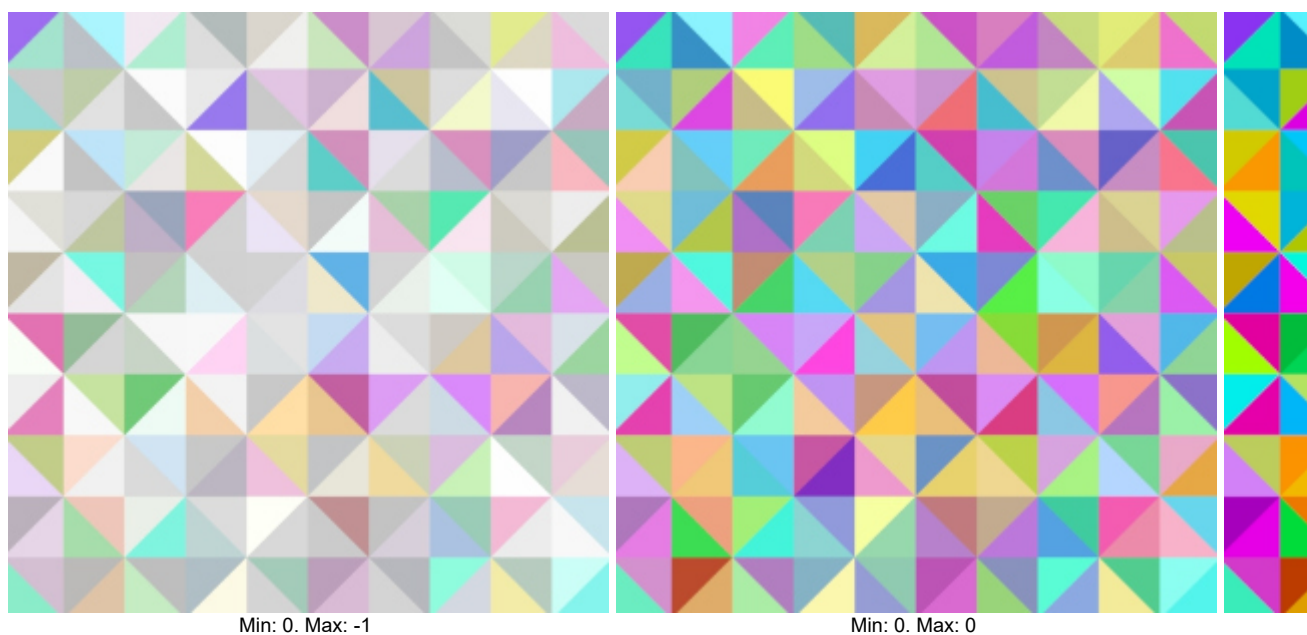
Hue Min/Max

Specify the range of a random value to be added to hue. The random value is generated based on a primitive id.



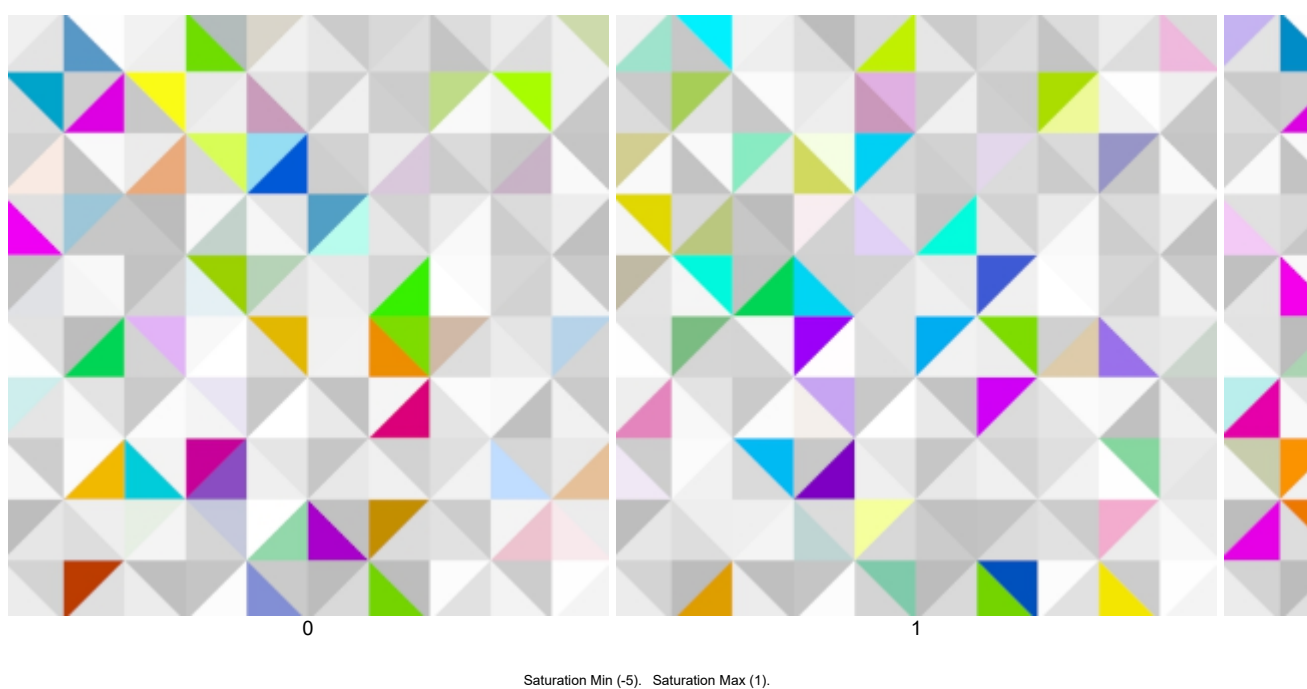
Saturation Min/Max

Specify the range of a random value to be added to saturation. The random value is generated based on a primitive id.



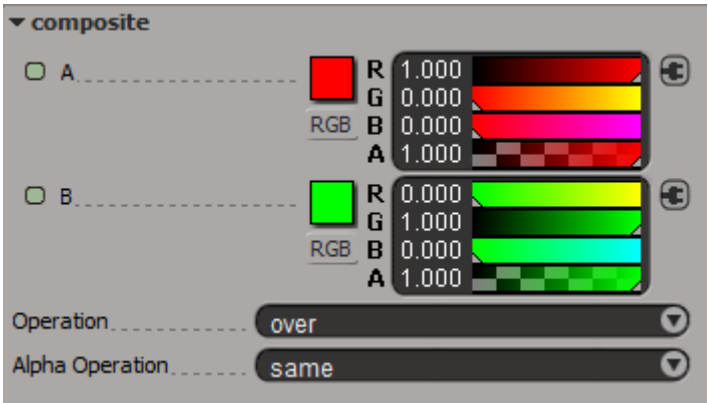
Seed

Gives a different color variation. This only affects the randomization due to a primitive id.





Composite



The composite shader mixes two RGBA inputs according to a blend mode.

Operation

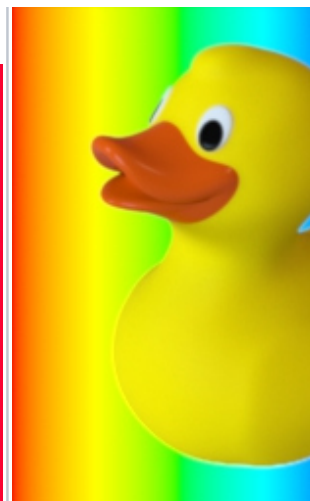
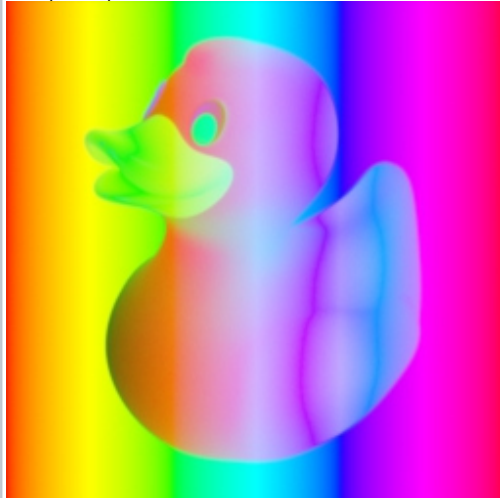
The following blend modes are available:

A Passthrough A	B Passthrough B	Atop $A * B.a + B * (1 - A.a)$
		Disjoint Over $A + B * (1 - A.a) / B.a, A + B$

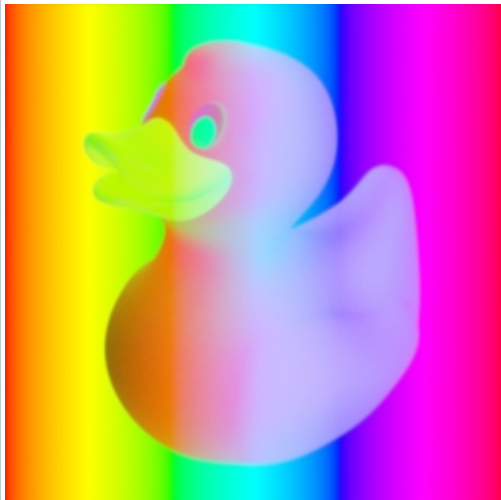
Cojoint Over
 $A + B * (1 - A.a) / B.a, A \text{ if } A.a > B.a$



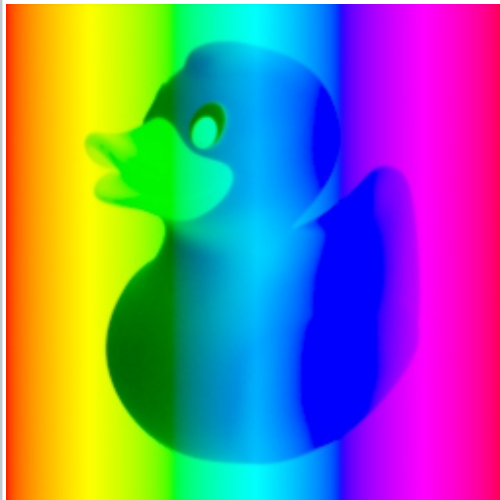
Difference
 $\text{abs}(A - B)$



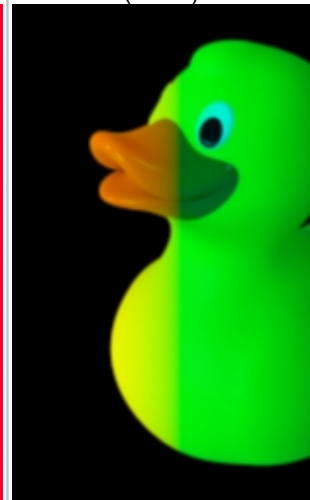
Exclusion
 $A + B - 2 * A * B$



From
 $B - A$



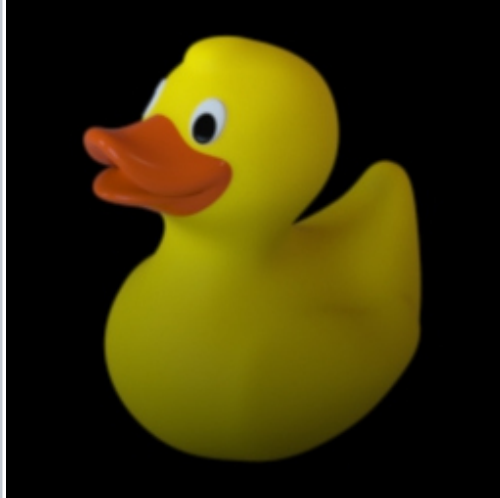
Geometric
 $2 * A * B / (A + B)$



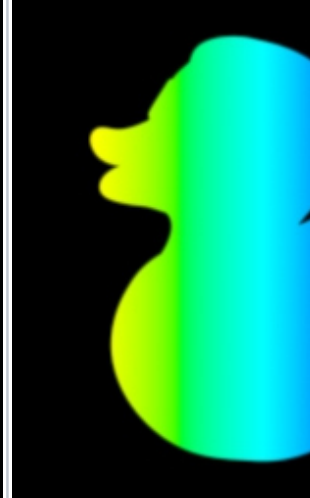
Hypot Diagonal
 $\text{sqr}(A * A + B * B)$



In
 $A * B.a$



Mask
 $B * A.a$



Max
 $\text{max}(A, B)$

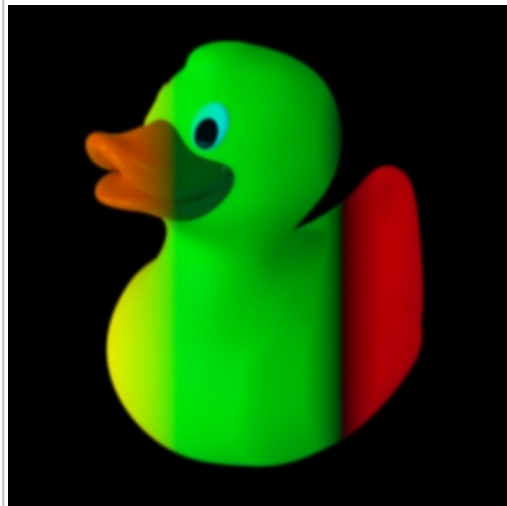
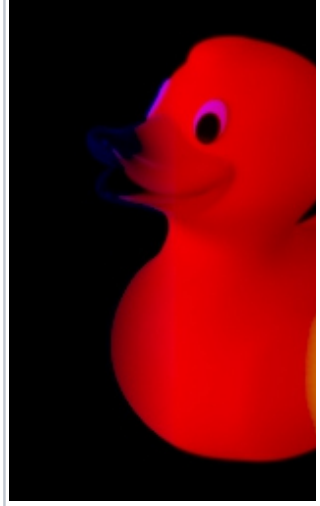
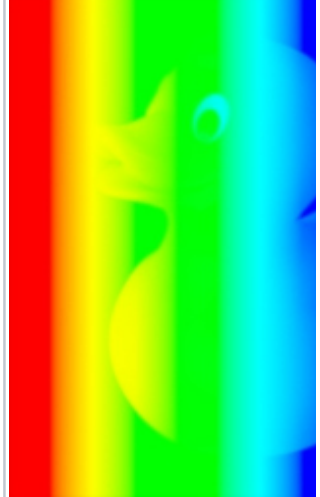


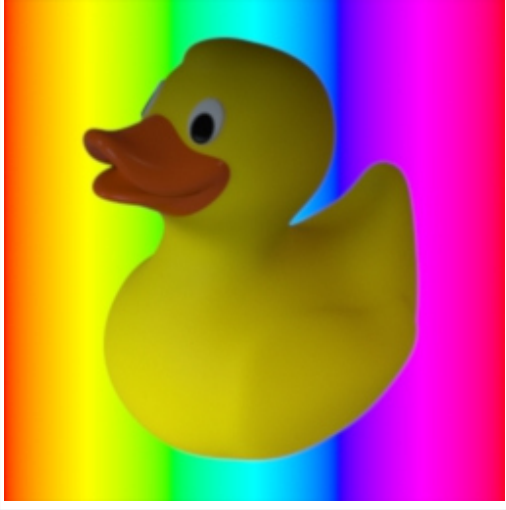
Min
 $\text{min}(A, B)$



Minus
 $A - B$



		
Out $A * (1 - B.a)$	Over $A + B * (1 - A.a)$	Overlay Screen, $A * B$ if $B < 0.5$
		
Screen $A + B - A * B$ if A and B in $0..1$, else $\max(A, B)$	Soft Light Softer version of Hard Light	Stencil $B * (1 - A.a)$
		
Xor $A * (1 - B.a) + B * (1 - A.a)$		



Alpha Operation

The alpha channel operation can be:

- Same: Apply the same operation to Alpha as on RGB.
- A: Use the alpha from A.
- B: Use the alpha from B.

Conversion

The following shaders allow for conversions between data types:

Float to Int

Convert float to an integer value.

Float to Matrix

Construct a 4x4 matrix from float values for each element. The `input_xy` parameters set the value in column `x` and row `y`.

Float to RGB

Create RGB color from individual R, G, B float values.

Float to RGBA

Create RGBA color from R, G, B, A float values.

RGB to Float

Convert RGB color to float value.

RGB to Vector

Convert RGB color to vector.

RGBA to Float

Convert RGB color and alpha value to float.

Vector to RGB

Convert vector to RGB color.

•

Arnold Data Type Conversions

The following table shows how data types are converted within Arnold itself.

Key:

(empty)	conversion is disallowed/ignored										
copy	copy (no need to convert)										
expand	all components set to the same source value										
shallow copy	for pointer types (pointer, node, array, matrix), copy the pointer, not the actual value										
boolean cast	zero means false, non-zero means true, and vice-versa										
upcast	smaller integer type just gets stuffed into a larger integer type										
downcast	larger integer type gets truncated into a smaller integer type										
average	the average of the components are taken										

	•	•	•	•	•	<u>TARGET</u>	•	•	•	•	•
<u>SOURCE</u>	enum	boolean	byte	int	uint	float	RGB	RGBA	vector	point	point2
enum	copy	•	•	•	•	•	•	•	•	•	•
boolean	•	copy	boolean cast	boolean cast	boolean cast	•	•	•	•	•	•
byte	•	boolean cast	copy	upcast	upcast	•	•	•	•	•	•
int	•	boolean	downcast	copy	copy	•	•	•	•	•	•



		cast									
uint	•	boolean cast	downcast	copy	copy	•	•	•	•	•	•
float	•	•	•	•	•	copy	r•=•flt g•=•flt b•=•flt	r•=•flt g•=•flt b•=•flt a•=•1	x•=•flt y•=•flt z•=•flt	x•=•flt y•=•flt z•=•flt	x•=•flt y•=•flt
RGB	•	•	•	•	•	average(rgb)	copy	r•=•r g•=•g b•=•b a•=•1	x•=•r y•=•g z•=•b	x•=•r y•=•g z•=•b	x•=•r y•=•g
RGBA	•	•	•	•	•	average(rgb)	r•=•r g•=•g b•=•b	copy	x•=•r y•=•g z•=•b	x•=•r y•=•g z•=•b	x•=•r y•=•g
vector	•	•	•	•	•	vector.x	r = x g = y b = z	r = x g = y b = z a = 1	copy	x = x y = y z = z	•
point	•	•	•	•	•	pnt.x	r•=•x g•=•y b•=•z	r•=•x g•=•y b•=•z a•=•1	x•=•x y•=•y z•=•z	copy	•
point2	•	•	•	•	•	pnt2.x	r = x g = y b = 0	r•=•x g•=•y b•=•0 a•=•1	x•=•x y•=•y z•=•0	x•=•x y•=•y z•=•0	copy
string	•	•	•	•	•	•	•	•	•	•	•
node	•	•	•	•	•	•	•	•	•	•	•
pointer	•	•	•	•	•	•	•	•	•	•	•
array	•	•	•	•	•	•	•	•	•	•	•
matrix	•	•	•	•	•	•	•	•	•	•	•

•
•

•	•	•	•	TARGET	•
SOURCE	string	node	pointer	array	matrix
enum	•	•	•	•	•
boolean	•	•	•	•	•
byte	•	•	•	•	•
int	•	•	•	•	•
uint	•	•	•	•	•
float	•	•	•	•	•
RGB	•	•	•	•	•
RGBA	•	•	•	•	•
vector	•	•	•	•	•
point	•	•	•	•	•
point2	•	•	•	•	•
string	copy	•	•	•	•
node	•	•	shallow copy	•	•
pointer	•	•	shallow copy	•	•
array	•	•	•	shallow copy	•
matrix					



	•	•	•	•	shallow copy
--	---	---	---	---	--------------

-
-

A few notes:

- The node type (AI_TYPE_NODE) is currently resolved at scene creation time, and node-to-node shader connections are not formally allowed since they internally are turned into a connection based on the output type of the shader node. However, whenever one wants to pass a node along, they can pass it as a generic pointer type instead of using other mechanisms such as shader message passing. The vast majority of the time, if you are using pointer passing inside of your shaders, however, you should re-think your design.
- No automatic conversions exist between integer types and floating-point types.
- No automatic conversions exist between enumerations and any other type.

-
-

Environment Shaders

The following environment shaders are available for SltoA:

- [Physical Sky](#)

Physical Sky



This shader implements a variation of the [Hosek-Wilkie](#) sky radiance model, including the direct solar radiance function. You can plug it into the environment, or to the color input of a Skydome light (which benefits from efficient importance sampling, sending more rays to bright areas around the sun) or add it directly as an environment shader. Currently, this shader is invisible to GI diffuse and specular rays. To use it as a light source you must attach it to a [skydome_light](#) with sufficient resolution to capture the small solar disk.



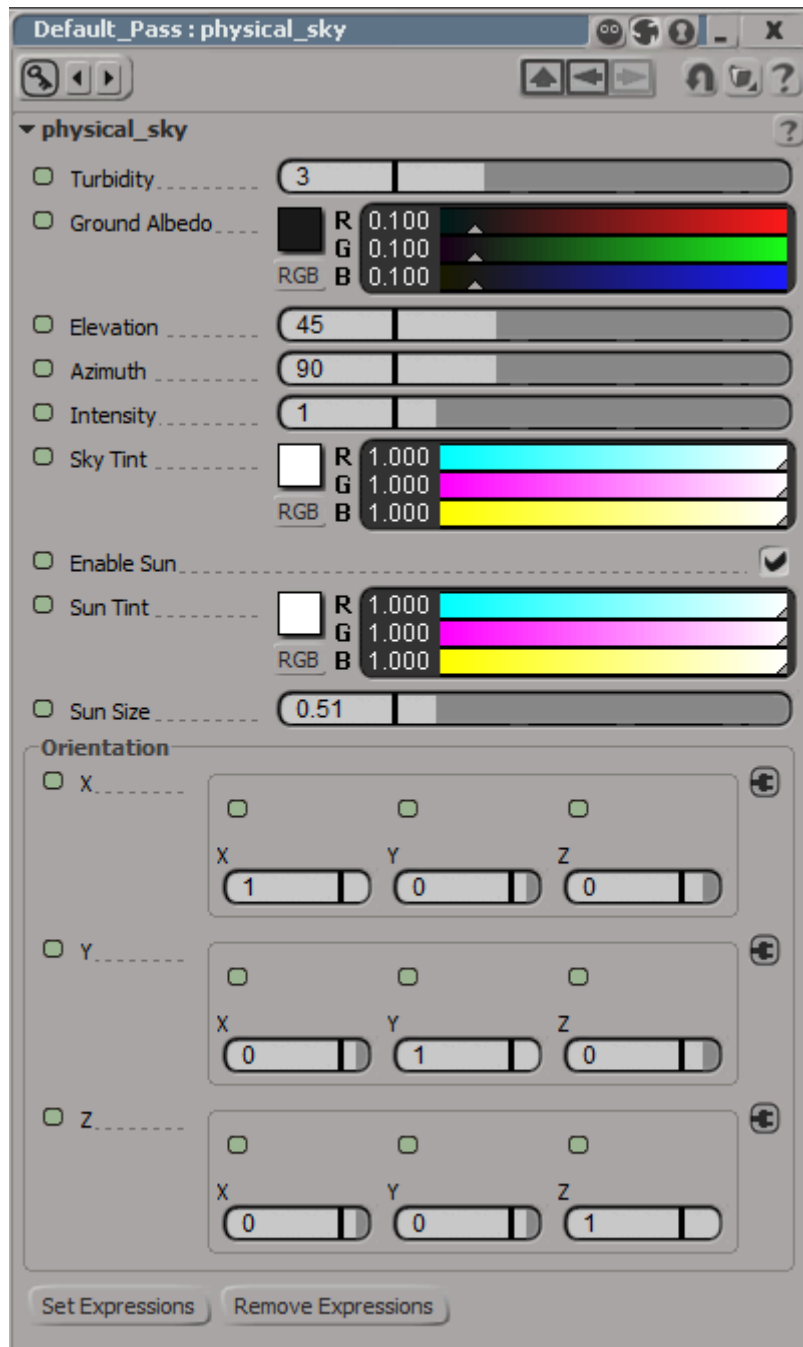
This is an experimental shader. Although it has been tested extensively, please be warned before using it in production.



If you experience flickering during an animation, this is likely because the skydome_light resolution is too low to capture the sun. Try increasing the resolution of the skydome_light.

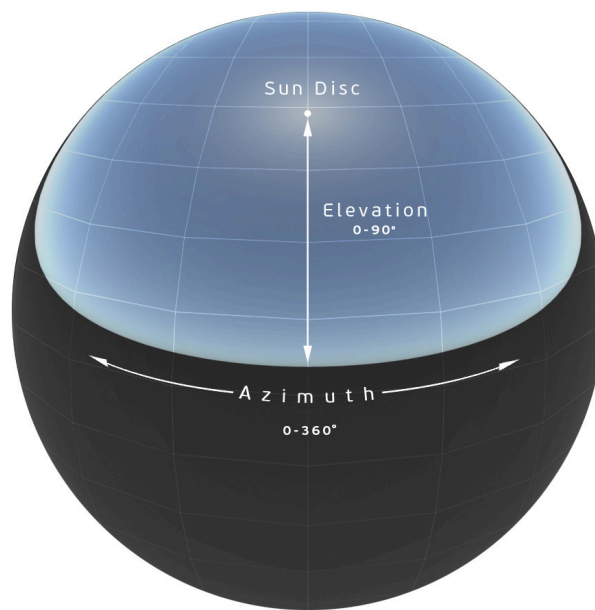


Due to the inherent nature of the physical_sky shader, there will be a horizon line, and depending on the camera position, the ground plane below the horizon will always appear dark.



Physical sky attributes

The physical sky uses a polar coordinate system. Elevation has an angle between 0 degrees to 180 degrees (sunrise to sunset). The sun is at the zenith at 90 degrees. The azimuth has an angle between 0 degrees to 360 degrees.

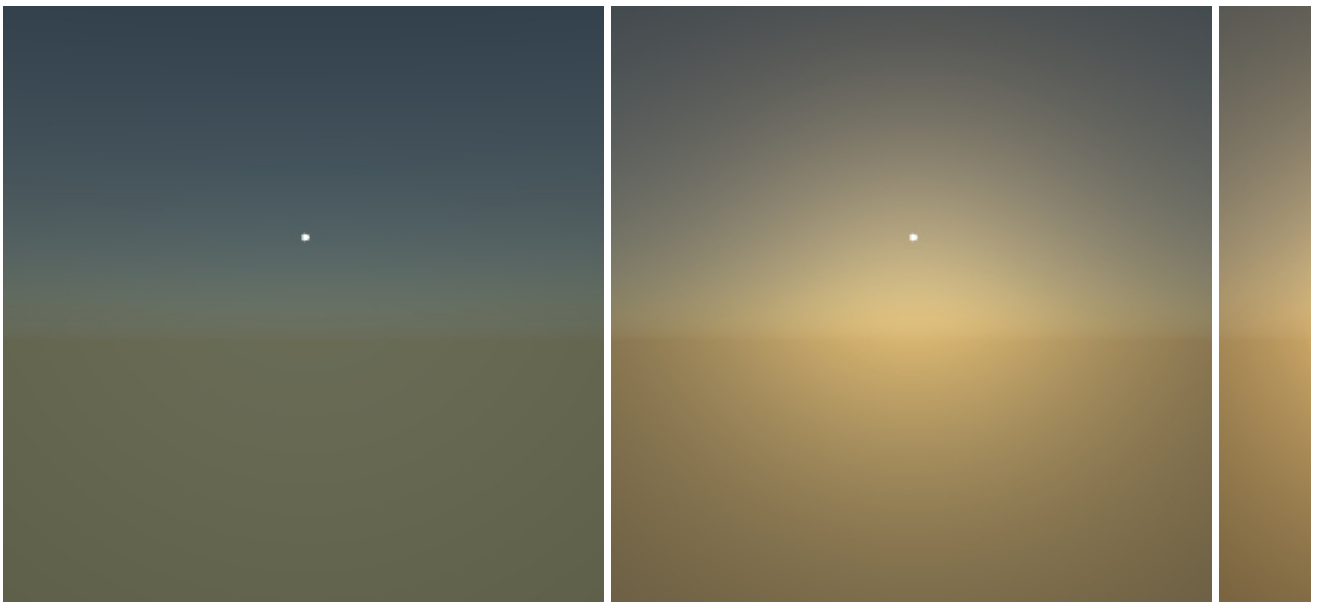


Turbidity

Turbidity determines the overall aerosol content (dust, moisture, ice, fog) of the air. It is used to easily define sky appearance and affects the color of the sun and sky.

Turbidity values range from 1 to 10:

- 2: Yields a very clear, Arctic-like sky.
- 3 (default): A clear sky in a temperate climate.
- 6: A sky on a warm, moist day.
- 10: A slightly hazy day.

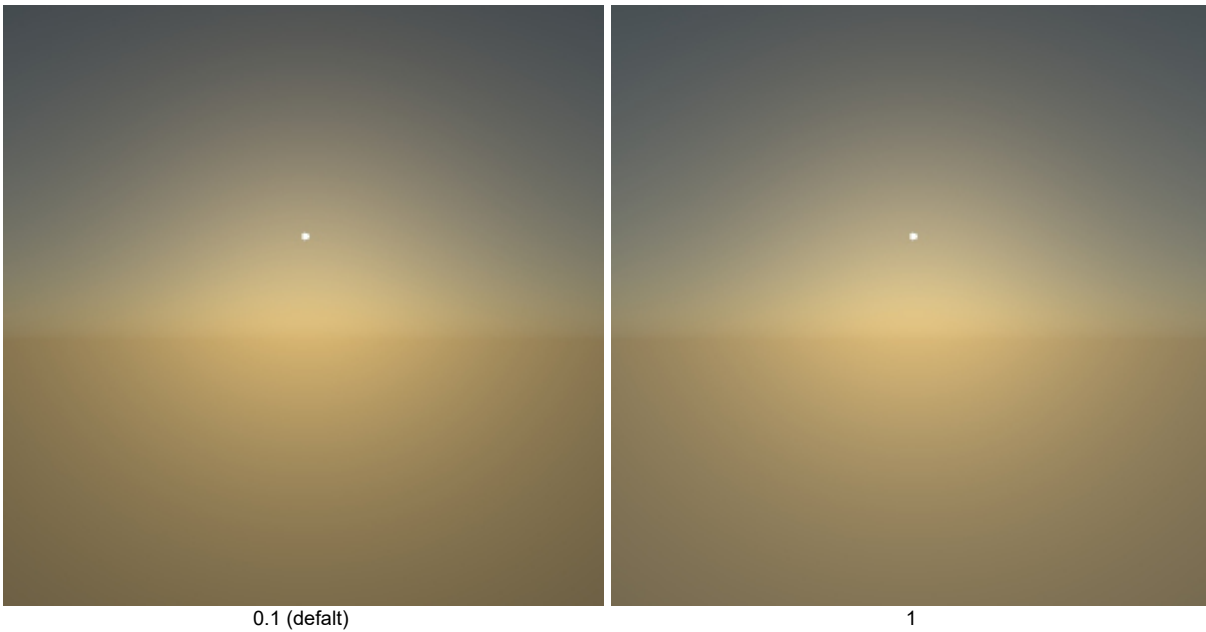


0

3 (default)

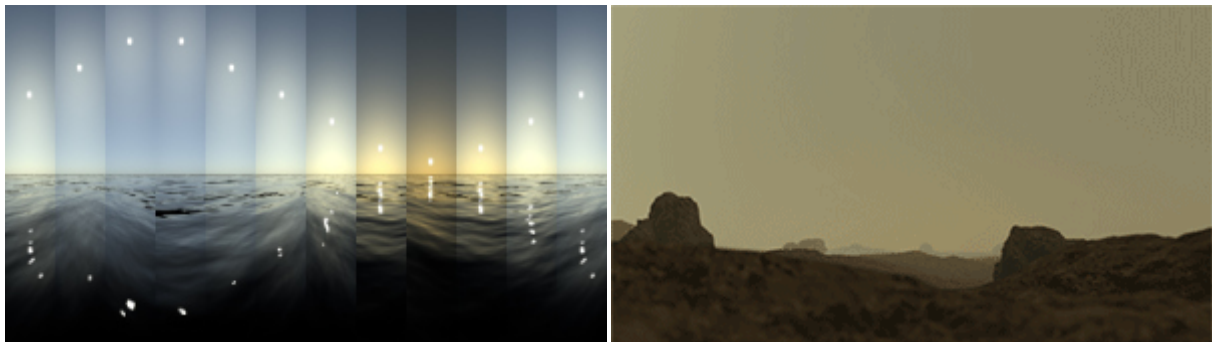
Ground Albedo

The amount of light reflected from the planet surface back into the atmosphere. This is an RGB value between 0 and 1, where 0 0 0 would correspond to a black ground, and 1 1 1 would be a white ground. Note, that this is a subtle effect (noticeable in the top corners of the images below).



Elevation

The angle between the sun and the observable horizon. The range is between 0 and 180 (90 to 180 is a mirror reflection of 0 to 90).



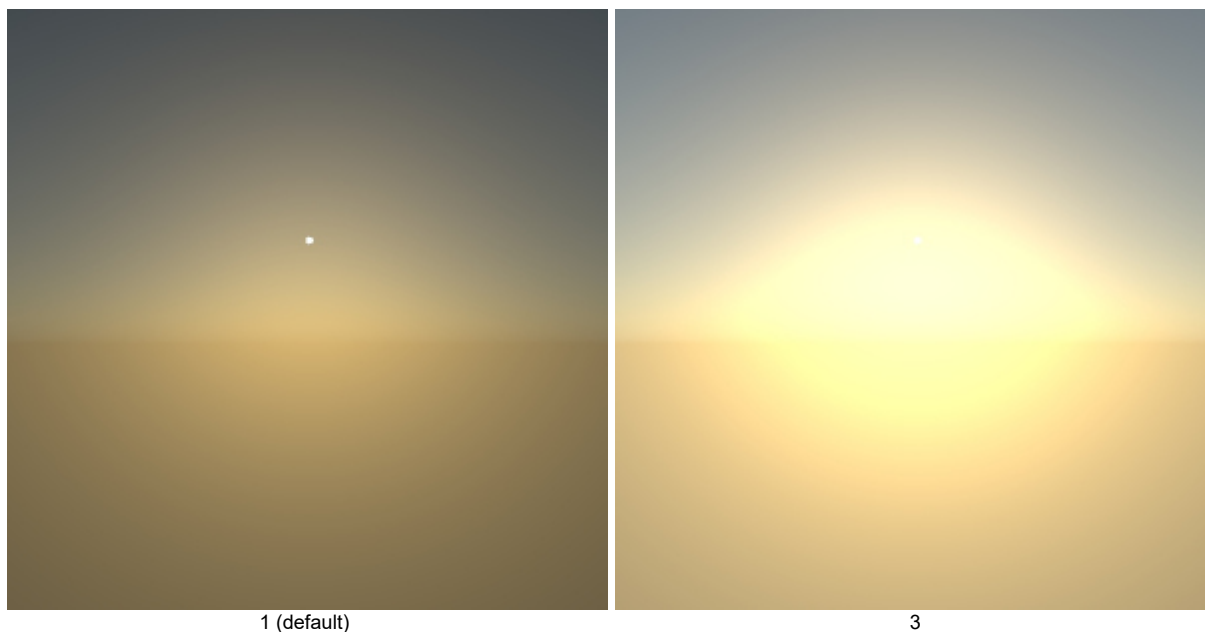
Left - Sun Elevation: 0 to 48 degrees (azimuth 0 to 360 degrees). Right - Sun Elevation: 0 to 20 degrees.

Azimuth

The angle of the sun around the horizon. Measured from the north, increasing towards the east (0 to 360 degrees).

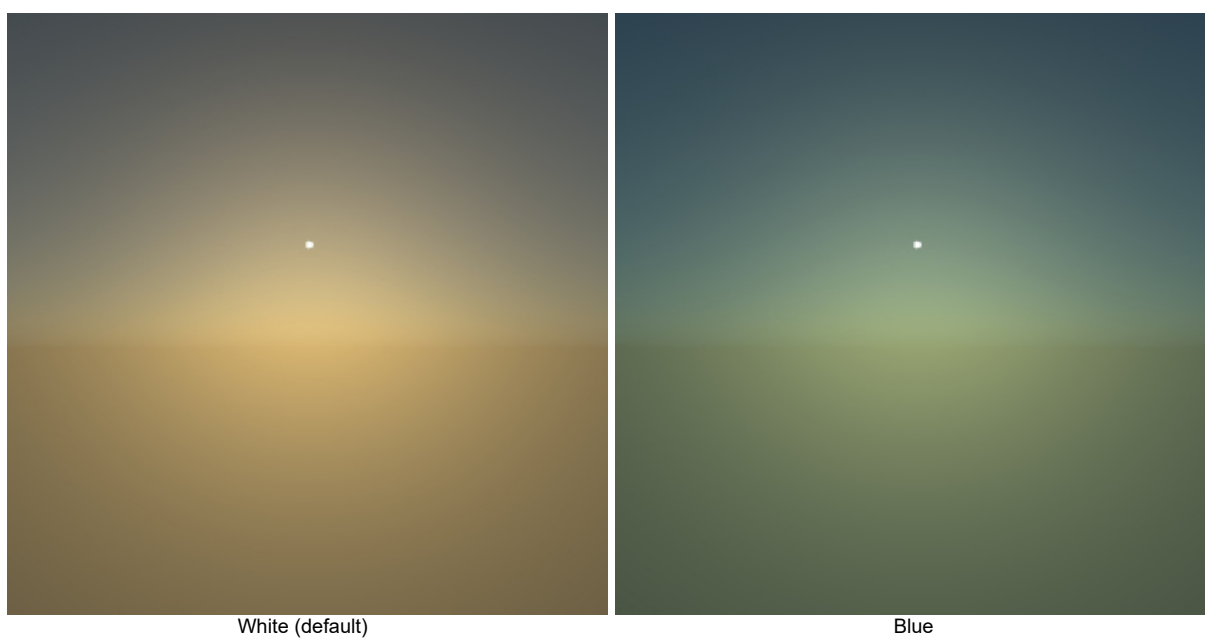
Intensity

Intensity is a scalar multiplier for the sky radiance. This value is similar to the sky_tint. However, the sky_tint uses RGB values whereas intensity uses scalar values (easier to adjust).



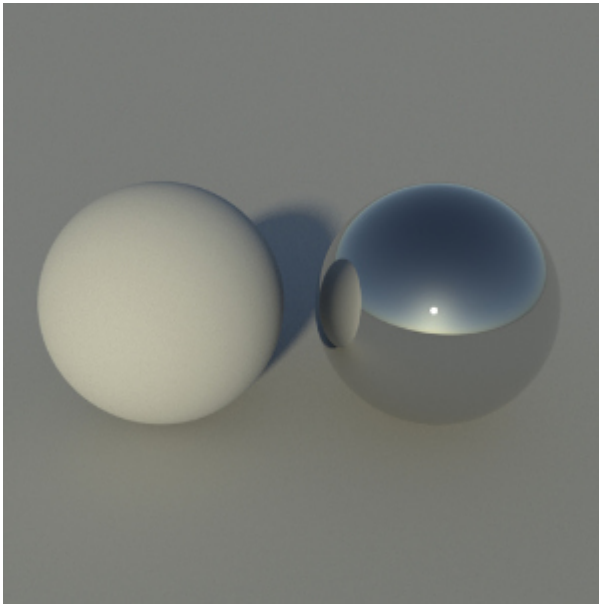
Sky Tint•

It allows you to set a color that 'tints' the sky color. The sky tint should be used for subtle coloration. For example, making the sky a slightly deeper blue color.

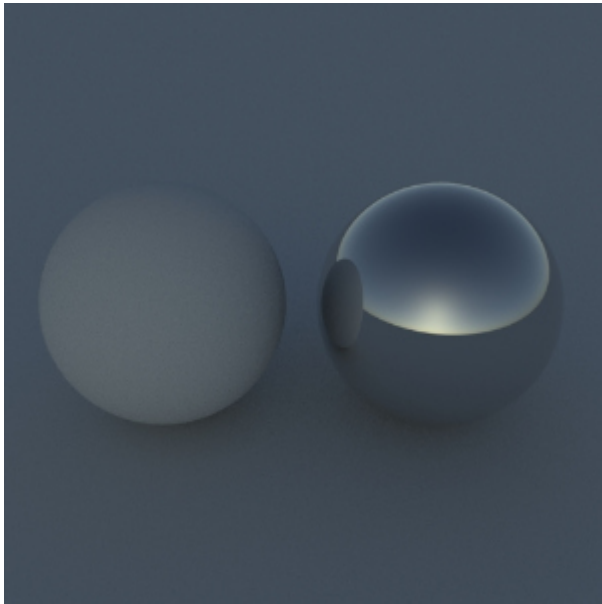


Enable Sun

The toggle switch that enables/disables the visibility of the sun.



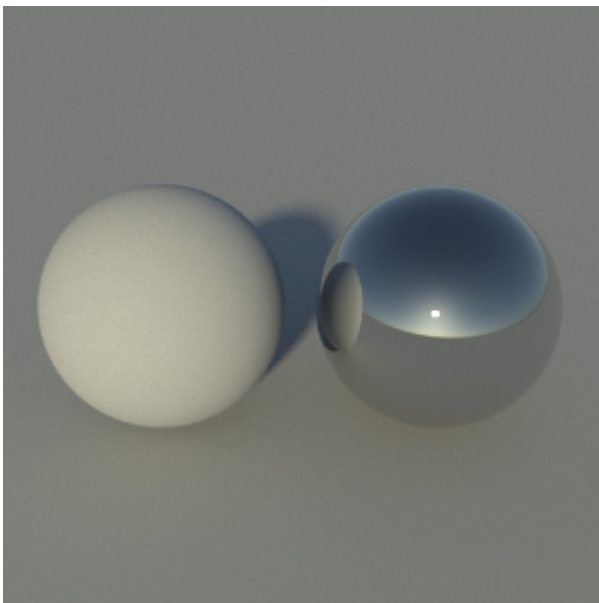
Enabled (default)



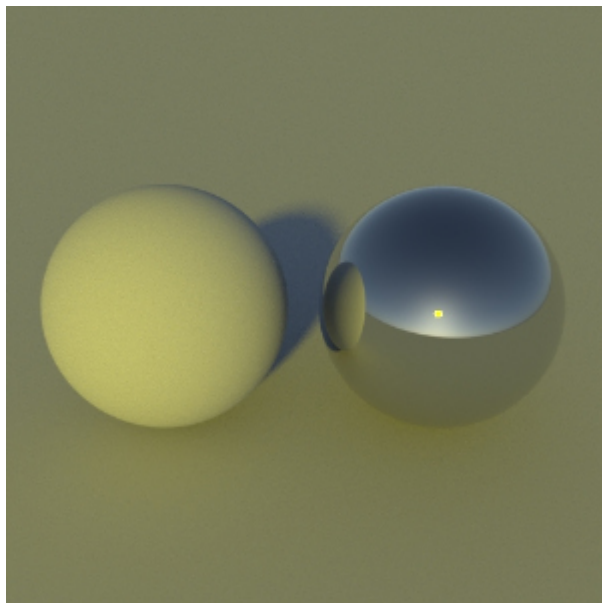
Disabled

Sun Tint

It allows you to set a color that 'tints' the color of the sun. This is an RGB multiplier that should be used for subtle coloration. Any non-subtle coloration will give physically inaccurate results and will produce a strange inconsistency between the sun and the sky colors (unless both tints are exactly the same).



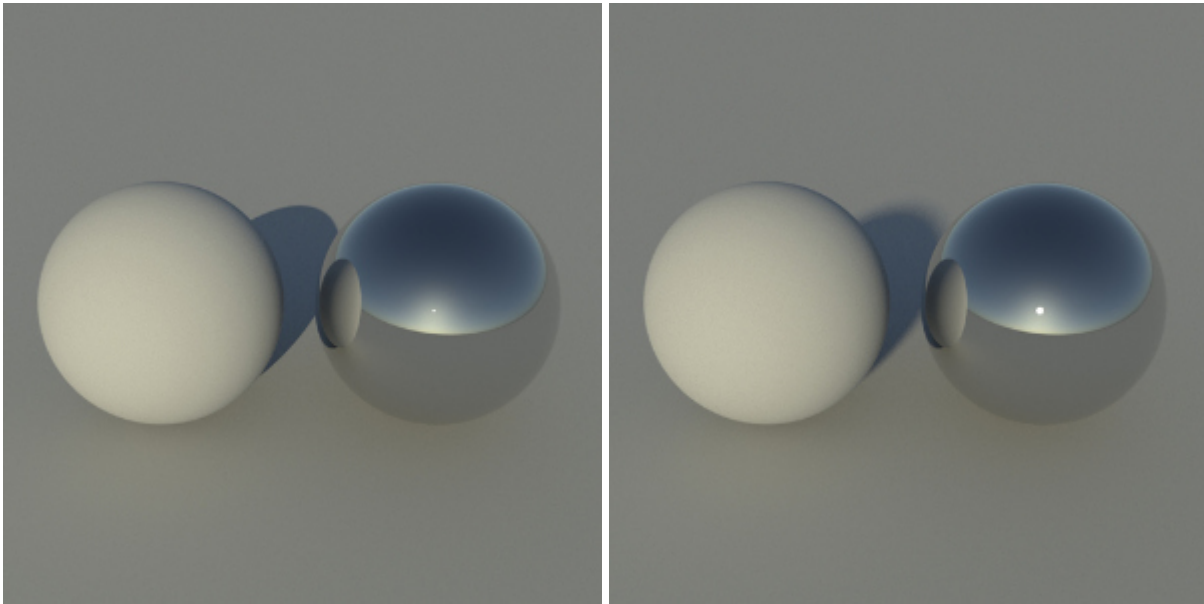
White (default)



Yellow

Sun Size

Sets the size of the visible solar disk. It is possible to change the size of the sun for 'artistic' reasons. However, 0.51 is the solid angle (degrees) of the sun, as seen from the earth. Increasing this value increases the area size of the sun and will, therefore, create softer shadows.



0.51 (default value)

5

Orientation XYZ

The **Set Expressions** button can be used to link the XYZ axes against an object's orientation. Once picked, the rotation axes of an object is transformed, by a set of expression, into the corresponding axes set. This way, you can simply rotate the object to interactively adjust the sky orientation. To unlink the axes from the object, click **Remove Expressions**.

•

✔ Fireflies

Fireflies can appear when rendering scenes with glass surfaces and the `physical_sky` shader. They are caused by the bright sun disc from the `physical_sky` that is connected to the Background. A workaround is to connect a different `physical_sky` to the Background that has `enable_sun` disabled.

Fireflies caused by sun in `physical_sky`'Sun Disc' disabled for Environment -> `physical_sky`

••

•



Further examples

Math Shaders

A collection of mathematical shaders. Math shaders can work on color or vector inputs.

Abs

Return the **absolute value** of input.

Add

Return $\text{input}_1 + \text{input}_2$.

Atan

Return the **arctangent** of y/x . The resulting value is in the range $[-\pi/2, \pi/2]$, using the signs of the two arguments to determine the quadrant of the result.

Compare

Compare input_1 and input_2 with the following operators and return true or false:

- Equal (==)
- Not Equal (!=)
- Greater Than (>)
- Less Than (<)
- Greater Than or Equal (>=)
- Less Than or Equal (<=)

Complement

Return **one's complement** $(1 - \text{input})$. Also known as **reverse video**.

Cross

Compute the **cross product** between two vectors, defined as the vector perpendicular to both input vectors, with its direction defined by the **right-hand rule**.

$$\mathbf{a} \times \mathbf{b} = \begin{pmatrix} a_y b_z - a_z b_y \\ a_z b_x - a_x b_z \\ a_x b_y - a_y b_x \end{pmatrix}$$

The length of the cross product can be interpreted geometrically as:

$$|\mathbf{a} \times \mathbf{b}| = |\mathbf{a}| |\mathbf{b}| \sin(\theta)$$

Divide

Return $\text{input}_1 / \text{input}_2$.

Dot

Compute the **dot product** between two vectors as follows:

$$\mathbf{a} \cdot \mathbf{b} = a_x b_x + a_y b_y + a_z b_z$$

The result is a scalar value that can be interpreted geometrically as:

$$\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}| |\mathbf{b}| \cos(\theta)$$

where the length of vector $\mathbf{a}$ is denoted by

$$|\mathbf{a}|$$

and the angle between $\mathbf{a}$ and $\mathbf{b}$ is θ .

Exp

Return the **exponential** of input, e^{input} . This is the inverse of **Ln**, see also **Pow**.

Fraction

Returns the **fractional part** of input. For example, an input of 123.456 would return 0.456.

Is Finite

Return false if `input` is either `infinity` or `NaN`, and true otherwise.

Length

Return the length of the `input` vector, with three possible distance definitions:

Euclidian

The "ordinary" length of the vector: $\sqrt{x^2+y^2+z^2}$

Quadrance

Euclidian distance squared, which is cheaper to compute: $x^2+y^2+z^2$

Manhattan

Measures distance following only axis-aligned directions, which is even cheaper to compute: $|x|+|y|+|z|$

Log

Return the `logarithm` of `input` to base. The argument must be greater than zero. This is the inverse of `Pow`.

Max

Return the per-component maximum of `input1` and `input2`.

Min

Return the per-component minimum of `input1` and `input2`.

Modulo

Return `input` modulo `divisor`. This is the remainder of the division of `input` by `divisor`.

Multiply

Return `input1 × input2`.

Negate

Return `input`.

Normalize

Return a normalized input vector, ie. a `unit vector` pointing in the same direction.

Pow

Return `baseexponent`. This is the inverse of `Log`, see also `Exp`.

Random

The Random shader outputs a random color from various types of inputs. It is useful to do variations of colors or shader properties for example.

Reciprocal

Return the `multiplicative inverse` of `input`, ie. $1/\text{input}$ or `input-1`.

Sign

- Return -1 if `input < 0`
- Return 0 if `input == 0`
- Return 1 if `input > 0`



Sqrt

Return the **square root** of **input**, ie. •

$\sqrt{\text{input}}$

Subtract

Return $\text{input}_1 - \text{input}_2$.

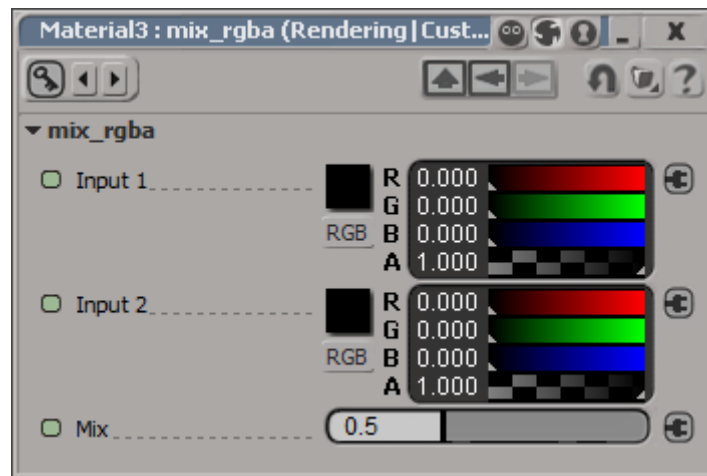
Trigo

Perform various trigonometric functions on **input**. The **frequency** and **phase** parameters make the most sense for the sine, cosine and tangent functions, but are available on all functions for orthogonality. The **units** parameter lets you choose between **radians** and **degrees** for the argument of sine, cosine, and tangent and for the result of the inverse functions. It has no effect on the hyperbolic functions.

Function	Formula	Units Affects	Output Range
Cosine	$\cos(\text{input} \times \text{frequency} + \text{phase})$	• argument	[-1, 1]
Sine	$\sin(\text{input} \times \text{frequency} + \text{phase})$		[-1, 1]
Tangent	$\tan(\text{input} \times \text{frequency} + \text{phase})$		[-,]
Arccosine	$\arccos(\text{input} \times \text{frequency} + \text{phase})$	• result	[0,] or [0°, 180°]
Arcsine	$\arcsin(\text{input} \times \text{frequency} + \text{phase})$		[-/2, /2] or [-90°, 90°]
Arctangent	$\arctan(\text{input} \times \text{frequency} + \text{phase})$		[-/2, /2] or [-90°, 90°]
Hyperbolic Cosine	$\cosh(\text{input} \times \text{frequency} + \text{phase})$	(nothing)	[1,]
Hyperbolic Sine	$\sinh(\text{input} \times \text{frequency} + \text{phase})$		[-,]
Hyperbolic Tangent	$\tanh(\text{input} \times \text{frequency} + \text{phase})$		[-1, 1]

•

Mix RGBA



The `mix_RGBA` is used to blend or add two colors or textures. It returns a [linear interpolation](#) of `input1` and `input2` according to the `mix_weight` attribute. A `mix_weight` value of 0 outputs `input1`, a value of 1 outputs `input2`, and a value of 0.5 mixes evenly between `input1` and `input2`.

Input 1

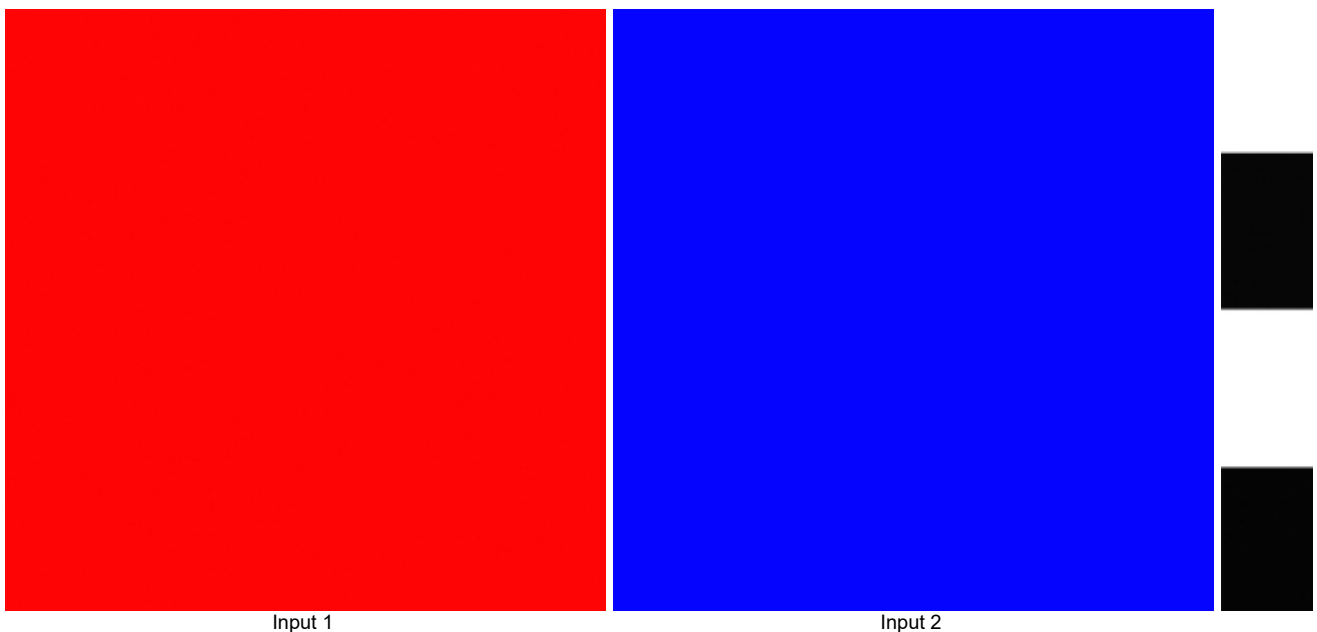
Input for color one.

Input 2

Input for color two.

Mix

Allows you to control the amount of blending between colors. Connect a texture to the `mix_weight` to control the blending between `input1` and `input2`.

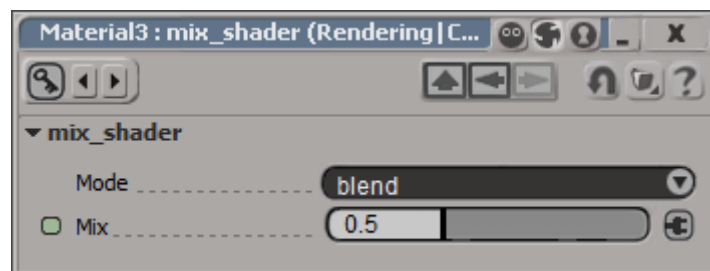


Mix Shader



Rollover images to view mix_weight image

The `mix_shader` shader is used to blend or add two shaders (including light AOVs). It returns a [linear interpolation](#) of `shader1` and `shader2` according to the `mix_weight` attribute. A `mix_weight` value of 0 outputs `shader1`, a value of 1 outputs `shader2`, and a value of 0.5 mixes evenly between `shader1` and `shader2`.



Shader 1

Input for shader one.



Shader 2

Input for shader two.

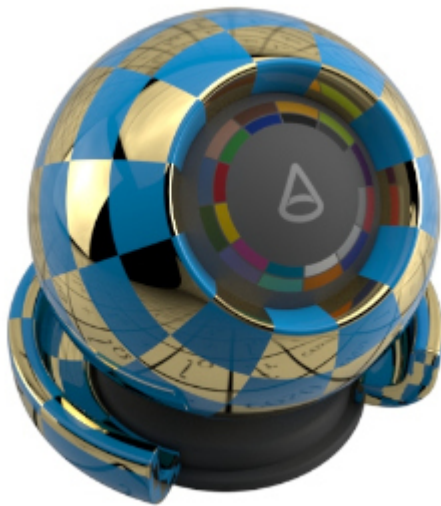


•

Mode

The mode by which the shaders are layered. Choose between blend or add.

The add mode can break energy conservation if not used carefully. It is intended for adding two volume shaders, or two shaders that only emit light while adding two surface shaders like `standard_surface` is not physically correct. •



Blend (default)



Add

Mix

Controls the amount of blending between shaders.

i You may need to insert a Range shader when connecting a bitmap to Mix.



0



0.5 (default)

You can connect a texture to the mix_weight to control the blending between shader1 and shader2.

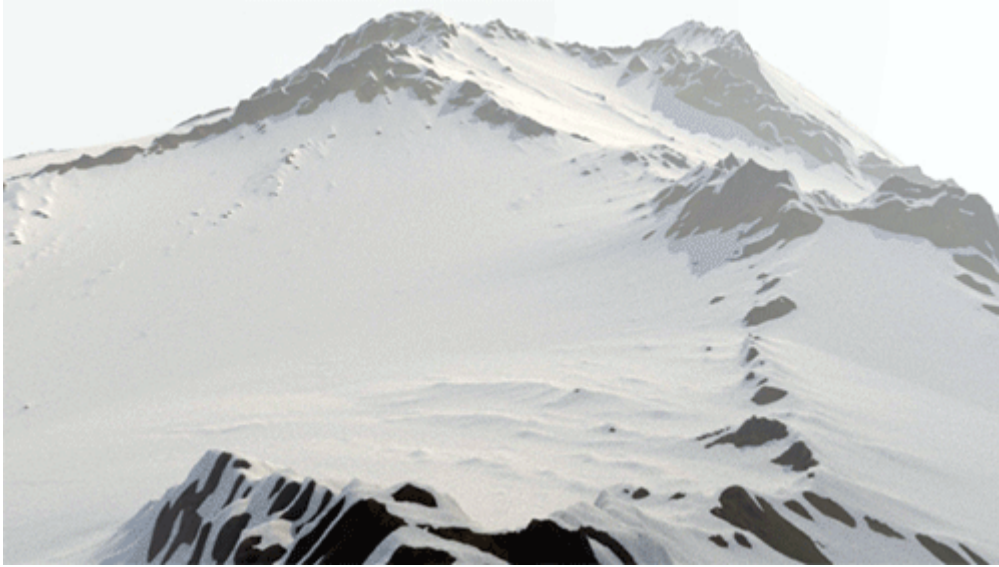


Shader 1



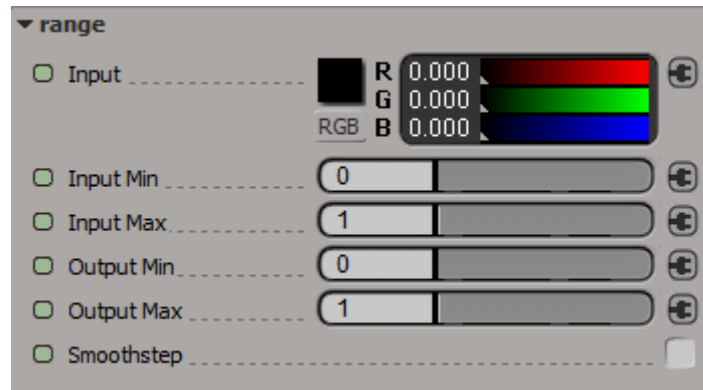
Shader 2

[Workflow Example](#)



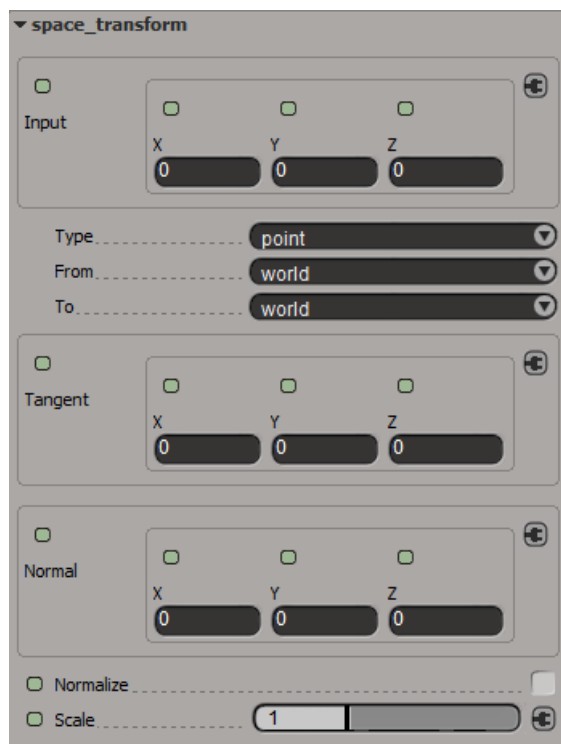
Animated Facing Ratio shader -> Mix Weight. Snow shader -> shader1, Mountain shader ->shader2.

Range



Remap input from the [input min, input max] range to the [ouput min, output max] range linearly. The result is not clamped unless **smoothstep** is on, and the result is interpolated smoothly and the result is **clamped** in the [output min, output max] range.

Space Transform



Convert the **Input** coordinates from one space to another:

- World : Absolute world coordinates.
- Object : Local coordinates.
- Camera: Defined such that +x is right, +y is up and looks down -z.
- Screen: Defined such that +x is right, +y is up and looks down +z.
- Tangent : Defined by a tangent reference frame that is built based on the input **Tangent** vector and the shading normal. If **Tangent** is 0, the shader attempts the following actions to build the frame:
 1. Look for vector user data named "tangent" and "bitangent".
 2. Use the UV derivatives.
 3. Build its own local frame.

Input is interpreted according to **Type** when transformed:

- Point : input is a position in space.
- Vector : input is a direction, the translation part of the **transformation** is left out.
- Normal : input is a normal, transformed by the inverse transpose of the transform to preserve its angle to the surface.

The input channels can be shuffled (**Order**) and inverted (Invert X, Invert Y, Invert Z). Also, they can be remapped to the $[-1, 1]$ range, for instance when the input is an 8-bit map to be used as signed normal or vector displacement map.

The output vector is scaled by **Scale** and can be set as the shading normal (Set Normal).

Shading State

The State nodes allow access to ray and geometric properties such as the surface normal, the UV surface parameters, ray depth etc. The nodes are separated into float, int, matrix, rgb, and vector data types.

•

NAME	VARIABLE	DESCRIPTION
State Float		
Screen X / Y	sx / sy	x/y image-space coordinate of this ray, in [-1,1)
Shutter Time	time	absolute time, between shutter-open and shutter-close
Sample Weight	we	sample weight
Ray Length	Rl	ray length (Ro-P)•
Barycentric U / V	bu / bv	barycentric coordinate (aka alpha, or u/v)
Surface U / V	u / v	u/v surface parameter
Light Distance	Ldist	distance from P to light source•
Shaded or Avg. Displaced Area	area	in•AI_CONTEXT_SURFACE•this is the world-space area being shaded; in•AI_CONTEXT_DISPLACEMENT•it's the average area of all un-displaced triangles shared by the current vertex
U / V Derivative wrt Screen X / Y	dudx / dudy / dvdx / dvdy	u/v derivative wrt screen u/v-axis (not normalized)•
Shutter start / end	•	start and end point of the motion range
State Int		
Raster X / Y	x / y	raster-space coordinate of this ray tree
Subpixel Sample Index	si	subpixel sample index
Path or Light Sample Index	i	path index (or light sample index for shadow rays)
Number of Samples	n	total number of samples to take
Ray Type	Rt	ray type
Transparency Index	transp_index	transparency index
Thread ID	tid	thread ID
Ray Depth	Rr	recursion level for the ray that created this hit
Reflection Depth	Rr_refl	ray reflection depth level•
Refraction Depth	Rr_refr	ray refraction depth level•
Diffuse Depth	Rr_diff	ray diffuse depth level
Glossy Depth	Rr_gloss	ray glossy depth level
Force Hemispherical Lighting •	fhemis	force hemispherical lighting•
Primitive ID	fi	primitive ID (triangle, curve segment, etc)•
Number of Active Lights	nlights	number of active lights
Inclusive Traceset	inclusive_traceset	is the trace-set inclusive?
Skip Shadow Rays	skip_shadow	if true, don't trace shadow rays
Shading Context	sc	type of shading context
State Matrix		
Object To World Matrix	M	object to world space matrix transform•
World To Object Matrix	Minv	world to object space matrix transform•
Camera To World	•	camera to world space matrix transform

Matrix		
World To Camera Matrix	•	world to camera space matrix transform
Screen To World Matrix	•	screen to world space matrix transform
World To Screen Matrix	•	world to screen space matrix transform
State Vector		
Ray Origin•	Ro	ray origin (typically the camera position)
Ray Direction	Rd	ray direction
Shading Point in Object-Space	Po	shading point in object-space
Shading Point in World-Space	P	shading point in world-space•
Surface Derivative wrt Screen X / Y•	dPdx / dPdy	surface derivative wrt screen x/y-axis (not normalized)
Shading Normal•	N	shading normal
Face-Forward Shading Normal•	Nf	face-forward shading normal
Geometric Normal•	Ng	geometric normal
Face-Forward Geometric Normal•	Ngf	face-forward geometric normal
Smoothed Normal no bump	Ns	smoothed normal (same as N but without bump)
Light Direction•	Ld	incident direction from light source
Surface Derivative wrt U / V•	dPdu / dPdv	surface derivative wrt U/V parameter (not normalized)
Ray Direction Derivative wrt Screen X / Y•	dDdx / dDdy	ray direction derivative wrt screen x/y-axis (not normalized)
Surface Normal Derivative wrt Screen X / Y•	dNdx / dNdy	derivative of surface normal with respect to x/y-axis

Surface

Below is a list of surface shaders available in SltoA:

•

- [Ambient Occlusion](#)
- [Car Paint](#)
- [Complex IOR](#)
- [Flakes](#)
- [Lambert](#)
- [Matte Shader](#)
- [Shadow Matte](#)
- [Standard Hair](#)
- [Standard Surface](#)
- [Thin Film Shader](#)
- [Two Sided](#)

Ambient Occlusion

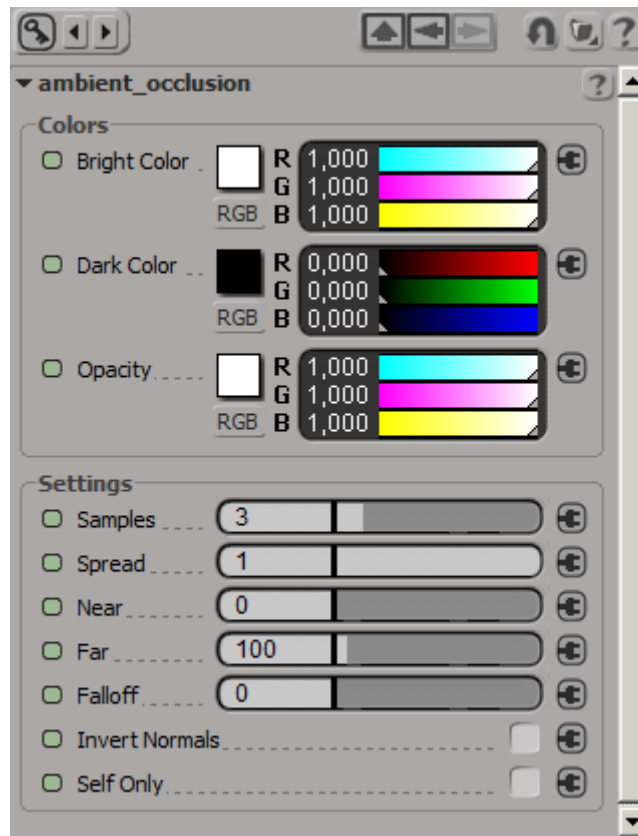


Scene rendered with ambient occlusion

•

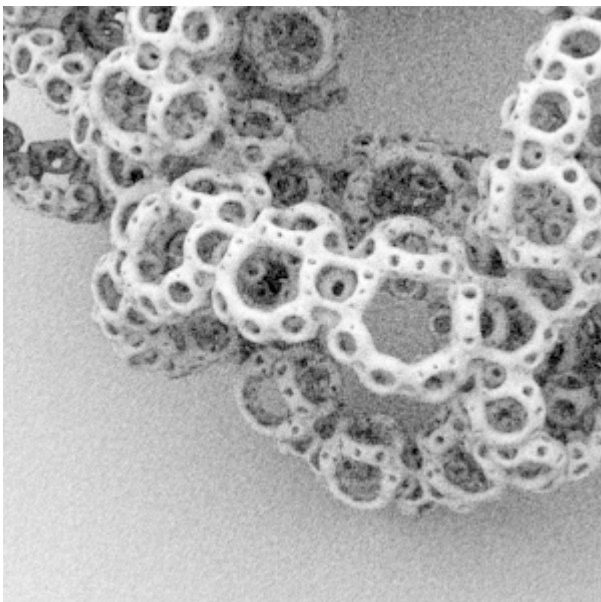
Ambient occlusion is an approximation of global illumination that emulates the complex interactions between the diffuse inter-reflections of objects. While not physically accurate (for that use full global illumination), this shader is fast and produces a realistic effect.

The ambient occlusion shader essentially fires a number of rays in the upper hemisphere defined by the tangent plane of the shading point and returns the ratio of the hits divided by the total rays as a color. When the ratio of the ray hits/total rays = 0, the output is considered fully unoccluded or 'Bright' (with white color by default). When the ratio of the ray hits / total rays = 1, the output is considered fully occluded or 'Dark' (with black color by default). In between, there is a linear interpolation of the two values.

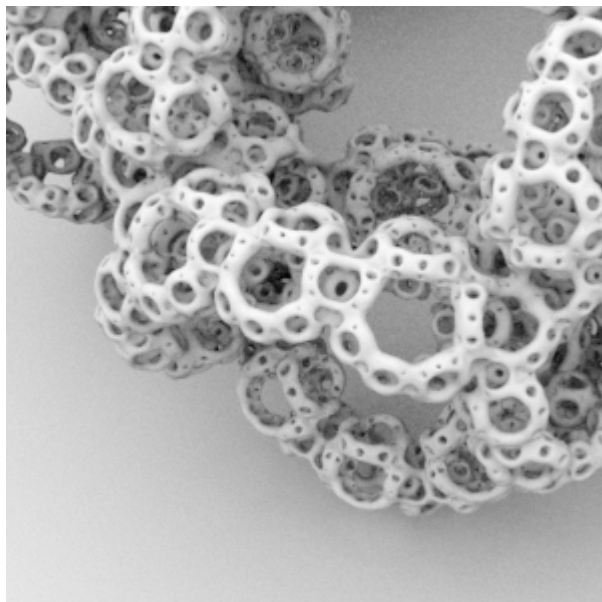


Samples

Controls the number of rays that will be fired to compute the ray hits/total ratio. Increasing the number of samples will reduce noise and give a better quality result.



1

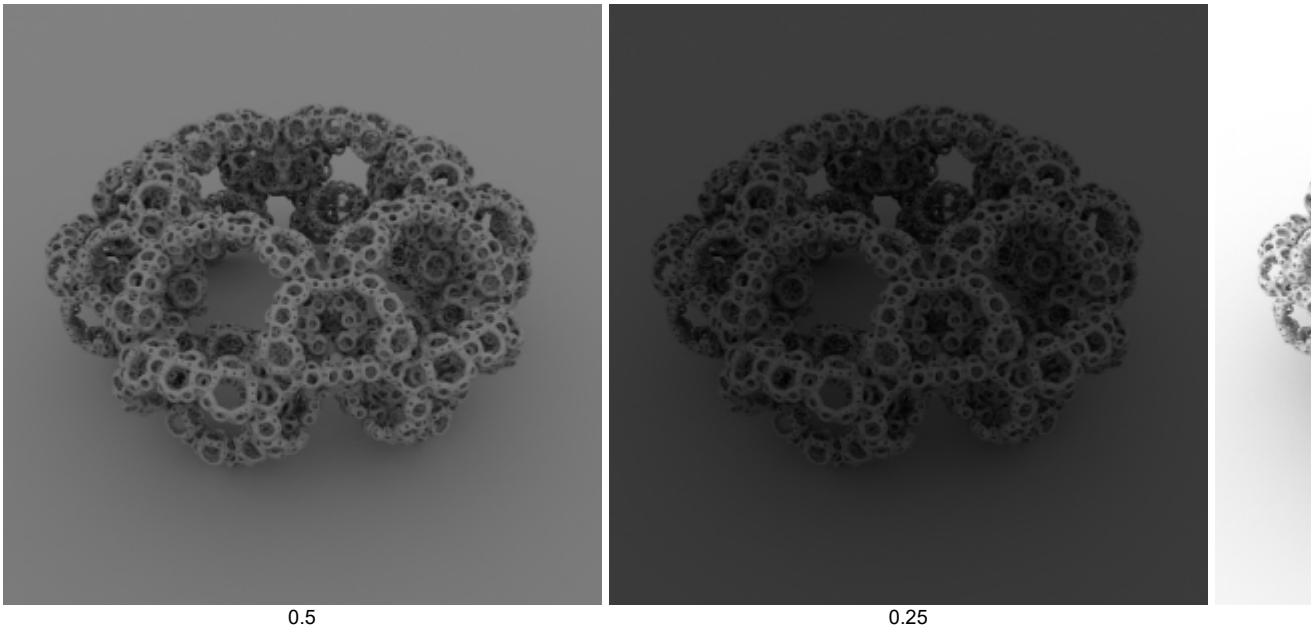


3 (default)

Bright

When the ratio of the ray hits/total rays is zero usually the output is considered fully unoccluded or Bright.

The output color when the ratio of the ray hits/total rays is zero (fully unoccluded).



It is possible to change the white attribute to a color other than white to achieve a specific look:



Dark

The output color when the ratio of the ray hits/total rays is one (fully occluded). It is possible to change the black attribute to colors other than black for specific looks. This can be useful for tinting the color of a floor shadow for example:



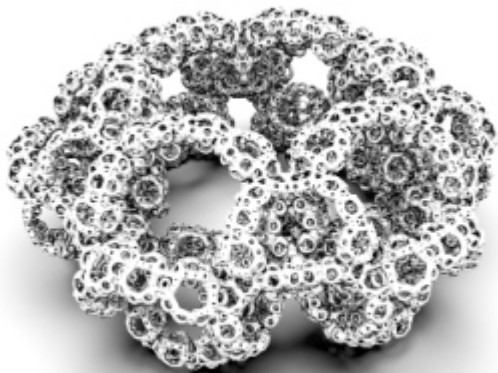
Black color



Black color tinted with a color

Spread

Angular spread around the normal vector N , in the range $[0,1]$, where 1 maps to 90 degrees (the full hemisphere). 1.0 is the most common value. Other values may produce results that are not useful.



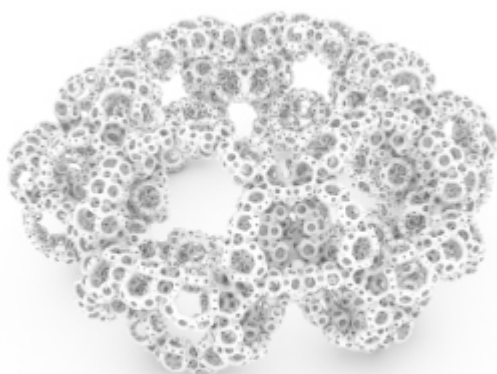
0.25



0.5

Falloff

An exponential falloff rate for the occlusion along the ray distance. It is not compatible with semi-opaque surfaces hit by the occlusion rays.



1



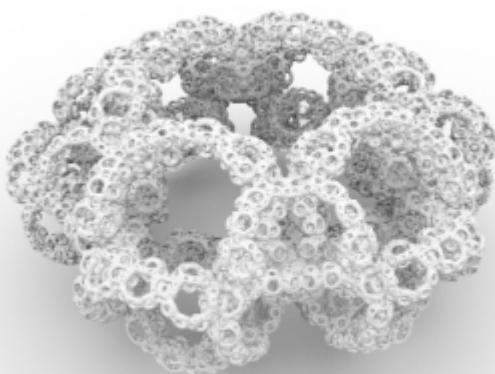
2

Near Clip

The minimum occlusion distance that is sampled.



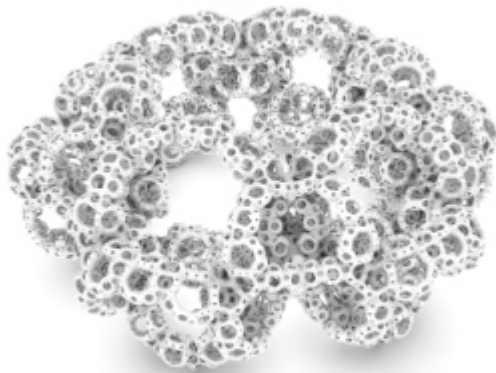
0 (default)



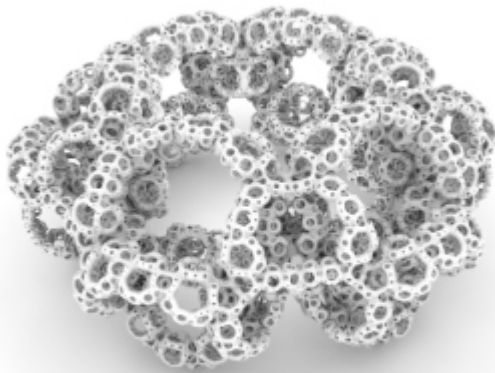
1

Far Clip

The maximum occlusion distance that is sampled.



1



2.5

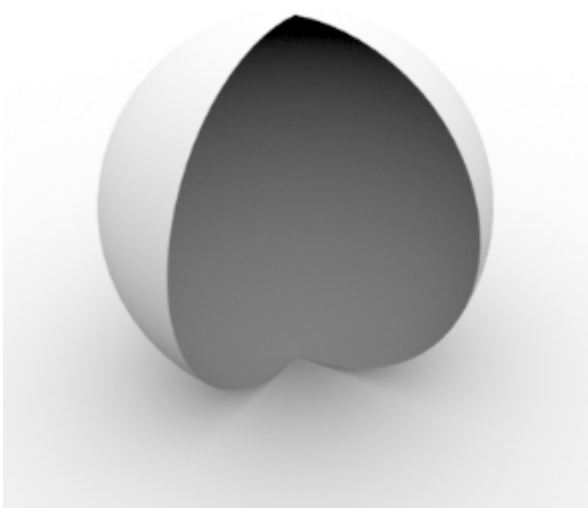
Opacity

MultiExcerpt named 'opacity ' was not found

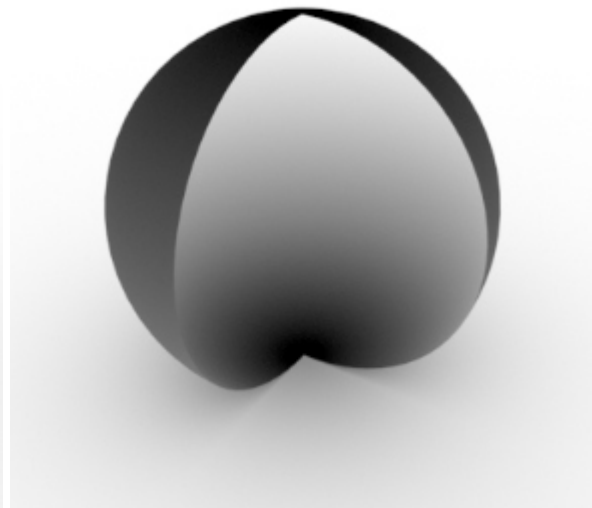
The page: ambient_occlusion was found, but the multiexcerpt named 'opacity ' was not found. Please check/update the page name used in the 'multiexcerpt-include macro.

Invert Normals

This attribute changes the direction of the rays being traced. When it is off the rays are traced outside of the surface. When it is on the rays are traced inside the surface.

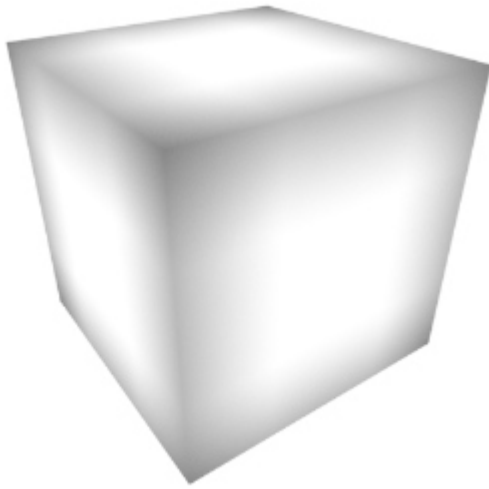


Disabled (default)

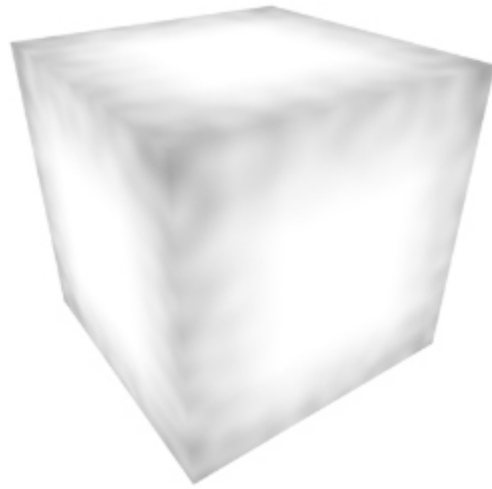


Enabled

The most common use for this is to simulate dirt or erosion, as corners and cracks will become darker. •

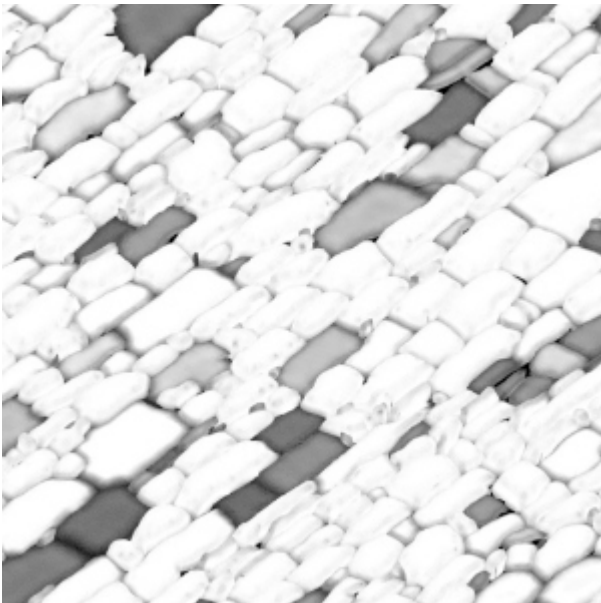


'Invert Normals' enabled (Far Clip 0.3).



Noise texture connected to 'Black' attribute

The example below shows a brick road with an ambient_occlusion pass (with invert_normals enabled) multiplied on top of the beauty render to add 'dirt' in-between the bricks.



'Invert Normals' enabled on AO



Beauty render

Self Only

Only gathers occlusion against the same object.



Disabled (default)



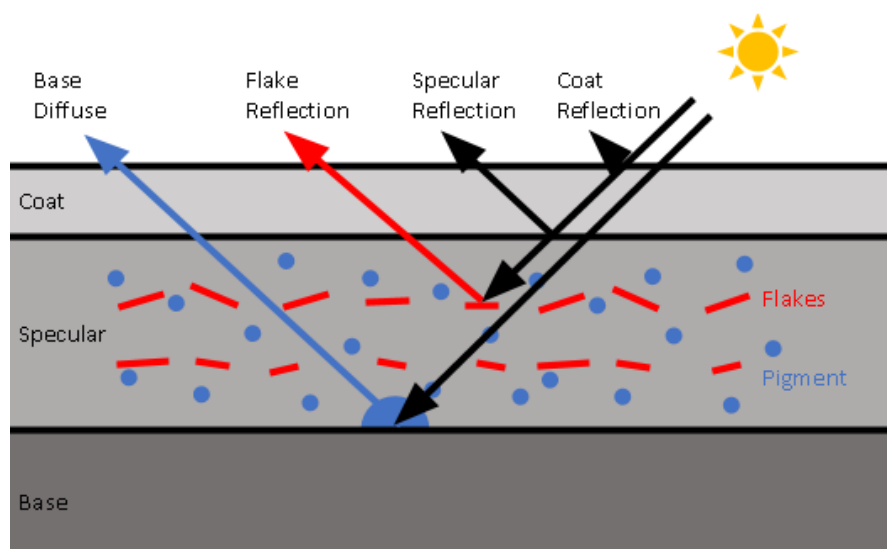
Enabled

-
-

Car Paint



This simple to use shader is useful for creating car paint materials. It supports three layers: `base_layer`, `specular` and `clear_coat`. These layers work similarly to those in the `standard_surface` shader. However, there are some unique attributes for specifically controlling color ramps, light facing color, and falloff that are unique to the `car_paint` shader. Metallic flakes can also easily be added to the specular layer.



When flakes become smaller than a pixel, they can introduce rendering artifacts during an animation. Increasing Camera (AA) samples can help to reduce flakes from 'shimmering' over distance.



.

Base

Weight

The primer layer color weight.



0



0.5

Color

The color of the primer layer.



Red



Green

Roughness

The primer layer follows an Oren-Nayar reflection model with surface roughness. A value of 0.0 is comparable to a Lambert reflection.



0



0.5 (default)

Specular

Weight

The base coat color weight.



0



0.5

Color

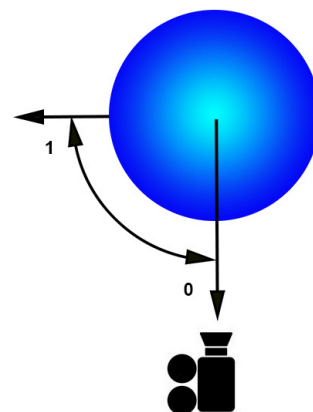
The color the specular reflection will be modulated with. Use this color to 'tint' the specular highlight from the base coat layer.



Specular Color: White (default). Rollover image to view orange.

Flip Flop

Connect a ramp shader here to modulate the specular reflection from the base coat depending on the viewing angle. This can be used to mimic a pearlescent effect.



Rollover image to view without a ramp (solid color). The color changes depending on the viewing angle.

Light Facing Color

Modulates the base coat specular color of the area facing the light source.



White (Specular Color: Blue)



Yellow (Specular Color: Blue). Car paint appears more green in light facing areas.

Pearlescent car paint effects can be created by combining light_facing_color with base_color.



Cyan (Base Color: magenta)

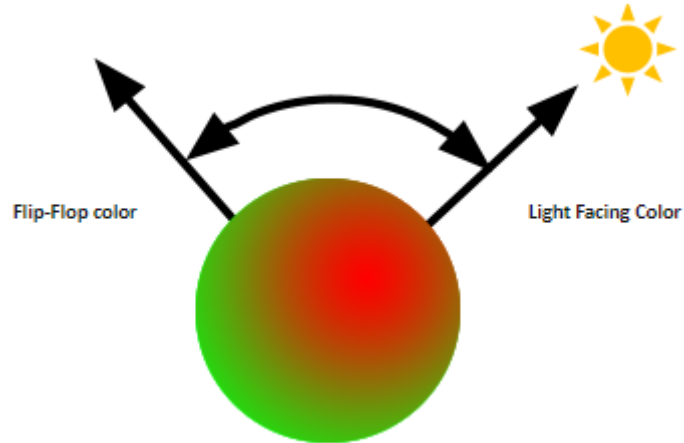


Blue (Base Color: red)

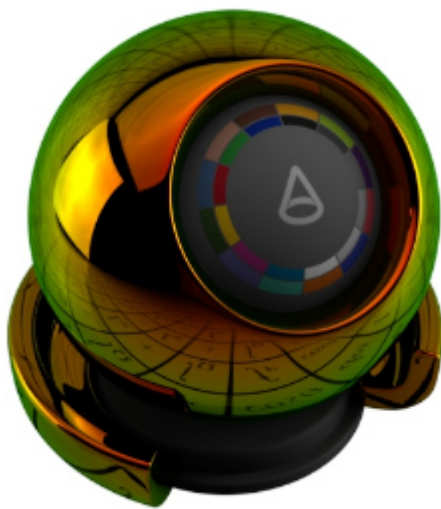
Falloff

The falloff rate of the light facing color of the base specular coat. The higher the value, the narrower the region. It also determines how the ramp color and light facing color are blended.

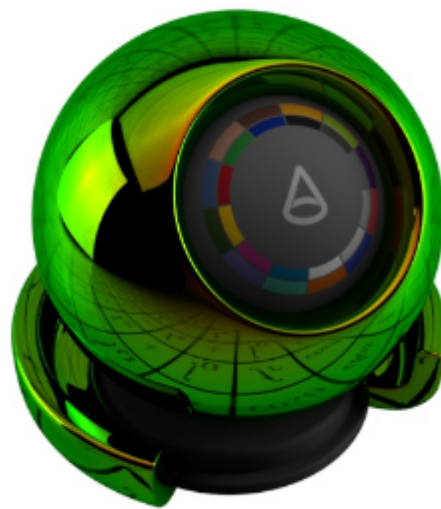
In the example below, the Flip-Flop color has been set to green and the Light Facing Color to red.



Setting the Falloff to 1 disables the Light Facing Color (red):



0



0.5



Falloff: 0 to 1. Flip-Flop color: Magenta, Light Facing Color : Red

Roughness

Controls the glossiness of the specular reflections from the base coat layer. The lower the value, the sharper the reflection. In the limit, a value of 0 will give you a perfectly sharp mirror reflection, while 1.0 will create reflections that are close to a diffuse reflection.*



0



0.25

IOR

Determines the index of refraction for the base coat.



1.52 (default)



2

Transmission Color

This simulates light attenuation due to pigments. The lower the value, the denser pigments.



Red:0.1



Red:0.5

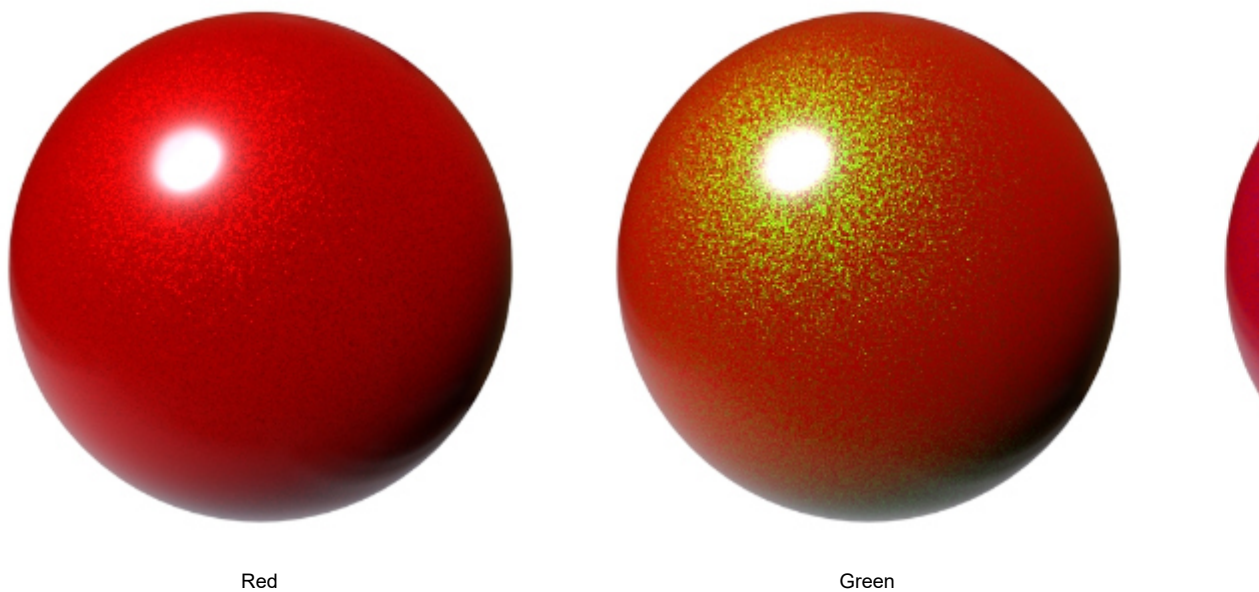
The transmission_color also affects the appearance of the flakes. For example, when the base_color is blue and the transmission_color is yellow it becomes multiplied and renders as green, with the flakes tinted yellow.



Flakes

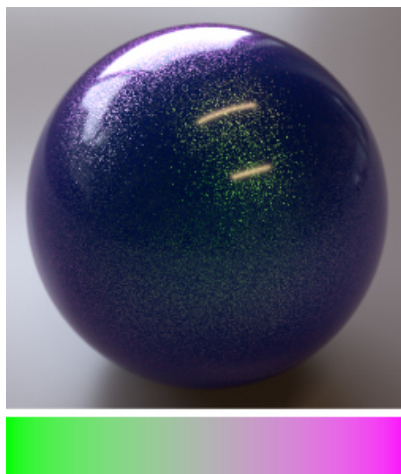
Flake Color

The color the specular reflection will be modulated with. Use this color to 'tint' the specular highlight from flakes.



Flip Flop

Connect a ramp shader here to modulate the specular reflection from flakes depending on the viewing angle.



Ramp connected to Flake Flip Flop

Light Facing Color

Modulate the specular reflection color from flakes of the area facing the light source.



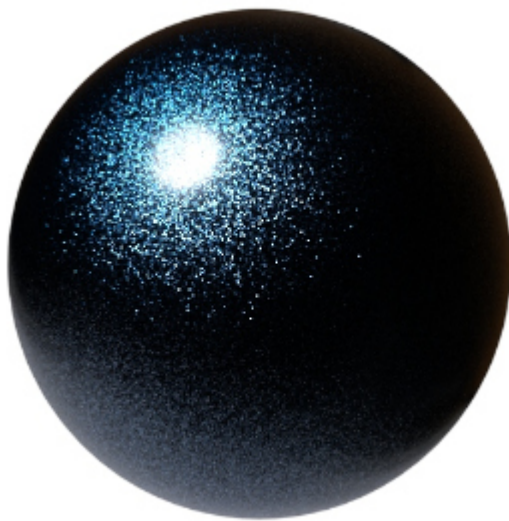
Red



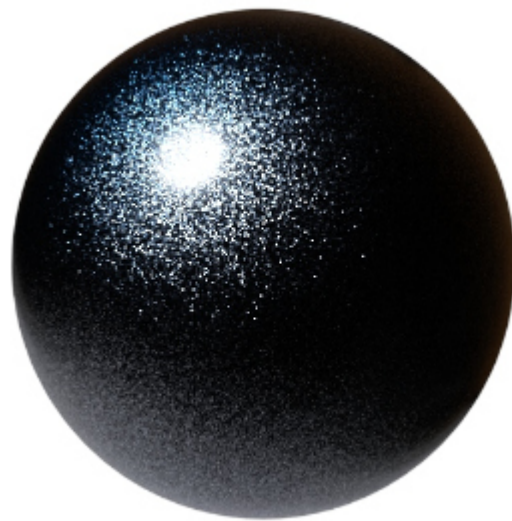
Green

Falloff

The falloff rate of the light facing color of flakes. The higher the value, the narrower the region.



0



0.5

(Light Facing Color: blue)

Roughness

Controls the glossiness of the specular reflections from flakes. The lower the value, the sharper the reflection. In the limit, a value of 0 will give you a perfectly sharp mirror reflection, while 1.0 will create reflections that are close to a diffuse reflection. •



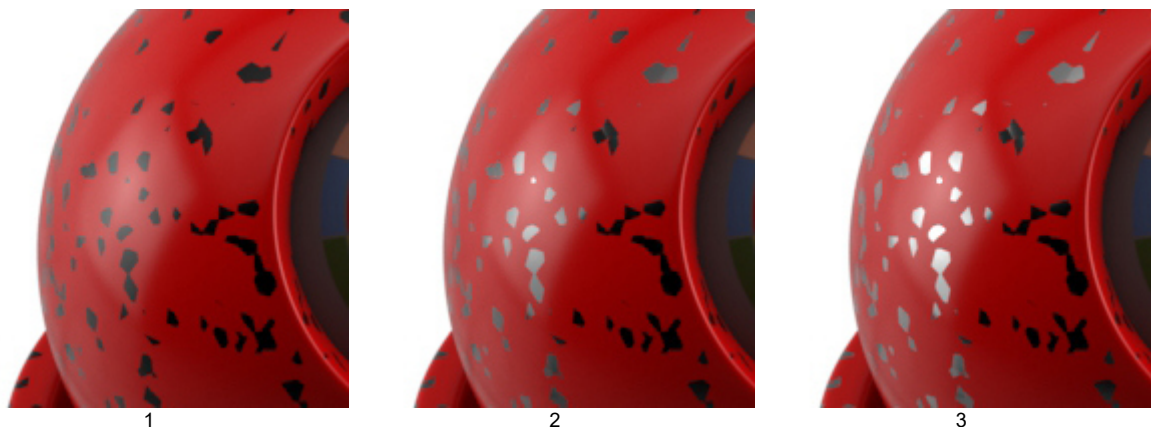
0.2



0.4 (default)

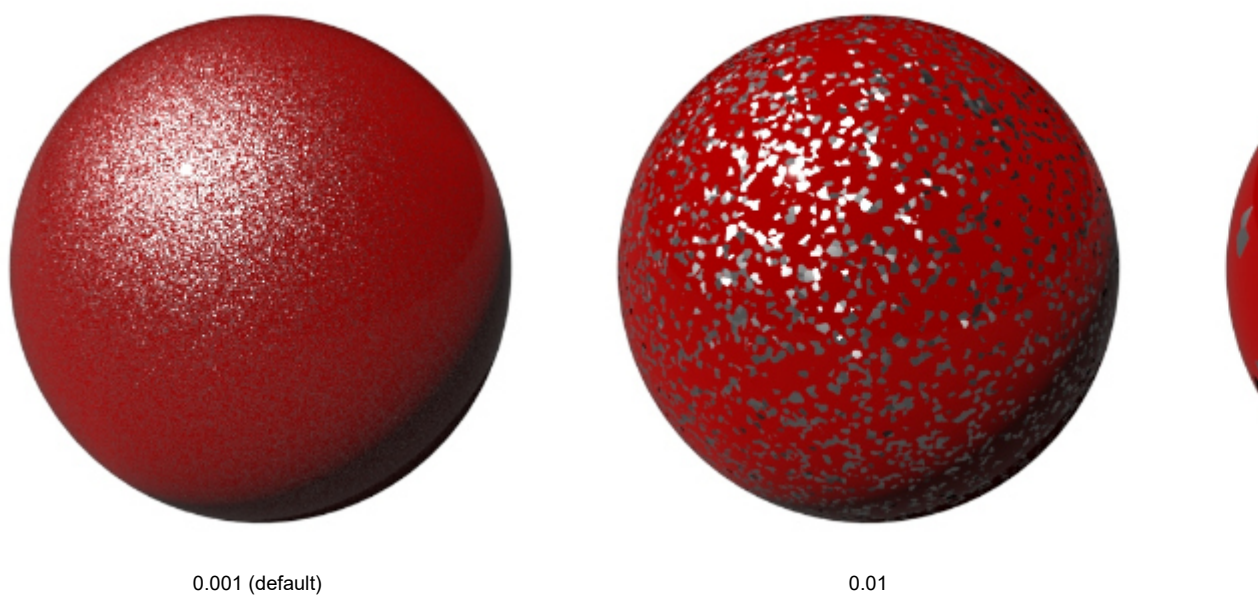
IOR

Determines the index of refraction of flakes.



Scale

Scales the flake structure up or down. Smaller values zoom out of the map, giving a larger number of flakes.



Density

Controls the density of flakes. There will be no flakes if it is 0. The surface is fully covered with flakes at 1.



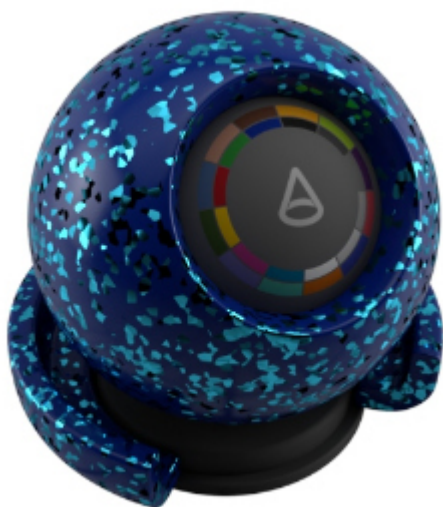
0 (default)



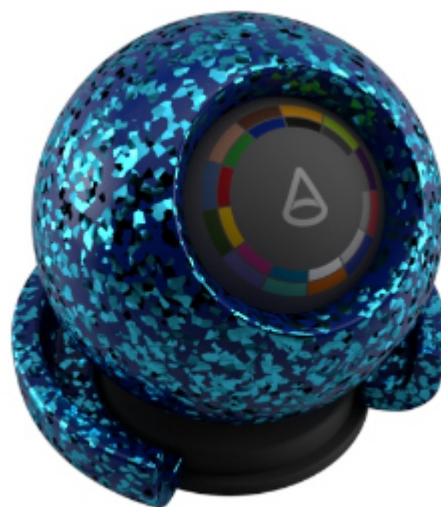
0.1

Layer

Specify the number of flake layers. The flakes at a deep layer are covered by the ones closest to the surface.



1 (default)



2

Normal Randomize

Randomize the orientation of flakes.



0.2

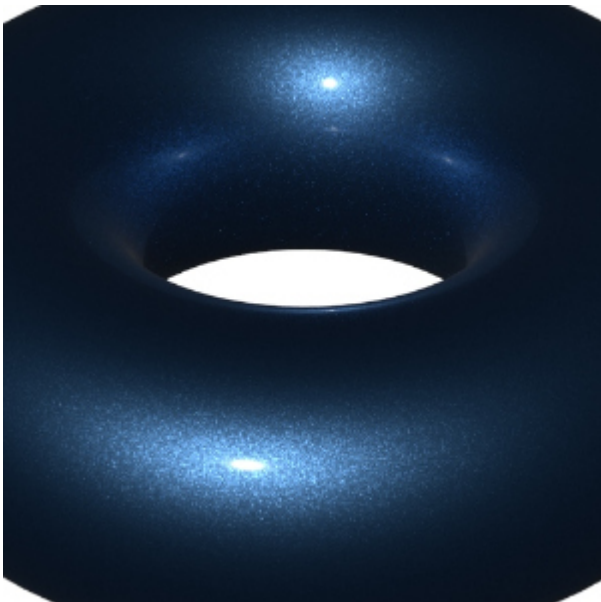


0.4

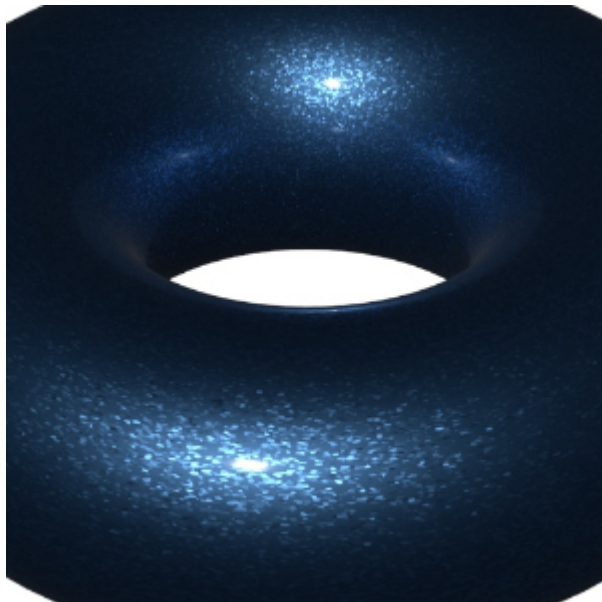
Coord Space

Specifies the coordinate space used for calculating the shapes of flakes.

- World: points are relative to the global origin of the scene.
- Object: points are expressed relative to the local origin (center) of the object.
- Pref (default): short for 'vertex in reference pose'. The plugin can pass these vertices to Arnold (in addition to the regular, deformed vertices) which can, in turn, be queried by the shader so that the texture 'sticks' to the reference pose and does not swim as the mesh deforms. (Pref does not work with NURBS surfaces.)
- UV: the texture coordinates.



Pref (default)



UV

Coat

Weight

This attribute is used to coat the material. It acts as a clear-coat layer on top of the base coat and primer layers. The coating is always reflective (with the given roughness) and is assumed to be dielectric.



Coat weight: 1 (rollover image to view 0). Specular Roughness: 0.5.

Color

This is the color of the coating layer's transparency.



Coat Color: White (default). Rollover image to view red.

Roughness

Controls the glossiness of the specular reflections. The lower the value, the sharper the reflection. In the limit, a value of 0 will give you a perfectly sharp mirror reflection, while 1.0 will create reflections that are close to a diffuse reflection. You should connect a map here to get variation in the coat highlight.



0



0.25

IOR

The IOR parameter (Index of Refraction) defines the material's [Fresnel reflectivity](#) and is by default the angular function used. Effectively the IOR will define the balance between reflections on surfaces facing the viewer and on surface edges.



1



2

Normal

The **Coat Normal** affects the Fresnel blending of the coat over the base, so depending on the normal, the base will be more or less visible from particular angles. Uses for **Coat Normal** could be a bumpy coat layer over a smoother base. This could include a rain effect, a carbon fiber shader or a car paint shader where you could use different normals (using e.g. flakes) for the coat layer and base layers.



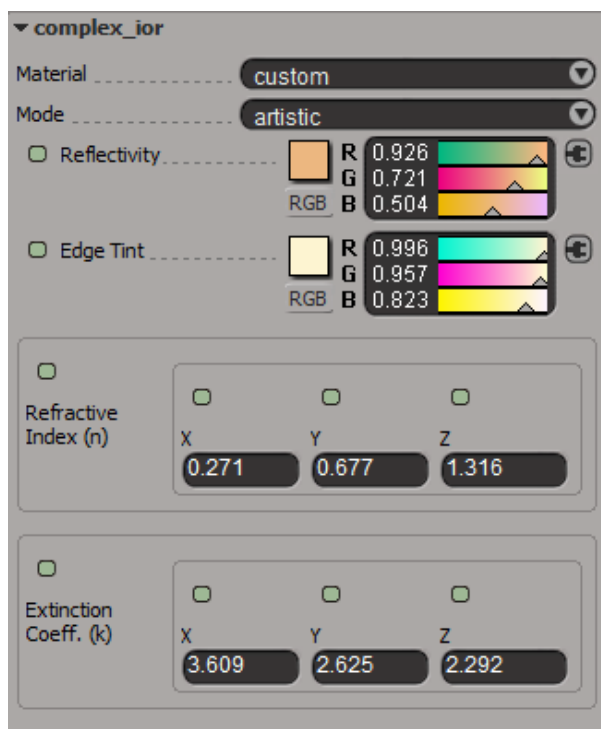
Complex IOR

 This shader is deprecated and should only be used with the Standard shader.



This shader can be used to render materials with complex index of refractions. The **Standard** shader can compute the Fresnel effect for dielectric materials like plastic and glass based on the refractive index of the material; however, metals have a more complicated Fresnel reflective curve that also depends on another parameter called extinction coefficient.

 You should not use the Complex IOR shader with the Standard Surface shader. The Standard Surface has the complex Fresnel built-in when the metalness is > 0 , using the base and specular color.



Material

Presets of different materials. The custom mode allows you to define a custom setting. By clicking on the Edit button, the selected preset overrides the Reflectivity, Edgetint and n, k fields, allowing you to refine the settings.



Sodium



Platinum



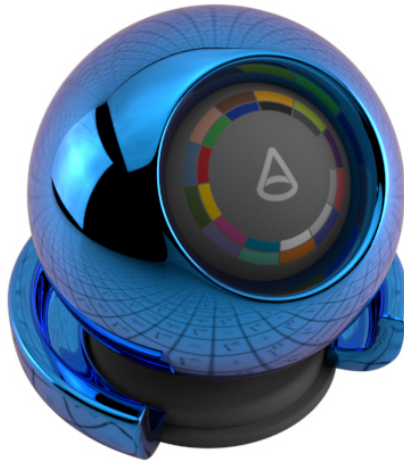
Iron



Gold

Mode

Select between the artist friendly (Reflectivity and Edgetint colors) and physical (n and k) controls.



Reflectivity: blue. Edgetint: purple.

Reflectivity

The reflectance at normal incidence.

Edgetint

Controls the color bias as the viewing direction becomes parallel to the surface.

n

The refractive index value for the red/green/blue wavelengths (e.g. 0.65, 0.55 and 0.45 micrometers).

k

The extinction coefficient value for the red/green/blue wavelengths (e.g. 0.65, 0.55 and 0.45 micrometers).



The site <http://refractiveindex.info/> has measured n and k values for many materials like copper, gold etc.

Flakes



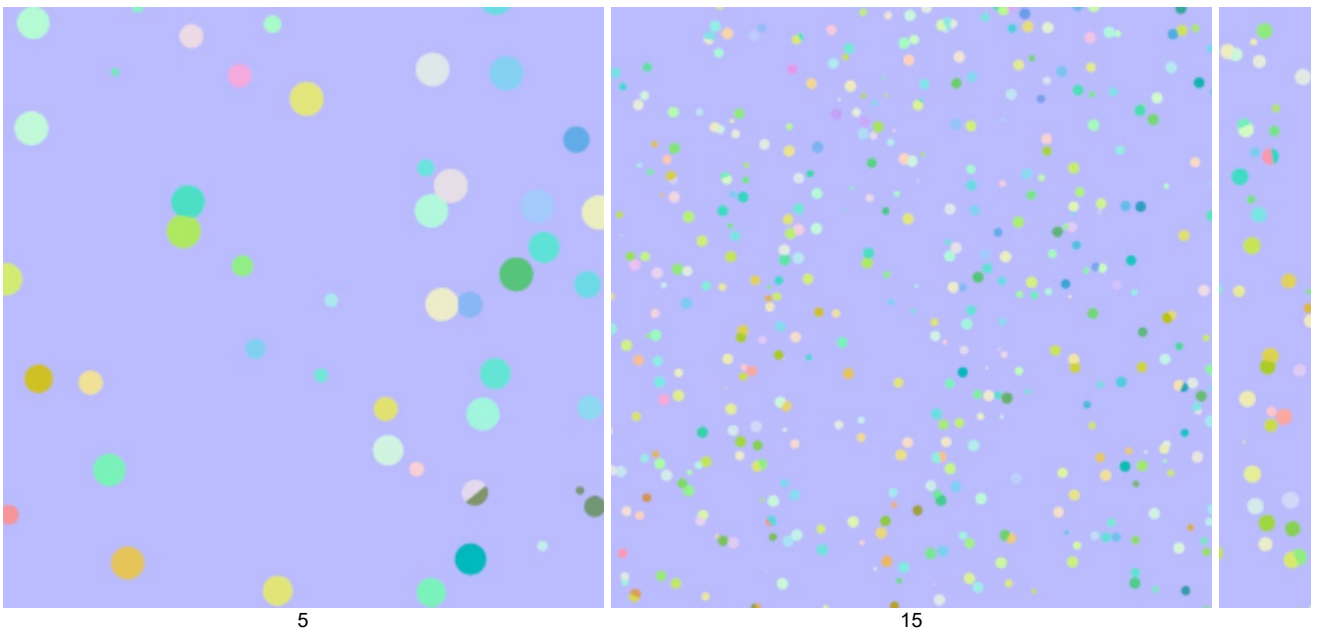
• This shader creates a procedural flake normal map that can be used for materials such as car paint. Note that the map does not perform any filtering, so small flakes require more AA samples to become clean.

When connected to a `normal_map` shader use `tangent` as the `output space` on flakes, `tangent` as the input space (Mode) on the `normal_map` shader and disable `Interpret 8/16 bit normal map as linear`.

•

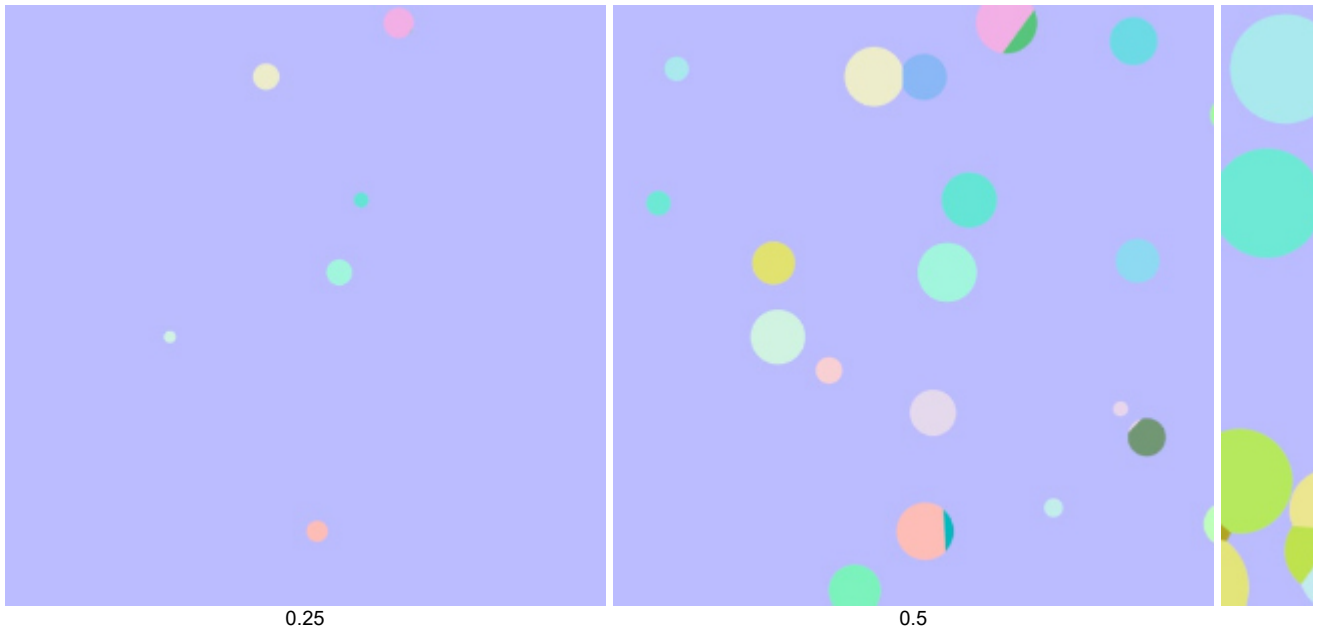
Scale

Scales the flake structure up or down. Larger values zoom out of the map, giving a larger number of flakes.



Size

The relative size of the flakes.



Normal Randomize

Blends between the smooth surface normal (0) and the random flake normal (1).



0.5



0.1

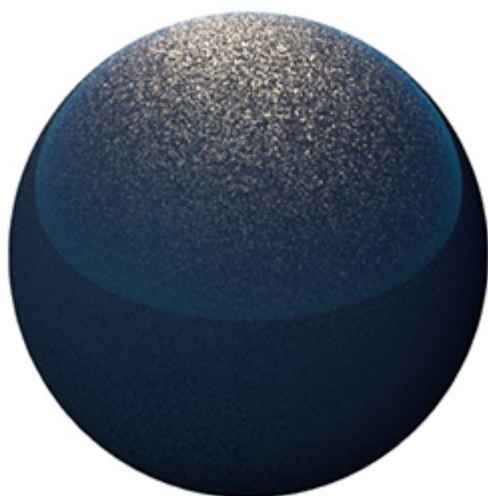
Coord Space

Specifies the coordinate space used for calculating the random normals.

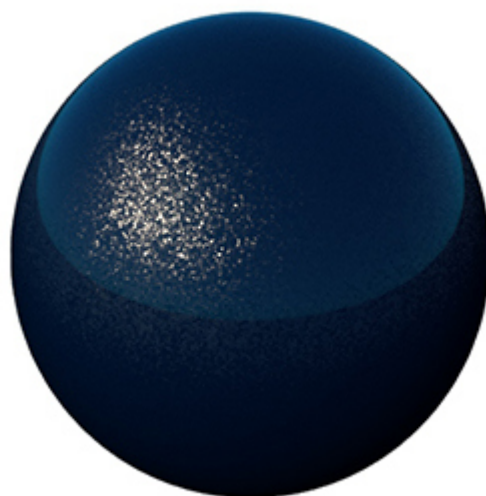
- World: points are relative to the global origin of the scene.
- Object: points are expressed relative to the local origin (center) of the object.
- Pref: short for 'vertex in reference pose'. The plugin can pass these vertices to Arnold (in addition to the regular, deformed vertices) which can, in turn, be queried by the shader so that the texture 'sticks' to the reference pose and does not swim as the mesh deforms. (Pref does not work with NURBS surfaces.)
- UV: the texture coordinates.

Output Space

Specifies the space of the output normal vector.



World



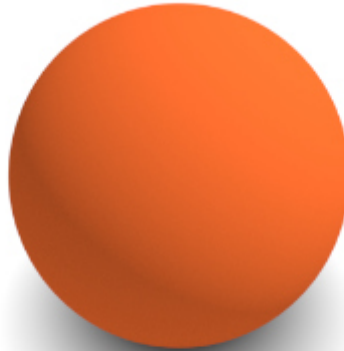
Tangent

Rollover images

-
-

Lambert

•



Simple Lambertian reflectance model. Outputs a simple color (RGB).

••



Color

Diffuse color.

Weight

Diffuse weight.

Opacity

The opacity of the material expressed as a color value. •

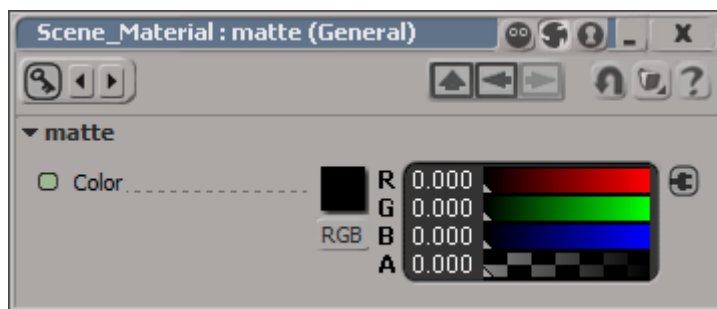
Normal

Connect a normal map here (usually exported from Mudbox or ZBrush).

Normal mapping works by replacing the interpolated surface normal by the one evaluated from an RGB texture, where each channel (Red, Green, Blue) corresponds to the X, Y and Z coordinates of the surface normal. It can be faster than bump mapping since bump mapping requires evaluating the shader underneath at least three times.



Matte Shader



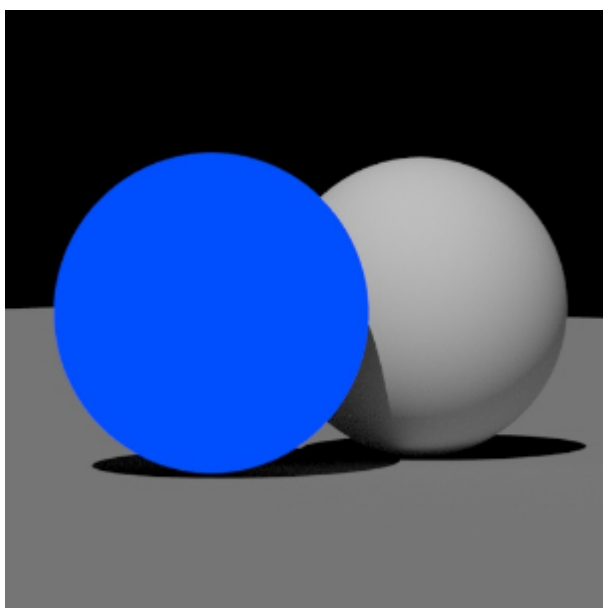
The matte option enables you to create holdout effects by rendering the alpha as zero.

Passthrough

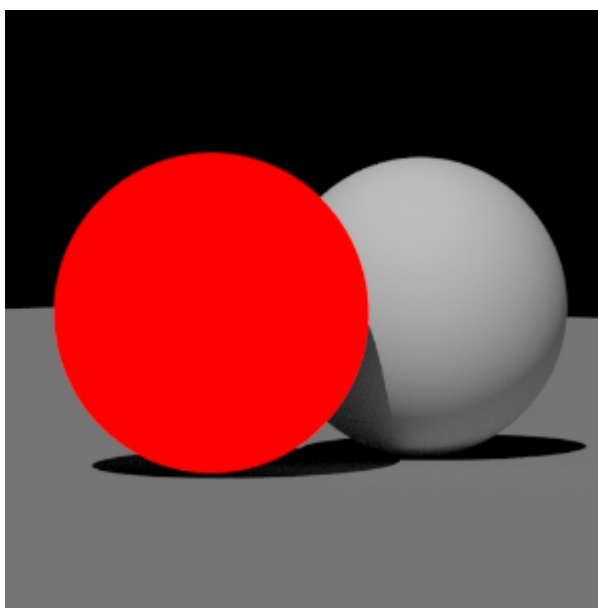
Allows parallel evaluation in a shader network.

Color

Changes the color of the matte.



Matte Color: Blue



Matte Color: Red

•

MultiExcerpt named 'matte_example ' was not found

The page: Matte was found, but the multiexcerpt named 'matte_example ' was not found. Please check/update the page name used in the 'multiexcerpt-include macro.

Shadow Matte

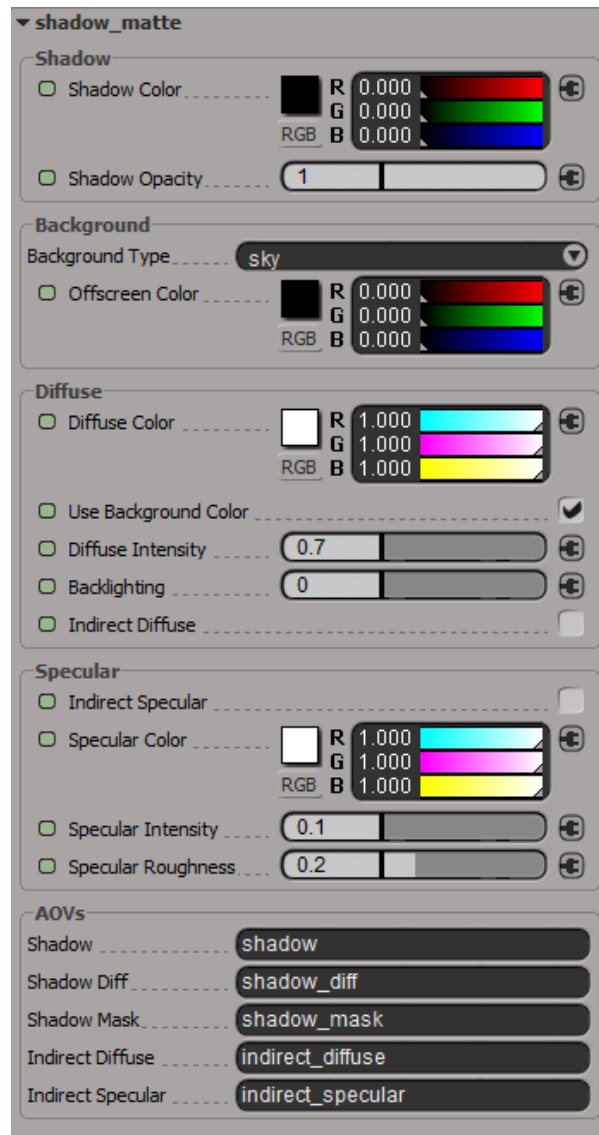


shadow_matte shader used to capture shadow on the floor plane (rollover image)

- The shadow_matte is a specific shader, used typically on floor planes to 'catch' shadows from lighting within the scene. It is useful for integrating a rendered object onto a photographic background. You can create a custom shadow pass with this shader. You can also render out shadows separately for use in a compositing package. For example, you may want to change the shadow density or color of the shadow to match some live-action footage.



The Shadow Matte only collects shadows from light sources and not from the Sky shader (which is not a light).



Shadow

Shadow Color

The color of the shadow. You can use it to 'tint' the color of the shadows to match a photographic backplate.



Black (default)



Red

Shadow Opacity

Determines how 'opaque' or dark the shadow appears. Higher values will produce a lighter shadow. •The captured shadow is visible in the alpha channel. •If you have a very hilly or bumpy ground, you may need to increase the Transparency Depth in the Render Settings.



0



0.5

Background

Background Type

MultiExcerpt named 'background_type ' was not found

The page: shadow_matte was found, but the multiexcerpt named 'background_type ' was not found. Please check/update the page name used in the 'multiexcerpt-include macro.

Offscreen Color

There may be areas visible in the specular reflections which are outside of the background plate; so-called offscreen areas. This parameter defines the color used for these offscreen areas; you can link a texture for instance.



offscreen_color: black (default)



Background image linked to offscreen_color

Diffuse

Diffuse color

The color used to determine the overall indirect diffuse contribution in the scene.



Diffuse Color: Red

Use Background

If enabled, the background color is used to determine the overall indirect diffuse contribution in the scene. Otherwise, the color defined in the diffuse color is used.



Enabled (default). Background color used.



Disabled. Red indirect diffuse color visible on floor.

Diffuse • Intensity

The amount of diffuse contribution.



1



5

Backlighting •

When enabled, it takes into account the backlighting illumination. •Backlighting provides the effect of a translucent object being lit from behind (the shading point is 'lit' by the specified fraction of the light hitting the reverse of the object at that point). It is recommended that this only be used with thin objects (single-sided geometry) as objects with thickness may render incorrectly.



0 (default)



0.5

Indirect Diffuse

Switch for enabling/disabling capture of the indirect diffuse light.



Disabled (default)



Enabled

Specular

Indirect Specular

Switch for enabling/disabling capture of the indirect specular light.



Disabled (default)



Enabled

Specular Color

The color the specular reflection will be modulated with.



White (default)



Red

Specular Intensity

The specular weight. Influences the brightness of the specular highlight.



0



0.5

Specular Roughness

Controls the glossiness of the specular reflections. The lower the value, the sharper the reflection.



0.1 (default)



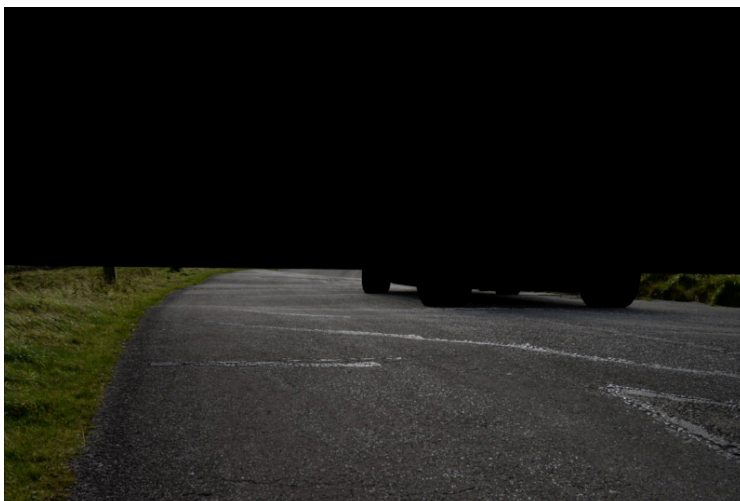
0.25

AOV names

A list of available **AOVs** available for the Shadow Matte shader. Each option creates a separate AOV render pass for that component. Note that the **AOV** must also be enabled in the **Render Settings** window.

Shadow

Direct light shadow AOV.



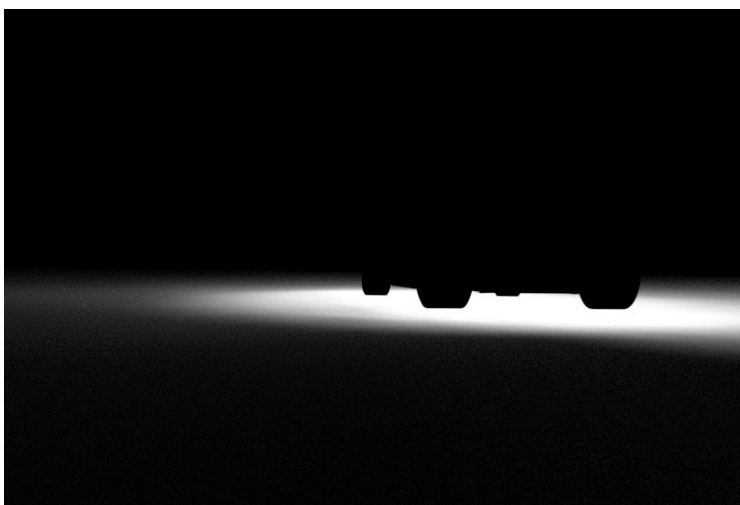
Shadow Diff

A difference AOV which can be used to eliminate the shadow from the direct component.



Shadow Mask

This AOV can be used in comp to localize and tweak the shadow.



Indirect Diffuse•

MultiExcerpt named 'aov_Indirect_Diffuse ' was not found

The page: shadow_matte was found, but the multiexcerpt named 'aov_Indirect_Diffuse ' was not found. Please check/update the page name used in the 'multiexcerpt-include macro.'

Indirect Specular •

MultiExcerpt named 'aov_Indirect_specular ' was not found

The page: shadow_matte was found, but the multiexcerpt named 'aov_Indirect_specular ' was not found. Please check/update the page name used in the 'multiexcerpt-include macro.'

⚠ Skydome Light Sampling

The Shadow Matte shader requires more than 1 sample to avoid noisy shadows. This is because it is computing a ratio of shadowed /unshadowed lighting, and to get a good estimate of that it requires multiple samples.



Samples: 1. Shadows on the floor plane (shadow matte) look poor compared to those on the platform



Samples: 2. Shadows on the floor plane appear the same as on the platform

Standard Hair



Hair groom by S ohrab Esfehiani

Standard_hair is a physically-based shader to render hair and fur, based on the d'Eon model for specular and Zinke model for diffuse. Realistic results can be obtained by setting a few simple parameters for the base color, roughness, and index of refraction.

The standard_hair shader is designed to be used with Arnold's curves shape node, and will not work well on other types of shapes. Other shaders like Lambert also work on curves, but cannot accurately simulate hair scattering.



When rendering hair it is recommended that you use curves in ribbon mode. Thick mode is meant for special effects such as tentacles, spaghetti, etc, and non-transparent shading.

Texturing Hair

When connecting textures to hair, the intended workflow is to set melanin to 0 and link the texture to the base_color. Diffuse can normally be left at 0. The base_color will affect the secondary specular and transmission, just like melanin. They are just two different ways to control the absorption in hair. Melanin can be seen as a way to more easily pick a base_color appropriate for human hair, but if you have a texture, then melanin can be disabled.



It is not possible to directly map hair shader parameters to standard_hair, due to the very different shading models. The specular, specular2, and transmission colors are still separately controllable but are now automatically energy-conserving using Fresnel and need to be made brighter to get a similar look. The diffuse component needs to be made brighter to match hair. However, for accurate results, it is better to not use diffuse scattering or specular tints at all. These parameters provide artistic control beyond what is physically accurate and are usually not needed anymore.



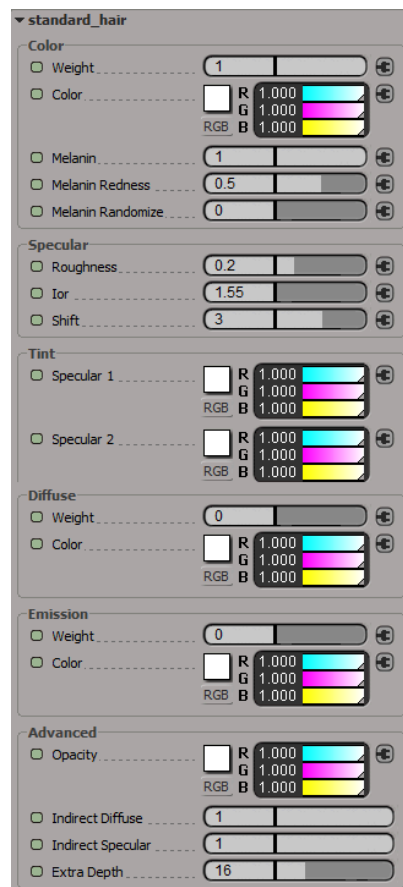
To reduce noise with white/light-colored hair, you will need to either increase the specular and/or the Camera (AA) samples.



Specular Samples: 2 (default)



Specular Samples: 5



Color

Weight

The brightness of the hair, a multiplier for the base_color.



Color

The color of the hair. This controls absorption inside hair fibers, which gives hair its color as light scatters around. Note that for blond or bright-colored hair, a higher number of specular bounces is needed to get accurate results.



It is recommended to leave the `base_color` as pure white and instead use the melanin controls. This guarantees that the color will be "plausible" in the sense that it more resembles the appearance of human hair.

This parameter can also be connected to a texture map to vary the hair color.



Ramp textures connected to `base_color`

Melanin

The melanin parameter is used to generate natural hair colors, by controlling the amount of melanin in the hair. Colors will range from blonde around 0.2 to red and brown around 0.5, to black at 1.0. To control the hair color with a texture map, set melanin to 0 and use base color instead.



0.2



0.5

Melanin Redness

Controls the redness of hair. Higher values increase the proportion of red pheomelanin (as found in red hair), relative to the amount of brown eumelanin (0.. 1).



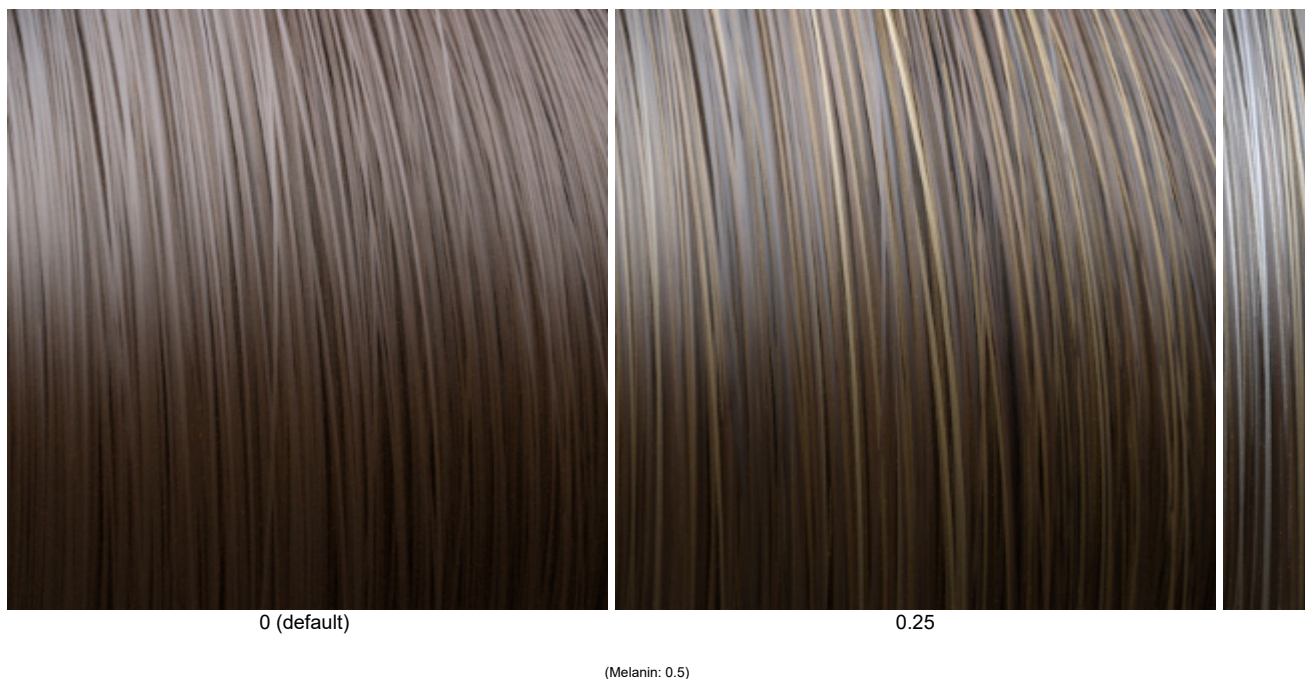
0



1 (max)

Melanin Randomize

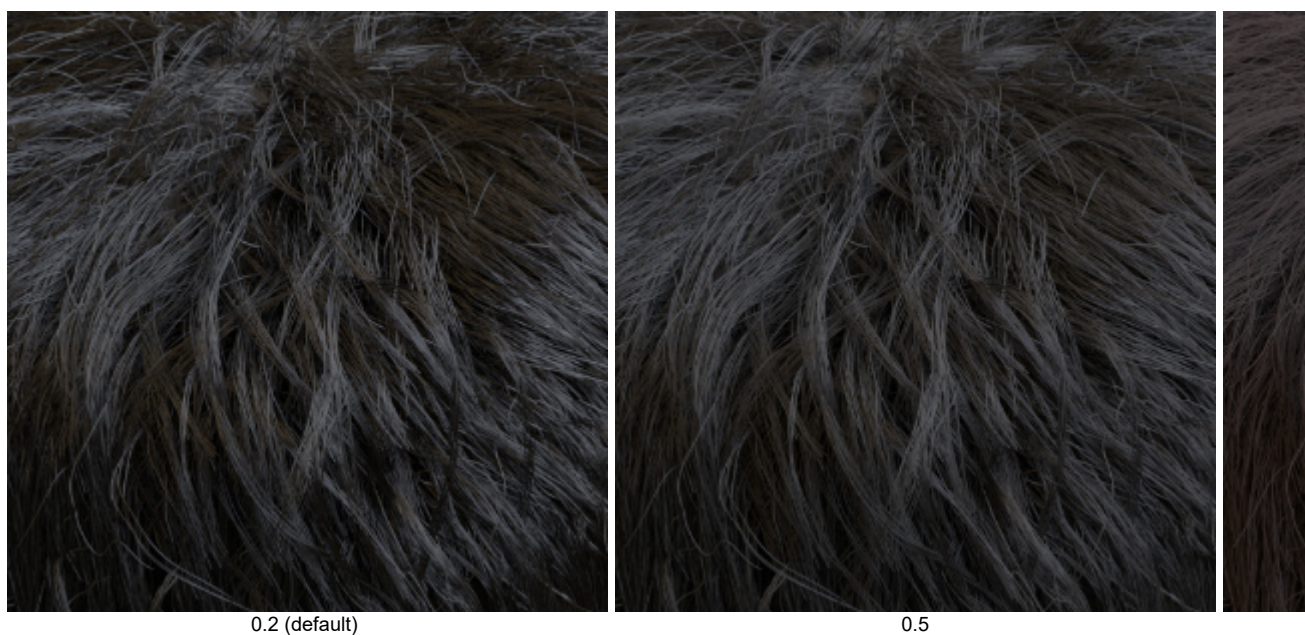
Randomizes the amount of melanin in hair fibers, for variation in hair colors (0.. 1).



Specular

Roughness

Controls the roughness of hair specular reflections and transmission. Lower values give sharper, brighter specular highlights while higher values give softer highlights.



IOR

Index of refraction. Each hair fiber is modeled as a dielectric cylinder, with hair reflecting off and transmitting into the fiber depending on the IOR. Lower IOR values give stronger forward scattering, and higher values give a stronger reflection.



1.3



1.55 (default)



You can use IOR values outside of 1.4-1.6 to render wet hair.

Shift

The angle of scales on the hair fiber, shifting the primary and secondary specular reflections away from the perfect mirror direction. For realistic results for human hair, a small angle between 0° and 10° should be used (values for animal fur may be different). For synthetic hair, such as a nylon wig, use a shift value of 0 since the surface of the fiber is smooth.



Rollover image

Some suggested values can be found below (taken from "[Studies of light scattering from ethnic hair fibers](#)"):

Piedmont: 2.8

Light brown European: 2.9

Dark brown European: 3.0

Indian: 3.7
Japanese: 3.6
Chinese: 3.6
African-American: 2.3

Tint

Specular 1

The scale of the primary specular contribution, which simply multiplies the primary specular color.



For realistic and clean hair, the color should be set to white. For realistic hair, it is recommended that this value is left at the default value (1.0/white).



White (default)



Magenta

Specular 2

The scale of the transmission contribution, which simply multiplies the transmission tint. For realistic and clean hair, this color should be set to white to let the base color tint the transmission.



For realistic hair, it is recommended that this value is left at the default value (1.0).



White (default)



Blue

Transmission•

The scale of the transmission contribution, which simply multiplies the transmission tint. For realistic and clean hair, this color should be set to white to let the base color tint the transmission. •



For realistic hair, it is recommended that this value is left at the default value (1.0).



White (default)



Blue

•

•

•

! The parameters below provide additional artistic control, but are not required to achieve realistic results.

Diffuse

Weight

Controls the diffuseness of hair, with 0 giving fully specular scattering, and 1 fully diffuse scattering. For typical realistic hair, no diffuse component is needed. Dirty or damaged hair might be approximated with diffuse scattering.

! Diffuse should not be used for realistic hair. Real hair does not have a diffuse component (unless makeup, spray, etc.). This parameter is not physically correct and is provided purely for additional artistic control. It is not needed to achieve realistic results.



0 (default)



0.1

Color

Diffuse scattering color.

! This parameter is not physically correct and is provided purely for additional artistic control. It is not needed to achieve realistic results.



Red



Green



Emission

Weight

The multiplier for the emission color.



This parameter is not physically correct and is provided purely for additional artistic control. It is not needed to achieve realistic results.



0 (default)



0.25



Color

The color used as emission.



This parameter is not physically correct and is provided purely for additional artistic control. It is not needed to achieve realistic results.



Red



Green



Advanced

Opacity

The opacity of the hair. This is set to full white by default, which means fully opaque hair, and for best performance, it should be left to the default.

Playing with this value to make the hair less opaque (or in other words, more transparent) can sometimes help make the hair look softer, at the expense of significantly increased render times. If not set to full white, be sure to disable the `opaque` flag in the `Arnold parameters` property of the hair object, or else Arnold will bypass any transparency-related calculations to optimize render times.



This parameter is not physically correct and is provided purely for additional artistic control. It is not needed to achieve realistic results.

Indirect Diffuse

The amount of diffuse light received from indirect sources only. Values other than 1 are not physically correct.



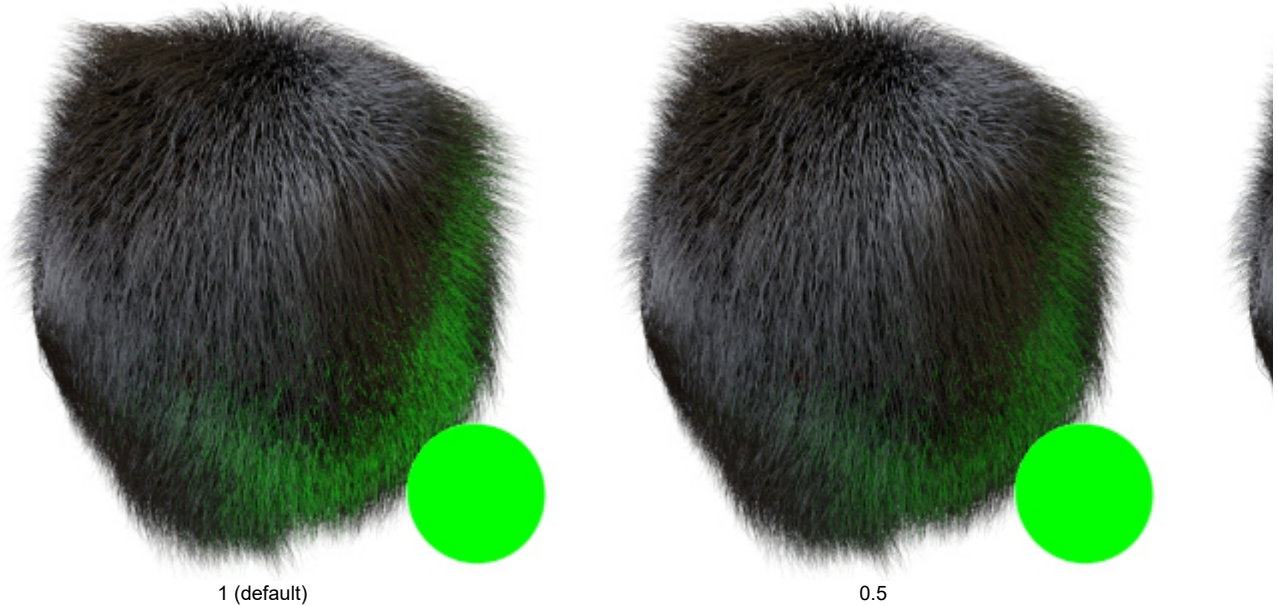
1 (default)



0.5

Indirect Specular

The amount of specularly received from indirect sources only. •Values other than 1 are not physically correct.



Extra Depth

Adds extra specular_ray_depth just for this shader. Blonde hair renders correctly by default, without needing to increase the GI_specular_samples first. •

i For blond or bright-colored hair, a high number of specular bounces must be used to accurately simulate multiple scattering. If this is too expensive to render, a diffuse component can be used as an approximation instead.





-
-
-
-

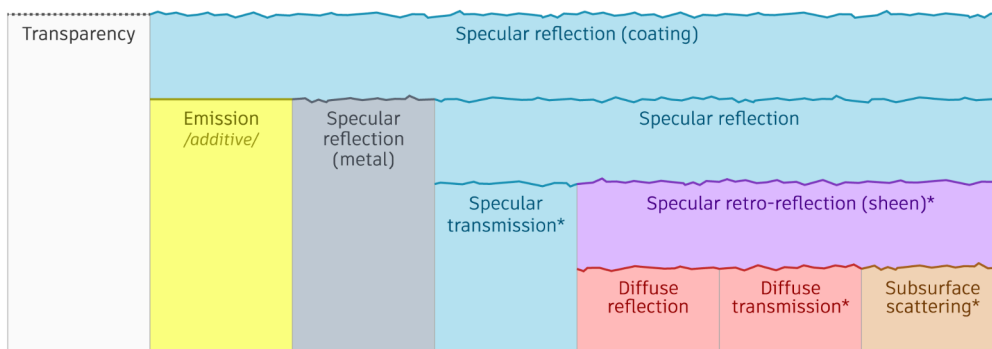
Many thanks to [Zeno Pelgrims](#) for the use of his hair groom.

Standard Surface



The standard_surface shader is a physically-based shader capable of producing many types of materials. It includes a diffuse layer, a specular layer with complex Fresnel for metals, specular transmission for glass, subsurface scattering for skin, thin scattering for water and ice, a secondary specular coat, and light emission.

This shader models a material consisting of ten components that are layered and mixed hierarchically as illustrated in the diagram below. The properties of individual components can vary across the surface.



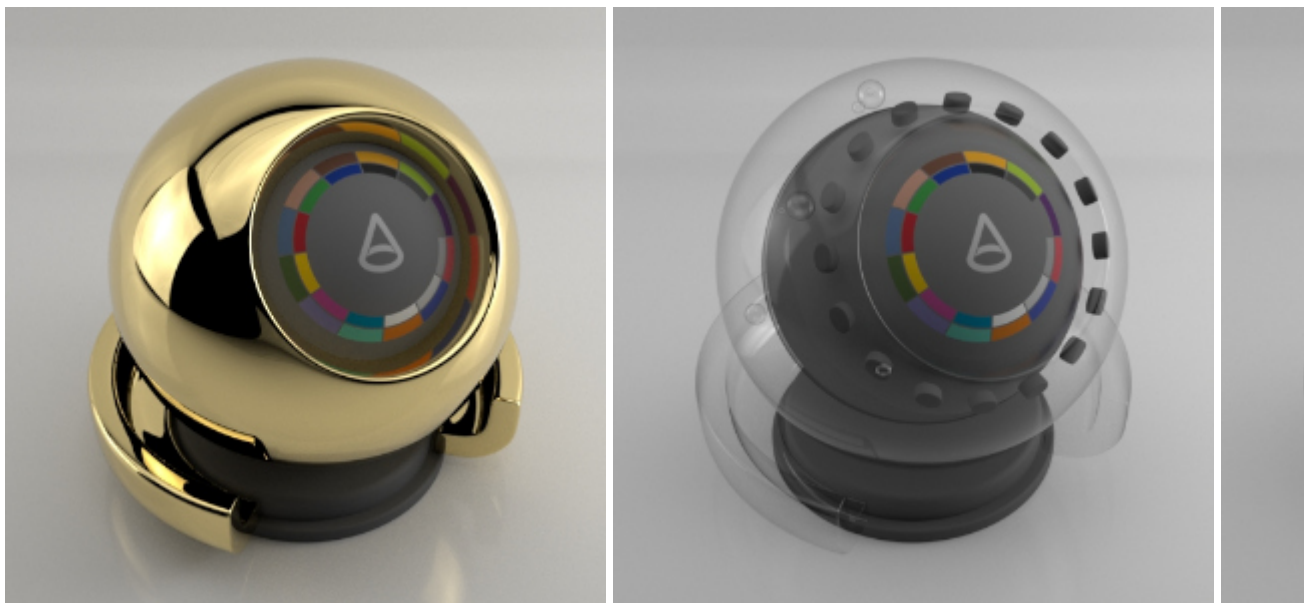
Schematic illustration of the idealized material model that standard_surface emulates. Horizontal stacking of components represents statistical mixture and vertical stacking represents layering. Components marked with an asterisk are optional.

Material Types

By default, the parameters are appropriate for materials such as plastic, wood or stone. By setting a few key parameters to 1, different types of materials can be quickly created:

- Metalness : gold, silver, iron, car paint.
- Transmission : glass, water, honey, soap bubble.
- Subsurface : skin, marble, wax, paper, leaves.
- Thin Walled : paper, leaves, soap bubble.

Parameter values between 0 and 1 may be used to create more complex materials that are a mix of basic material types.

Examples of `standard_surface` material presets found within Arnold

Energy Conservation

Standard Surface is energy conserving by default. All its layers are balanced so that the amount of light leaving the surface does not exceed the amount of incoming light. For example, as a surface is made more metallic and the specular layer contribution is increased, the diffuse layer contribution is reduced accordingly to ensure energy conservation.



Diffuse and rough (left) to metallic specular (right).



When using layer weights or colors with values higher than 1, energy conservation is broken. It is discouraged to create such materials, as they will not behave predictably under different lighting, and may lead to increased noise and poor rendering performance.

Due to the large number of controls, the Standard Surface shader is split up into several groups. The individual settings for each group are described in more detail in the pages below.

- [Base](#)
- [Specular](#)
- [Transmission](#)
- [Subsurface](#)
- [Coat](#)
- [Thin Film](#)
- [Emission](#)
- [Geometry](#)
- [Advanced](#)

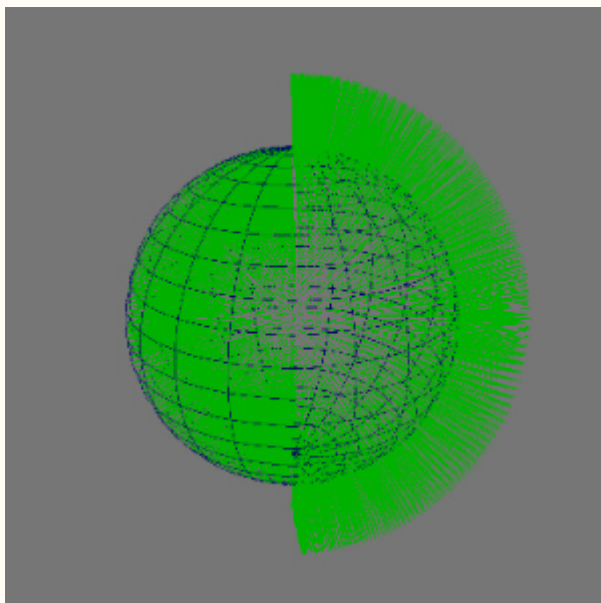


Further information about physically based rendering in Arnold can be found [here](#).

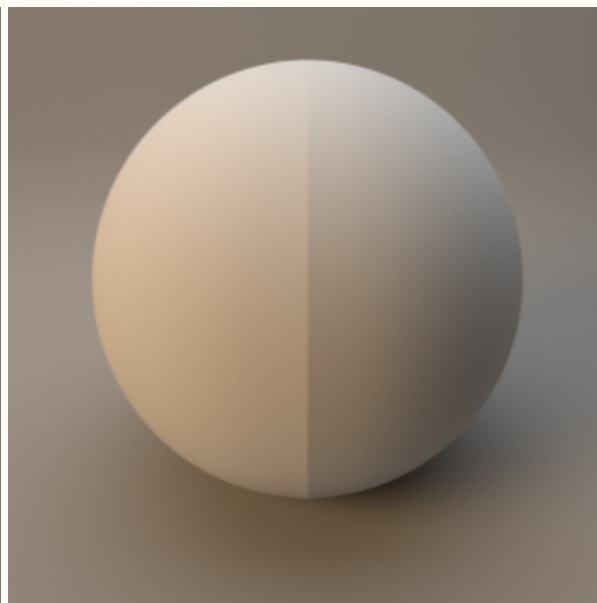


Surface Normal Direction

When rendering diffuse surfaces it is very important that the normals of the geometry face in the right direction. In the example below you can see the difference between normals that are facing inwards in the wrong direction (left side) versus those that are facing correctly in the outwards direction (right side).

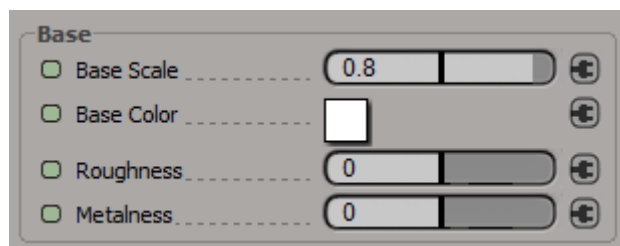


The normals are reversed on the left side of the sphere



The right side of the sphere renders correctly

Base



Scale

The base color weight.



0



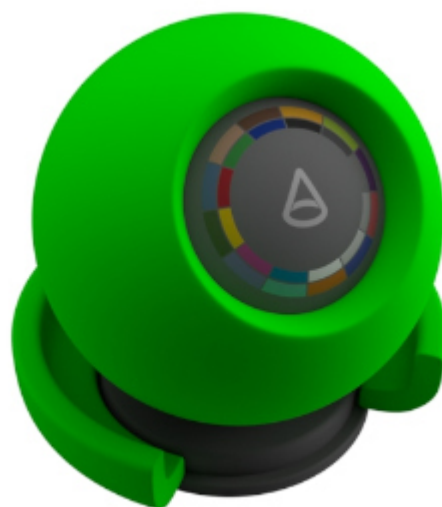
0.5

Color

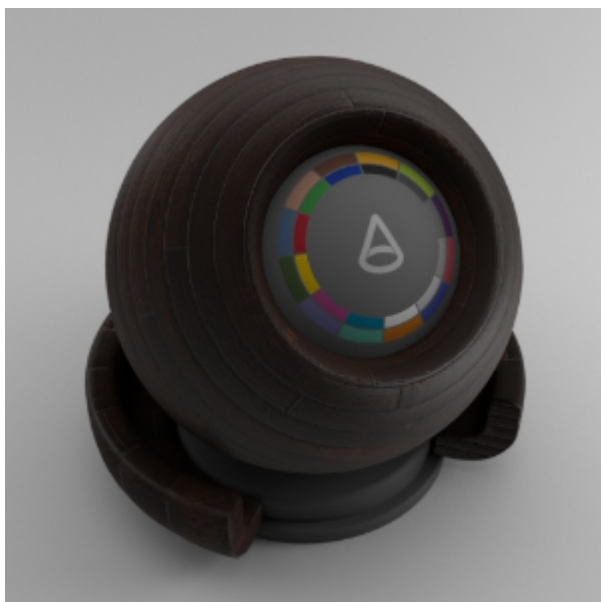
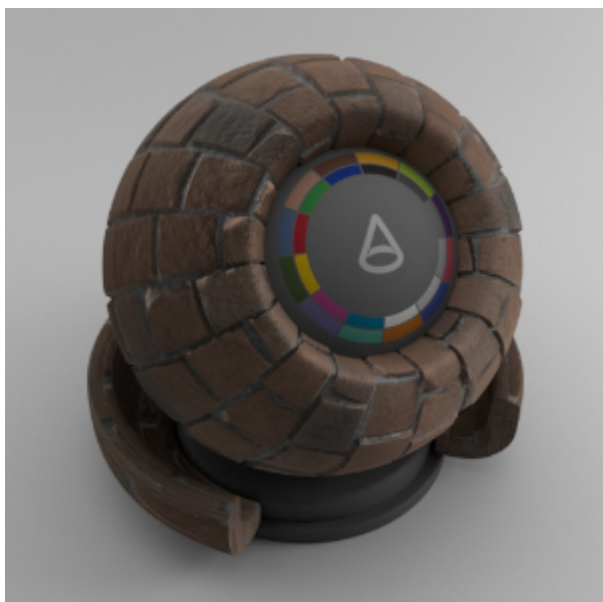
The `base_color` sets how bright the surface is when lit directly with a white light source (intensity at 100%). It defines which percentage for each component of the RGB spectrum does not get absorbed when light scatters beneath the surface. Metal normally has a black or very dark base color, however, rusty metal's need some base color. A `base_color` map is usually required.



Red



Green



Diffuse file texture -> base_color

Roughness

The base component follows an Oren-Nayar reflection model with surface roughness. A value of 0.0 is comparable to a Lambert reflection. Higher values will result in a rougher surface look more suitable for materials like concrete, plaster or sand.



0 (default)



0.5

Metalness

With metalness 1.0 the surface behaves like a metal, using fully specular reflection and complex fresnel.



0 (default)



0.5



For a perfectly sharp and mirror-like reflection, increase metalness to 1, and reduce specular_roughness to 0. The base_weight should also be set to 1.



When metalness is enabled, the specular_weight and specular_color only control the edge tint, while the base is still affected by roughness.

The metal appearance is controlled using the base_color (facing) and specular_color (edge tint) parameters. These are automatically translated to physical `and` values, to achieve the same look but with easily tweakable and texturable colors. Some examples of real-world values for metals can be found below.

	Base Color	Specular Color
Aluminium (Al)	☐ 0.912 0.914 0.920	☐ 0.970 0.979 0.988

Copper (Cu)	 0.926 0.721 0.504	 0.996 0.957 0.823
Gold (Au)	 0.944 0.776 0.373	 0.998 0.981 0.751
Iron (Fe)	 0.531 0.512 0.496	 0.571 0.540 0.586
Lead (Pb)	 0.632 0.626 0.641	 0.803 0.808 0.862
Mercury (Hg)	 0.781 0.779 0.779	 0.879 0.910 0.941
Nickel (Ni)	 0.649 0.610 0.541	 0.797 0.801 0.789
Platinum (Pt)	 0.679 0.642 0.588	 0.785 0.789 0.784
Silver (Ag)	 0.962 0.949 0.922	 0.999 0.998 0.998



Aluminium



Copper



Iron



Lead



Nickel



Platinum

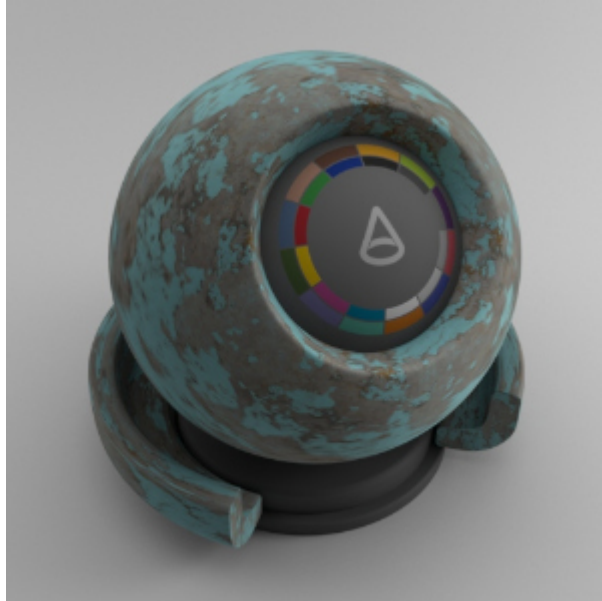
Metalness values between 0.0 and 1.0 can be used to texture surfaces like rusted iron, where different areas of the surface can have more reflective clean metal and more diffuse rust. PBR metalness maps from applications like Substance Painter can be connected to this parameter.



Rusted iron texture -> metalness



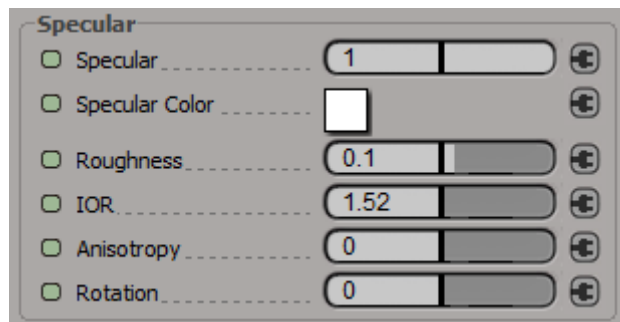
Greasy worn metal



Worn painted cement

Shaders using metalness textures

Specular



Specular

The specular_weight. Influences the brightness of the specular highlight.



0



0.5

Specular Color

The color the specular reflection will be modulated with. Use this color to 'tint' the specular highlight. You should only use colored specular for certain metals, whereas non-metallic surfaces usually have a monochromatic specular color. Non-metallic surfaces normally do not have a colored specular.



Red



Green

Roughness

Controls the glossiness of the specular reflections. The lower the value, the sharper the reflection. In the limit, a value of 0 will give you a perfectly sharp mirror reflection, while 1.0 will create reflections that are close to a diffuse reflection. You should connect a map here to get variation in the specular highlight.

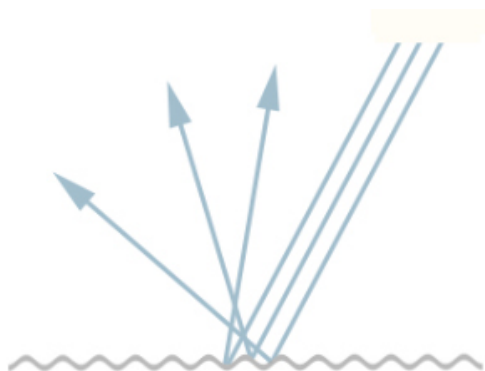


0.2 (default)



0.4

The 'microscopic' features of a surface affect the diffusion and reflection of light. This 'micro surface' detail has the most noticeable effect on specular reflections. In the diagram below, you can view parallel lines of incoming light commence to diverge when reflected from rougher surfaces when each ray hits a part of the surface with a different orientation. In summary, the rougher the surface becomes, the more the reflected light will diverge or appear 'blurred.'



'Microsurface' detail is represented as a general measure of roughness (this surface would have a high specular_roughness value).

The brightness of the specular highlight is automatically linked to its size due to the standard_surface shader's energy-conserving nature. In the example below, all of the materials are reflecting the same amount of light, but the rougher surface is spreading it out in multiple directions. However, with low amounts of roughness, the surface is reflecting a more concentrated amount of light.



0.1 (default)



0.4

To get variation in the highlights of the surface, a map should be connected to the Specular Roughness. This will influence not only the brightness of the highlight but also its size and the sharpness of the environmental reflection.



You can connect the file texture to a [range](#) shader to control the effect of the specular_roughness.



0



0.5

Fingerprint texture -> specular_roughness



The specular_roughness affects both specular reflection and refraction. There is also a transmission_extra_roughness parameter to add some additional roughness for refraction if required. You can, however, use coat to create a rough reflection layer over a sharp refraction.



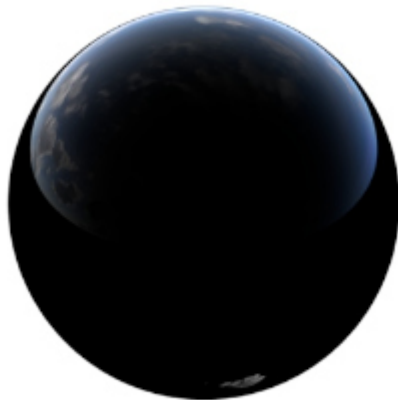
0



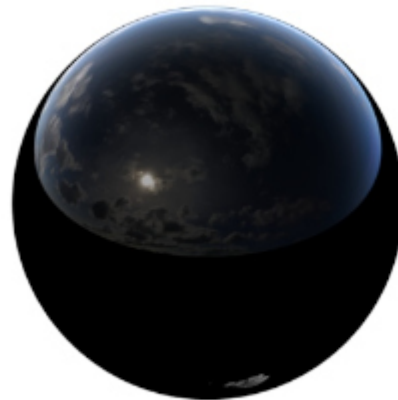
0.5

IOR

The IOR parameter (Index of Refraction) defines the material's [Fresnel reflectivity](#) and is by default the angular function used. Effectively the IOR will define the balance between reflections on surfaces facing the viewer and on surface edges. You can see the reflection intensity remains unchanged, but the reflection intensity on the front side changes a lot.



1



1.1

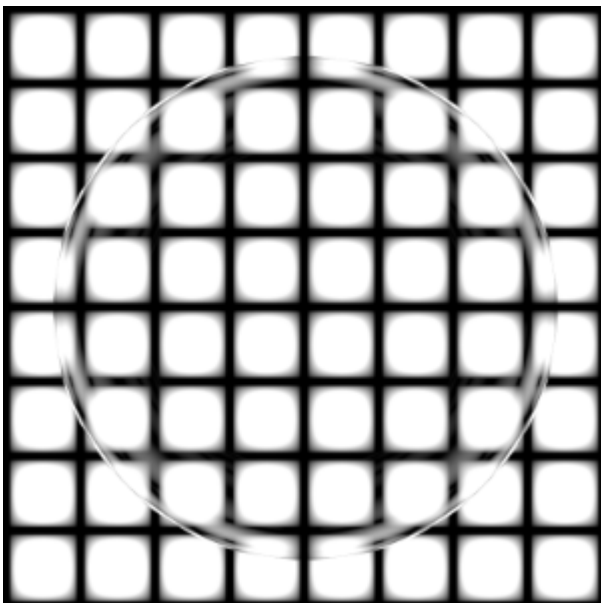


Using a very high IOR value can look quite similar to metalness. It looks the same if you set the `base_color` to the `specular_color` and the `specular_color` to black. The difference is that you get an extra reflection at the edges, with the `specular_color` controlling the edge tint. The metal fresnel works the same as in the new complex IOR shader, with the artistic parameters.

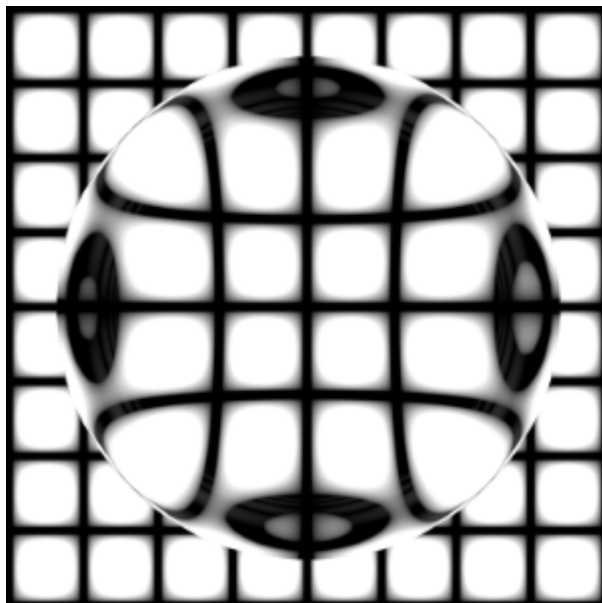
You should normally use IOR for materials like plastic, glass, or skin (dielectric fresnel) and `metalness` for metals (conductive fresnel with `complex_ior`). The other reason is that `metalness` is easier to texture since it's in the 0..1 range, and using textures from applications like Substance painter works best when using `metalness` rather than IOR.

Specular IOR with Transmission

The default value of 1.0 is the refractive index of a vacuum, i.e., an object with IOR of 1.0 in empty space will not refract any rays. In simple terms, 1.0 means 'no refraction'. The `standard_surface` shader assumes that any geometry has outward-facing normals, that objects are embedded in the air (IOR 1.0) and that there are no overlapping surfaces.



1

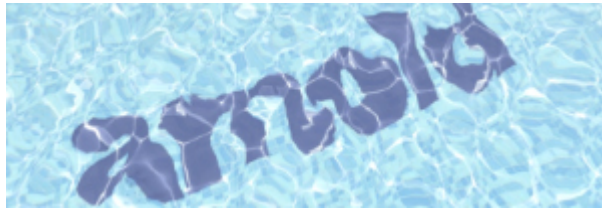


1.1





1



1.5 (default)



Normals

When rendering refractive surfaces, it is very important that the normals of the geometry face in the right direction. In the example below (left), you can see the difference between normals that are facing in the right direction (outward), versus those that are facing inwards (incorrect). This is especially important when rendering surfaces with double-sided thickness, such as glass. Normal direction is equally important when rendering single-sided surfaces such as a car windscreen (right).



Normals pointing outward (correct). Rollover image to view normals pointing inward (incorrect).



Windscreen model (single-sided). Rollover image to view normals pointing inward (incorrect).



If you see any black where there should be refraction, you may not have a high enough `•transmission_ray_depth•` value (found in the `•ray_depth•` section in the `•render settings•`). The default value is eight, which is sufficient for most cases.

Anisotropy

Anisotropy reflects and transmits light with a directional bias and causes materials to appear rougher or glossier in certain directions. The default value for anisotropy is 0, which means 'isotropic.' As you move the control towards 1.0, the surface is made more anisotropic in the U axis.



0 (default)



0.6

Anisotropy is suitable for materials that have a clear brush direction such as brushed metal which has tiny grooves in which form a 'stretched' anisotropic reflection.



Many small discs form together to create an effect which is the anisotropic highlight

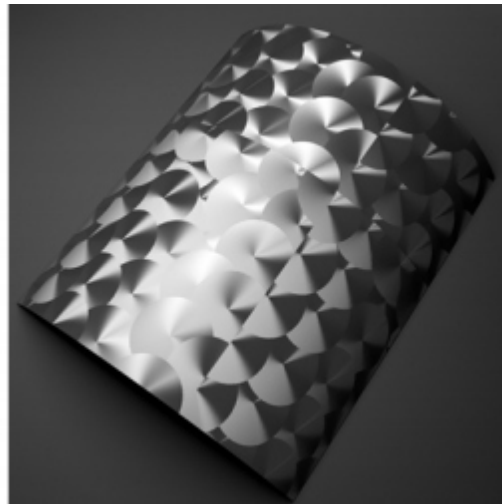
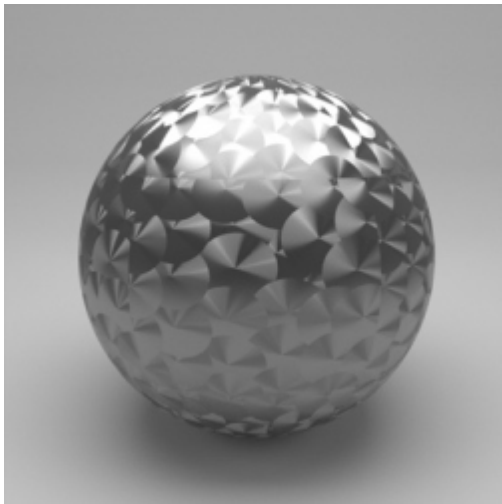


0.3



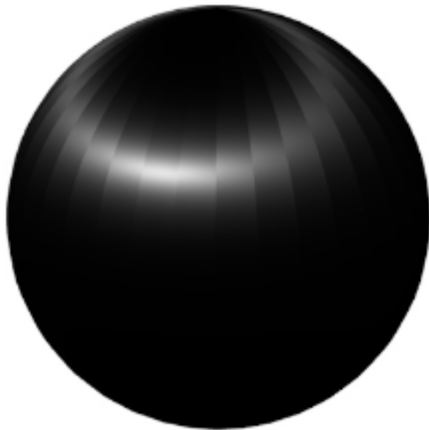
0.6

Anisotropic reflections are suitable for brushed metal effects such as in the example below:•

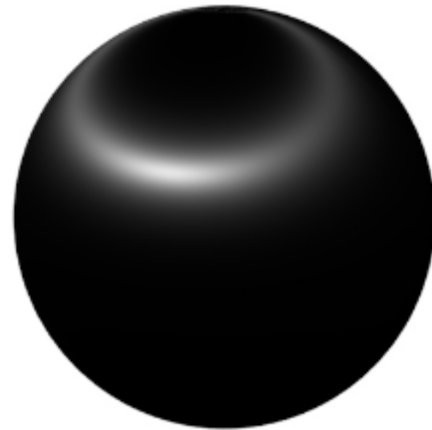


Texture assigned to anisotropic_rotation

You may notice faceting appears in highlights when using Anisotropy. It is possible to remove the faceted appearance by enabling smooth subdivision tangents (via Arnold `subdiv_smooth_derivs` parameter). Take into account this requires a subdivision iteration of at least one in the polymesh to work.



Faceting seen in the anisotropic highlight



Subdivision: enabled. Subdivision Iterations: 1. Smooth Tangents: enabled.

✓ Increase subdivision_iterations to remove anisotropic faceting.

i More information about Specular 'Anisotropy' can be found [here](#).

Rotation

The rotation value changes the orientation of the anisotropic reflectance in UV space. At 0.0, there is no rotation, while at 1.0 the effect is rotated by 180 degrees. For a surface with brushed metal, this controls the angle at which the material was brushed. For metallic surfaces, the anisotropic highlight should stretch out in a direction perpendicular to the brushing direction.

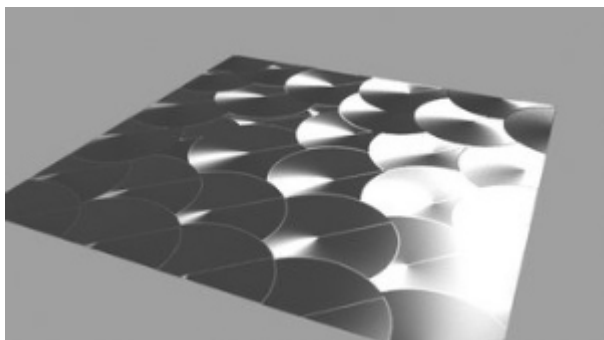


0 (default)

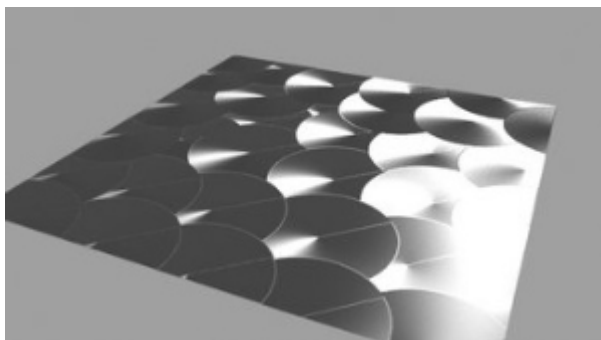


0.25

It is possible to assign textures to the rotation. When doing so, it is advisable to avoid texture filtering. This means disabling MIP-mapping and disabling the magnification filter, which by default is set to "smart bicubic." One way is to set the mipmap_bias of the image node to a strong negative value, like -8, which means "use 8 MIP levels higher resolution than usual".



Mipmap bias set to 0. Filtering set to Bicubic.

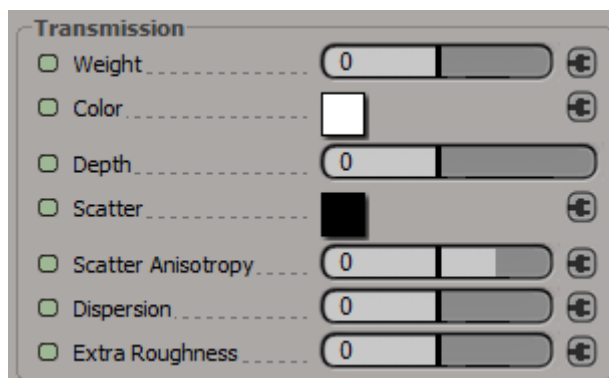


Mipmap bias set to -8. Filtering set to Bicubic.



No l

Transmission



Weight

The transmission allows light to scatter through the surface, for materials such as glass or water.



0 (default)



0.5



Note that you must also disable Opaque for the mesh that has been assigned the Standard Surface shader.



If you can only see black where there should be transparency, you may not have a high enough refraction ray depth value (found in the Ray Depth section of the Render Settings). The default value is two.

Color

This filters the refraction according to the distance traveled by the refracted ray. The longer light travels inside a mesh, the more it is affected by the transmission_color. Therefore green glass gets a deeper green as rays travel through thicker parts. The effect is exponential and computed with Beer's Law. It is recommended to use light, subtle color values.



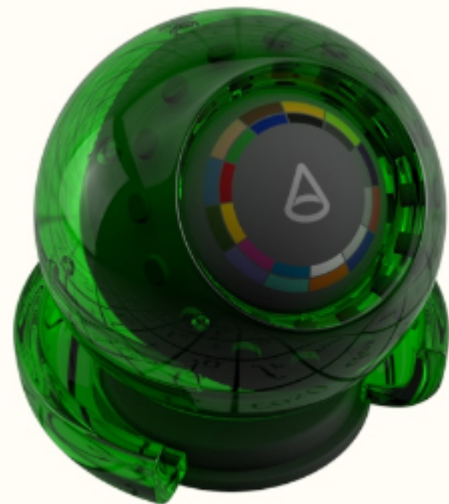
`transmission_color : red ( transmission_depth : 1 to 10)`



If you use a fully saturated color like (1, 0, 0), the way that is interpreted is that all red light is allowed to pass through, and no green and blue light. `transmission_color` values near 0 make the interior of the mesh very dense to block all light, and then set the depth multiplier to a small value like 0.001 might not make much of a difference because the depth is large anyway.



Red



Green

Fully saturated colors for `transmission_color` are not recommended

If this value has a color and shadows tinted with that color are required, then disable `opaque` for the mesh that has been assigned the `standard_surface` shader. In the example below, you can see that with `opaque` enabled the rays cannot pass through the sphere. Whereas with `opaque` disabled, the rays can pass through the sphere and absorb the color set by the `transmission_color`, thereby creating the effect of colored shadows.



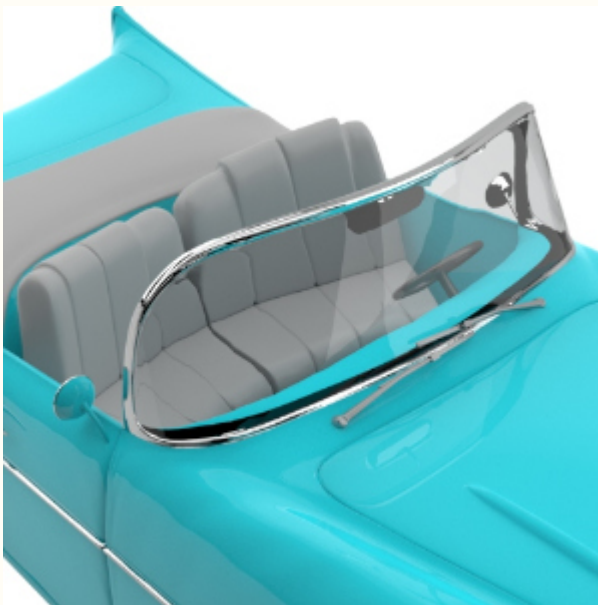
Opaque: Enabled



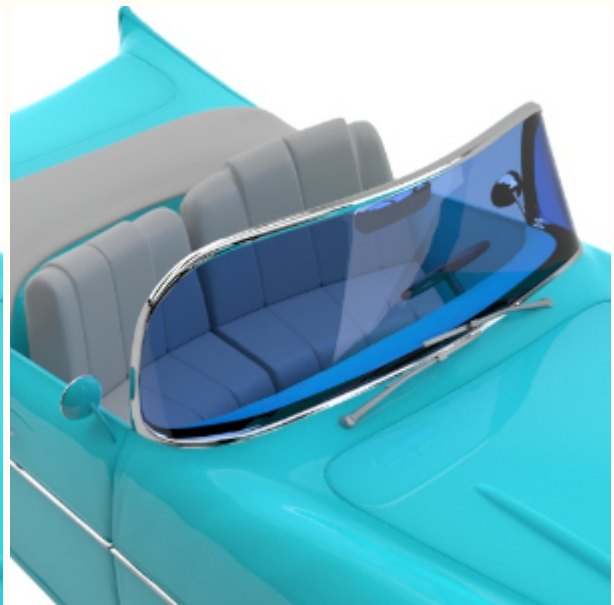
Opaque: Disabled



Note that `transmission_color` will not work for single-sided geometry unless `thin_walled` is enabled.



Disabled



Enabled. Transmission Color is visible.

Depth

Controls the depth into the volume at which the `transmission_color` is realized. Increasing this value makes the volume thinner, which means less absorption and scattering. It is a scale factor so that you can set a `transmission_color` and then tweak the depth to be appropriate for the size of your object.



1



2

The effect of increasing `transmission_depth` can be seen in the animation below. Note that a `transmission_scatter` color has also been used in this case.



Depth is scene scale-dependent and can have a dramatic effect on its appearance. The `transmission_color` and depth control transmittance/absorption, and that depends on the object scale. So for a small object to see anything you might need to set a quite low depth, or for a big object a high one. If you cannot see the effect of depth, then you may need to check the size of your scene.

When the scene scale is too small, the `transmission_color` (orange) appears incorrect (left image) with a `transmission_depth` of 1. Lowering the `transmission_depth` fixes it (right image). It is recommended to model to real-world scale to avoid these situations.



Scatter

Transmission_scatter is suitable for any liquid that is fairly thick or where there is enough of it for scattering to be visible, such as a deep body of water or honey. If you have a glass of water, there is not that much scattering, however, for an ocean, it is required. Other examples include materials like ice, opalescent glass, or milky glass.



Black (default)



Orange (suitable for thick viscous liquids like honey)



For scattering to work, ensure that `opaque` is enabled for the mesh that has been assigned the `standard_surface` shader.

Marble look using a file texture `transmission_scatter•`

Scatter Anisotropy

The directional bias, or anisotropy, of the scattering. The default value of zero gives isotropic scattering so that light is scattered evenly in all directions. Positive values bias the scattering effect forwards, in the direction of the light, while negative values bias the scattering backward, toward the light.



-0.5

0 (default)

Dispersion

Specifies the abbe number of the material, which describes how much the index of refraction varies across wavelengths. For glass and diamonds, this is typically in the range of 10 to 70, with lower numbers giving more dispersion. The default value is 0, which turns off dispersion. The chromatic noise can be reduced by either increasing the global Camera (AA) samples or the Refraction samples.



1



3



`transmission_dispersion` is ideal for gemstone materials like a diamond

Extra Roughness

Adds some additional blurriness of refraction computed with an [isotropic microfacet BTDF](#). The range goes from -1 to 1, where 0 means no roughness. It is computed as

`transmission_roughness = specular_roughness + transmission_extra_roughness.`



0 (default)



0.5

Negative values result in a lower roughness for transmission than for reflection.



0 (default)



-1

specular_roughness: 0.3



Transmission and Opacity

MultiExcerpt named '1' was not found

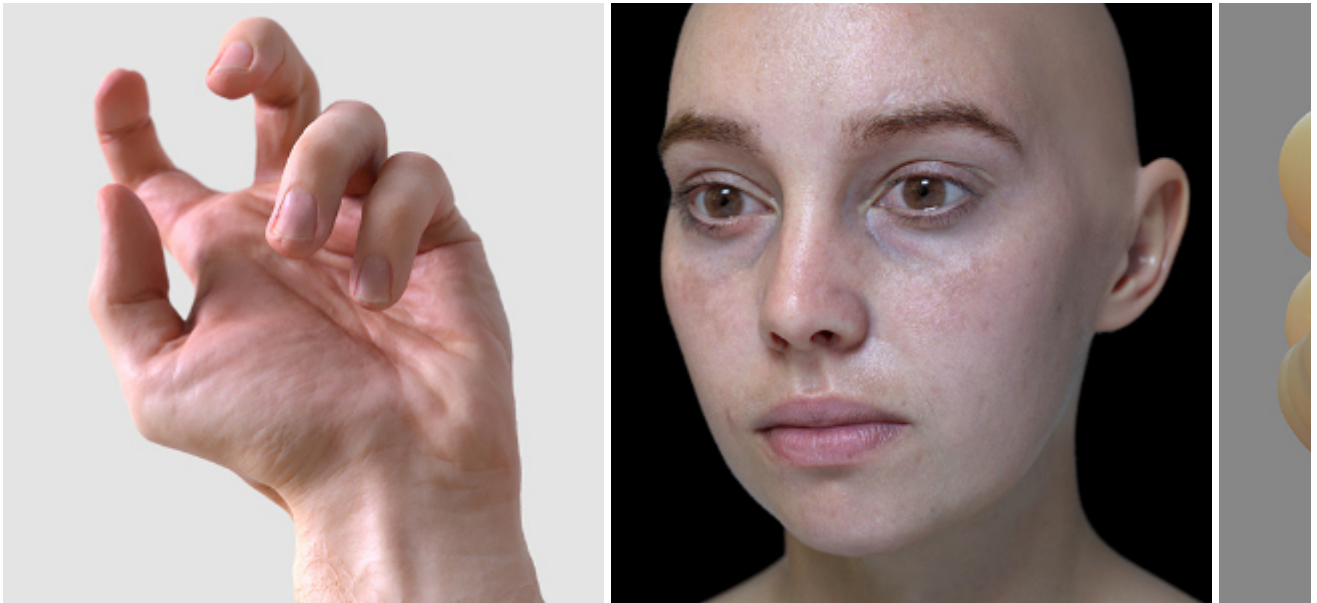
The page: Transmission and Opacity was found, but the multiexcerpt named '1' was not found. Please check/update the page name used in the 'multiexcerpt-include macro.

•

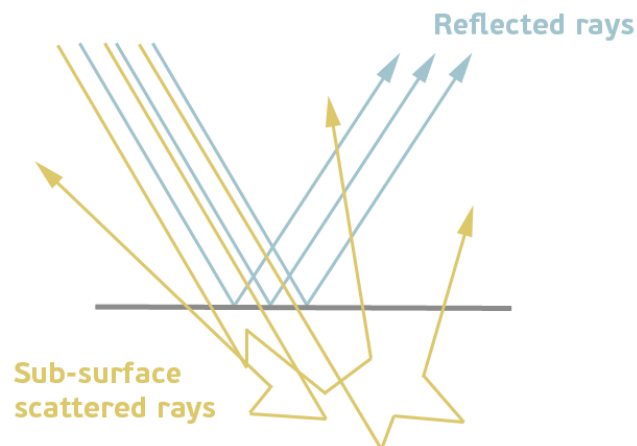
MultiExcerpt named '2' was not found

The page: Transmission and Opacity was found, but the multiexcerpt named '2' was not found. Please check/update the page name used in the 'multiexcerpt-include macro.

Subsurface



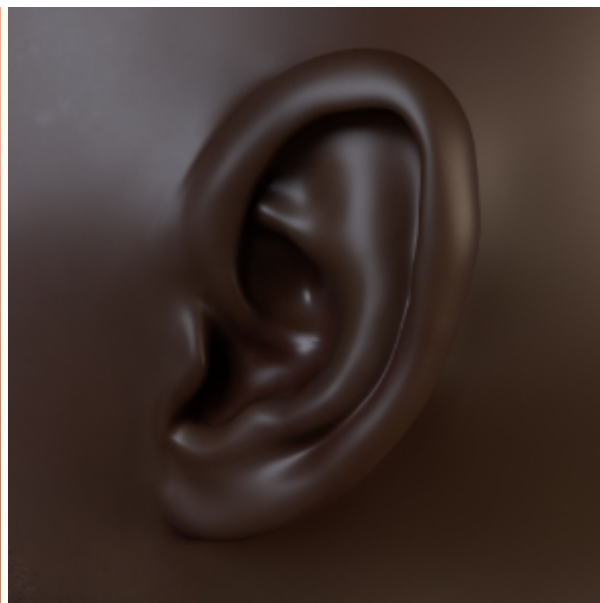
Sub-Surface Scattering (SSS) simulates the effect of light entering an object and scattering beneath its surface. Not all light reflects from a surface. Some of it will penetrate below the surface of an illuminated object. There it will be absorbed by the material and scattered internally. Some of this scattered light will make its way back out of the surface and become visible to the camera. This is known as 'sub-surface scattering' or 'SSS'. SSS is necessary for the realistic rendering of materials such as •marble, •skin, •leaves, •wax, and •milk. The SSS component in this shader is calculated using a brute-force raytracing method.



You must ensure that the geometry normals are pointing in the correct direction, otherwise, SSS will not render correctly.



Normals pointing outwards (correct)



Normals pointing inwards (incorrect)

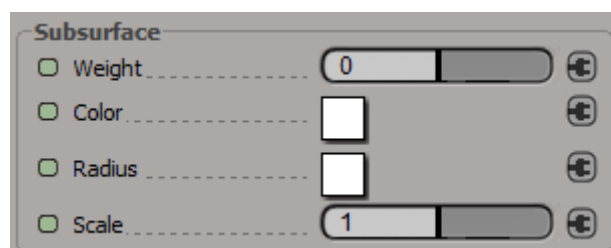
SSS is important when replicating realistic materials such as plastics, for example:



SSS disabled



SSS enabled



Weight

The 'blend' between diffuse and subsurface scattering. When set to 1.0, there is only SSS, and when set to 0 it is only Lambert. In most cases, you want this to be 1.0 (full SSS).



0 (default)



0.5

Color

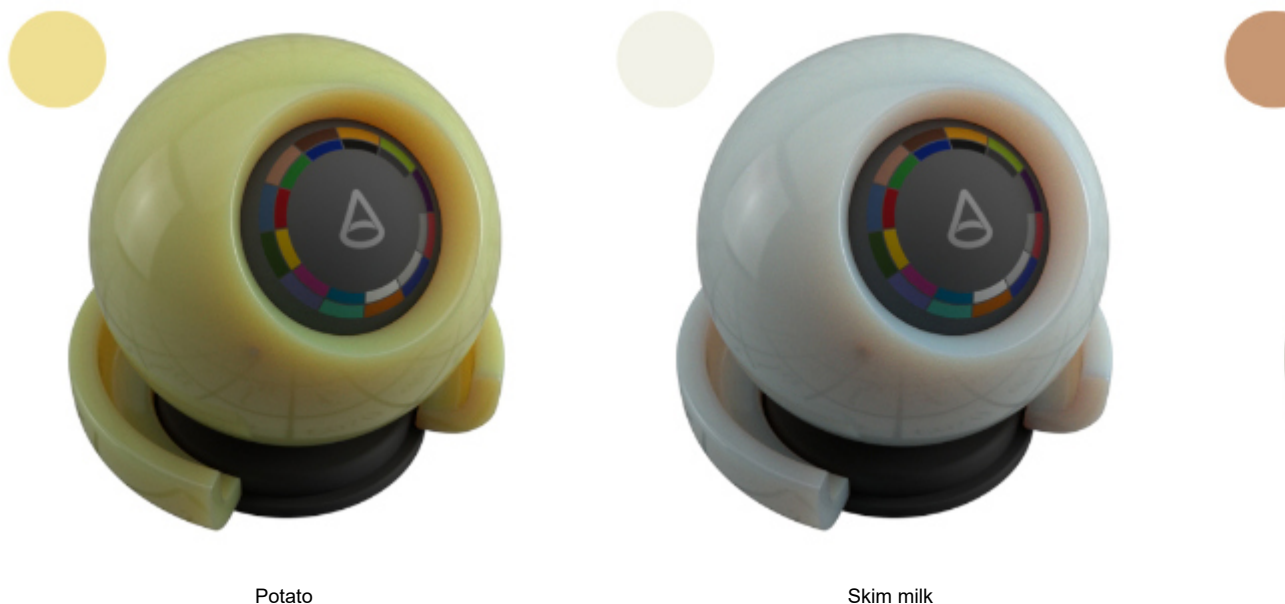
The color used to determine the subsurface scattering effect. For example, replicating a skin material would mean setting this to a fleshy color.



Apple



Chicken



Potato

Skim milk

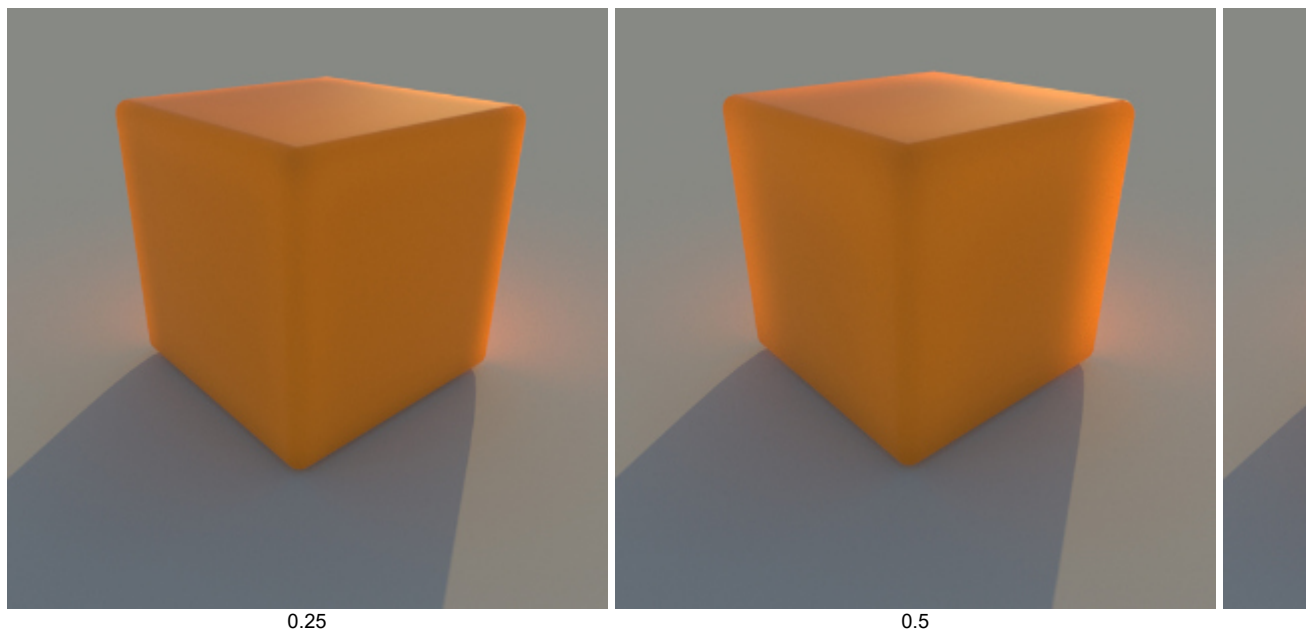


Skin Color

Generally, you would connect a color texture map to the `subsurface_color`, and set the subsurface radius to a value appropriate for skin (mostly a higher value for the red channel assuming red mostly comes from blood). The radius can also be textured if desired. Veins would be part of the color texture map, and that not only affects the color but also how the subsurface scattering behaves. `standard_surface` has a mix between base and SSS. It is your choice if you want to use diffuse with skin. It can be used to e.g. mix in a different texture with a mask for e.g. face paint, or to vary sharpness by connecting the same texture to both the `base_color` and `subsurface_color` and changing the blend factor.

Radius

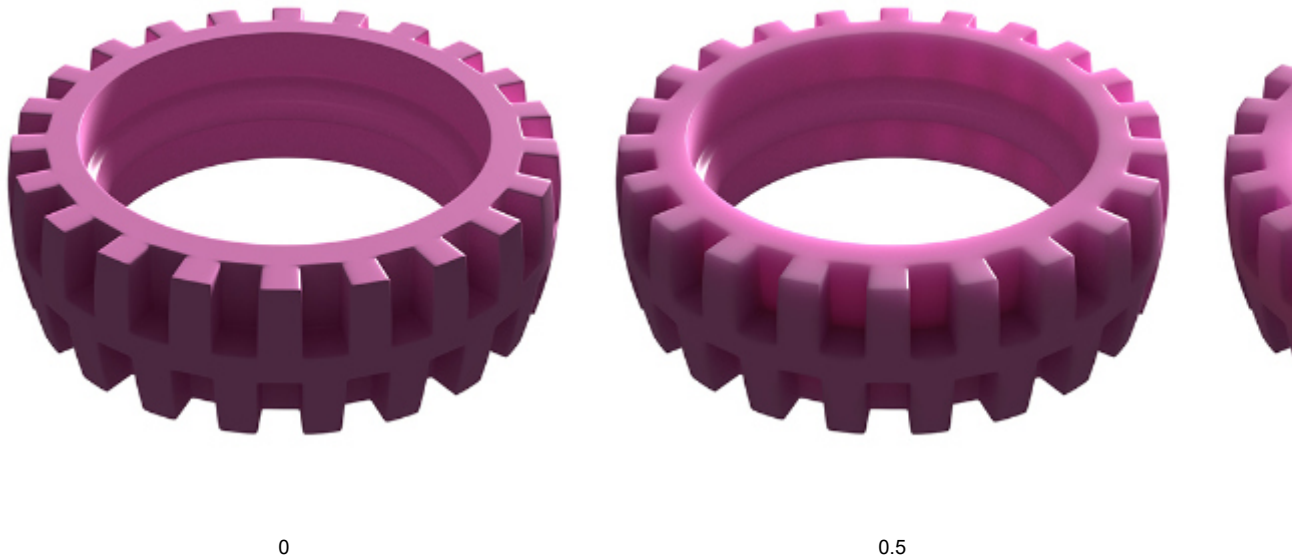
The approximate distance up to which light can scatter below the surface, also known as “mean free path” (MFP). This parameter affects the average distance that light might propagate below the surface before scattering back out. This effect on the distance can be specified for each color component separately. Higher values will smooth the appearance of the subsurface scattering, while lower values will result in a more opaque look.



0.25

0.5

The lighter the color, the more light is scattered. A value of 0 will produce no scattering effect:•



Increasing the radius value can radically change the appearance of the material, from looking like leather to marble. SSS is very scale-dependent. You will need to adjust the radius multiplier depending on the size of your model. If you were to render using the default SSS settings, you might get something that does not look correct. Alternatively, adjusting the scene scale can have similar results.



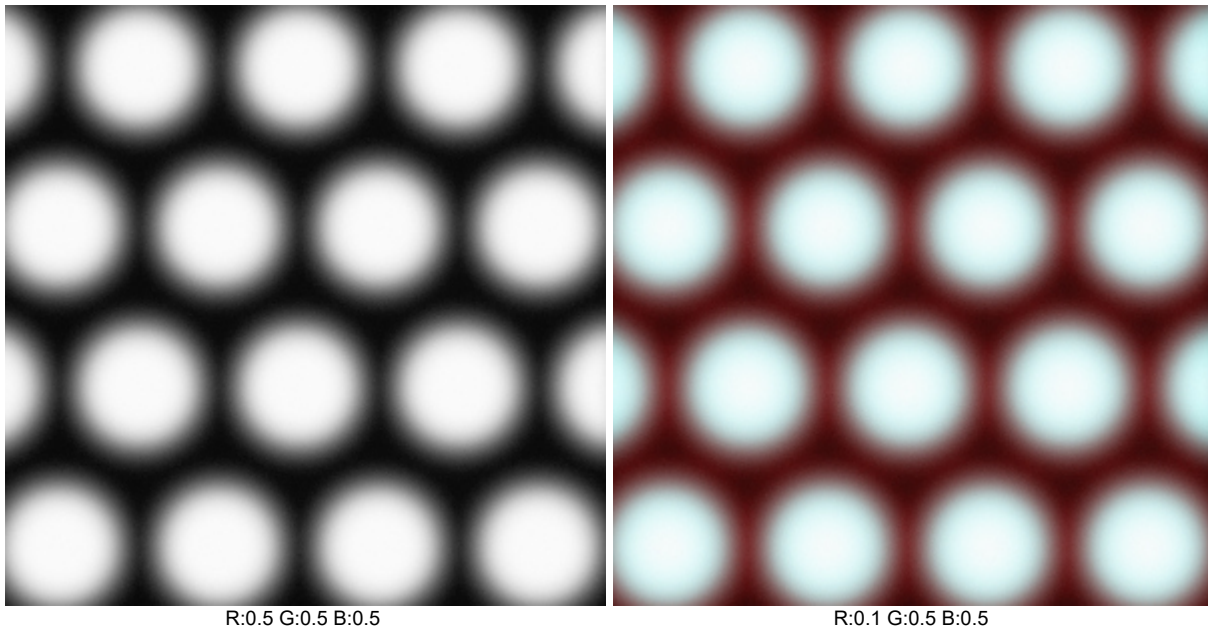
Instead of distributing all of the colors with the same amount, you can also choose different radius values for each of the RGB colors. For example, a material like clay or skin should have a higher red radius than green and blue.



When rendering skin, you should use a value like **1.0, 0.35, 0.2**, indicating that red should scatter deepest and green and blue less. This would replace the three-layer workflow (deep, mid, and shallow layers) for skin where you would make the deep layer red to indicate that deep should scatter with a larger radius. Setting `scatter_radius.R` to a larger value would correspond to that.



The images below show the effect when increasing the red color of the radius. Notice the colored 'fringing' effect around the edges of the circles. The same effect occurs when Gaussian blurring the red channel of the source image in a compositing package.



Scale

Controls the distance that the light is likely to travel under the surface before reflecting back out. It scales the scattering radius and multiplies the `sss_radius_color`.



0



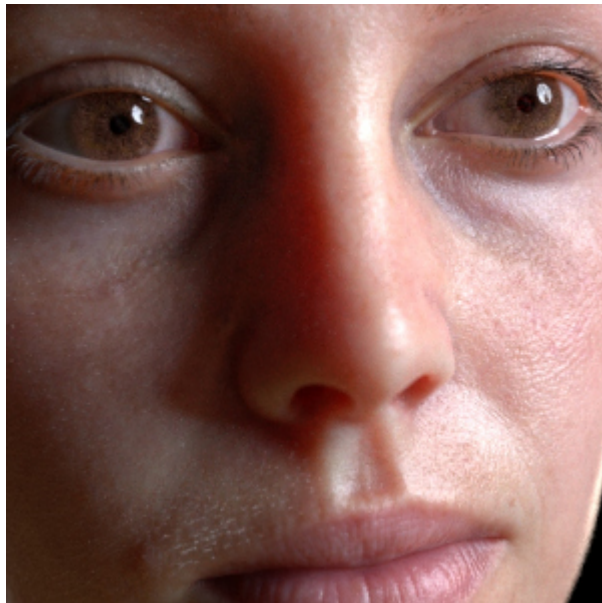
1 (default)



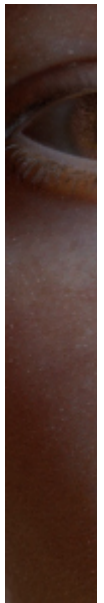
If the scene is in meters, then **SSS Scale** can be set to 0.01 to specify the **sss_radius** in centimeters. For example for skin, the **sss_radius** could be **0.37cm**, **0.14cm**, **0.07cm**.



0

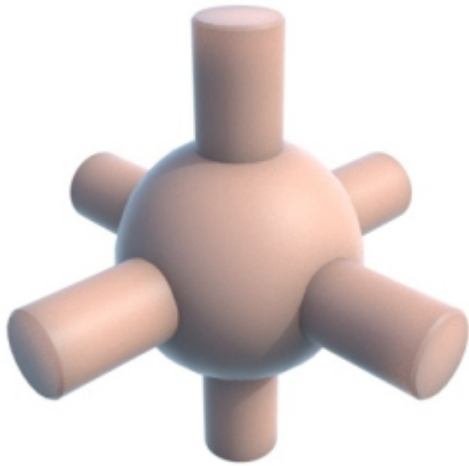


0.25

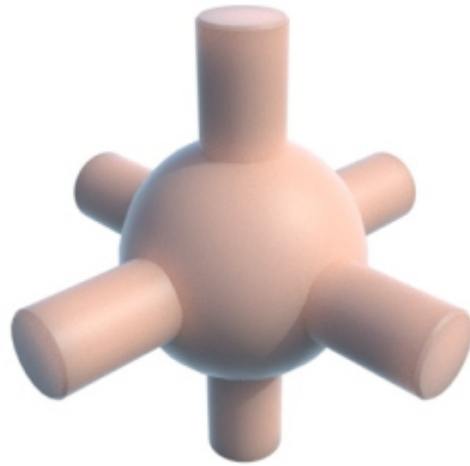


SSS Between Objects

It is possible to tag multiple objects as belonging to the same SSS 'set' so that illumination will blur across object boundaries. A common use-case might be blurring between teeth and gum geometry. It is enabled by adding the constant **STRING** **userdata:sss_setname** to the same value on the objects in the set.

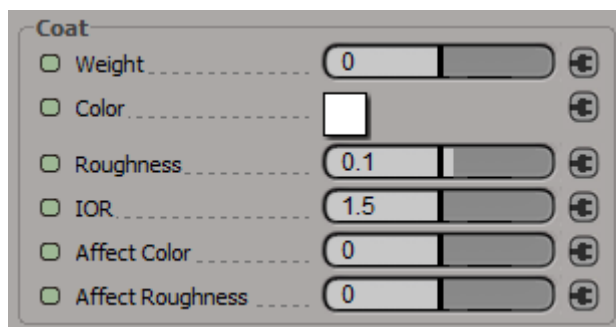


No sss_setname used (default)



Same sss_setname used across all objects

Coat



Weight

This attribute is used to coat the material. It acts as a clear-coat layer on top of all other shading effects. The coating is always reflective (with the given roughness) and is assumed to be dielectric. Examples would be the clear-coat layer for car paint or the sheen layer for a skin material. For example, for an extra oily layer or wet skin. Other examples would be objects that have been laminated or a protective film over an aluminum cell phone.



0 (default). Specular only.



0.5



The coat layer simulates a dielectric (think plastic, glass, resin/enamel, many liquids) which absorbs light and so tints all of the transmitted light (save maybe some very minor polarization effects), while metals, on the other hand, tend to filter the color of whatever it is they are reflecting, even at grazing angles. Therefore, if you want to render a bare metal, the coat_weight should be 0.



0 (default)



0.5

(Coat Color: Gold)

Coat can be used for materials such as metallic car paint, the reflective coating on a balloon, or a bubble for example.



Rollover images



When combining a coat (low roughness) on top of a specular (high roughness), the sharp coat disappears at the center due to [fresnel](#).



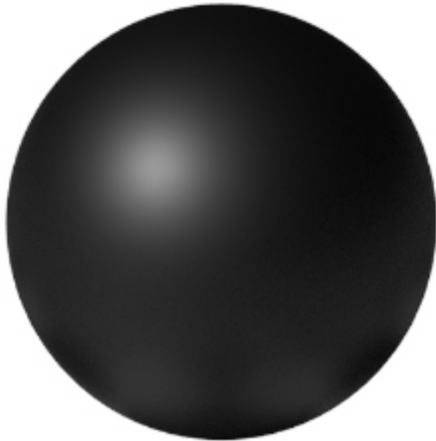
Color

This is the color of the coating layer's transparency.

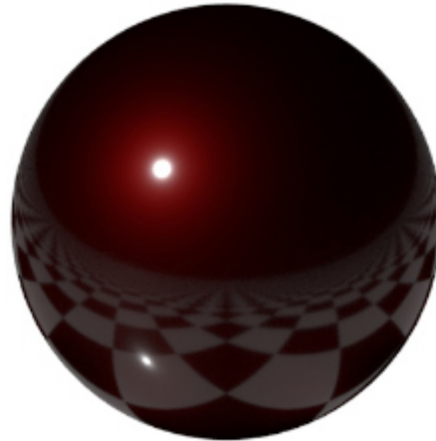


In typical real-world cases, the `coat_color` should always be white and is there mostly for artistic control.

In the example below, the red `coat_color` acts as a clear coat layer, tinting the specular reflection underneath.



Specular only (high Roughness)



Specular with Coat (low Coat Roughness and red Coat Color)

Texturing Example

In the example below, a blue and white checker texture has been applied to the `base_color` and the `coat_color` has been changed to yellow. When combined, the yellow clear-coat layer on top of the base color gives a green tint similar to a semi-transparent layer of paint or lacquer.



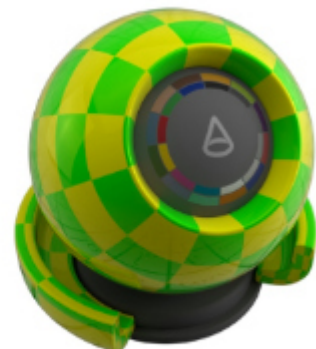
Base Color

+



Coat Color: Yellow

=

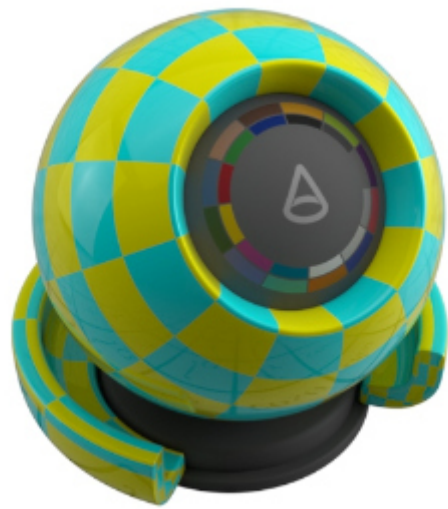


Coat and Base Color

You can also define how the coating affects the underlying `base_color` by connecting a texture map to the `coat_weight`.

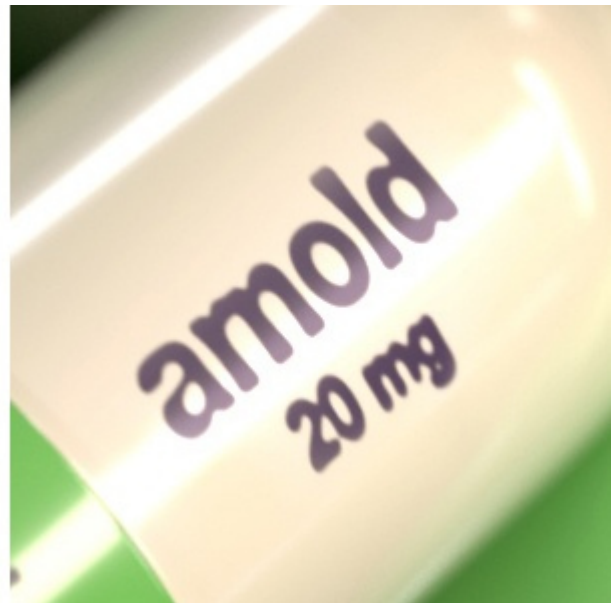


Checker connected to 'Coat Weight'



Yellow 'Coat Color' is now restricted to white areas of checker and blue 'Base Color' is unaffected

Coat_color can also be used to layer a stencil texture. In the example below, a type image has been connected to the coat_color, and the coat_roughness has been increased. Notice how it appears unaffected by the specular highlight that appears to 'sit' underneath it.



Rollover image (without coating)



Texture maps -> coat_color

Roughness

Controls the glossiness of the specular reflections. The lower the value, the sharper the reflection. In the limit, a value of 0 will give you a perfectly sharp mirror reflection, while 1.0 will create reflections that are close to a diffuse reflection. You should connect a map here to get variation in the coat highlight.



0.1 (default)

0.3

IOR

The IOR parameter (Index of Refraction) defines the material's [Fresnel reflectivity](#) and is by default the angular function used. Effectively the IOR will define the balance between reflections on surfaces facing the viewer and on surface edges. You can see the reflection intensity remains unchanged, but the reflection intensity on the front side changes a lot.



1



2

Affect Color

In the real world, when a material is coated there is a certain amount of internal reflections on the inside of the coating. This causes light to bounce onto the surface multiple times before escaping, allowing the material's color to have an enhanced effect. An example of this is varnished wood. This effect can be achieved by using `coat_affect_color`.



0

Rollover image

Affect Roughness

This causes the coating's roughness to have an effect on the underlying layer's roughness, simulating the blurring effect of being seen through the top layer. In the example below, a checker texture has been connected to the `coat_roughness` with Metalness set to 1.0. When set to 0, the coating is unaffected, whereas at 1 the coating_roughness is apparent.



0



0.5

Normal

The `coat_normal` affects the fresnel blending of the coat over the base, so depending on the normal, the base will be more or less visible from particular angles. Uses for `coat_normal` could be a bumpy coat layer over a smoother base. This could include a rain effect, a carbon fiber shader, or a car paint shader where you could use different normals (using e.g. flakes) for the coat layer and base layers. •



Rain drops



Flakes

`Coat_normal` is suited to situations where the layer may be uneven, for example, an oily or wet layer, rain on streets or glazing on food.



Rollover image (left). Puddles are created with a glossy texture map connected to `coat_normal` via bump shader (right).

When you specify a `coat_normal`, it affects just the `coat`, and none of the layers below it (diffuse, specular, transmission). In the example below right, the rock material appears to have a clear coat layer, when a bump texture is connected to the `coat_normal`.



Regular bump map. Specular: 1, Coat: 1.



Coat Normal map. Specular: 1, Coat: 1 (Specular Weight is unaffected by Coat Normal map).



Textures -> Coat Normal

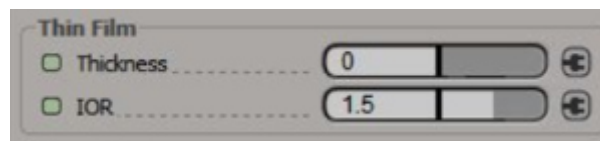
-
-
-
-

Thin Film



Thin Film•reproduces the effect of thin film interference on a surface.

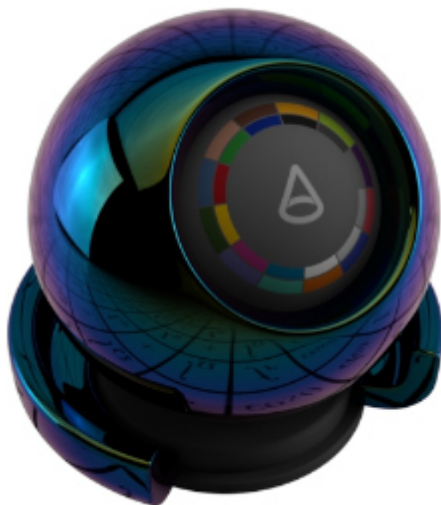
-
-



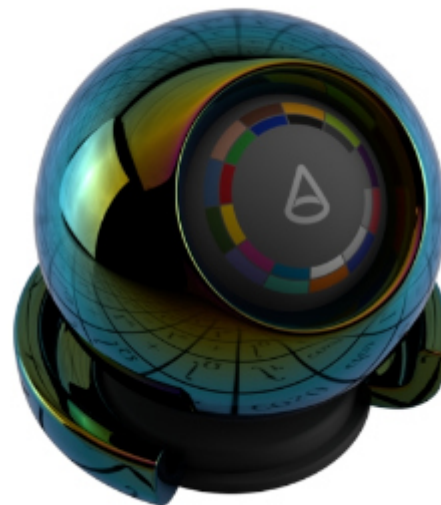
-

Thickness

Defines the actual thickness of the film between the specified min and max thickness (0 to 2000 (soft min/max)).•This affects the specular, transmission and coat components. Normally this would be something like a noise map to give some variation to the interference effect.•If the thickness becomes large like 3000 [nm], the iridescence effect will disappear, which is physically correct behavior.



300



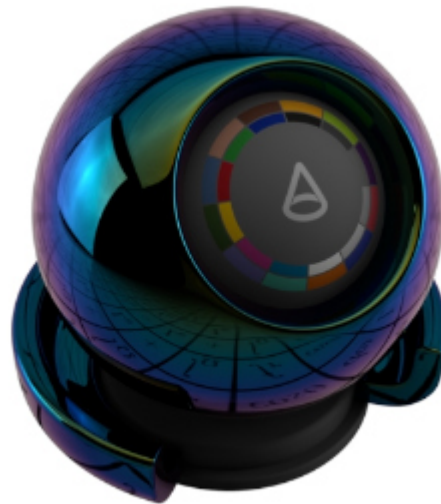
350

IOR

The refractive index of the medium surrounding the material. Normally this is set to 1.0 for air.**



1



1.5 (default)



The IOR for water is 1.33 and soap is about 1.5. Therefore the IOR for a soap bubble should be in the 1.33-1.4 range.



Soap Bubble

The following Standard Surface shader settings were used to create a soap bubble.

Base: 0

Specular: 1

Specular Color: 1 1 1

Specular Roughness: 0

Specular IOR: 1.0

Transmission: 1

Transmission Color: 1 1 1

Coat: 1

Coat IOR: 1.5

Thin Film: IOR 1.4

Thin Film Thickness: 500 [nm]

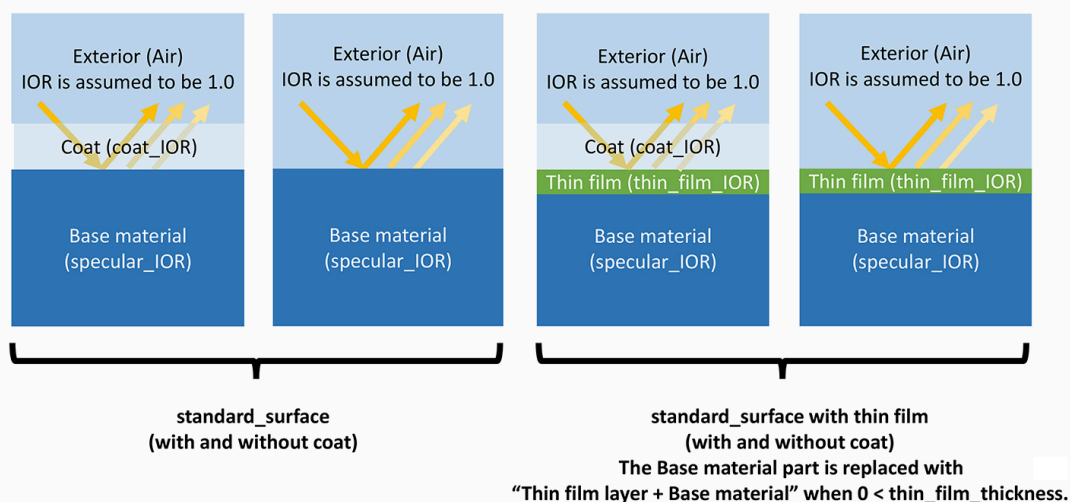
-
-

A grayscale map should be connected (through a Range shader) to Thin Film Thickness if variation in the Thin Film is required.

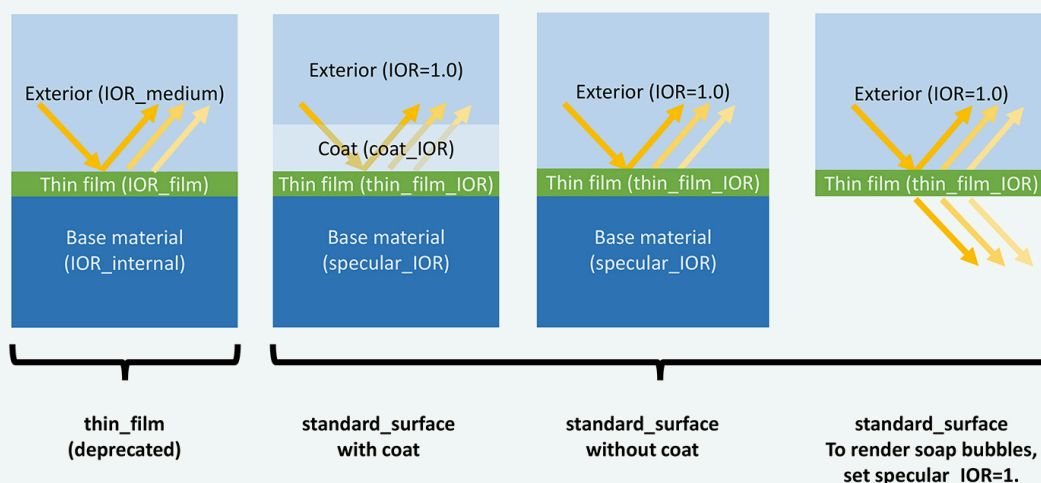


i More Information about soap bubble color and film thickness can be found [here](#).

i How Thin Film is Modeled



✓ How to Map IOR values from Thin Film to Standard Surface



Emission



Weight

Controls the amount of emitted light. It can create noise, especially if the source of indirect illumination is very small (e.g. light bulb geometry).



0 (default)



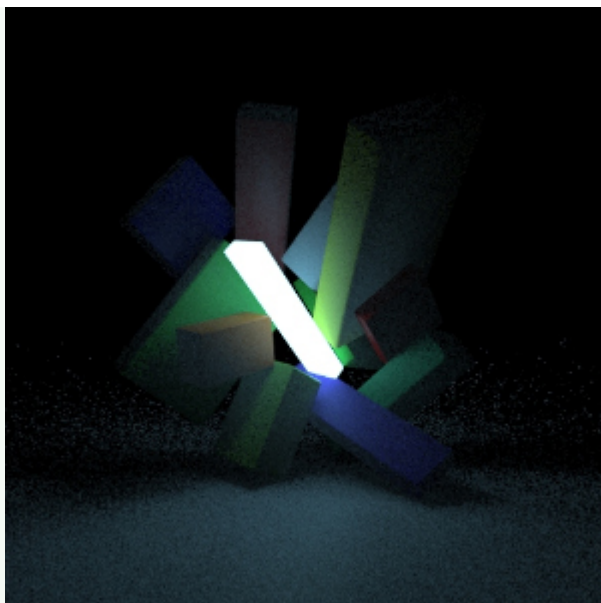
0.5



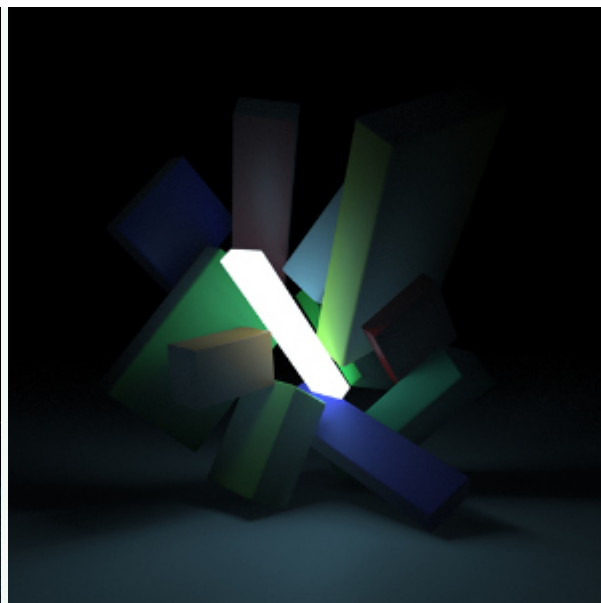
Bulb with emissive shader assigned to it (rollover image)



Increasing the number of diffuse samples will help to reduce any noise in dark, indirectly lit areas of a scene when using emission.



Diffuse Samples: 2



Diffuse Samples: 6

Color

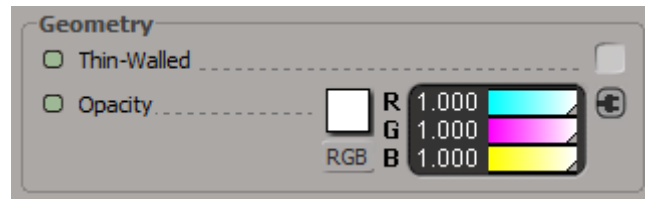
The emitted light color.



Texture map representing hot lava connected to emission_color

-
-
-
-
-
-

Geometry



Thin Walled

Thin_walled can also provide the effect of a translucent object being lit from behind (the shading point is 'lit' by the specified fraction of the light hitting the reverse of the object at that point). It is recommended that this only be used with thin objects (single-sided geometry) as objects with thickness may render incorrectly.**



Disabled



Enabled. Background behind object is not refracted.

Thin_walled is ideal for thin (single-sided) objects, like bubbles for example.



Disabled (default)

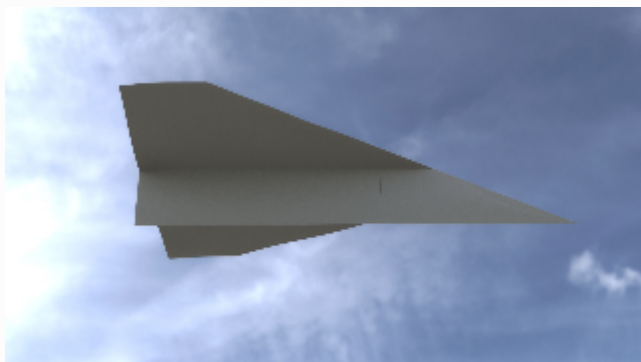
Enabled

Thin Walled Translucency

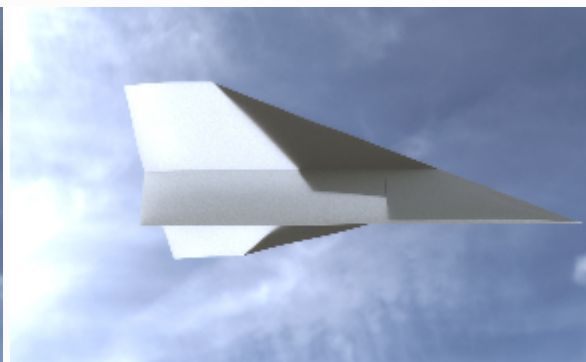
The appearance of this effect is like a thin sheet of paper letting some light through to the backside.



For diffuse surfaces such as paper, enable `thin_walled` and set `subsurface_weight` to, for example, 0.5 to have half the light reflected and half transmitted.

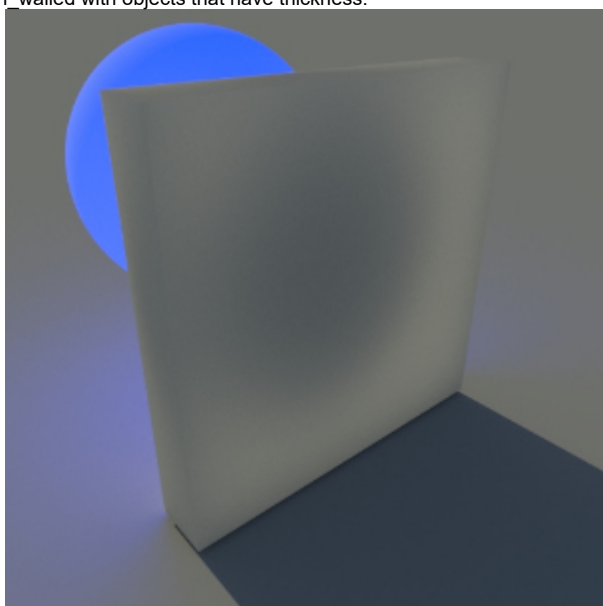


Disabled (default)

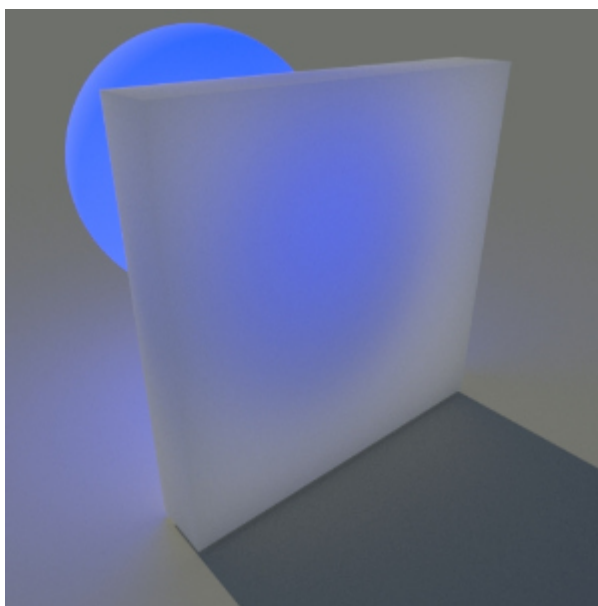


Enabled

In certain situations, `thin_walled` may work fine with thickness. However, you should ensure that the `diffuse_ray_depth` level is above 1 when using `thin_walled` with objects that have thickness.



Diffuse Ray Depth: 1 (default)



Diffuse Ray Depth: 2. Blue emission is visible.

Opacity

Controls the degree to which light is not allowed to travel through it. Unlike transmission, whereby the material still considers diffuse, specular, etc., the opacity will affect the entire shader. Useful for retaining the shadow definition of an object, while making the object itself invisible to the camera.



You must ensure that `opaque` is disabled for the mesh that the `standard_surface` shader is assigned to when using opacity.



1 (default)



0.5

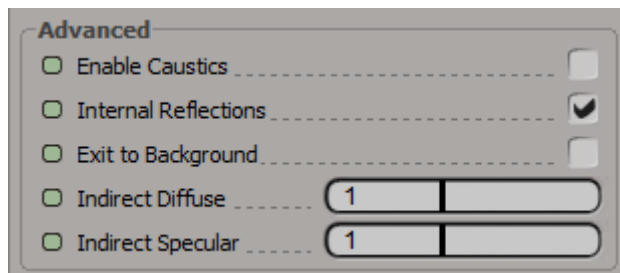


Checker map -> opacity

Anisotropy Tangent

Lets you specify a custom tangent for specular anisotropic shading. For example, if the mesh has tangent user data, that could be linked here.

Advanced



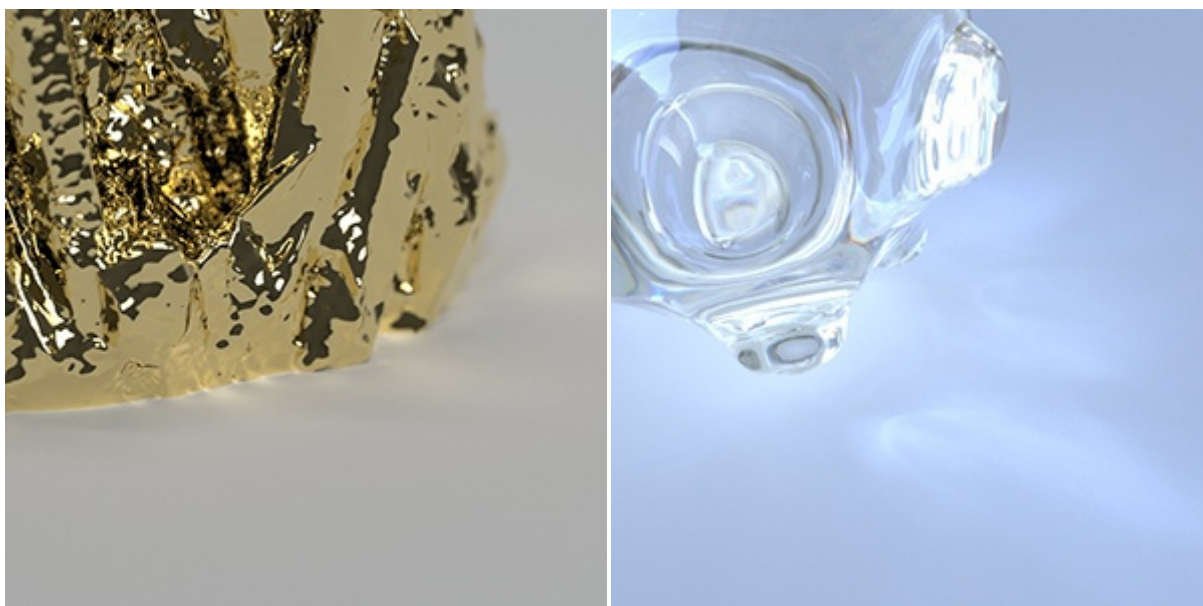
Enable Caustics

This switch in the standard_surface shader specifies whether specular or transmission bounces behind diffuse bounces are enabled or not. As caustics can be noisy, these are disabled by default.

To control noise from caustics, the global indirect_specular_blur setting may be increased, which blurs out caustics to reduce noise at the cost of accuracy.



Arnold will require a high number of diffuse samples to achieve a clean image when using caustics. Care should be taken when enabling this feature.



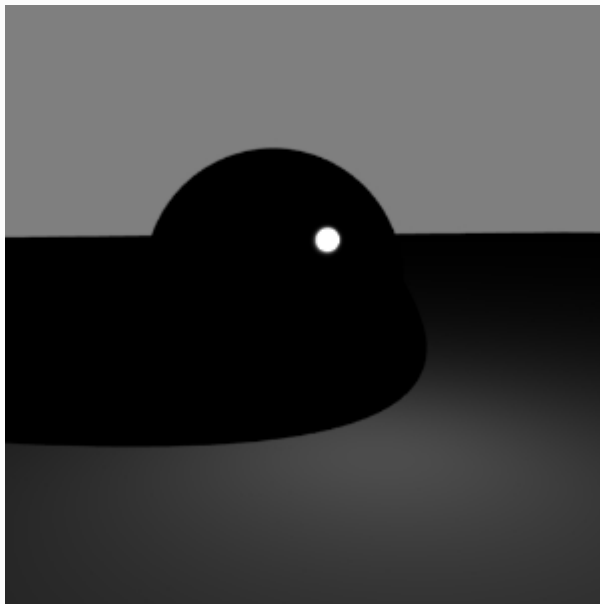
Reflective caustics (left), refractive caustics (right). Rollover images to view without Caustics.

Notice that the areas in the scenes below are darker inside the glass. This is because the illumination inside the glass is essentially all caustics. Enabling caustics for the glass shader fixes this issue. •

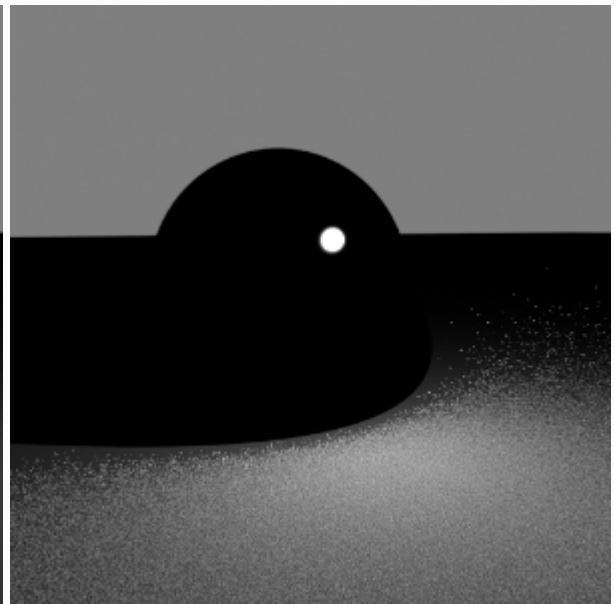


Rollover images to view with caustics enabled

i You may experience fireflies when using high specular_weight, and low Specular Roughness with Caustics enabled. To reduce this type of noise, increase the specular_roughness of the shader or indirect_specular_blur globally.



Disabled (default)



Enabled

✓ Enabling caustics can add more believability when rendering eyes.



Rollover image to view caustics enabled for outer cornea shader

Internal Reflections

Unchecking internal reflections will disable indirect specular and mirror perfect reflection computations when ray refraction depth is bigger than zero (when there has been at least one refraction ray traced in the current ray tree).

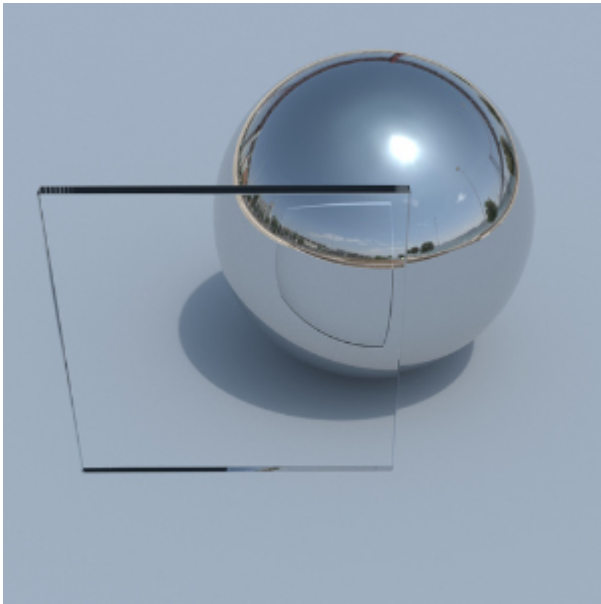


Enabled (default)

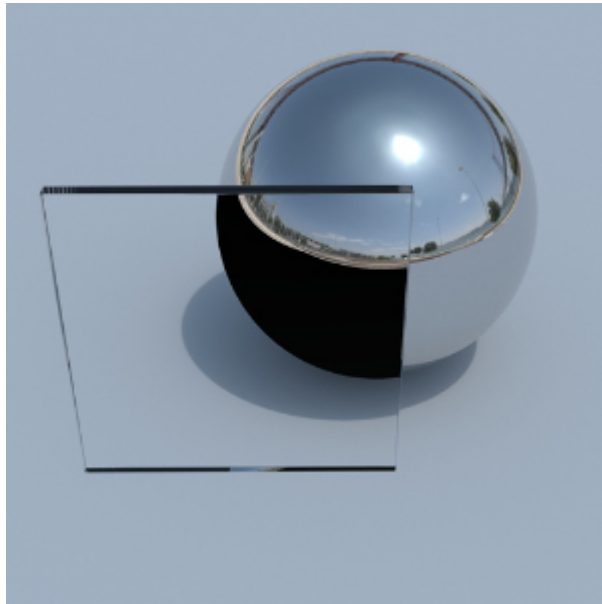


Disabled

In the right image below, the sphere appears black because `internal_reflections` is disabled in the `standard_surface` shader assigned to the sphere.



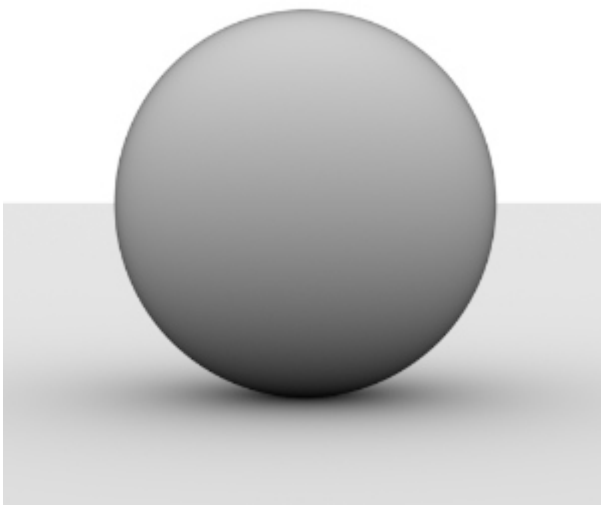
Enabled (default)



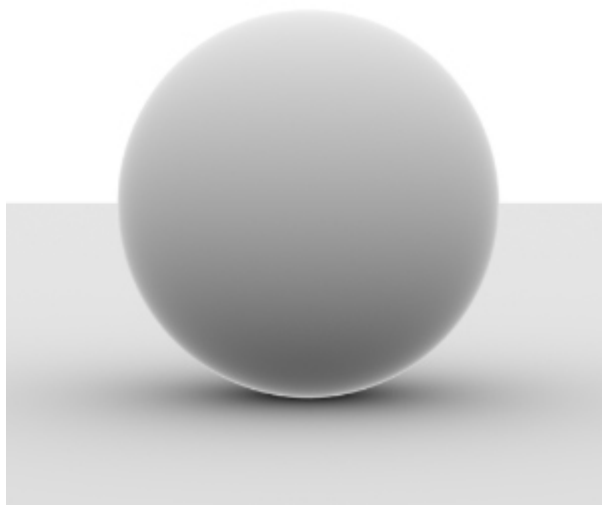
Disabled

Exit to Background

This will cause the standard_surface shader to trace a ray against the background/environment when the maximum GI reflection/refraction depth is met and return the color that is visible in the background/environment in that direction. When the option is disabled, the path is terminated instead and returns black when the maximum depth is reached.



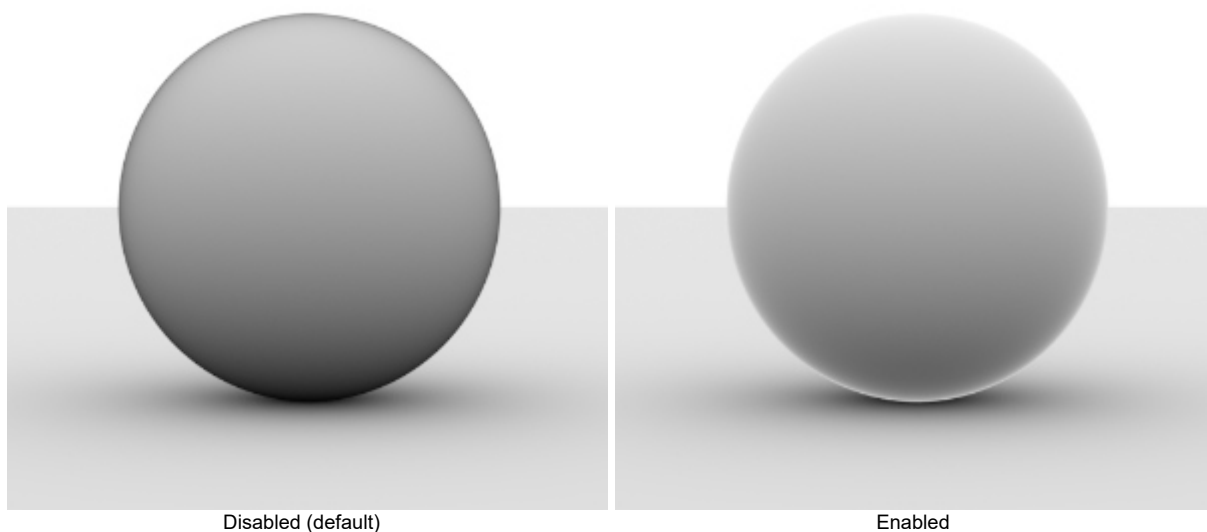
Disabled (default)



Enabled

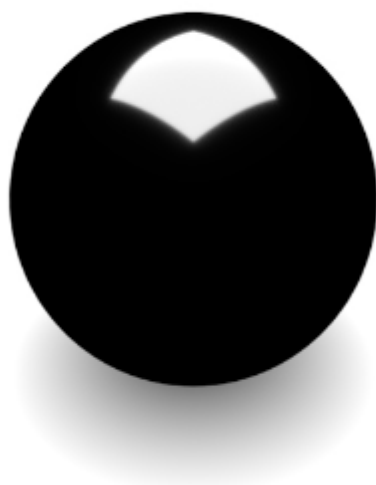
Indirect Diffuse

This will cause the standard_surface shader to trace a ray against the background/environment when the maximum GI reflection/refraction depth is met and return the color that is visible in the background/environment in that direction. When the option is disabled, the path is terminated instead and returns black when the maximum depth is reached.



Indirect Specular

The amount of specularity received from indirect sources only. •Values other than 1.0 will cause the materials to not preserve energy, and global illumination may not converge.



0



0.5

•
•

Thin Film Shader

 This shader is now deprecated and will be removed in a future release. You should instead use `thin_film` in the `standard_surface` shader.



This shader reproduces the effect of thin film interference on a surface. It can be linked to the `specular` or `reflection` color of a standard shader with the `fresnel` option disabled (the shader does its own fresnel calculations).

 The Thin Film shader is designed for the `Standard` shader, it does not work correctly with the `Standard Surface` shader at the moment.

• `MinThickness` •(nm)

The minimum thickness of the thin film, in nanometers.

`MaxThickness` •(nm)

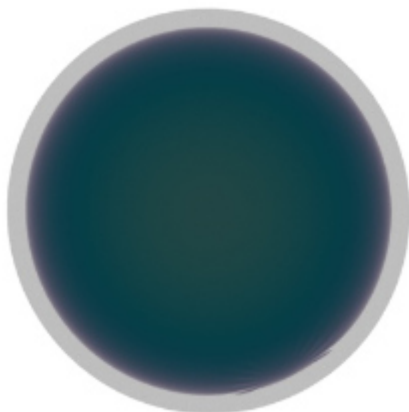
The maximum thickness of the thin film, in nanometers.

`Thickness`

Defines the actual thickness of the film between the specified min and max thickness values. Normally this would be something like a noise map to give some variation to the interference effect.

`Medium` •IOR•

The refractive index of the medium surrounding the material. Normally this is set to 1.0 for air.



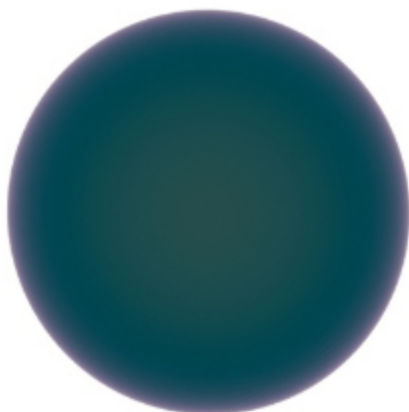
1.1



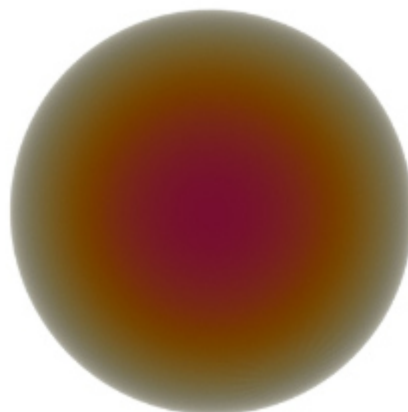
1.5

Film •IOR

The refractive index of the medium that the thin film is made of.



1.5 (default)



2

Internal •IOR

The refractive index of the medium below the thin film.



2



3

-
-
-

[Further examples](#)



Further examples

Two Sided



-

Applies two shaders on either side of a double sided surface.

Front

Input shader for front surface.

Back

Input shader for back surface.

-

Texture

Textures (image or procedural texture) can be used in a shader network of a mesh surface, as a displacement, bump or normal map, etc. •

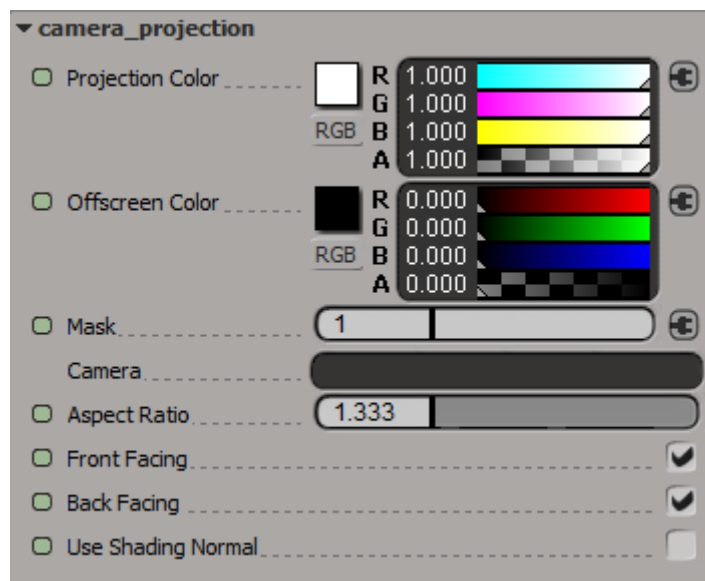
•

Below is a list of texture nodes available in SlttoA:

•

- [Camera Projection](#)
- [Checkerboard](#)
- [Image](#)
- [Noise](#)

Camera Projection



•

This shader performs camera projection of an image over an object.

Projection Color

The input (usually an image) to be projected.

Offscreen Color

The color returned outside of the projection camera viewing frustum.

Mask

Use this parameter to mask out areas that should not receive the projection.

Camera

The projecting camera. Leave blank to use the default render camera.

Aspect Ratio

The projection camera aspect ratio. By default, it is linked by expression to the selected camera's x/y resolution.

Front Facing

Apply the projection to polygons front-facing the projection camera. •

Back Facing

Apply the projection to polygons back-facing the projection camera.

Use Shading Normal

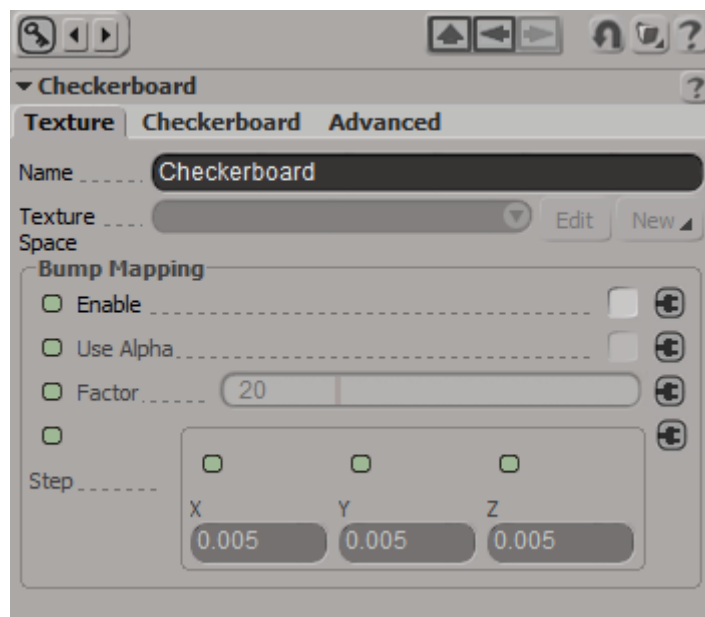
Use the shading normal for visibility tests instead of the geometric normal. This results in smoother but less accurate visibility boundaries that can produce seams if the render camera is used for the projection.

•

Checkerboard



This shader represents a checkerboard pattern.



Color 1

Color one used for the checkerboard squares.

Color 2

Color one used for the checkerboard squares.

U Frequency

Controls how many times the texture is repeated (tiled) in the U direction.

V•Frequency

Controls how many times the texture is repeated (tiled) in the V direction.

U•Offset

Offsets the checkerboard in the U direction.

V•Offset

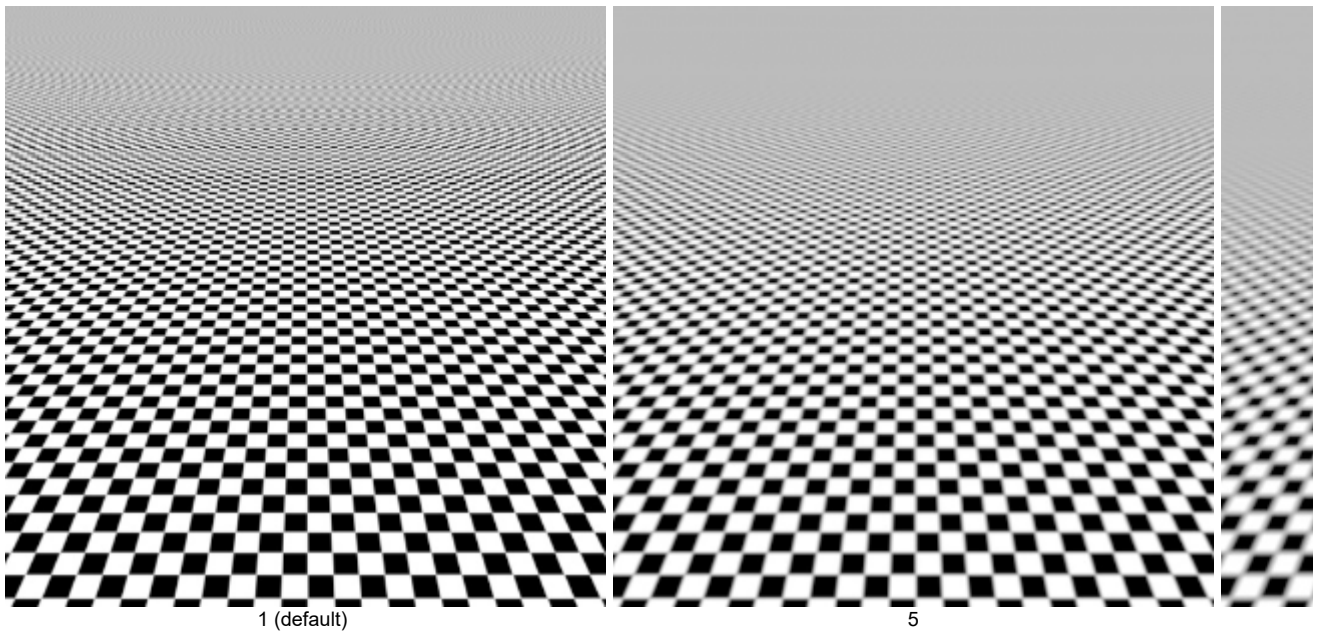
Offsets the checkerboard in the V direction.

Contrast

Contrast between the two checkerboard colors. The range is 0 (both colors are averaged) to 1. The default value is 1.

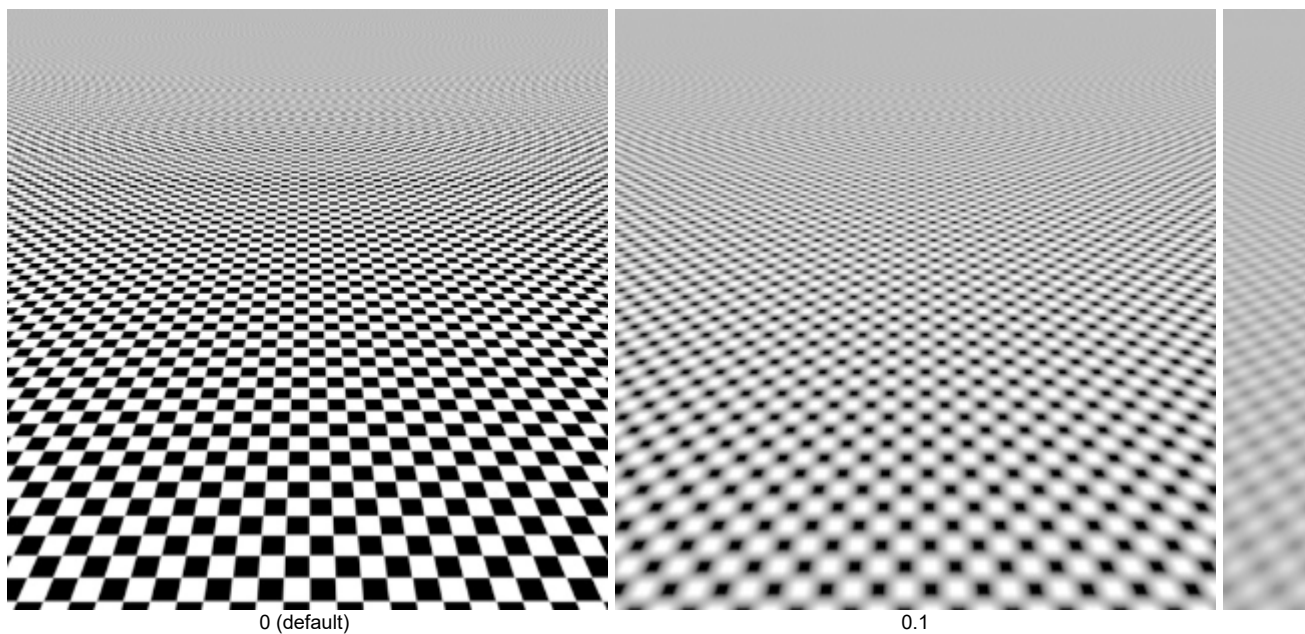
Filter Strength

Specifies the amount used to 'blur' the checkerboard.



Filter Offset

Scales the size of the filter.



UV Set

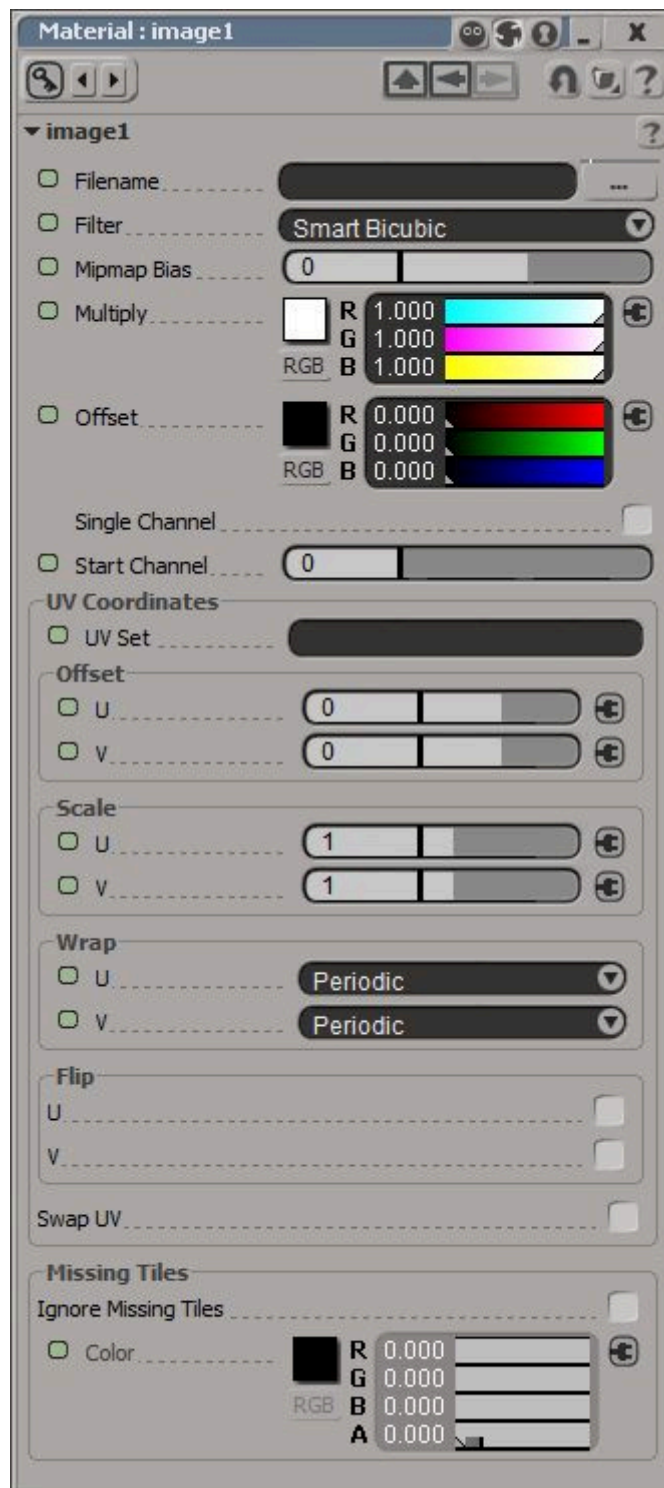
A string with the name of the UV set to use to sample the image. By default, when the `uvset` parameter is empty, the primary UV set in the polymesh will be used. Example: If you have created a UV set in a `polymesh` node called "UVset2", then you can use it by setting the `uvset` parameter to "UVset2".

-
-
-

Pepe model by Daniel M. Lara ([Pepeland](#)).

Image

The image node is a color shader that performs texture mapping using a specified image file. You can control the position, size, and rotation of this frame on the surface. You can control how the texture is tiled within the frame using the scale UV, flip, wrap, and swap attributes.



Filename

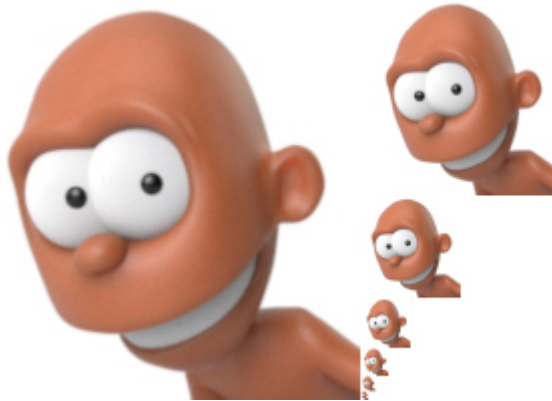
The image filename. UDIM token functionality is supported. For efficiency reasons, UDIM substitution is not supported if image.filename is linked.

Filter

Texel interpolation method used to filter the rendered image file. The options are closest, bilinear, bicubic, smart_bicubic, and the default is smart_bicubic. Cubic is slower but smoother, closest is faster but looks blocky when magnified.

Mipmap Bias

Mip-Mapping Bias offsets the computed Mip-Map level from which the image is sampled. A negative value will force a larger Mip-Map level (clearer image); a positive value will force a smaller Mip-Map level (blurrier image).



An example of mipmap image storage.

The first image on the left is shown with filtered copies reduced in size.



0 (default)

5

Multiply

Multiplies the image by a constant.



White (default)



Blue

Offset

Uniformly darkens or lightens the texture.



0 (default)



0.75

Single Channel

Read and outputs only the first channel of the image (which is usually R). You can force other channels by defining Start Channel.



Single Channel

Start Channel

This allows you to offset the lookup channel.



0 (default)



1

UV Set

A string with the name of the UV set to use to sample the image. By default, when the `uvset` parameter is empty, the primary UV set in the polymesh will be used. Example: If you have created a UV set in a `polymesh` node called "UVset2", then you can use it by setting the `uvset` parameter to "UVset2".

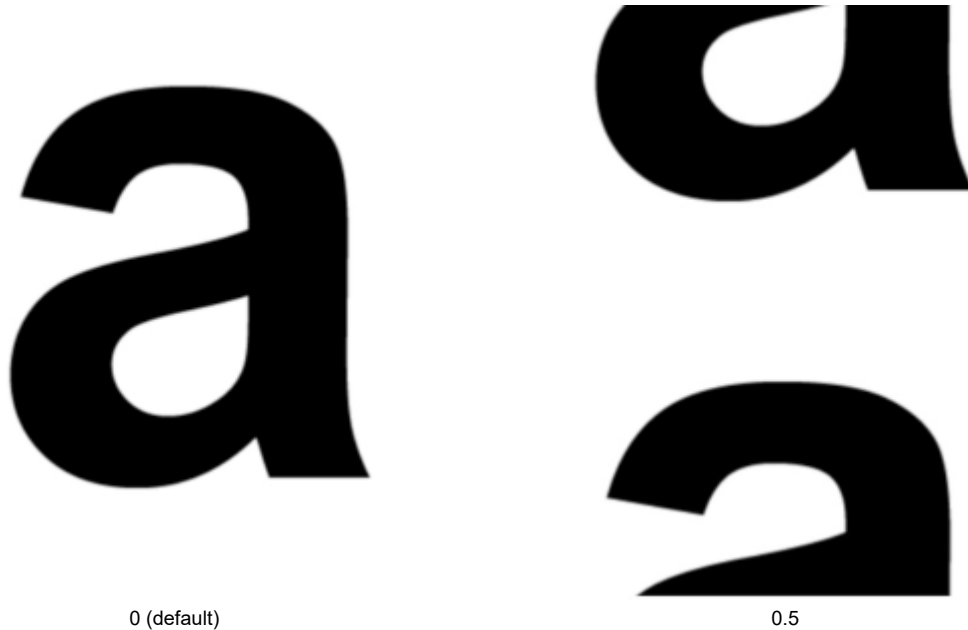
UV Coords

If uv coords are linked to a shader, the evaluation of the parameter will be used as the UV coordinates to sample the image, instead of the polymesh ones. Texture derivatives are not computed in this scenario, destroying texture mapping performance, especially in scenes with many, high-resolution textures.

•

Offset U

Offsets the image in the U direction. This offset takes place before scaling, flipping, or swapping of the S and T coordinates.



Offset V

Offsets the image in the V direction. This offset takes place before scaling, flipping, or swapping of the S and T coordinates.



Wrap U

Controls how a texture repeats in the U direction on a large surface. Choose between periodic, black, clamp, mirror and file. The default wrap mode is 'periodic'. The default wrap mode for tiles (UDIM, etc) is 'mirror'. See below.

Wrap V

Controls how a texture repeats in the V direction on a large surface. Choose between Periodic, Black, Clamp, Mirror and File. The default wrap mode is Periodic. The default wrap mode for tiles (UDIM, etc) is Mirror. The Missing wrap mode works similarly to Black, however, lookups to the image shader that are outside of the texture will use the missing_texture_color.

a a

a a a

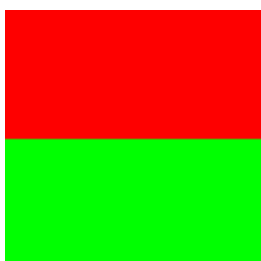


Periodic (default)

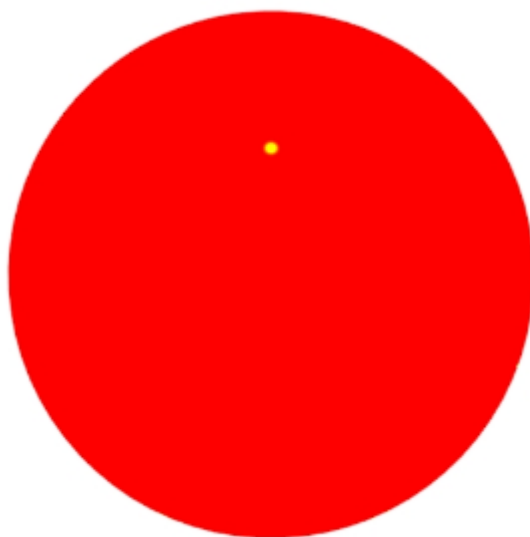
Black

File wrap mode uses the wrap setting encoded in the file itself (only applies to OpenEXR files). This is useful to preserve information about how a texture should wrap without manual correction afterward. In certain cases, 'file' mode can prevent edges. For that, to work you will need to make the correct choice when you generate mipmaps with 'maketx'.

The example below shows an OpenEXR file used as a lat-long skydome that has the file wrap modes set to 'periodic, clamp'. Generating mip maps with this mode prevents artifacts at the sphere poles. When you leave wrap modes with their default values (periodic, periodic), you can see a dot at the top of the sphere. However, if you set wrap modes to 'file' (and use a file that has the correct wrap modes in the metadata), the dot will no longer be visible.



OpenEXR file that has the file wrap modes set to 'periodic, clamp'



Wrap: Periodic (artifact visible at pole of sphere)



Wrap: File (artifact is gone)

Scale U

Scales the image.

Scale V

Scales the image.

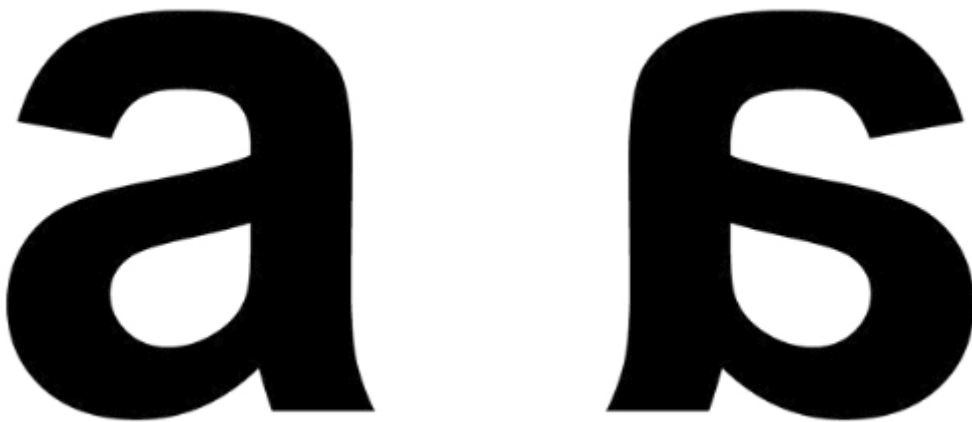


Scale U: 1, Scale V: 1 (default)

Scale U: 2, Scale V: 2

Flip U

Flip (mirror) the image in the horizontal direction.



Disabled (default)

Enabled

Flip V

Flip (mirror) the image in the vertical direction.



Disabled (default)



Enabled

Swap UV

Swaps the axes.



Disabled (default)



Enabled

Ignore Missing Tiles

MultiExcerpt named 'ignore_missing_tiles ' was not found

The page: image was found, but the multiexcerpt named 'ignore_missing_tiles ' was not found. Please check/update the page name used in the 'multiexcerpt-include macro.

Missing Tile Color

MultiExcerpt named 'missing_tile_color ' was not found

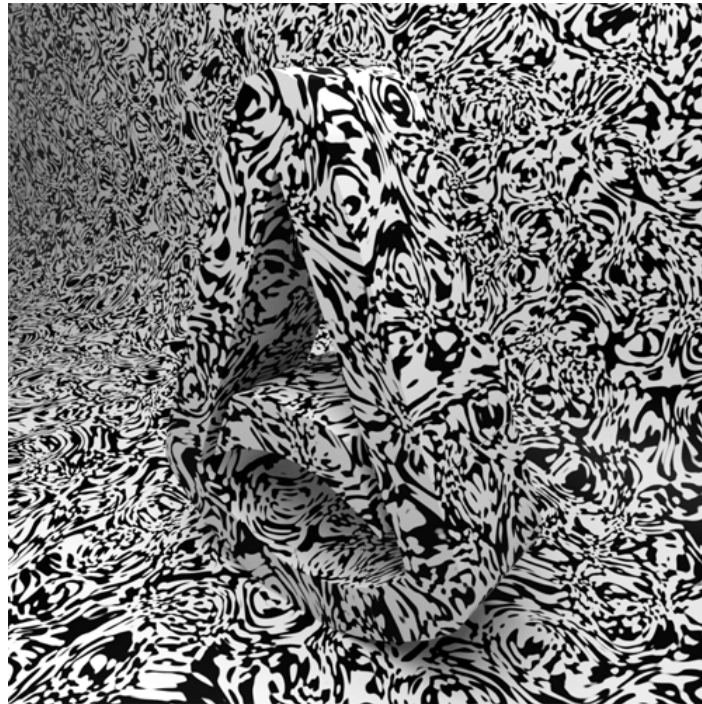
The page: image was found, but the multiexcerpt named 'missing_tile_color ' was not found. Please check/update the page name used in the 'multiexcerpt-include macro.



-
-
-

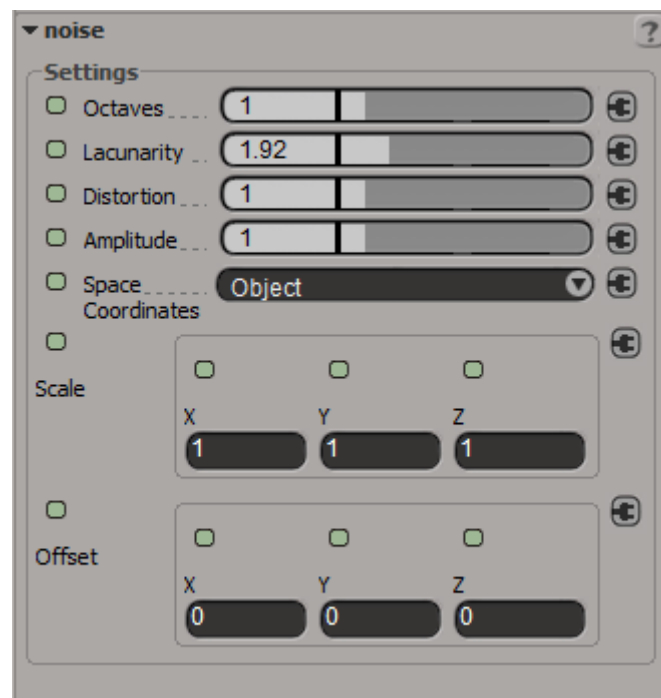
Pepe model by Daniel M. Lara (Pepeland)

Noise



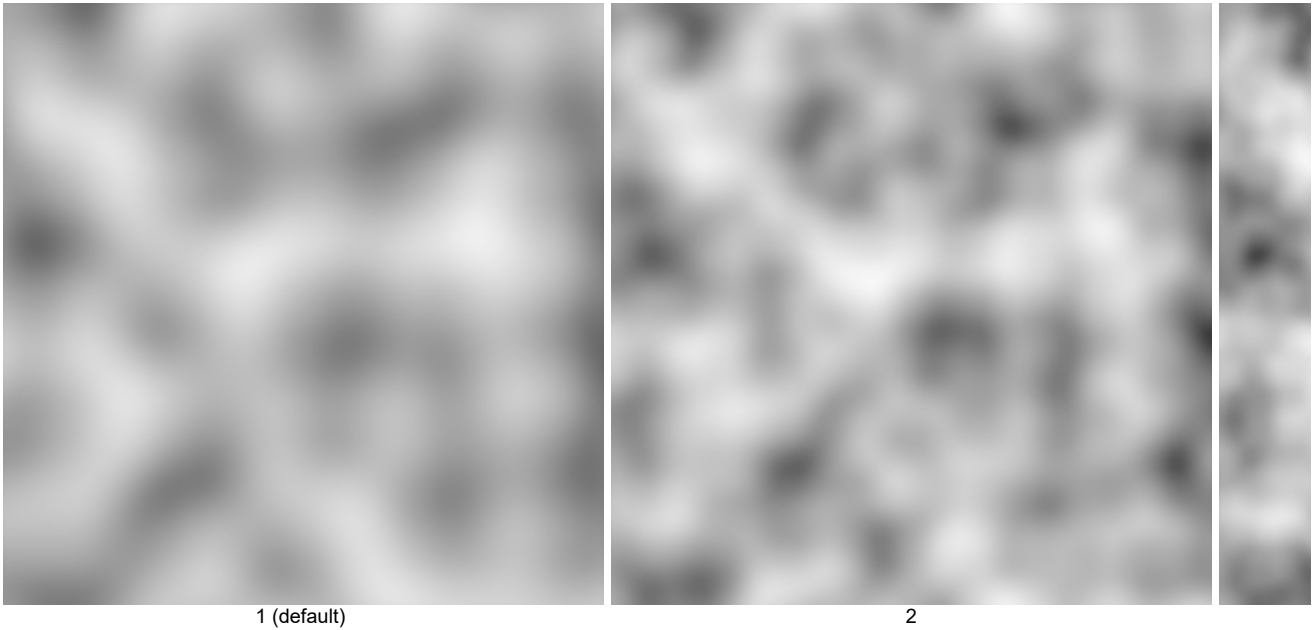
This is a shader that evaluates a coherent noise function (potentially summed over several octaves, giving fractal noise, sometimes known as a turbulence function). The output is between 0 to 1 (actually $[0,1)$), which can be used as input to several other shader nodes to produce various effects. This is useful for creating many real-world patterns (marble, granite, leather, etc.)

Settings



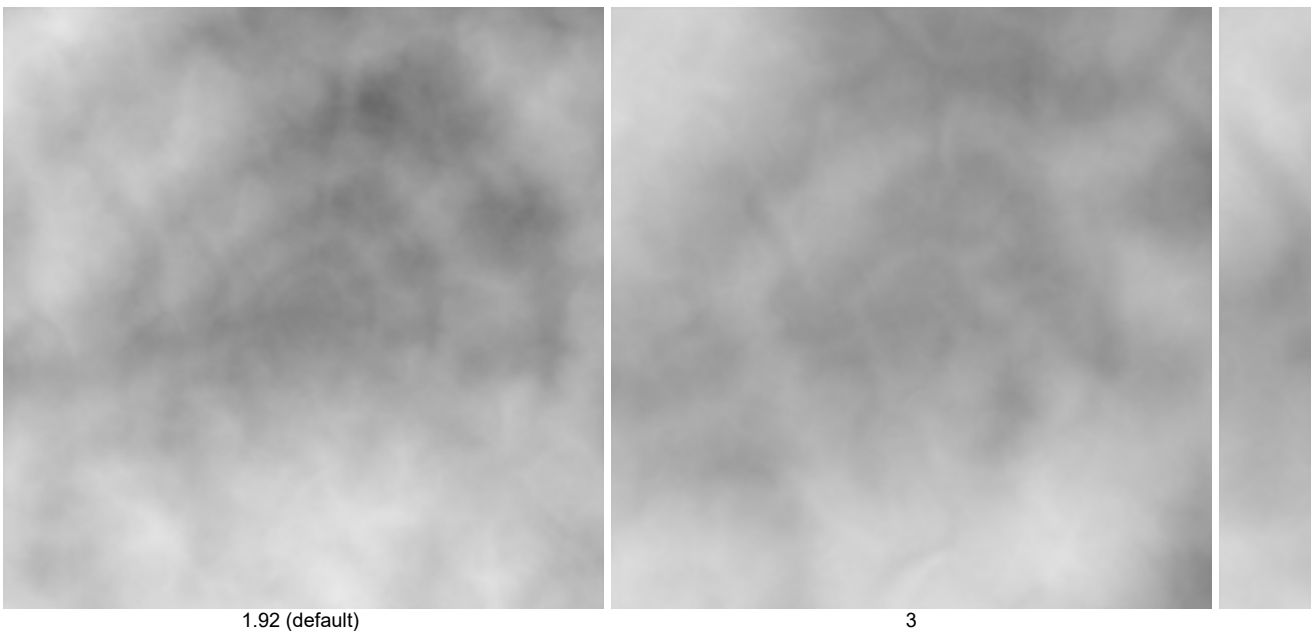
Octaves

The number of octaves over which the noise function is calculated (the fractal noise function is repeated at multiple frequencies, known as octaves; normally each octave is at about twice the frequency, i.e., half the size, of the previous one, but you can alter this with the lacunarity control).



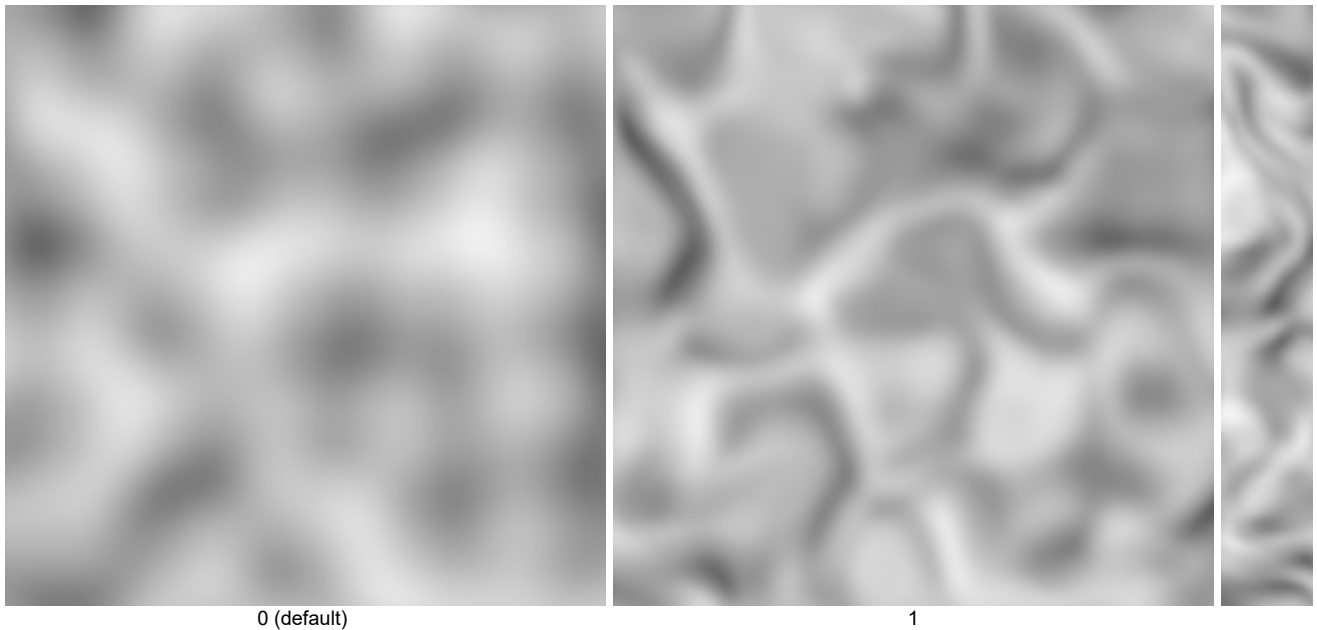
Lacunarity

Controls the average size of gaps in the texture pattern produced. Lacunarity means the change in scale between each octave - in musical scales this would be 2.0, meaning each octave is at twice the frequency/half the scale of the previous one, but for our purposes numbers that are near 2.0, but not actually 2.0 are best. In most cases, we recommend using the default value of 1.92.



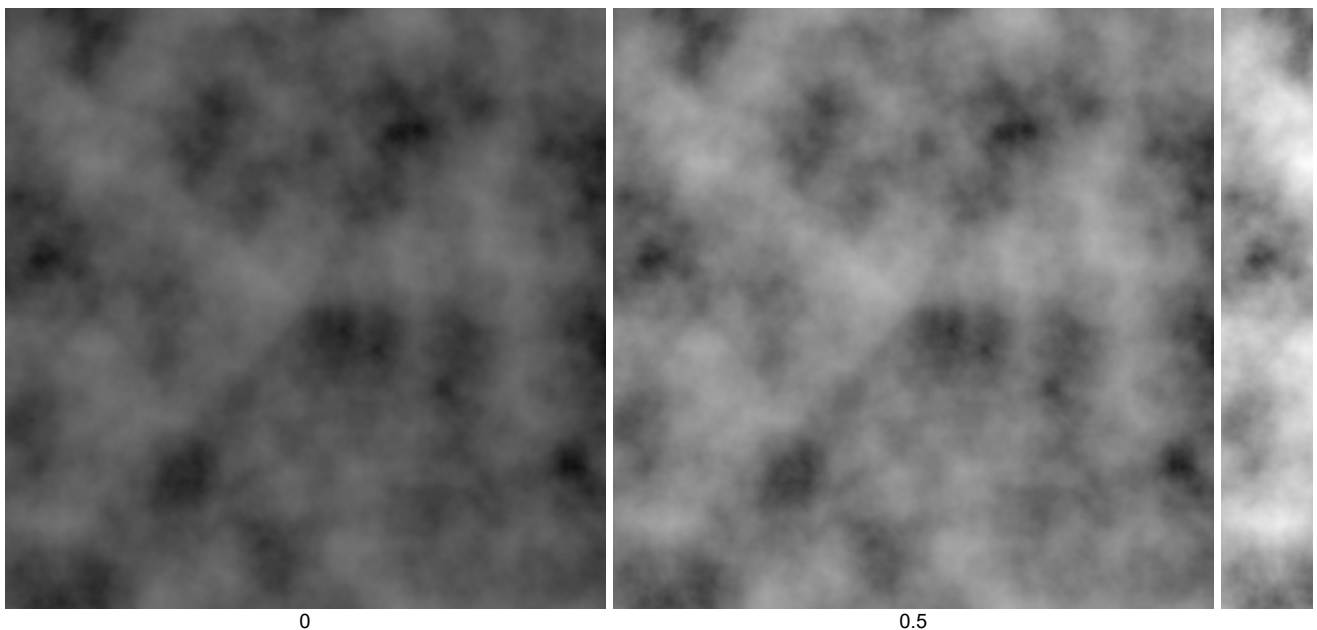
Distortion

Defines a degree of random displacement applied to each point as part of the noise calculation, giving a different aesthetic feel.



Amplitude

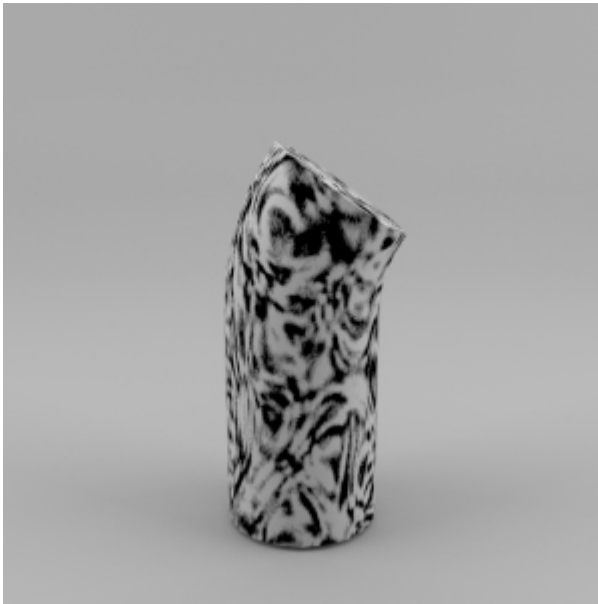
Controls the amplitude, or range, of the output. Normally the output has values between 0 and 1; the amplitude control multiplies this.



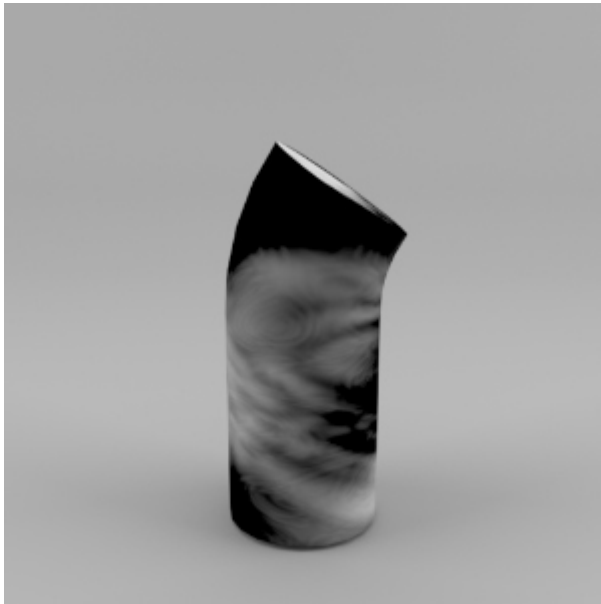
Space Coordinates

Specifies the coordinate space to use. These include World, Object, Pref, and UV space coordinates. •Pref is short for 'vertex in reference pose'. The plugin can pass these vertices to Arnold (in addition to the regular, deformed vertices) which can, in turn, be queried by the noise shader so that the noise 'sticks' to the reference pose and does not swim as the mesh deforms.

- Object space, where points are expressed relative to the local origin (center) of the object.
- World space, where points are relative to the global origin of the scene.
- Pref, which isn't really a space, but rather a reference to a bind pose (note Pref does not work with NURBS surfaces).
- UV, for texturing using the object's local UV coordinates. Note that this calls the faster 2D noise API, not the 3D noise like all other coordinate spaces.



Coord Space: Object (default)



Coord Space: UV

Below are animations that show `pref_space` coordinates in use with a deforming object. Note how the texture sticks to the object when using Pref space coordinates, compared to Object/World space coordinates where the texture swims.



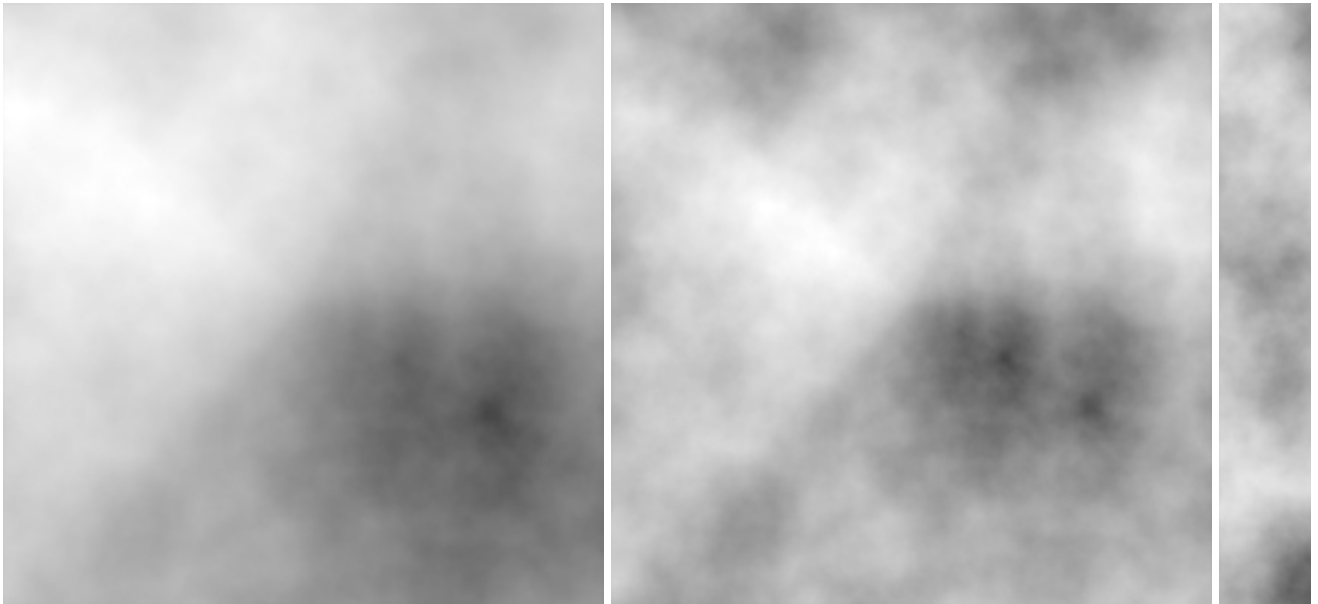
Using World/Object Space - Texture Swims



Using Pref Space - Fixes Texture Swimming

Scale

Controls the scale of the noise function in x, y, and z directions.

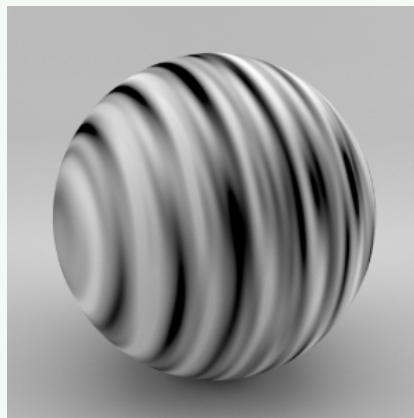


1 (default)

2



Scaling the noise in one direction only can produce interesting results.

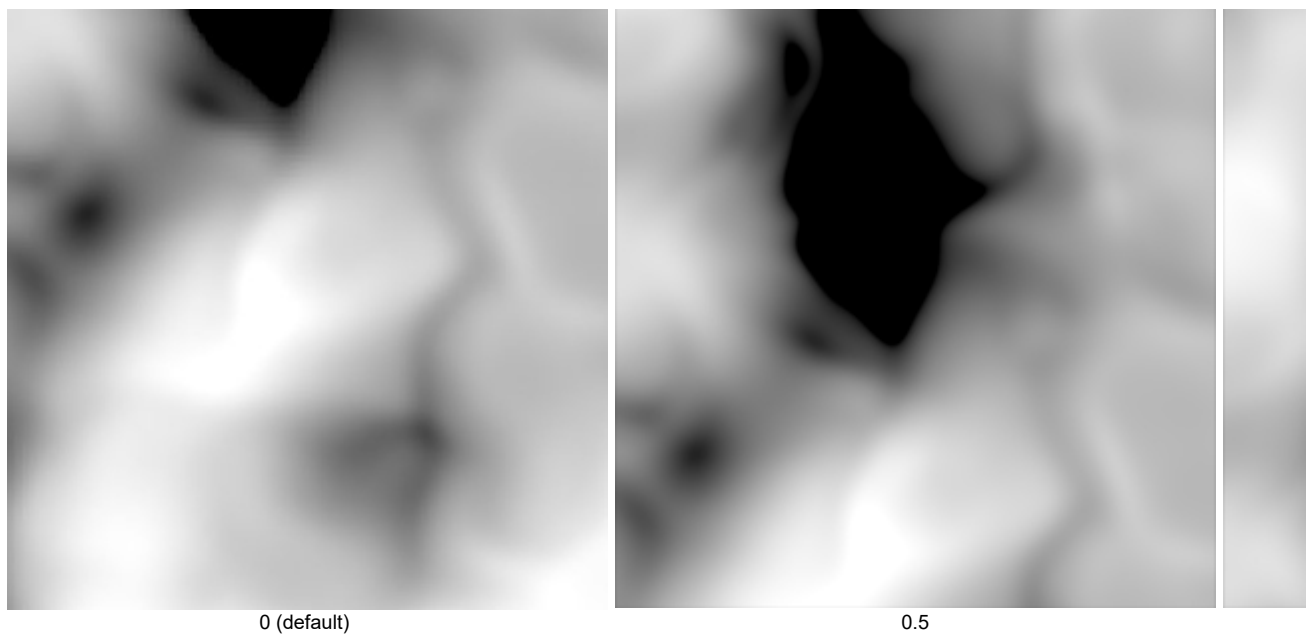


Scale X:1, Y:0.1, Z:0.1

Offset

-

Offset the noise in x, y, or z directions.



-
-

Utility Shaders

Below is a list of available Utility shaders in SltoA:

•

- [Blackbody](#)
- [Cache](#)
- [Clamp](#)
- [Curvature](#)
- [Facing Ratio](#)
- [Flat](#)
- [Motion Vector](#)
- [Passthrough](#)
- [Ray Switch](#)
- [Ray Switch RGBA](#)
- [Switch](#)
- [Trace Set](#)
- [Utility](#)
- [UV Transform](#)
- [Wireframe](#)

Blackbody



blackbody (temperature 0-3000) -> emission_color

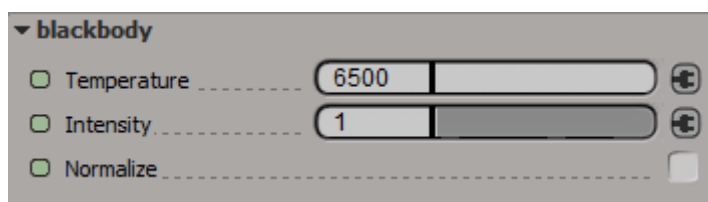
The blackbody shader emits a color based on color temperature. It is useful for fire and explosions, or for light emission using the blackbody spectrum. For volumes, a volume sample shader reading from a temperature channel should be connected to the blackbody shader.



The blackbody shader is a copy of the Houdini Pyro shader and is thus tailored for the output of pyro simulations.



You must ensure that a volume sample shader is connected to the blackbody shader.



Temperature

The temperature corresponding to an input value of 1. For example, if this parameter is 6500 (the default 6500K, is considered as the "daylight" white point), then an input value of 1 means a temperature of 6500 degrees Kelvin. The color ranges from red, through to white and then to blue. Values above 6500 will give a cool color, whilst values below will show a warm color.

Intensity

This controls the intensity of the blackbody emission. For physically correct results intensity 1 must be used. This can result in extremely bright light, however, and lower values may be used to reduce the intensity.

Normalize

By enabling normalize, the intensity is discarded, and a normalized color is output instead. This can be used to control the intensity separately from the color.

Cache



.

The cache shader can be used to avoid re-evaluating costly parts of the shading network with diamond connections. As that part of the network won't be calculated twice for the same shading point it will speed up renders, especially those with volumes.

To use the cached result, the evaluations must share the same:

- shading point
- shading normal
- ray origin
- UVs
- geometry node

Clamp



Clamps the input between min and max.

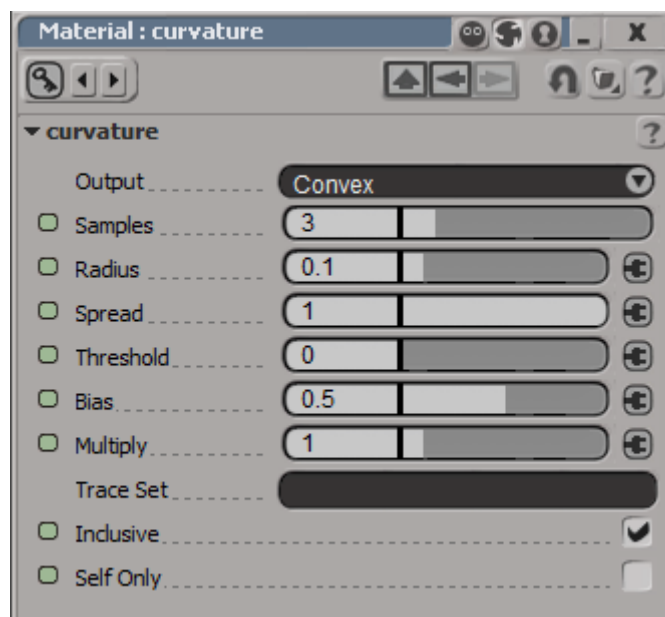
Curvature



Rollover image •

This shader will sample around the shading point within a given radius to output the curvature. You can adjust the radius, falloff, and spread of the curvature sampling as well as specify a trace set to exclude or include objects. This shader is useful for creating procedural wear or dirt maps in conjunction with a noise shader.

•



Output

Whether to output the convex (positive) curvature or concave (negative) curvature only as grayscale, or output both with the convex curvature in the red channel and concave curvature in the green channel.



Convex (default)



Concave

Samples

Controls the number of rays that will be fired to compute the curvature. Increasing the number of samples will reduce noise and give a better quality result. The actual number of rays is the square of this value.



3 (default)



6

Radius

The radius of the sphere around the shading point within which the curvature will be estimated. Smaller values show finer details, while larger values give a smoother result.



0.01



0.05

Spread

Lower values of spread will shoot rays more vertically with respect to the surface, and the default value of 1 means that rays are shot in all directions. This acts as a threshold to remove unwanted small variations of the curvature.*



0.85



0.9

Threshold

Consider only curvature above this normalized angle, with (0, 1) mapping to (0°, 180°). This parameter is useful to prune geometry "bubbles" behind the surface for convex curvature.

Bias

The bias will affect the falloff of the curvature. With the default of 0.5, the curvature sample importance decreases linearly with the distance to the shading point. Higher values will cover more and exaggerate the curvature, whereas smaller values will emphasize small details.



0.001



0.002

Multiply

Multiplies the curvature by this factor. Note that before this multiplication the output values range from 0 to 1, corresponding to angles in (0°, 90°).



0.5



1 (default)

Trace Set

For fine-grained control of which geometry should be hit or avoided when sampling curvature, you can tag objects to be in a trace set. Put an existing trace set in this parameter to make the probe ray avoid or hit corresponding tagged geometry. You can control the avoidance behavior with the Inclusive parameter.

Inclusive

When a trace set is exclusive, rays are traced against all geometry except the tagged nodes. When a trace set is inclusive, rays are traced against tagged nodes, but also against nodes that are not tagged at all.



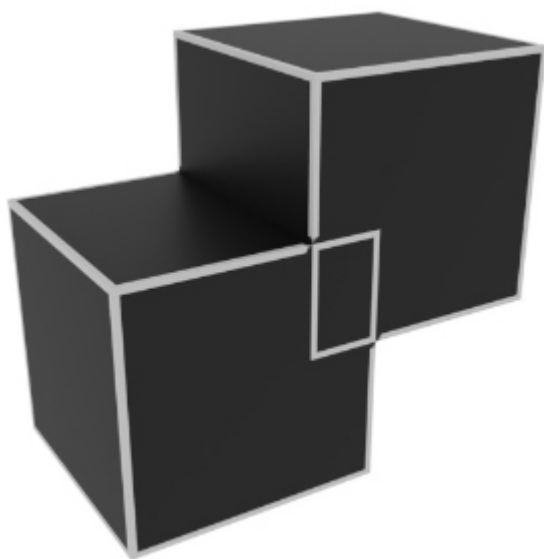
Veins included (default)



Veins excluded

Self Only

Limits curvature sampling to the object being shaded.



Disabled (default)



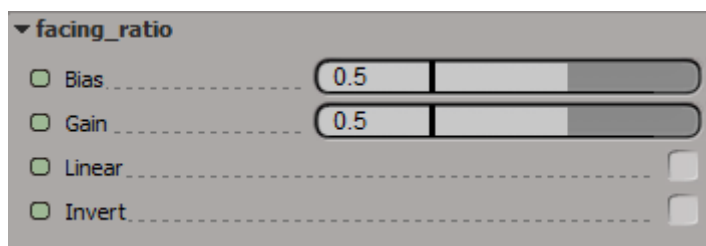
Enabled

Facing Ratio



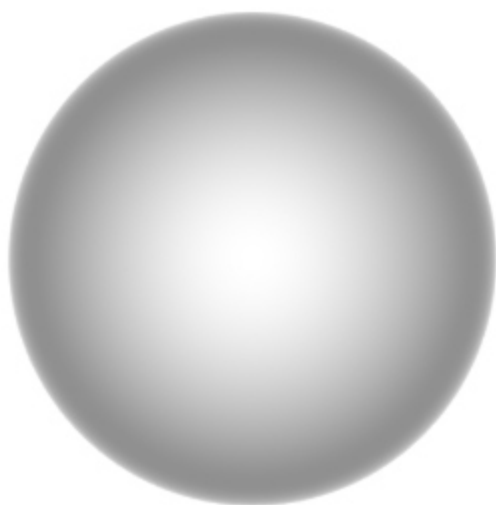
Tinted car paint effect using `facing_ratio` (rollover image)

This shader returns the absolute value of the dot product between the shading normal and the incoming ray direction. It is also named `incidence` in other renderers. Unlike the `utility` shader used in `ndoteye` mode, `facing_ratio` works for any type of ray, not just camera rays. The returned values are always in the `[0..1]` range.

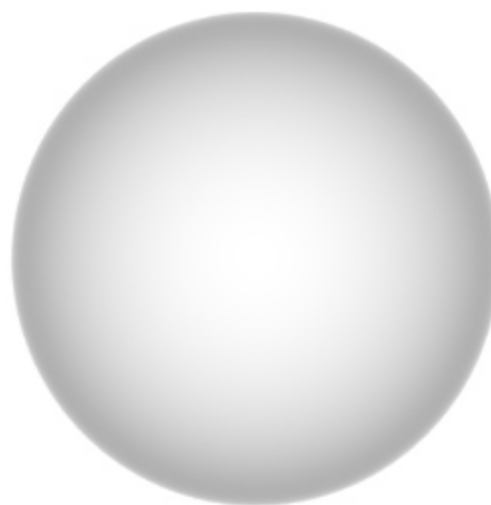


Bias

Push or pull values by altering the slope at the beginning of the range. Bias values below 0.5 decrease the slope and lower values overall. Above 0.5, the slope is higher, and the value grows more quickly. A value of 0.5 has no effect.



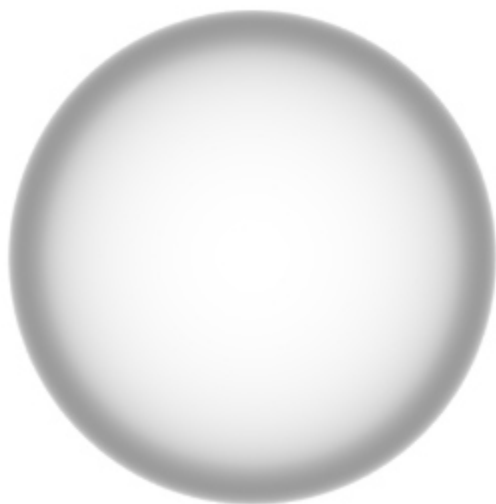
0.25



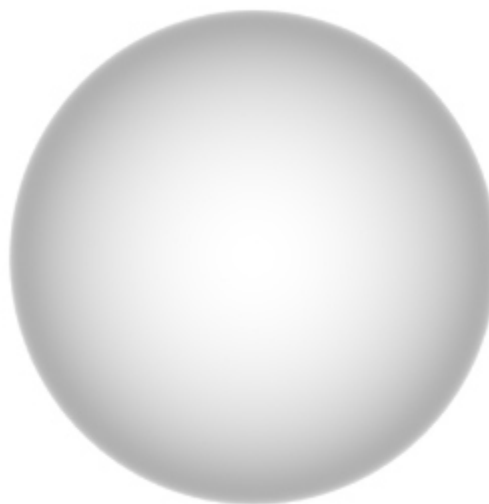
0.5 (default)

Gain

Increase or decrease the slope of the mid-range values. Gain values below 0.5 increase the contrast whereas values above 0.5 flatten the mid-range values. A value of 0.5 has no effect.



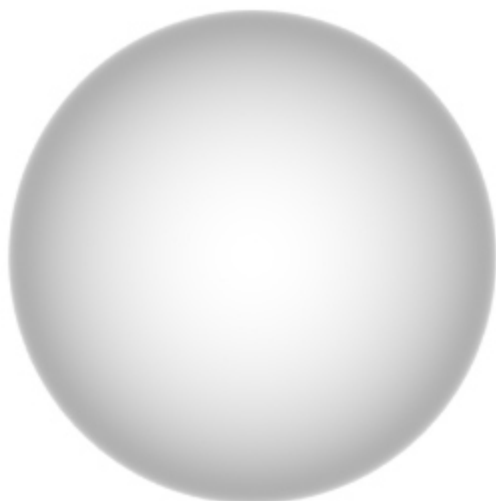
0.25



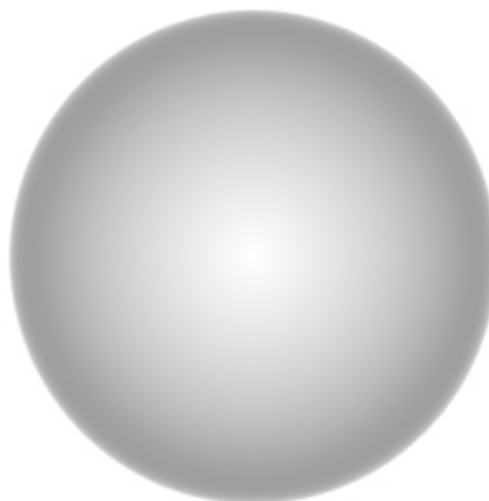
0.5 (default)

Linear

When enabled, the `facing_ratio` shader will output a linear, normalized angle instead of N·Eye.



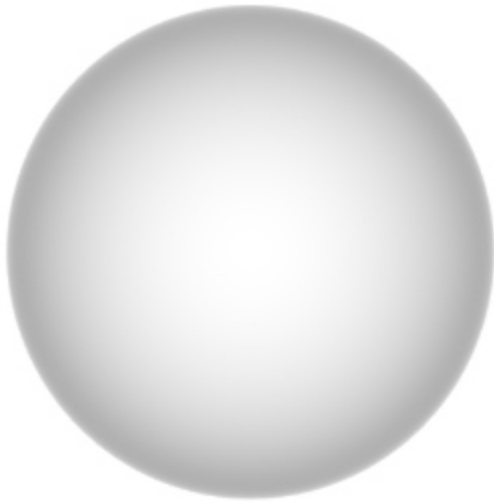
Disabled (default)



Enabled

Invert

Return the complement of the result ($1 - x$).



Disabled (default)



Enabled

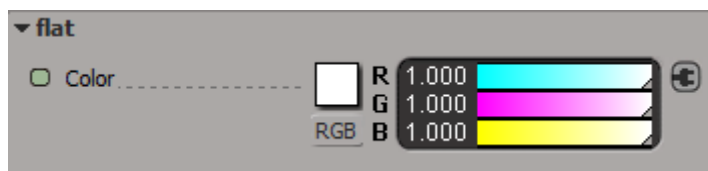
-
-

Flat

•



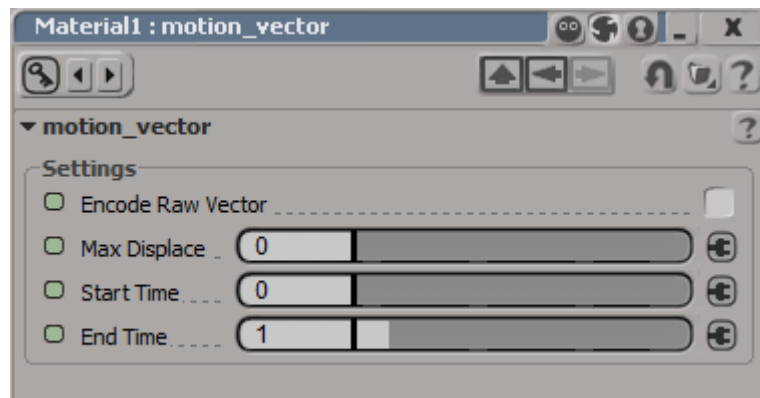
A simple color shader node which just allows a color with no other effects.



Color

The input color. •

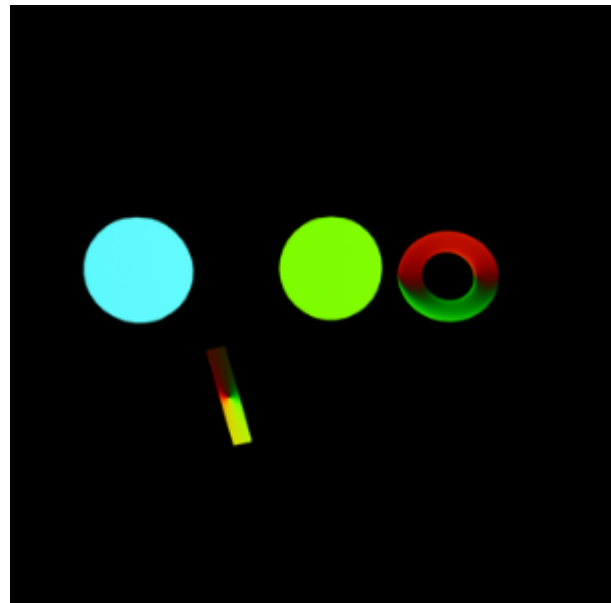
Motion Vector



This shader will encode a vector representing the motion of the object in the Red and Green components. The following two images illustrate the effect. The first image shows the original scene with motion blur rendered normally. The second shows the false-color effect of the motion vector.

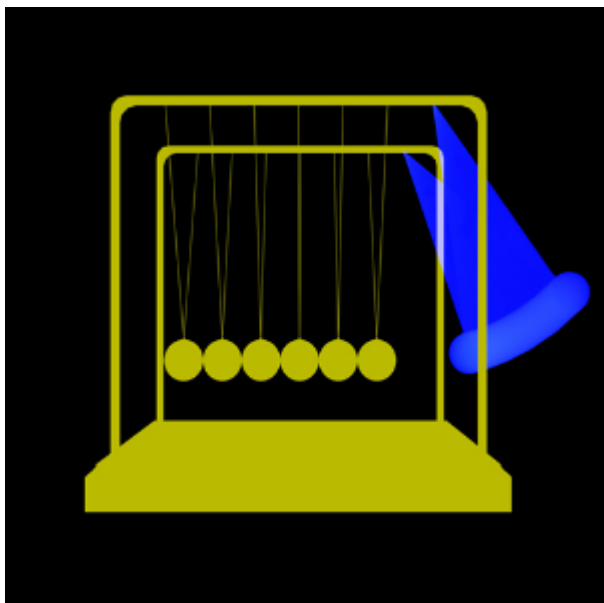


Original scene with 3D motion blur.

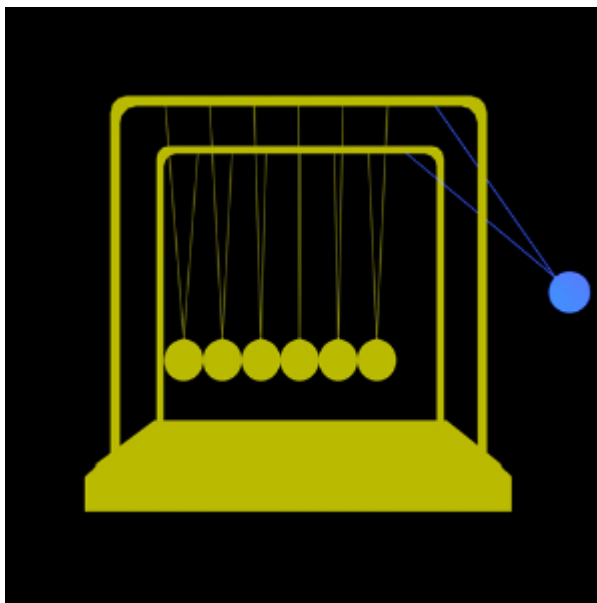


Using the motion_vector shader. The left sphere has a max_displace of 0 (hence blue). Other objects have max_displace of 1.

i To use this shader correctly, you must enable instantaneous_shutter For Motion Vectors (Motion Blur settings).



Instantaneous Shutter For Motion Vectors: Disabled (incorrect)



Instantaneous Shutter For Motion Vectors: Enabled (correct)

Camera Shutter

MultiExcerpt named 'notes ' was not found

The page: motion_vector was found, but the multiexcerpt named 'notes ' was not found. Please check/update the page name used in the 'multiexcerpt-include macro.



You must add the shutter_start and shutter_end via an Arnold user options property.

Encode Raw Vector

The vector won't be normalized.

Max Distance

When greater than 0, this is the maximum amount of screen-space motion. The motion vector encoded in the RG components is then normalized to this value. This is better when writing to 8 or 16-bit formats like JPEG, TIFF, etc. On the other hand, when this value is less than or equal to zero, magnitude encoding is used instead: RG holds the unit direction vector, and B holds the magnitude. This is better when writing to floating-point formats (EXR etc.).

Start Time

Sample time for shutter start.

End Time

Sample time for shutter end.



Passthrough

Evaluate any shading network connected to the `eval[1-20]` inputs. The `passthrough` input is evaluated first and is passed as-is to the output. This shader is useful to force the evaluation of parallel shading networks that are not part of the beauty pass, whose resulting value you need to store in an AOV for example.

-

Ray Switch

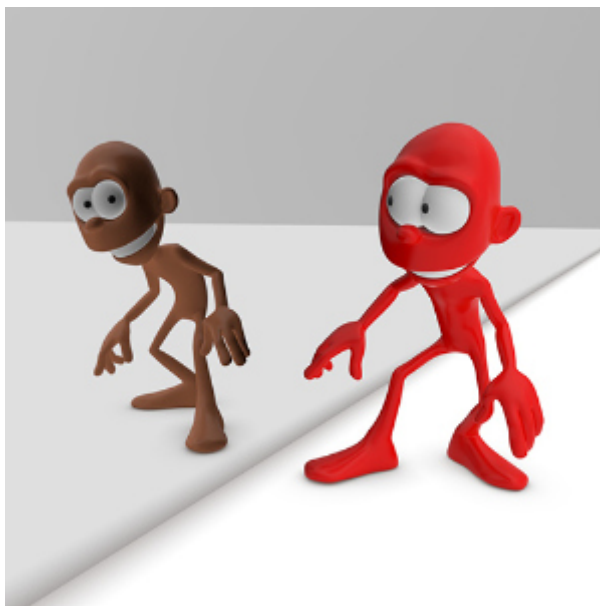
This shader makes it possible to evaluate different shader trees per ray. This decreases the shading complexity of a scene and thus the render times, and increases artistic control. It can be used to remove unnecessary secondary rays (specular/sss), make speculars even more glossy in specular rays, control the color of opacity in shadow rays to fake light scattering through tissue or add a second specular lobe in-camera rays only.

 There is also the [Ray Switch RGBA](#) shader for adjusting colors instead of shaders.

 It is also possible to connect the `Ray Switch` shader to the `Background Environment`.

Camera...

Plug the output of the shader you wish to use when calculating camera rays here.



Red standard_surface shader connected to camera attribute of ray_switch_shader

Shadow

The shader evaluation that happens for transparent shadows on objects. A use for this parameter could be to connect a `ray_switch` shader to the opacity parameter of a `standard_surface` shader. That way you can get a shadow that's different than the actual transparency of the object. For example, to reduce the amount of shadow cast by the object, or to use a different cutout opacity pattern.



Body object's standard_surface shader has red transmission_color (opaque is disabled)

Diffuse Reflection ...

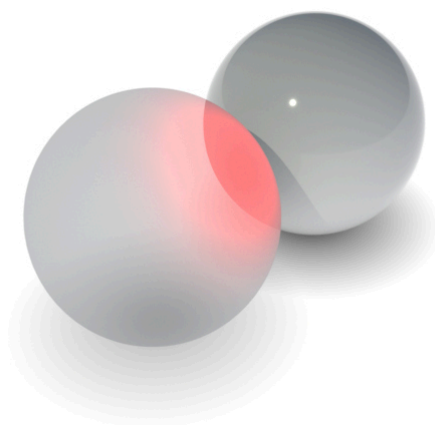
Plug the output of the shader you wish to use when calculating diffuse reflection rays here.



Red emissive shader -> diffuse_reflection of ray_switch_shader

Diffuse Transmission

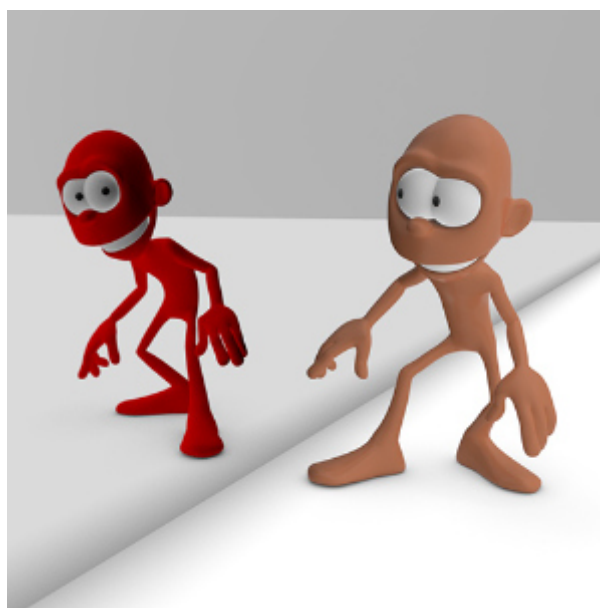
Plug the output of the shader you wish to use when calculating diffuse_transmission rays here.



Red emissive shader -> diffuse_transmission of ray_switch_shader

Specular Reflection

Plug the output of the shader you wish to use when calculating glossy rays here. •



Red standard_surface shader -> specular_reflection of ray_switch_shader

Specular Transmission

Plug the output of the shader you wish to use when calculating specular_transmission rays here. •



Red standard_surface shader -> specular_transmission of ray_switch_shader

Volume

Plug the output of the shader you wish to use when calculating volume rays here.*

-
-
-
-

Pepe model by Daniel M. Lara*[\(Pepeland\)](#)

Ray Switch RGBA



This shader makes it possible to evaluate different color trees per ray.



There is also the [Ray Switch](#) shader for connecting and adjusting shaders instead of colors.

Camera

Plug the output of the color you wish to use when calculating camera rays here. •

Shadow

Plug the output of the color you wish to use when calculating shadow rays here. • The shader evaluation that happens for transparent shadows on objects.

Diffuse Reflection

Plug the output of the color you wish to use when calculating diffuse_reflection rays here. •

Diffuse Transmission

Plug the output of the color you wish to use when calculating diffuse_transmission rays here.

Specular Reflection

Plug the output of the color you wish to use when calculating specular_reflection rays here.

Specular Transmission

Plug the output of the color you wish to use when calculating specular_transmission rays here.

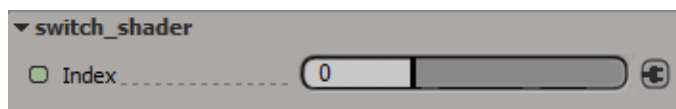
Volume



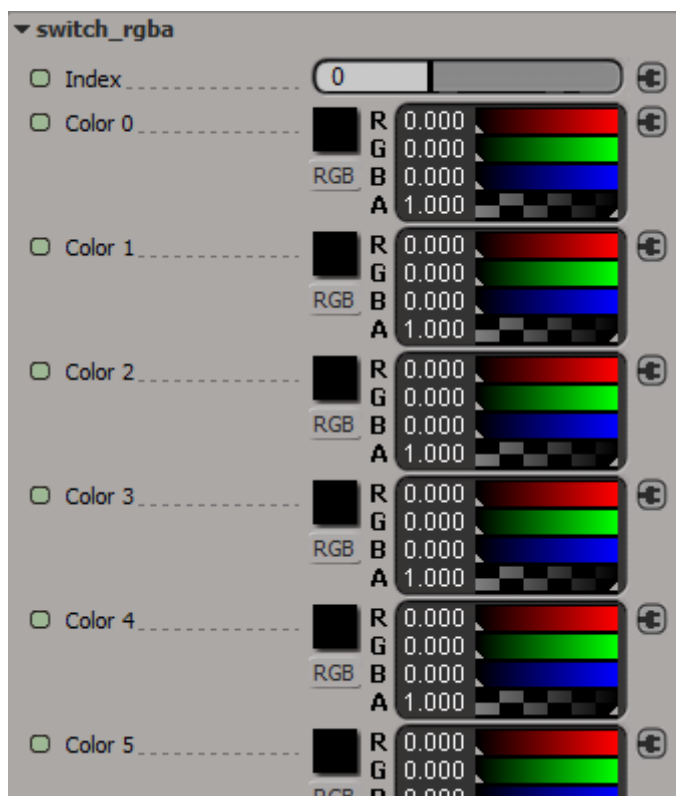
Plug the output of the shader you wish to use when calculating volume rays here. •

-
-

Switch



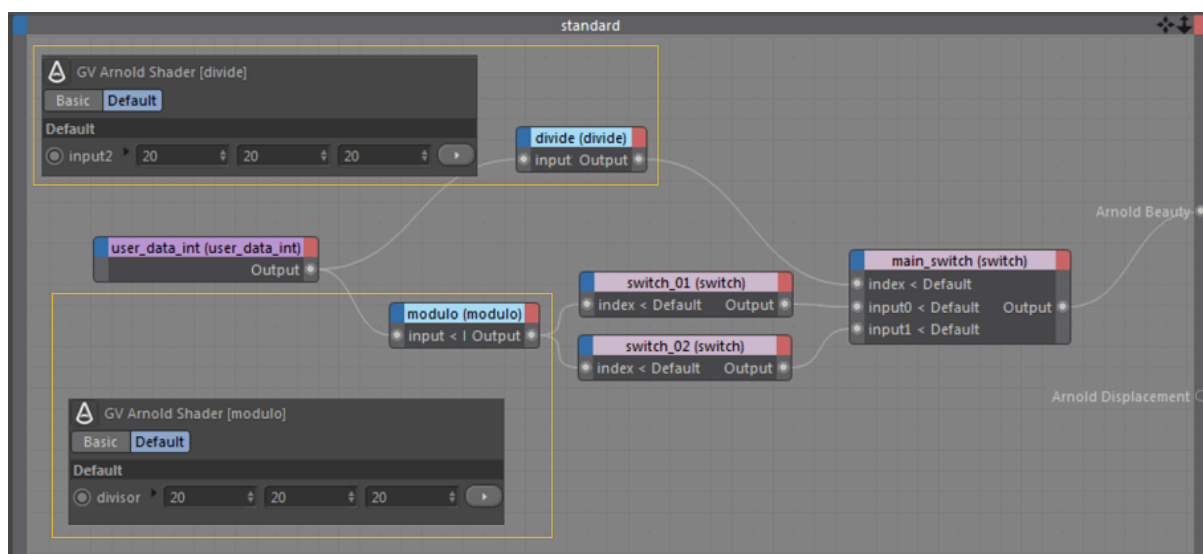
Switch shader



Switch RGBA

Evaluate and return $input_{index}$ as selected by the $index$ parameter, ranging from 1 to 20. The non-selected inputs will not be evaluated.

The following setup can be used in extreme situations if you need more than 20 inputs. This example demonstrates selecting inputs from two switch nodes (switch_01 and switch_02). You need a third switch node (main_switch) to control which of your switches are active. The integer input (user_data_int in this example) is connected to a divide and a modulo node, both have a divisor of 20 (the size of a switch node). The divide node is responsible to select the active switch, so connected to the main switch. The modulo node is responsible to select the input of the active switch, so connected to all other switch nodes.

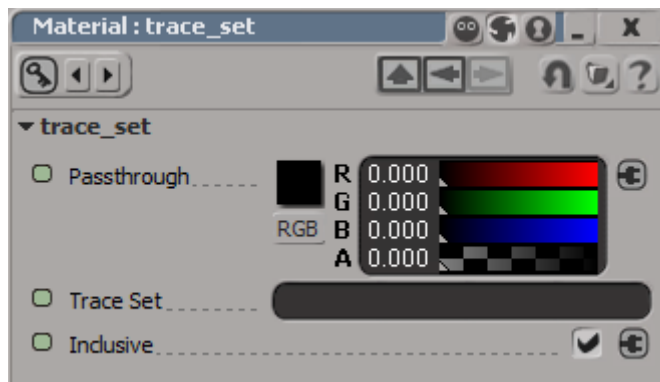


Trace Set

This shader allows you to tag the rays originating from the owner object, so that they can hit or avoid the tagged objects. There are two types of trace sets:

- When a trace set is exclusive, rays are traced against all geometry except the tagged nodes.
- When a trace set is inclusive, rays are traced against tagged nodes, but also against nodes that are not tagged at all.

•



Passthrough

The shader that gets called after setting the label

Trace Set

The string label defining the set of objects to be traced or avoided. Objects are labeled using the [Arnold Parameters](#) property.

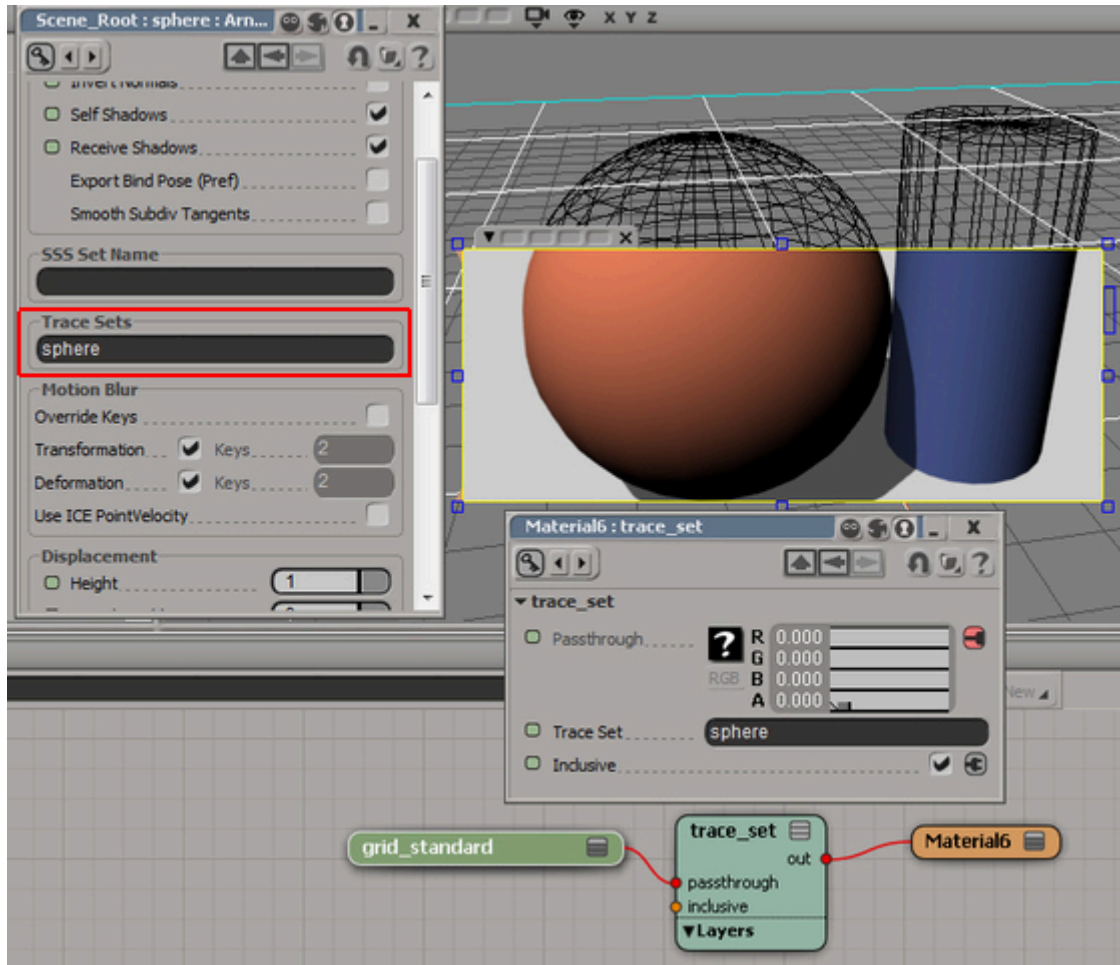
Inclusive

If on, the tracing works in inclusive mode, else in exclusive, as described above.

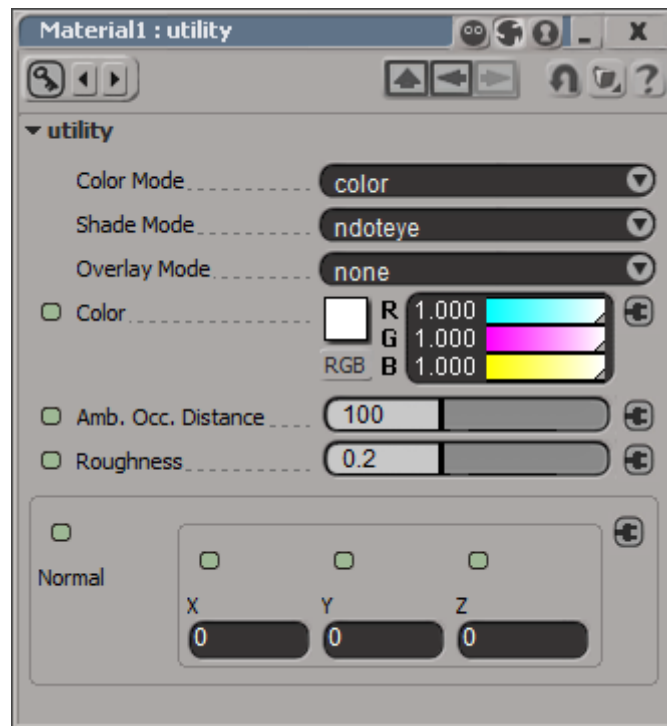
•
•

Example

In the following example, the sphere is labeled as "sphere", the cylinder as "cylinder". The grid has a standard shader, that, before being called, defines its trace set as inclusive for the label "sphere". As a result, only the sphere casts shadows over the grid, while the cylinder is ignored.



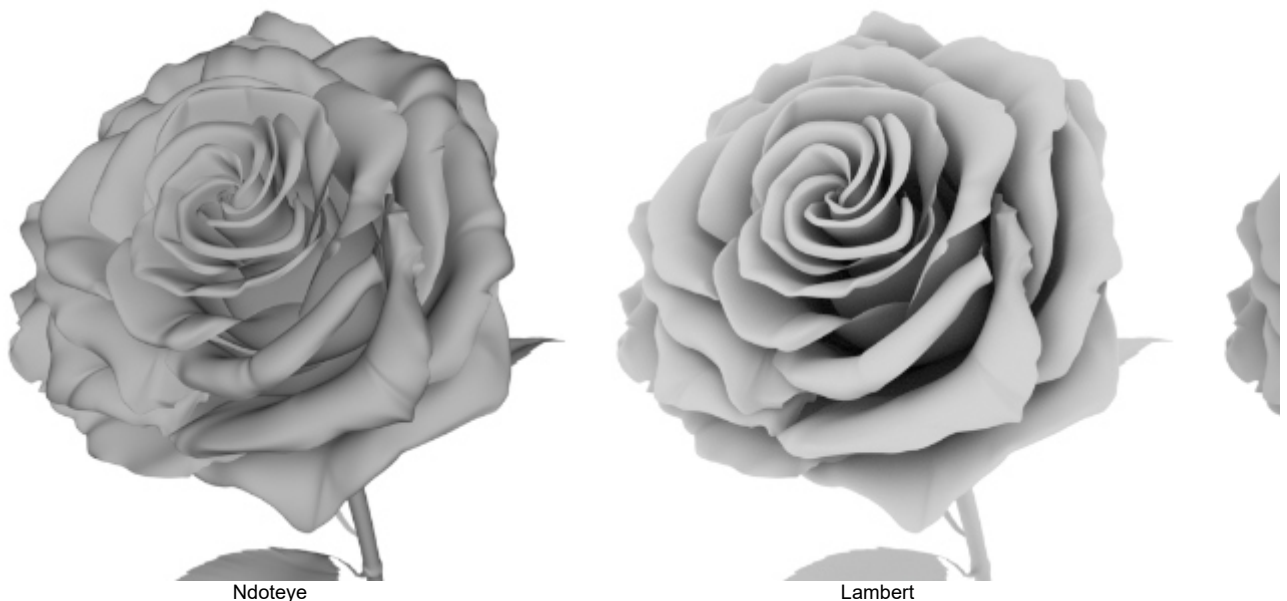
Utility



A general 'all-purpose' utility node shader that can be used to create passes for use within a compositing package. It can also be useful for debugging scenes.

Shade Mode

The shading mode used to render the mesh. ndoteye renders using the eye vector mode. Lambert renders using a simple Lambertian shading model. Flat renders the model as a pure, solid, flatly lit, and shaded color. ambocc renders the model using an ambient occlusion technique.



Ndoteye:• Uses a dot product between the Normal and the Eye vector.

Lambert:• Uses a Lambertian shading model.

Flat: Renders the model as a pure, solid flatly lit, and shaded color.

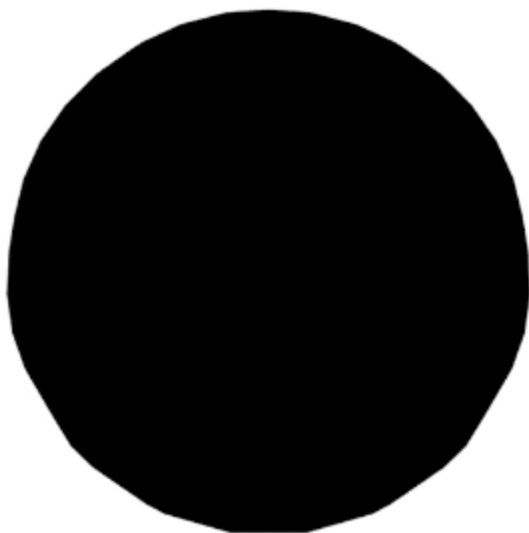
Ambocc:• Renders the model using an ambient occlusion technique. •ambient_occlusion has a distance parameter (ao_distance) to control the length of ambocc rays for scenes where the default of 100 units is too short.

Plastic:• Has both diffuse (0.7) and specular (0.1) components. The specular component is hardcoded to a Cook-Torrance BRDF with MIS support. This mode can be helpful when debugging and optimizing glossy materials, as it can be quickly assigned to all objects in the scene with the simple•kick•commands:•-is -sm plastic.

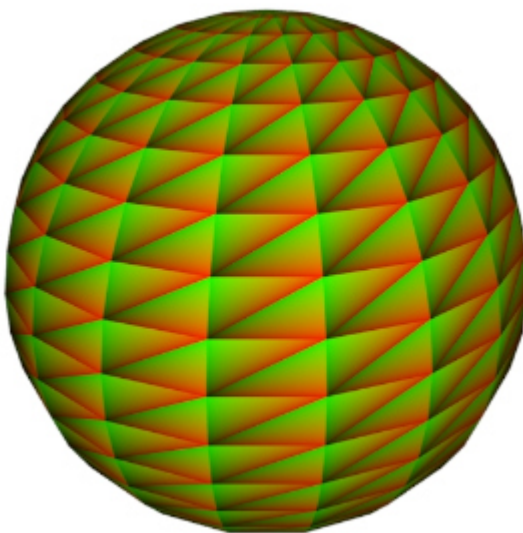
Metal: Similar in use to plastic shade mode, but without the diffuse component (specular only).

Color Mode

The mode used to color the mesh. You can either choose a flat color as specified by the color setting, or the output color can be chosen to represent various things such as the geometry normal, the shading normal, a wireframe view, etc.



Bad UVs



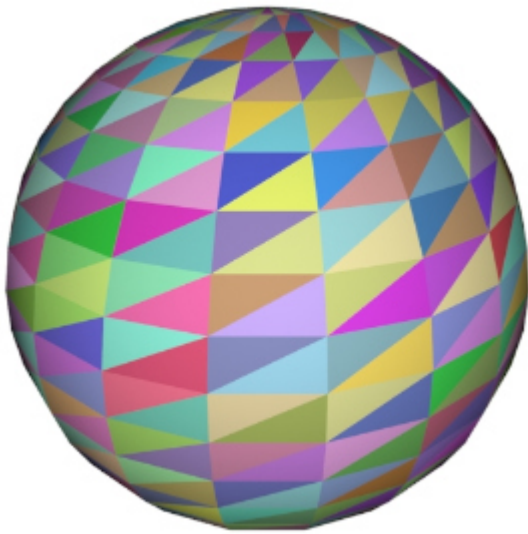
Barycentric Coords



Geometric Normal



Normal



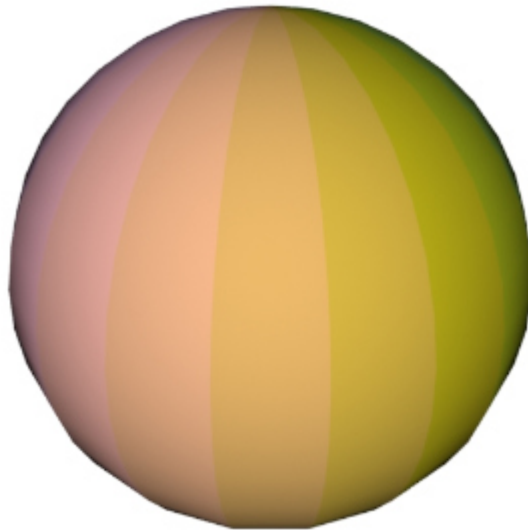
Primitive ID



Reflection Lines



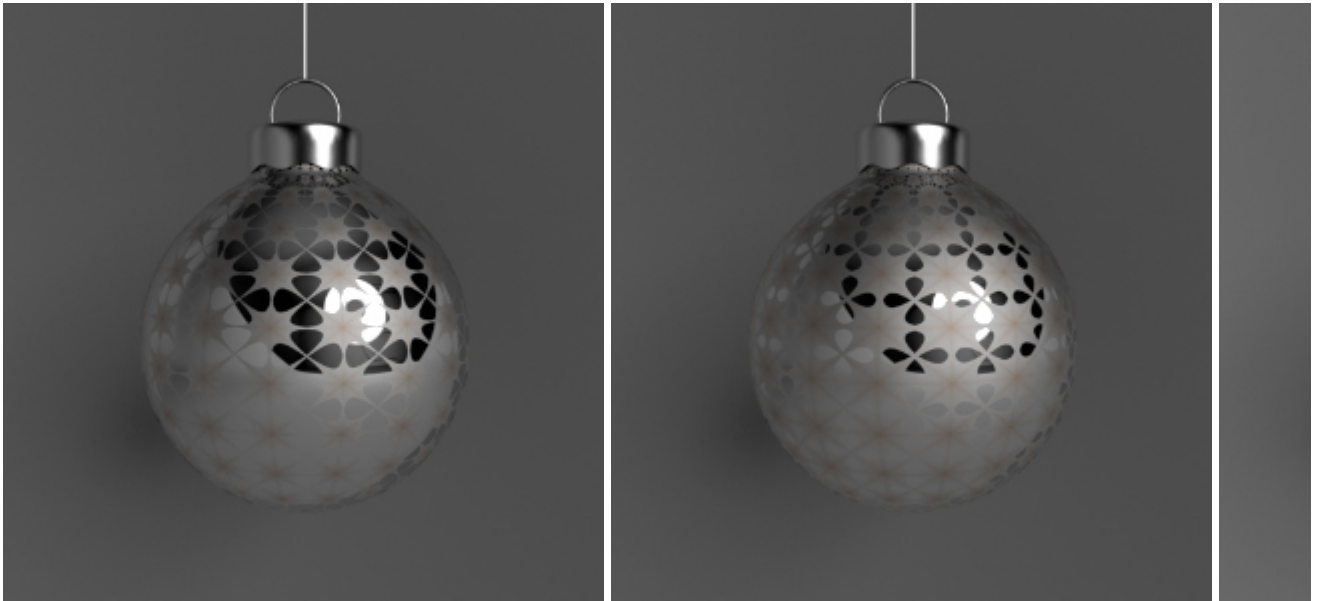
Un-bumped Normal



U Surface Derivative (dPdu)

Bad•UVs: Returns magenta in the UV's of the primitive that is degenerated.

Barycentric Coords: Barycentric coordinates (bu corresponds to red and bv to green) of the primitive.

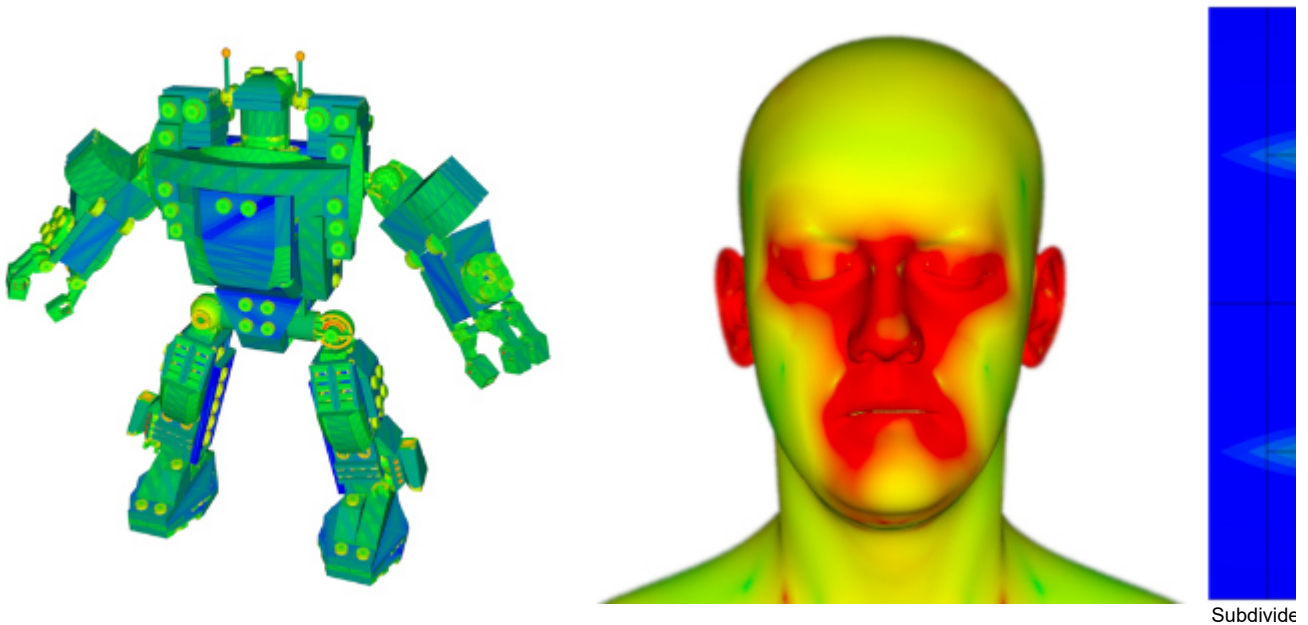


Utility shader ('bary') connected to specular roughness of a Standard shader

Bump Difference : This mode shows how far the bump and autobump normals vary from the base smooth-shaded normals as a heatmap (blue is the same, going through green to red, varying up to 90 degrees away). This is useful for debugging the balance between subdivision iterations with displacement vs autobump making up for the rest.

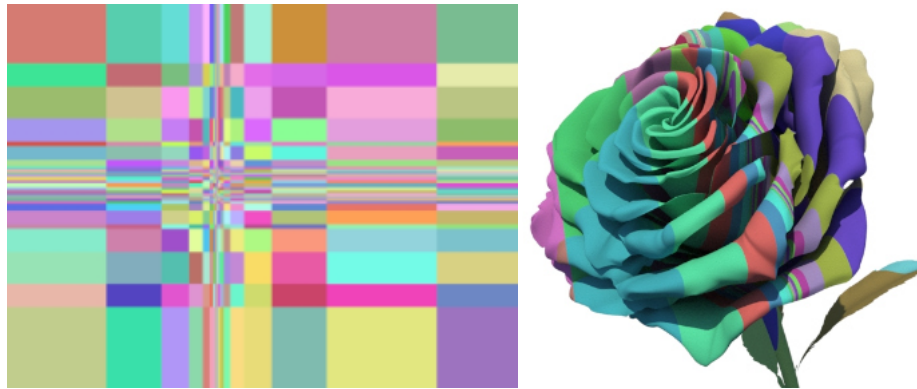
Color :•Single color output.

Edge Length : Shows the edge length of the primitive as a heatmap. This visualization is useful for detecting over-tessellated objects.



Subdivide

Floatgrid : A color is mapped around a Hash function based on the Shading Point.

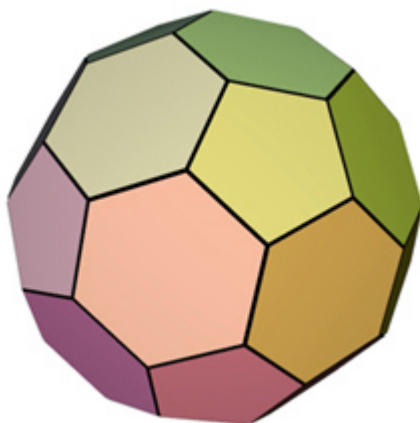


Geometric Normal : Shader normals in world space.

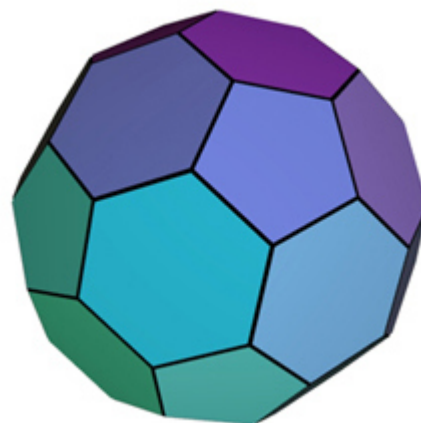


Geometric Normal

Normal : Geometry normals in world•space.



Normals facing outwards



Normals facing inwards

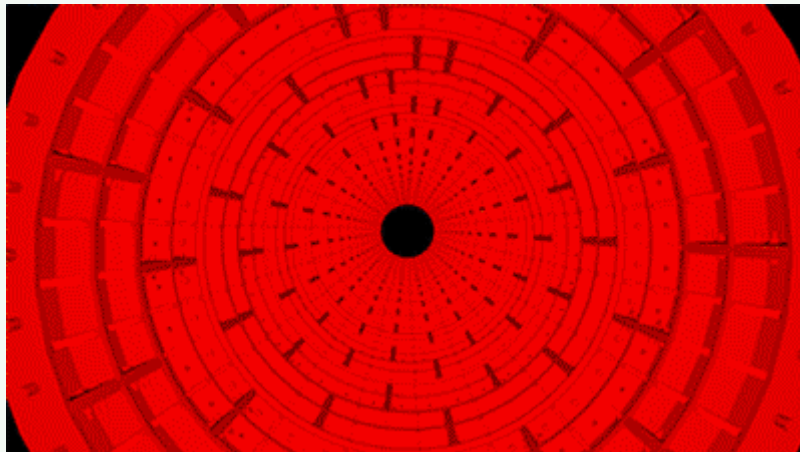
Number of lights :•Shows the relative number of lights considered at the shading point, which is helpful when debugging scenes with many lights. This is visualized with a 'heatmap' color gradient that goes from red (all lights in the scene are considered), passing through yellow and green, to blue (very few lights are considered), and finally black (no lights considered). A utility shader is assigned to the geometry below with `color_mode` set to `number_of_lights`. It is recommended to use `shade_mode.flat`.



The colors become warmer as the moving light approaches the center, indicating light contribution in the scene

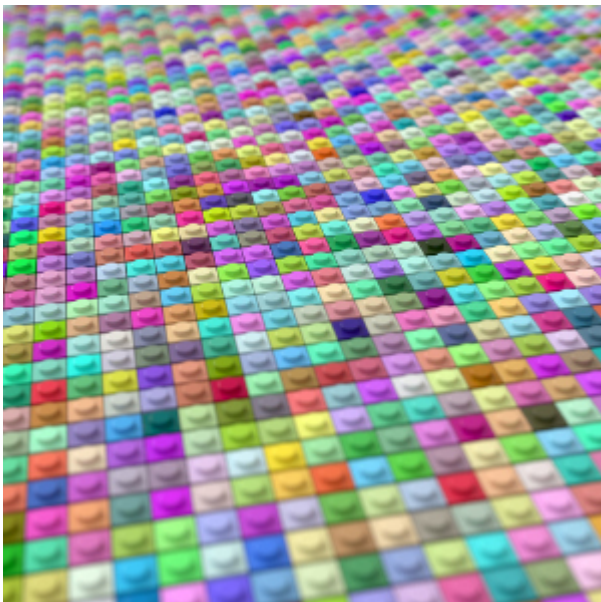


The Utility shader's nlights color mode can be used to see how many lights are influencing a shading point when adjusting the low_light_threshold.



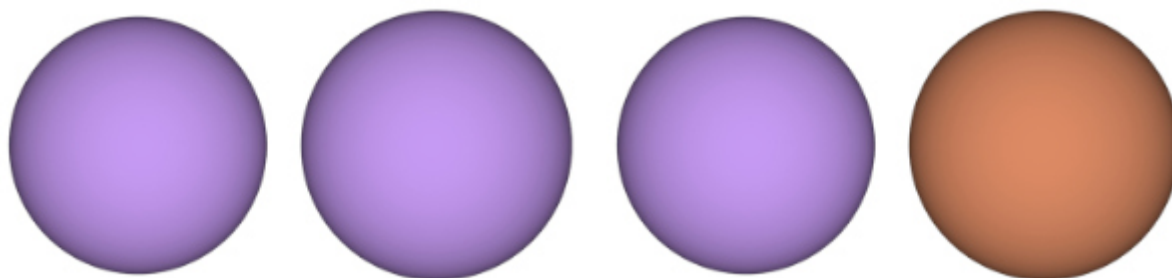
low_light_threshold : 0 to 1. Red = all lights, blue = low values, black = no lights.

Object : Object mode uses the name of the shapes to compute the color.



Color mode set to Object

Object ID : Commonly used when outputting a render per material. ID mode uses the ID parameter shapes have to compute the color. To change the ID, select the mesh and type 'id 1,2,3, etc.' in the user_options of the object.



Both objects set to ID1

Right object set to ID2

Polygon Wireframe: •Renders a polygon wireframe of the mesh.

Primitive ID: •Each primitive ID is represented as a different color.

Reflection Lines: This can be used to diagnose contour lines and spot continuity problems when evaluating models. It can also give you an idea of how real-life reflections would behave. This sort of shading is especially useful when modeling cars for example, where surface evaluation is critical.



reflection_lines reveals continuity error in hood of car geo (rollover image)

Shading Point (Relative to BBox): Shading point, relative to the Bounding Box.



Shading Point (Relative to BBox)

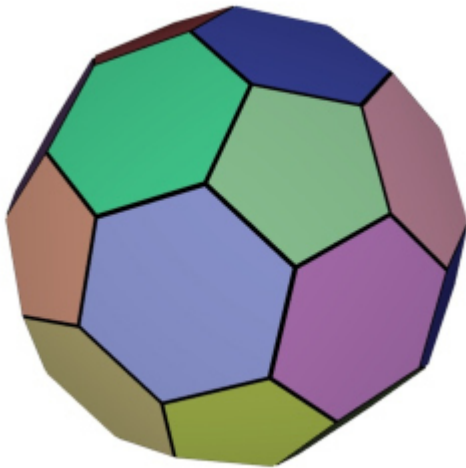
Subdivision Pixel Error : Shows as a heatmap mode, the edge length of the primitive based on how well the polygon matches the subdiv_pixel_error (only visible when the adaptive subdivision is enabled for the mesh).

Triangle Wireframe: Renders a triangulated wireframe of the mesh.

U Coords: U coordinate mapped to the red, green, and blue channels.

Un-bumped Normal: Smooth un-bumped normals in screen space.

Uniform ID: Allows you to color by patch instead of by polygon and by curve instead of the curve segment.



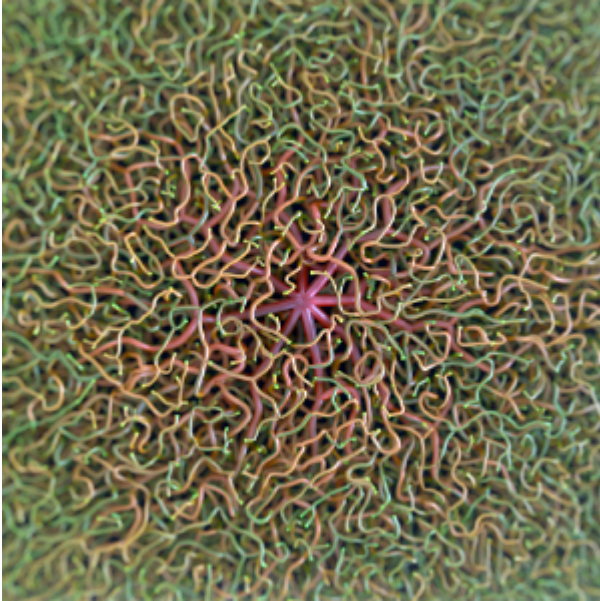
Polygon faces and curves are colored individually using uniform_ID

U Surface Derivative (dPdu): Surface derivative with respect to u coordinate.

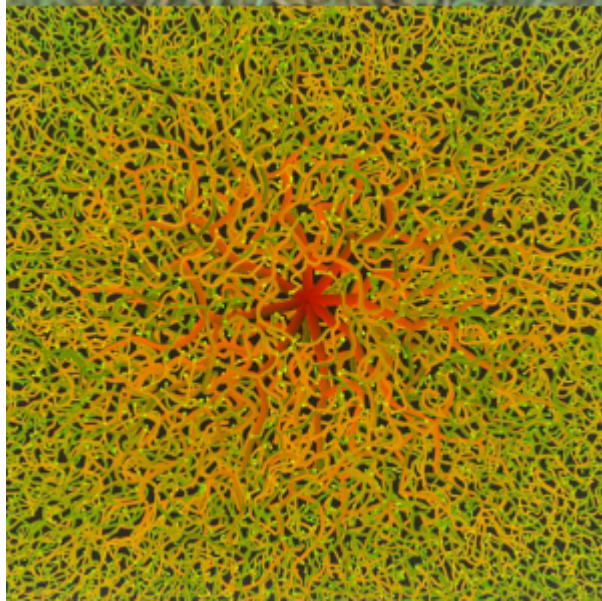


U Surface Derivative (dPdu)

UV Coords: UV coordinates (u corresponds to red and v to green) of the primitive.



UV Coords (shade mode: Plastic)



UV Coords (shade mode: Flat)

V Coords: V coordinate mapped to the red, green, and blue channels.

V Surface Derivative (dPdv): Surface derivative with respect to v coordinate.

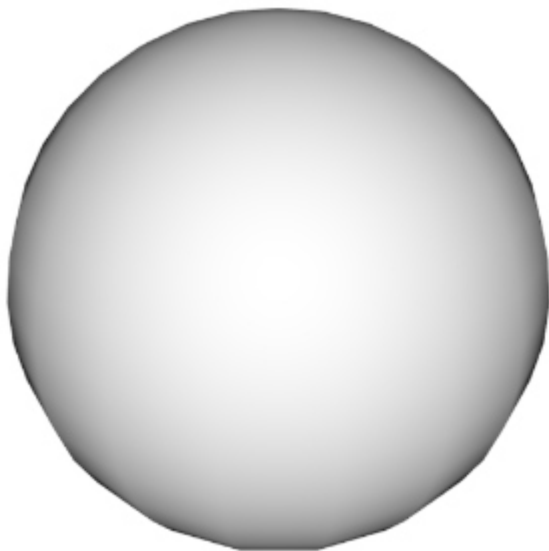


V Surface Derivative (dPdv)

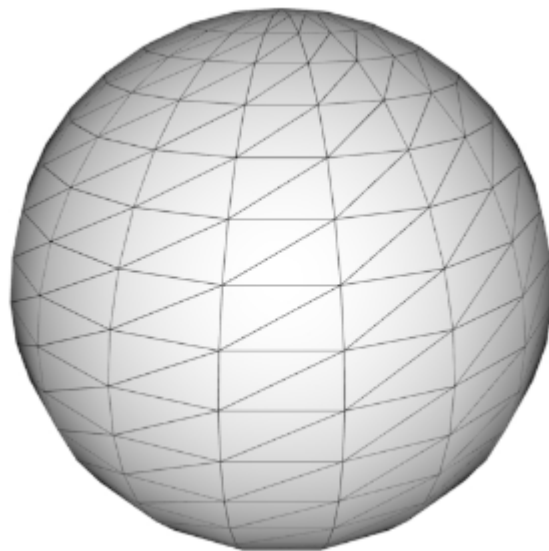
 Click [here](#) to see more information about Object ID mode.

Overlay Mode

This allows you to overlay a wireframe on top of the regular color and shading modes.



None (default)



Wire

Color

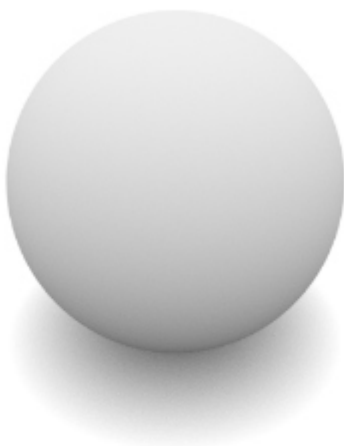
The color used as the shading mode for the model.

Opacity

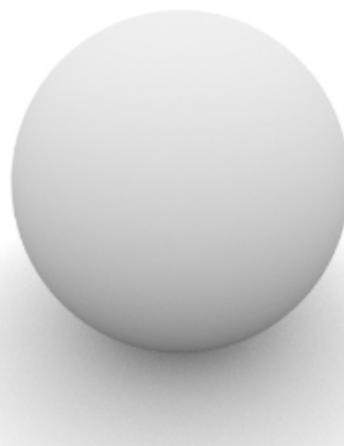
Sets the opacity of the shader.

AO Distance

The maximum occlusion distance that is sampled.



0.5



1

Roughness

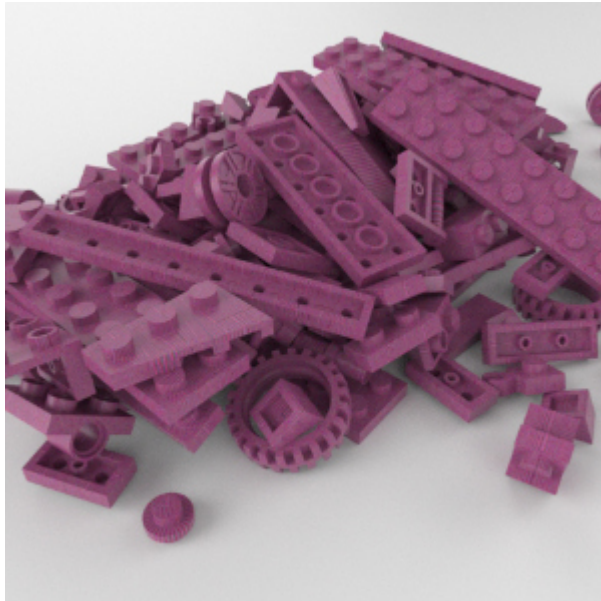
Controls the glossiness of the specular reflections. The lower the value, the sharper the reflection. In the limit, a value of 0 will give you a perfectly sharp mirror reflection, while 1.0 will create reflections that are close to a diffuse reflection. You should connect a map here to get variation in the specular highlight.

•

Further examples



Bad UVs



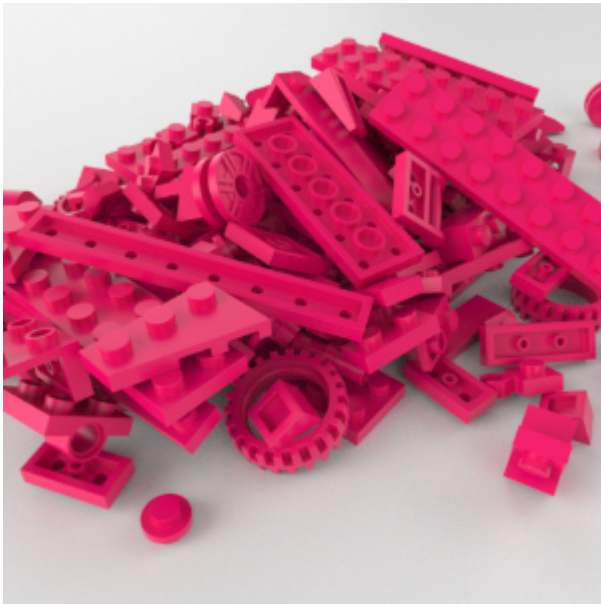
Barycentric Coords



Edge Length



Float Grid



Number of Lights



Object ID



Primitive ID



Reflection Lines



U Coords



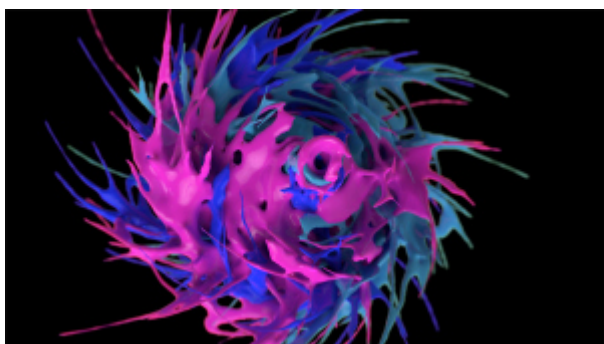
U Surface Derivative dPdu



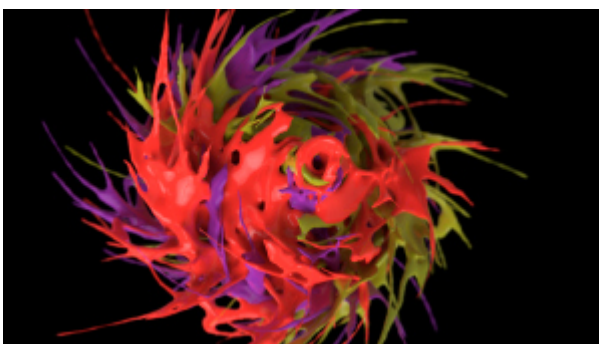
UV Coords



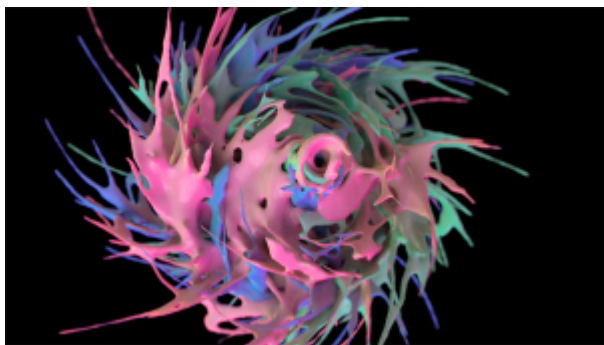
V Coords



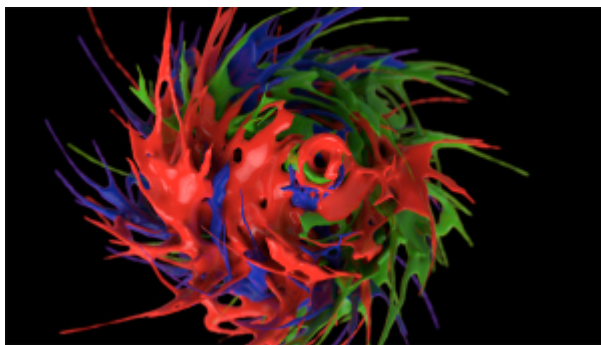
Color Mode: Bump Difference



Color Mode: Color



Color Mode: Normal

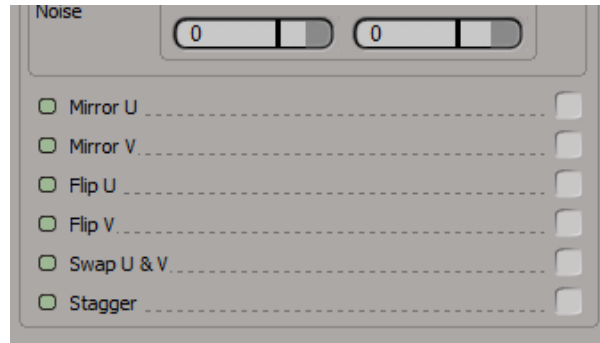


Color Mode: UV Coords



UV Transform





This shader enables you to modify UVs locally, thereby controlling how a 2d texture is placed onto a surface (textures are placed on a surface using the surface's UV coordinates). You can control the position, size, and rotation of an image on the surface using the Scale, Translate Frame and Rotate Frame attributes. You can also control how the texture is tiled within the frame using the Repeat, Rotate, Offset, Mirror, Stagger, and Wrap attributes.

Passthrough

The `passthrough` input is evaluated first and is passed as-is to the output.

Unit

Set rotation angle units to `radians`, `degrees` or `normalized`.

UV Set

A string with the name of the UV set to use to sample the image. By default, when the `uvset` parameter is empty, the primary UV set in the polymesh will be used. Example: If you have created a UV set in a polymesh node called "UVset2", then you can use it by setting the `uvset` parameter to "UVset2".

Frame

Coverage

Controls how much of the surface will be covered by the texture frame.

Scale Frame

Controls how much of the surface will be covered by the texture. Its default values, (1, 1, 0) will cover the entire surface. Reduce these values to make the texture smaller. For example, if you change the Scale U to 0.5, then the texture will only cover half the surface in the U direction.

Translate Frame

Controls the position of the texture, relative to the Pivot frame value (default 0, 0, 0). Increase these values to move the texture frame away from the Pivot. For example, when Translate Frame U is set to 0.5, the texture will be shifted half-way along the surface.

Rotate Frame

Controls how much the texture frame is rotated relative to the surface.

Pivot Frame

Controls the origin of the texture frame relative to the surface.

Wrap Frame U

Flip (mirror) the image in the horizontal direction.



Disabled (default)



Enabled

Wrap Frame V

Flip (mirror) the image in the vertical direction.



Disabled (default)



Enabled

Wrap Frame Color

Defines the color used in the color-wrap mode. Linking a shader to this parameter is also allowed.

Transform

Repeat

Controls how many times the texture is repeated.

Offset

Swaps the axes.



Disabled (default)

Enabled

Rotate

Controls how much the texture is rotated within the texture frame (the frame itself is not rotated). To rotate the frame, use the Rotate Frame attribute instead).

Pivot

The origin value at which modifications to any values are based around.

Noise

The 2D noise for U and V.

Mirror U

Page not found for multiexcerpt macro.

The page: A5noderef:UV_Transform was not found. Please check/update the page name used in the 'multiexcerpt-include macro.

Mirror V

Controls how texture tiles are placed beside each other. When enabled, each copy of the texture will be the mirror image of the ones on the V side of it, with their shared edge being the (U) axis of reflection.

Flip U

Flip (mirror) the image in the horizontal direction.



Disabled (default)



Enabled

Flip V

Flip (mirror) the image in the vertical direction.



Disabled (default)



Enabled

Swap UV

Swaps the axes.



Disabled (default)



Enabled

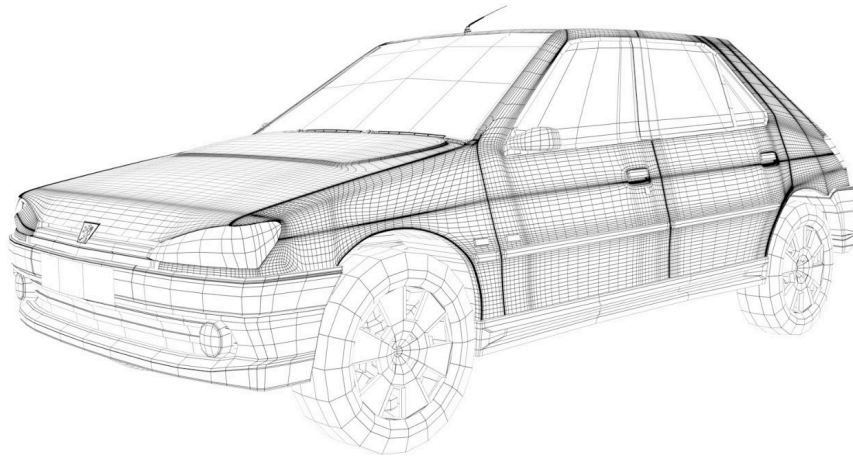
Stagger

Controls the alignment of texture tiles placed alongside each other. When enabled, every other row of texture tiles is offset by one-half of its width, similar to the way bricks are stacked in a brick wall. •

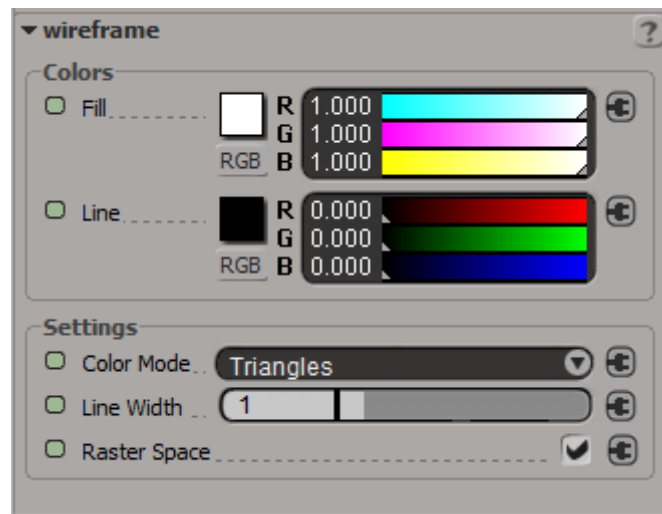
•

Wireframe

•



Color shader which produces a wire-frame style output (as RGB).



Edge Type

Determines how the mesh is represented. Quads will render as polygon faces. If triangles is selected, polygons will be broken up into their triangular tessellation. Patches are not currently supported.



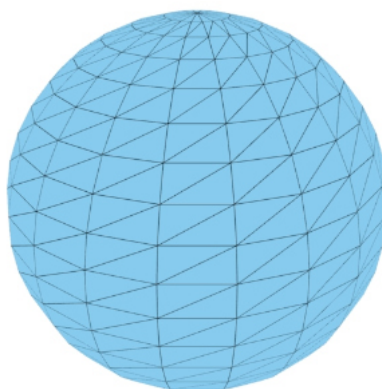
Edge Type: Triangles (default)



Edge Type: Polygons

Fill Color

The color used to represent the polygon faces.



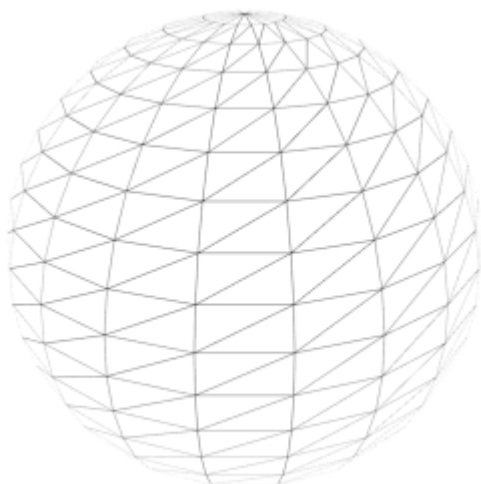
Line Color

The color used to represent the edges.

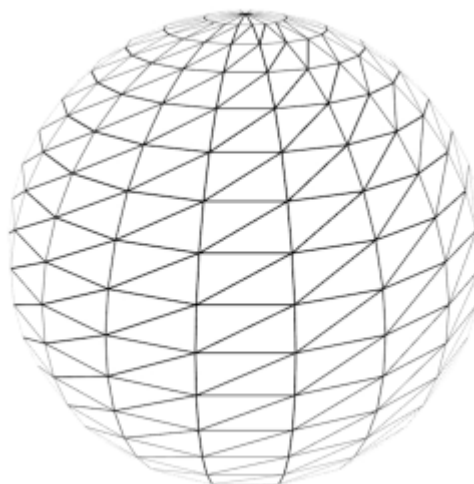


Line Width

The thickness of the edge lines of the polygon faces.



1 (default)



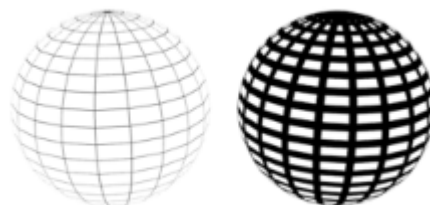
2

Raster Space

When enabled, the line width will be set in screen space pixels instead of world units.



Scale: 1



Scale: 0.1

Right sphere has raster space disabled

•

Volume Shaders



Volume shaders allow you to visualize a 3D scalar data field (e.g. density field) by sampling field values and mapping to color and opacity. You can assign volume shaders to [volume shapes](#).

Atmosphere shaders are special volume shaders to simulate light scattering effect in the scene, like fog. Atmosphere shaders have to be set in **Arnold Render settings > Main > Environment > Atmosphere** field.

 For more information on volume workflows refer to the [Volumes](#) page.

Below is a list of volume shaders available in SItOA:

- [Standard Volume](#)
- [Volume Sample Float](#)
- [Volume Sample RGB](#)

 It is recommended that volume shading networks are kept as lean as possible. This is important for render times in the volume context because it is evaluated so often.

Standard Volume



Walt Disney Animation Studios Cloud Data Set available [here](#). Tutorial [here](#).

The `standard_volume` is a physically-based volume shader. It provides independent control over volume density, scatter color and transparent color. Blackbody emission is used to render fire and explosions directly from physics simulations.

Each component can be controlled by a volume channel coming from the volume object, with other parameters acting as multipliers on the channel. Optionally the channel can be left empty, and a custom shader like `Volume Sample` or a procedural texture may be connected instead, to manipulate each component with more control.

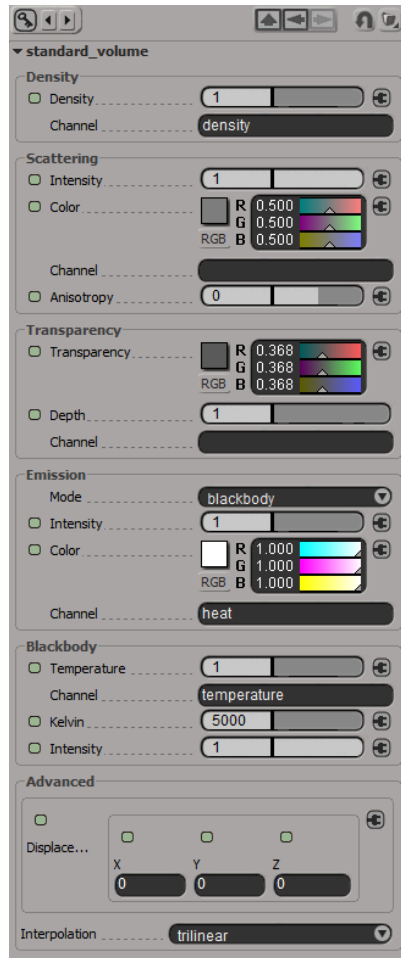
However, be warned that the evaluation of a shader network for volume rendering is much more expensive than for surface shading because the shader network is called many times per ray, once per ray march sample. So, in a production environment, it's best to use as few shaders as possible, ideally having just the `standard_volume` shader doing all the work.



The images in the examples below use a GI Volume Depth of 8 (defaults to 0).•This is useful when rendering volumes such as clouds for which multiple scattering has a large influence on their appearance.



It is recommended that volume shading networks are kept as lean as possible. This is important to render times in the volume context because it is evaluated so often.



Density

Density

The density of the volume, with low density resulting in thin volumes and high density in thick volumes. It acts as a multiplier on the density channel, or if no density channel is specified, it may be used to connect a shader like a volume sample or a procedural texture.



0.1

0.25

Channel

Density channel to read from the volume object.

Scattering

Intensity

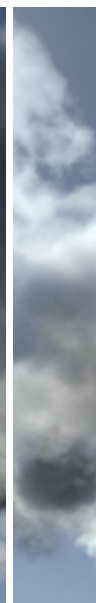
The brightness of the volume under illumination. This is the ratio of light that is scattered rather than absorbed, for energy conservation values in the range 0..1 must be used.



0.1



0.5



Color

The color of the volume under illumination.



White (default)



Light orange tint

Channel

Scatter channel to read from the volume object. This acts as a multiplier on the scatter color, to texture the color of the volume.

Anisotropy

The directional bias, or anisotropy, of the scattering. The default value of zero gives isotropic scattering so that light is scattered evenly in all directions. Positive values bias the scattering effect forwards, in the direction of the light, while negative values bias the scattering backward, toward the light. This shader uses the Henyey-Greenstein phase function.

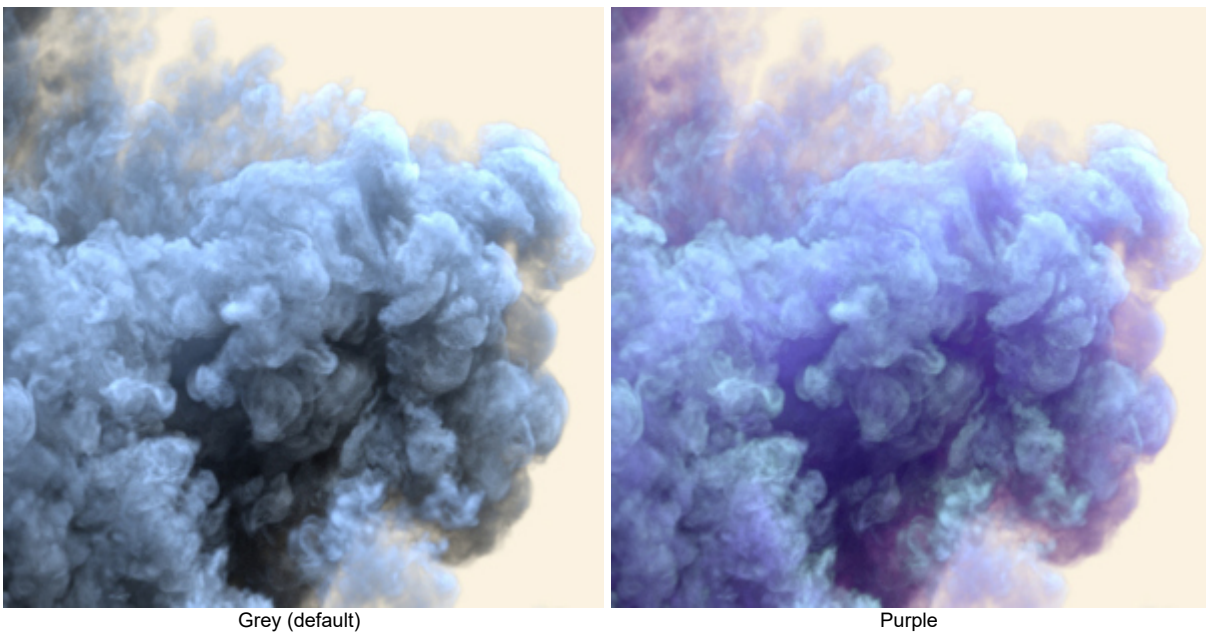
Note that values very close to 1.0 (above 0.95) or -1.0 (below -0.95) will produce scattering that is so directional that it will not be very visible from most angles, so such values are not recommended.



Transparent

Transparency

Additional control over the density of the volume, to tint the color of volume shadows and objects seen through the volume.



Depth

Additional control over the density of the volume, to control the depth into the volume at which the transparent color is realized.



1 (default)



2

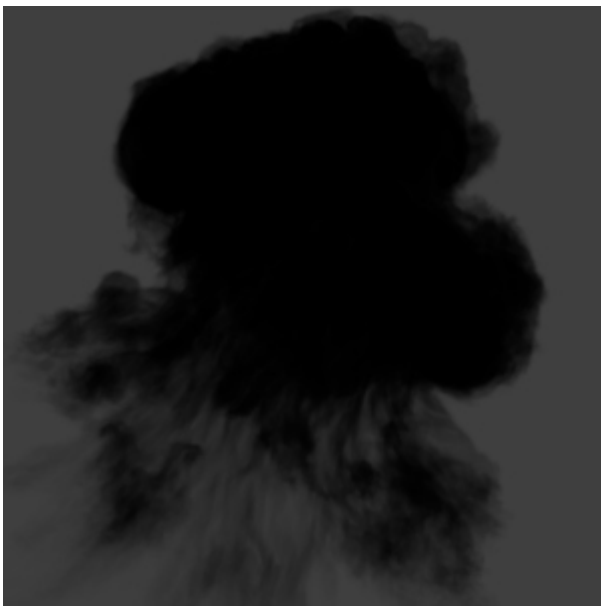
Channel

Transparent color to read from the volume object. It acts as a multiplier to texture the transparent color in the volume.

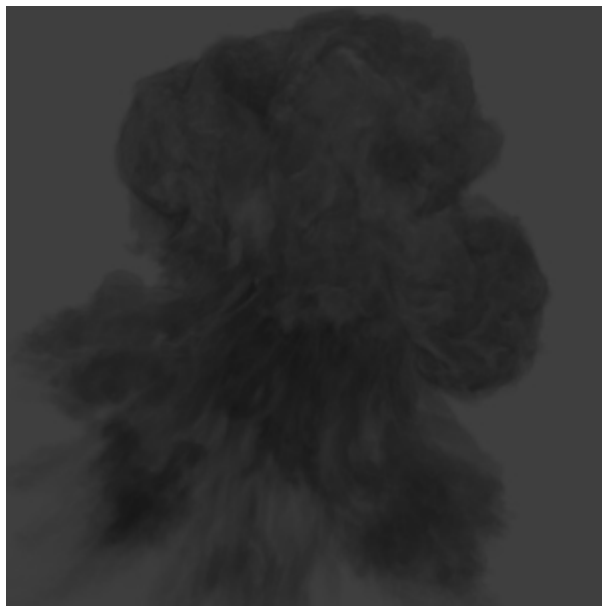
Emission

Mode

- None: emit no light.
- Channel : emit light using a specified emission channel, or using a shader connected to the emission rate or color parameters.
- Density : emit light using the density channel, modulated by an optional emission channel.
- Blackbody : emit color and intensity based on temperature, for rendering fire and explosions.



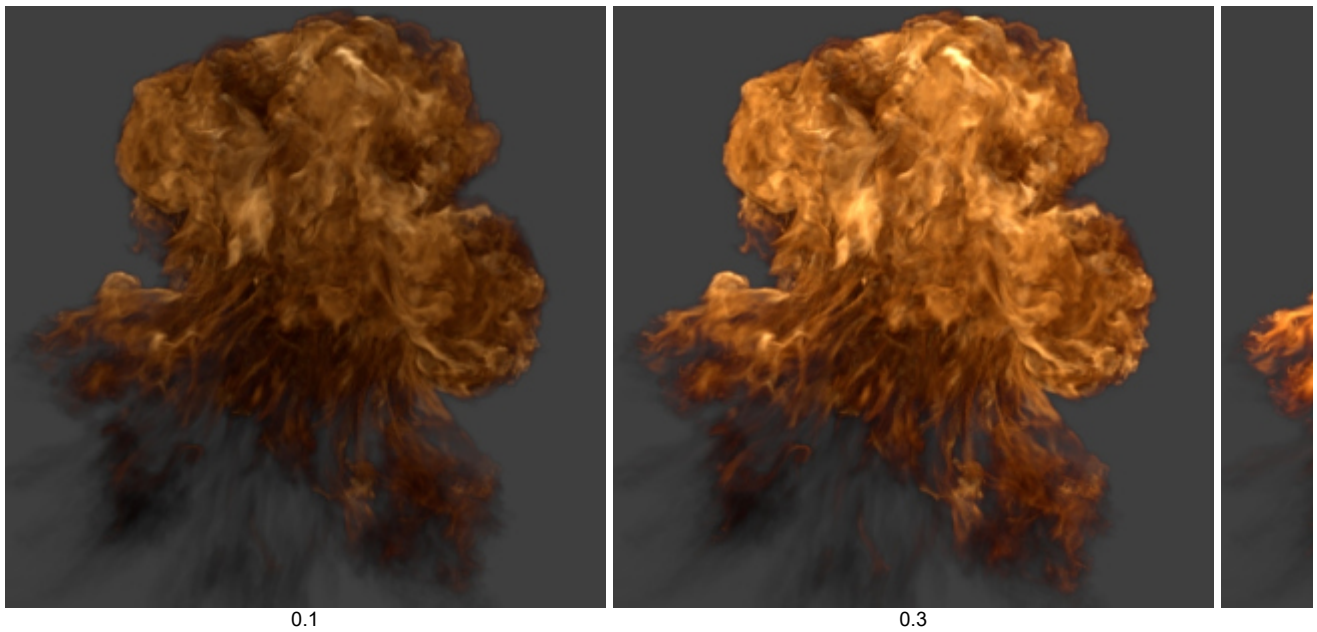
None



Density

Intensity

Emission is the rate at which a volume emits light. If a density channel, emission channel, or blackbody channel is used for emission, this acts as a multiplier to decrease or increase the emission. Alternatively, if no such channel is used, a shader like a volume sample or a procedural texture may be connected to this parameter to control where light is emitted.



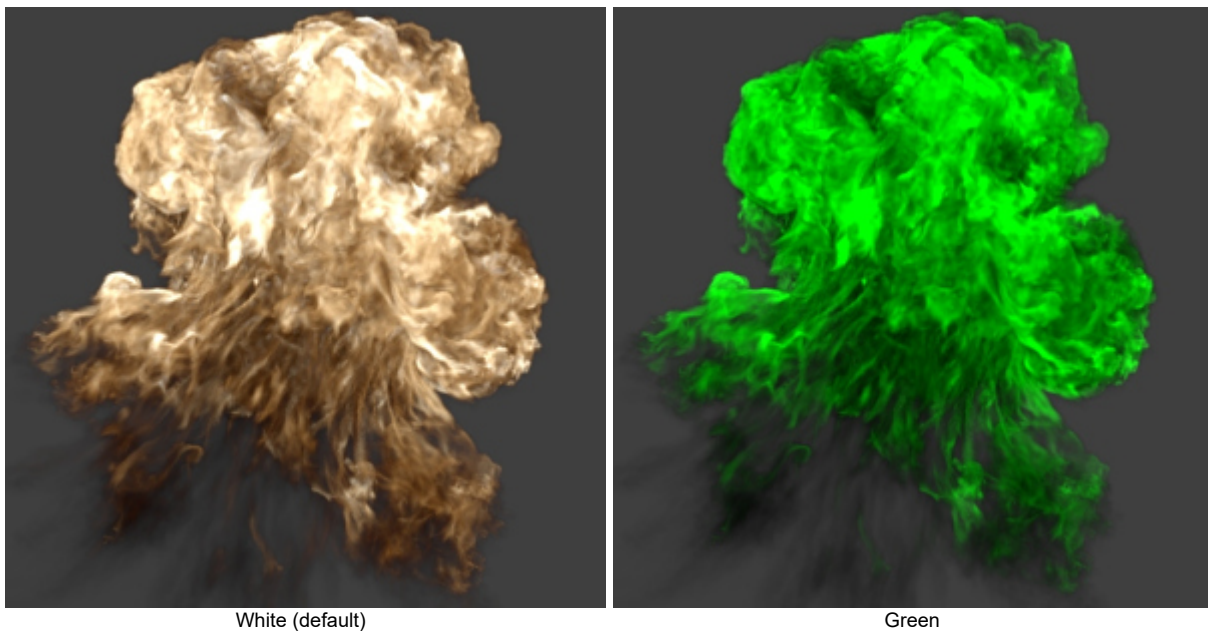
(Emission Mode: Blackbody)



Volumes can only receive indirect lighting from other emissive volumes when the **Volume Ray Depth** is above 0.

Color

A color to tint (multiply to) the emission.



Channel

In Channel emission mode, the emission channel to sample the emission rate from.

When not empty, this channel will be used to sample the emission values for the volume instead of the Emission parameter. The sampling tab has additional parameters to control how data is sampled from volume channels.

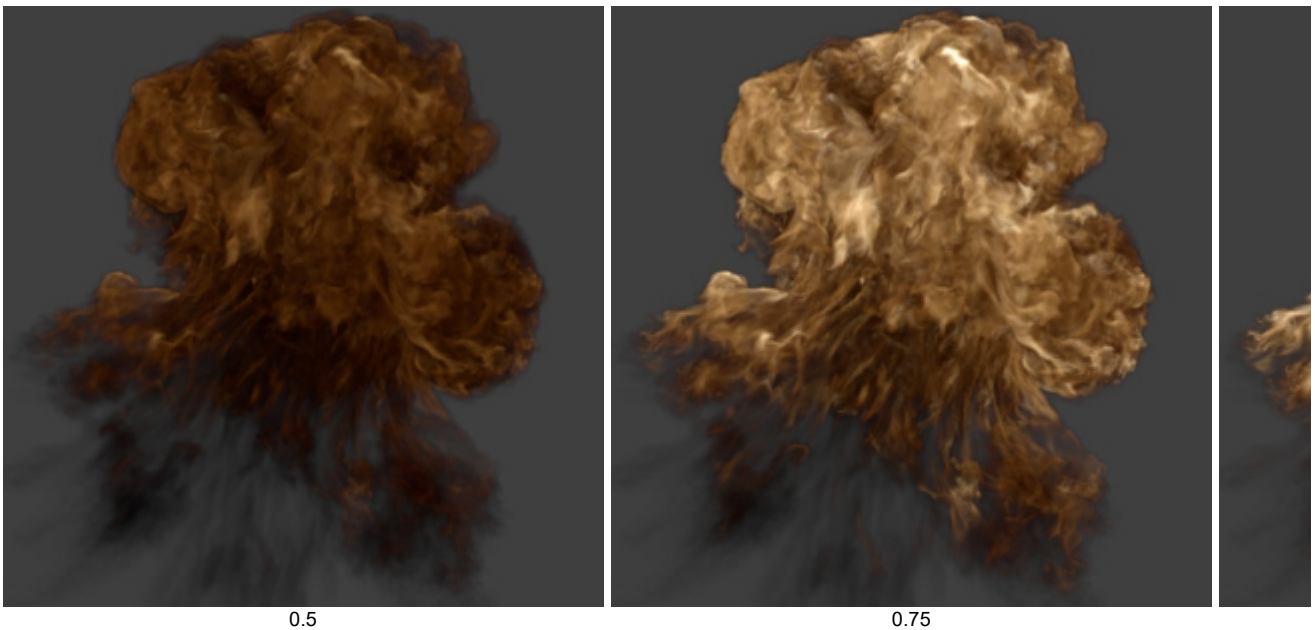


When using the channel emission mode, ensure that the emission attribute is driven by a VDB grid. If it is a fixed parameter value, you may get artifacts.

Blackbody

Temperature

If a blackbody channel is used, this acts as a multiplier for the blackbody temperature. Alternatively, a shader like a volume sample or a procedural texture may be connected to this parameter to control the blackbody temperature.

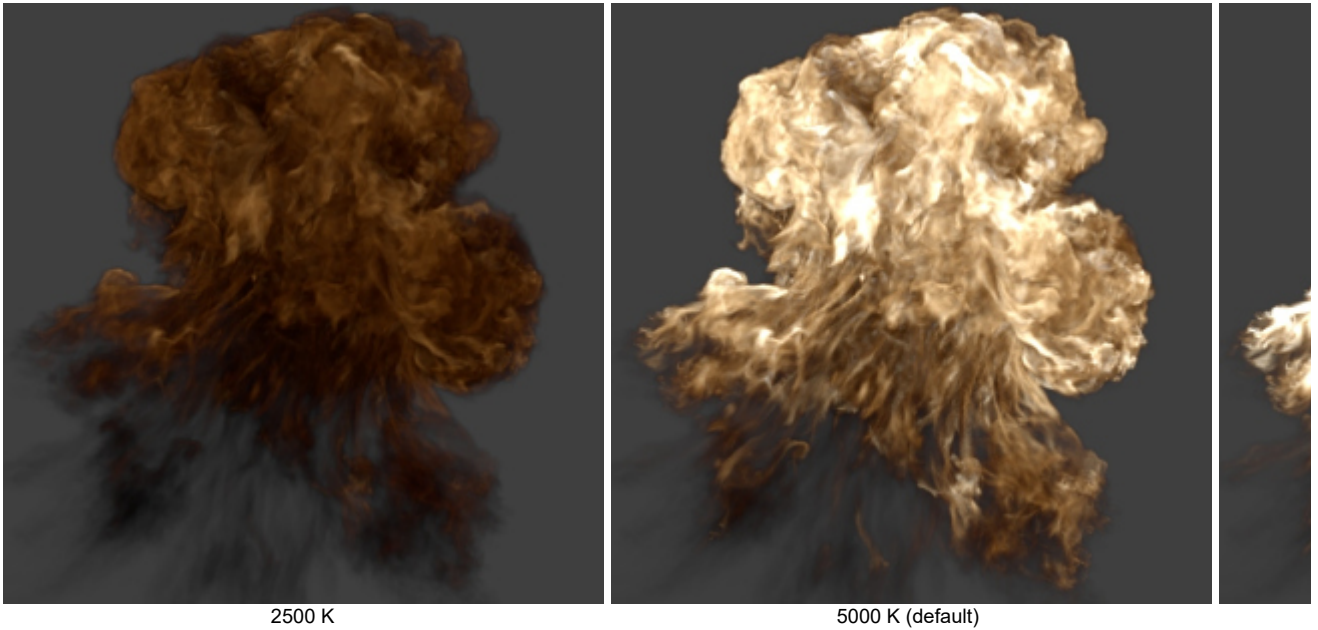


Channel

In the blackbody emission mode, the temperature channel to read from the volume object. Typically the temperature channel comes from a pyro physics simulation.

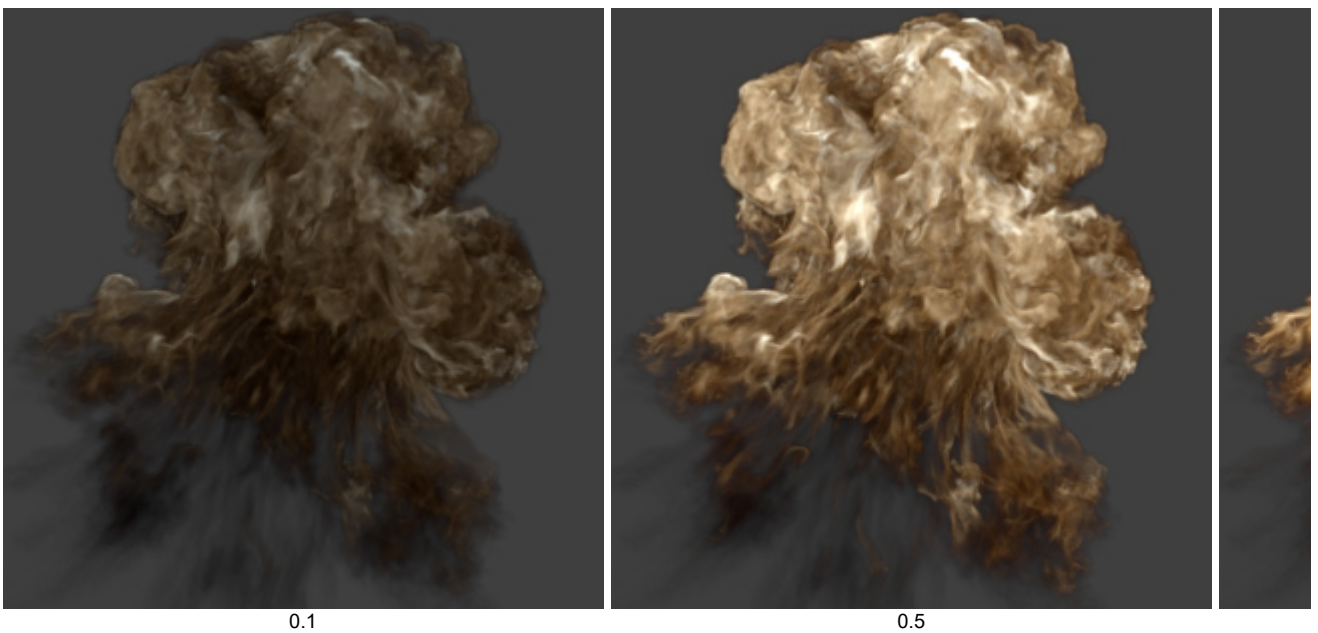
Kelvin

The multiplier for the temperature from the temperature volume channel. Usually, the temperature is stored in a 0..1 range, and this parameter can be used to scale it to a temperature in Kelvin.



Intensity

Control the intensity of the blackbody emission. For physically correct results intensity 1 must be used. This can result in extremely bright light, however, and lower values may be used to reduce the intensity.



Advanced

Displacement

Connect a noise texture or other shaders here to displace the volume, similar to displacement on surfaces. This can be used to add additional detail to the volume or to warp it. The connected texture will be applied as an object space offset to the volume sampling position, for all volume channels.



No displacement



Noise -> Displacement

- ✓ When using the noise shader with displacement, ensure that its mode is set to vector.



Mode: Scalar (default)



Mode: Vector

Noise shaders -> displacement

- ✓ When using displacement, ensure that the volume_padding of the volume is increased to account for the increased size of the volume.



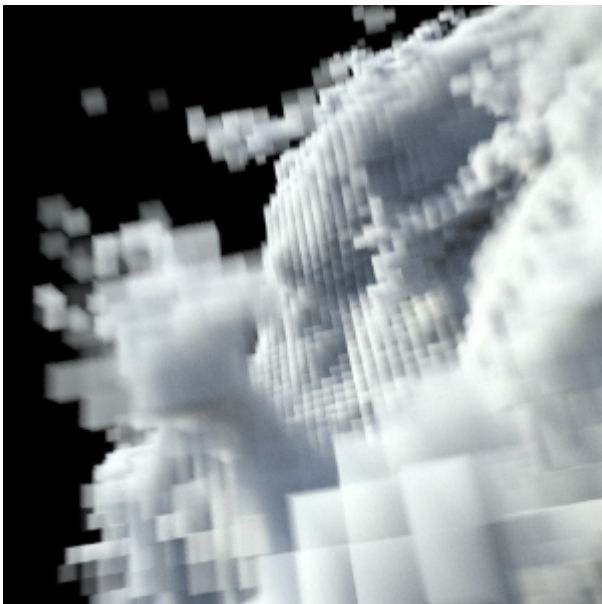
Volume Padding: 0



Volume Padding: 10

Interpolation

The voxel interpolation to use when sampling the volume data using named channels.



Closest



Trilinear (default)

-
-
-



Further examples

-
-

Volume Sample Float



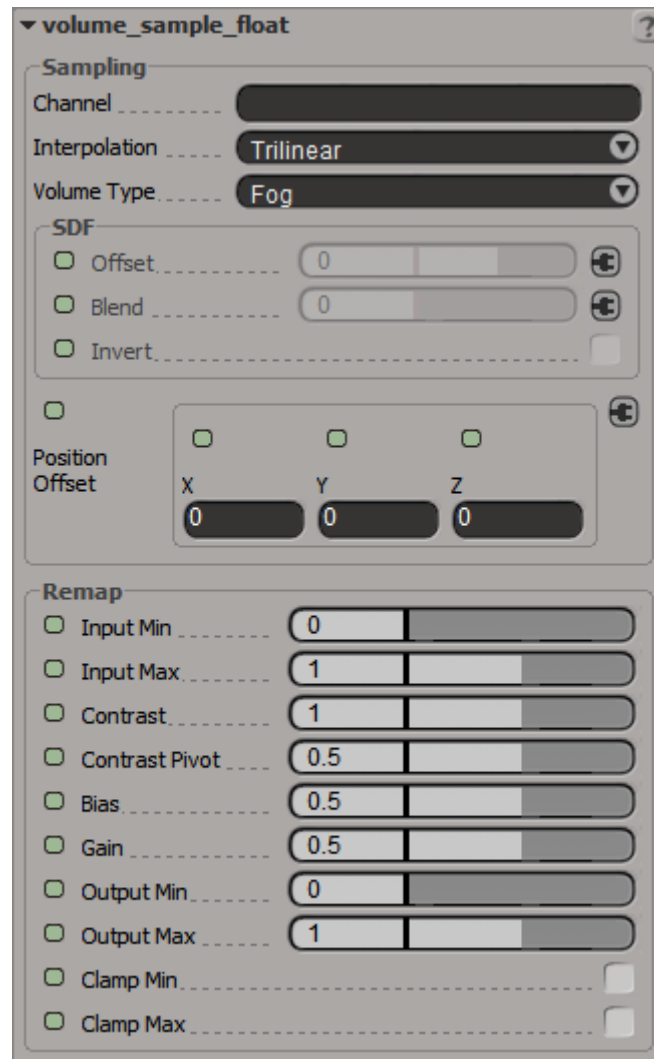
The `volume_sample_float` shader will sample a volume channel of the new volume API. It should be plugged into one of the components of the `standard_volume` shader if you need to further edit the result of the reading of a channel, for instance to color correct it, etc. For more information on volume workflows refer to the `volume` page.



Both the `volume_sample_float` and `volume_sample_RGB` shaders share the same sampling controls, the only difference being the type of the output, RGB or float.



These parameters are not texturable due to performance reasons.



- ## Sampling

- ### Channel

- When not empty, this channel will be used to sample the scattering values for the volume instead of the scattering parameter.

- ### Interpolation

- The voxel interpolation to use when sampling the volume data using named channels.



Tricubic

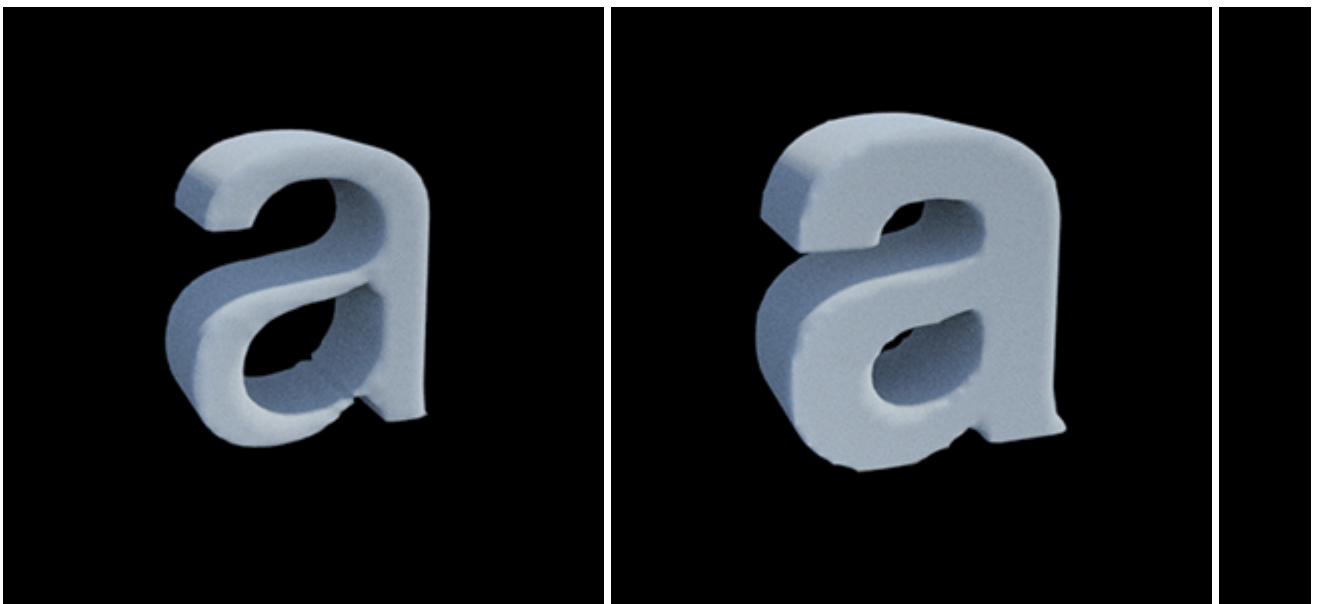
Trilinear (default)

Volume Type

Interpret the volume data as fog (density) or SDF (signed distance function). In SDF mode, the inside is converted to a constant density of 1.

SDF Offset

Offset the SDF values by this distance, allowing to dilate or erode the SDF boundary shape.

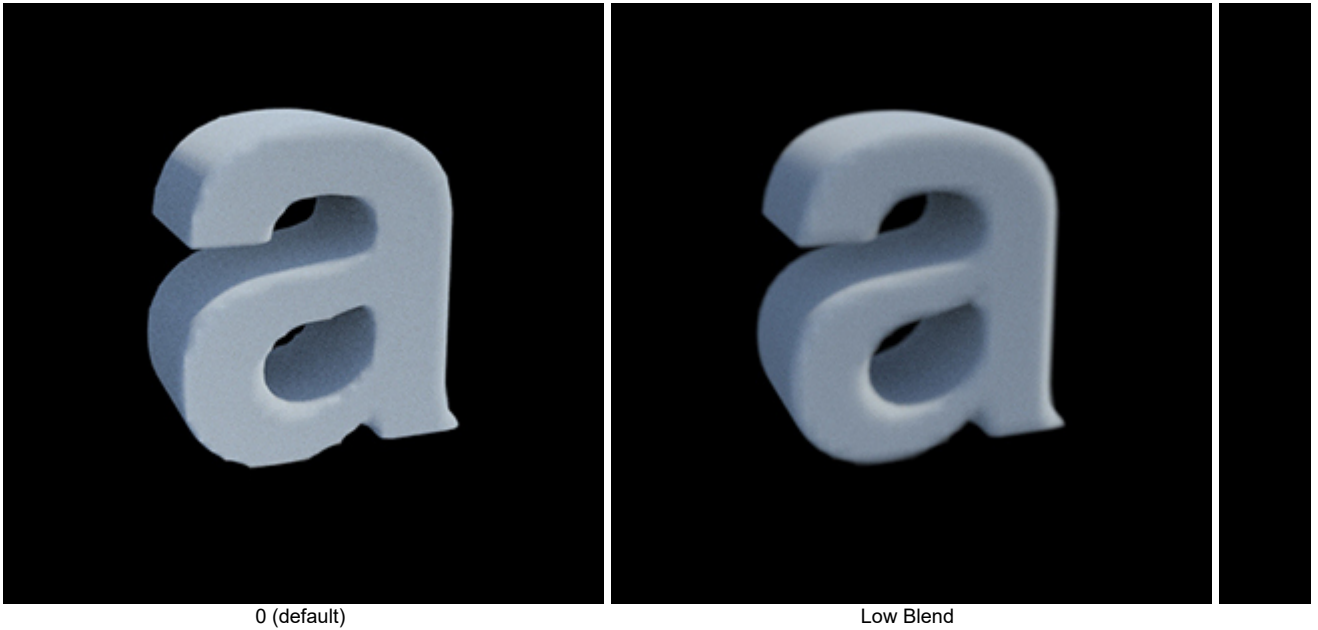


Negative Offset

Zero Offset (default)

SDF Blend

The distance over which to smooth out the edges of the SDF. •



0 (default)

Low Blend

Invert •SDF

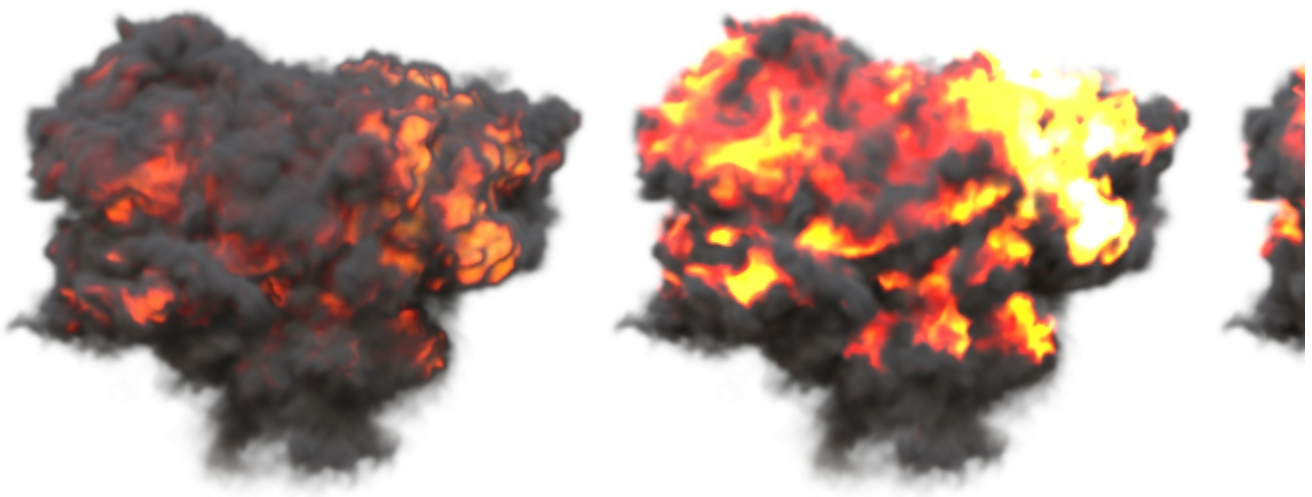
By default, positive distances represent space outside of the boundary while negative distances mean inside. This parameter lets you invert this convention.

Position Offset

An object space offset to apply to the volume sampling position when using named channels to fetch the volume data. This is useful to displace the volume data.



You will need to connect a noise texture to the position offset to see any result.



Y: 0

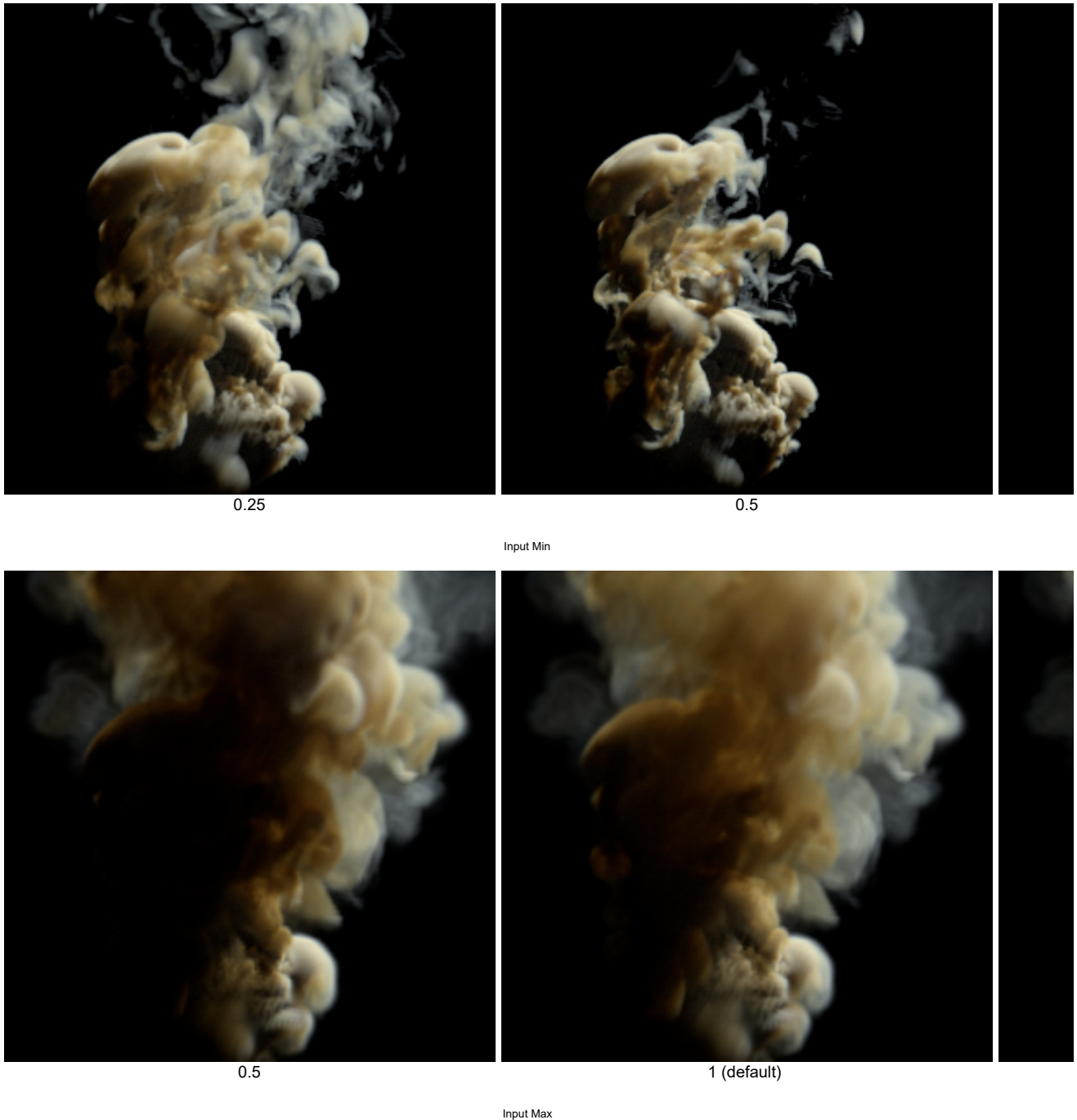
Y: -10

Remap••

These are useful parameters to remap the float sampled values to an arbitrary range . The operations are applied in the same order as the parameters. •

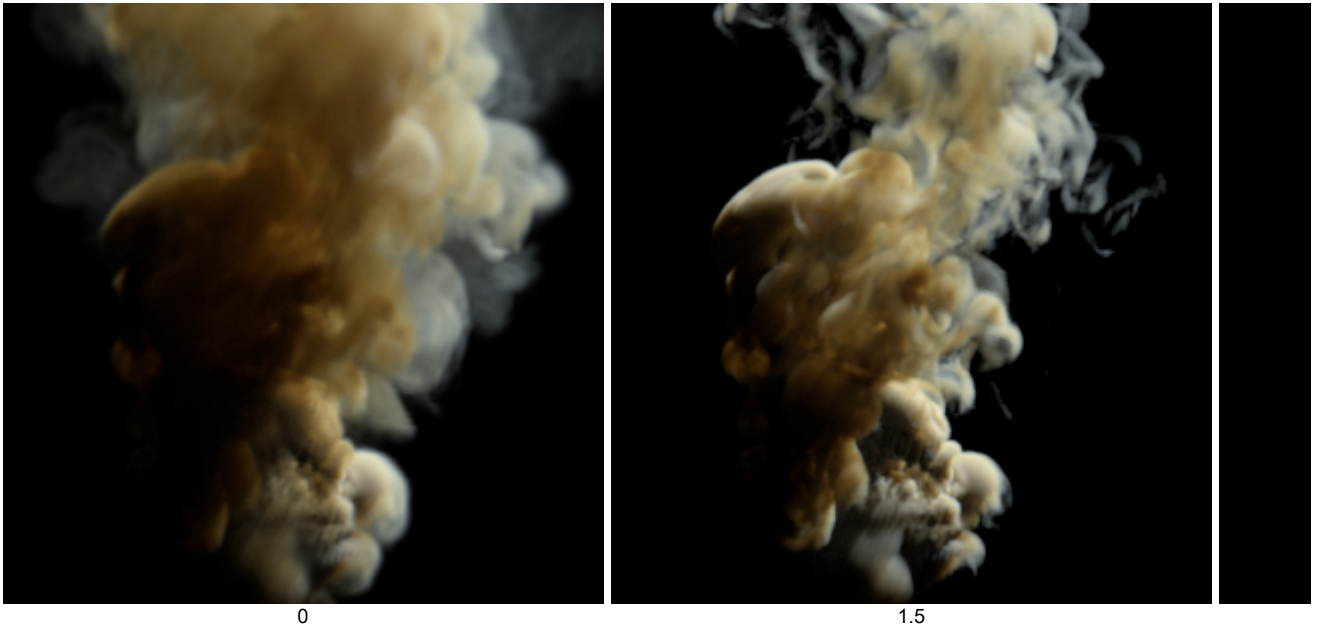
Input Min / Max

The range of input values that will be mapped internally to [0, 1]. The internal values are not clamped.



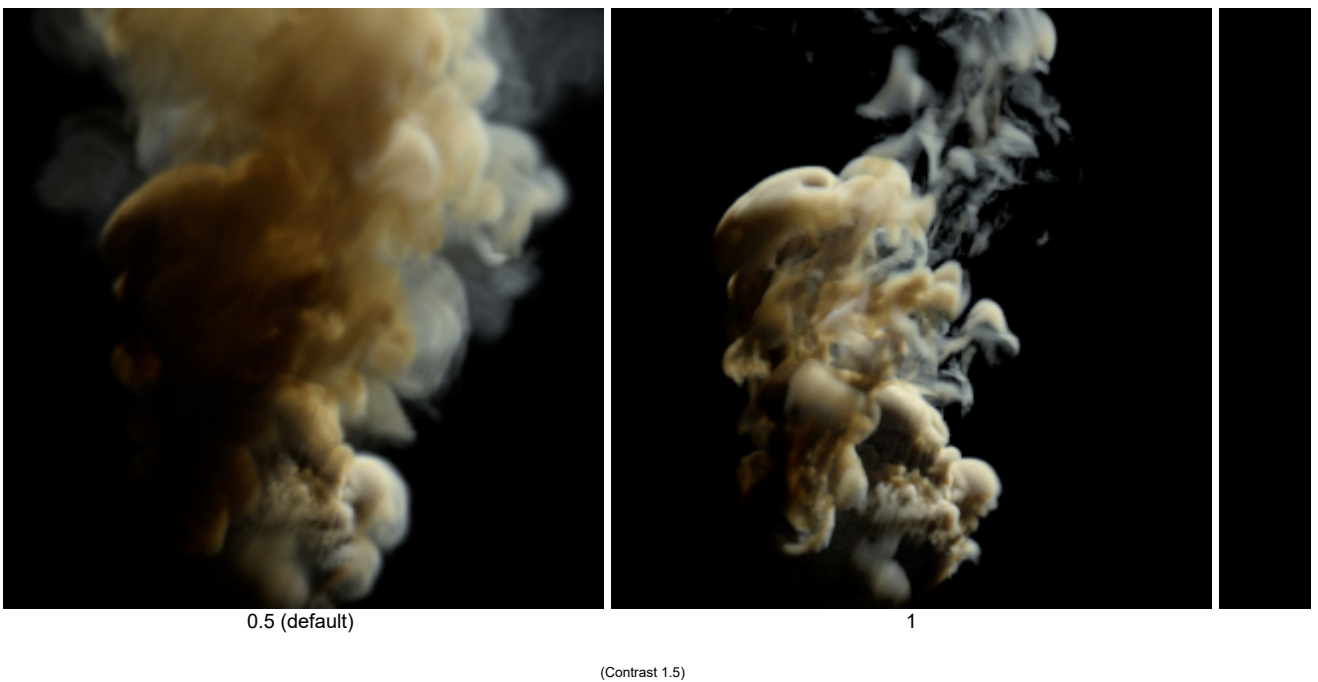
Contrast

Scale values around the Contrast Pivot.



Contrast Pivot

The origin of the contrast scaling.



Bias

Push or pull values by altering the slope at the beginning of the range. Bias values below 0.5 decreases the slope and lower values overall. •Above 0.5, the slope is higher, and the value grows more quickly. A value of 0.5 has no effect.



0.5 (default)



1.5

Gain

Increase or decrease the slope of the mid-range values. Gain values below 0.5 increase the contrast whereas values above 0.5 flatten the mid-range values. A value of 0.5 has no effect.



0



0.25

Output Min / Max

The range to remap the internal values to on output. No clamping is done by default, see Clamp Min/Max.



0.5

1

Output Min



800

8000

Output Max

Clamp Min / Max

When enabled, the output values will be clamped at•Output Min•and/or•Output Max.

-
-

Volume Sample RGB



The volume_sample_RGB shader will sample a volume channel of the new volume API. It should be connected to one of the components of the volume_standard_shader if you need to further edit the result of the reading of a channel, for instance to color correct it, etc.

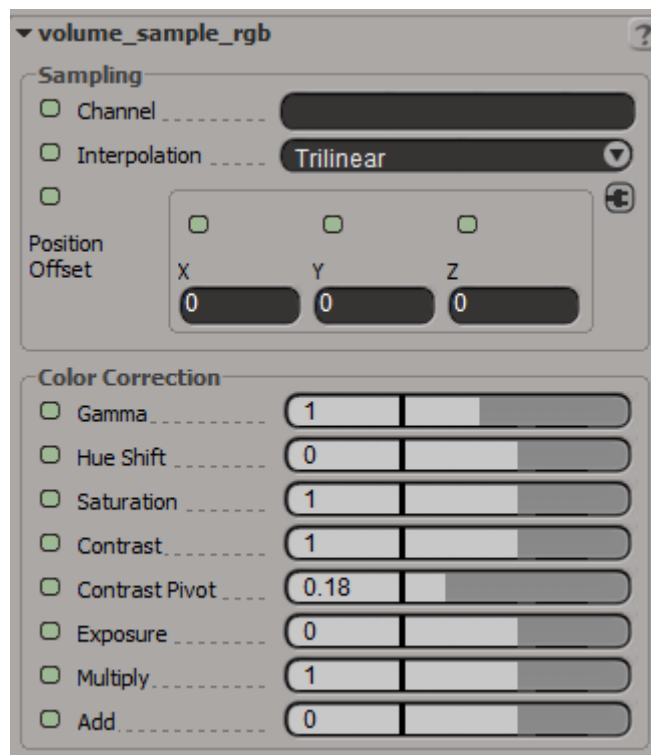


Both the volume_sample_float and volume_sample_RGB shaders share the same sampling controls, the only difference being the type of the output, RGB or float.



These parameters are not texturable due to performance reasons.

Sampling



Channel •

When not empty, this channel will be used to sample the scattering values for the volume instead of the scattering parameter.

Position Offset

An object space offset to apply to the volume sampling position when using named channels to fetch the volume data. This is useful to displace the volume data.



You will need to connect a noise texture to the position offset to see any result.



-10



-5

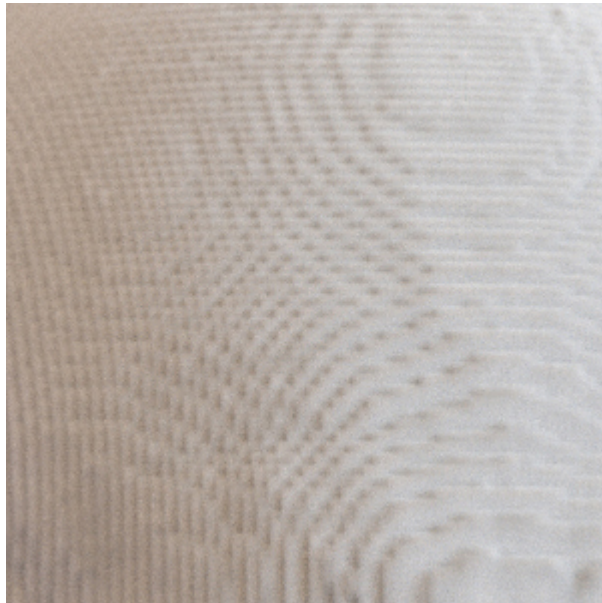


Interpolation

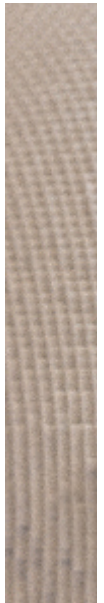
The voxel interpolation to use when sampling the volume data using named channels.



Tricubic



Trilinear (default)



Color Correction

Gamma

Apply a gamma correction to the color.



0.5



1 (default)



Hue Shift

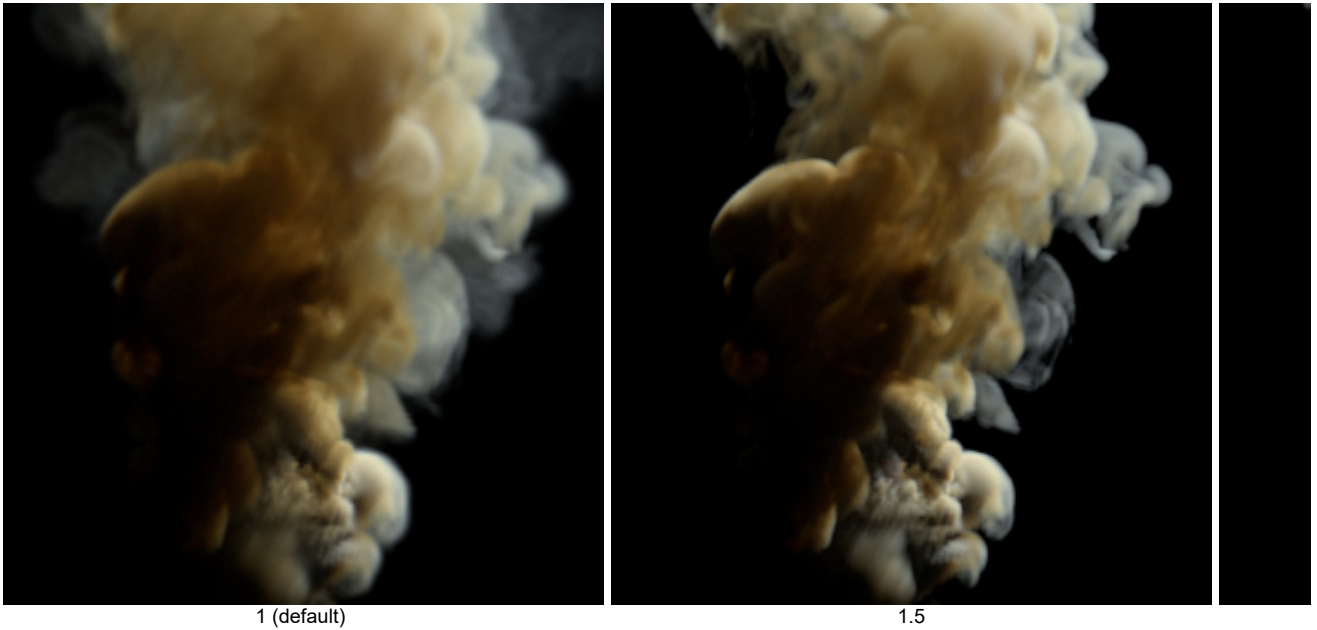
Rotate the color hue, a value of 1 means a full rotation.

Saturation

Scale the saturation by this factor.

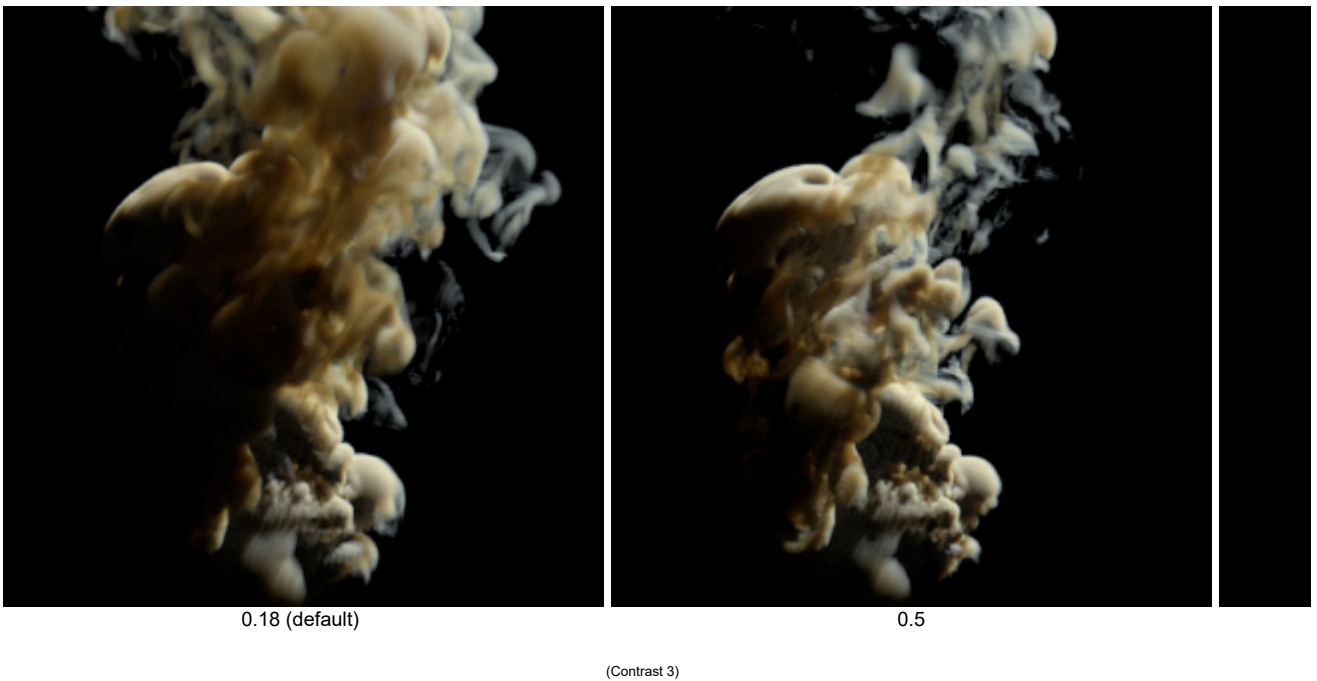
Contrast

Scale values around the `contrast_pivot`.



Contrast Pivot

The origin of the contrast scaling. The default is 0.18, the average perceptual mid-gray.



Exposure

Multiply the color using a photo f-stop scale, where an increment of 1 doubles the luminosity.



5



0 (default)

Multiply

Multiply the color by this factor.



0.1



1 (default)

Add

Add this value to the color.



-0.1



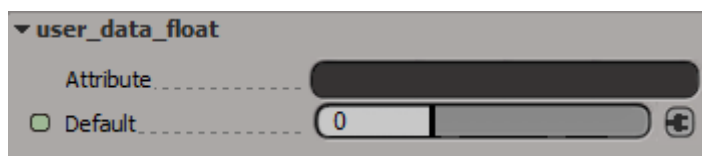
-0.2

User Data Shaders

These nodes can be used to retrieve attributes from the attached geometry. When the scene is translated to Arnold, attributes are not exported automatically. They must be named in the Attributes of the Arnold Parameters. On export, SlttoA translates them into user attributes on the Arnold geometry. To retrieve this data, a node is then required of the correct type. While there are more types than listed below, these should cover the type dimensions, (int, string, 1,3 and 4 floats). In Arnold, the attribute is selected by name only. The nodes below are very accepting of inputs. If, for example, an attribute is named in user_data_rgb but the input passed is actually an integer, then it will still work and just convert the integer to float and pass it to each of the R, G & B values.

It is also possible to read user data fields from volumetric shapes, allowing things like per-particle user data on volumetric spherical point clouds to affect the result of volumetric shading.

User Data Float



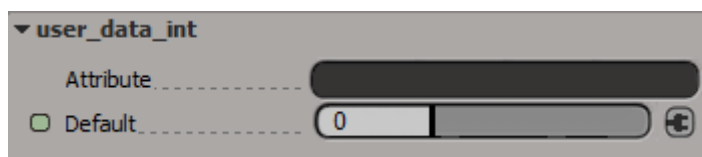
Attribute

Read float value from shape user data, at the current shading point on the surface.

Default

Output value to use if user data with the specified name is not available.

User Data Int



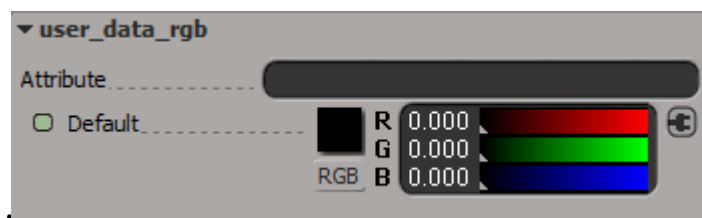
Attribute

Read integer value from shape user data, at the current shading point on the surface.

Default

Output value to use if user data with the specified name is not available.

User Data RGB



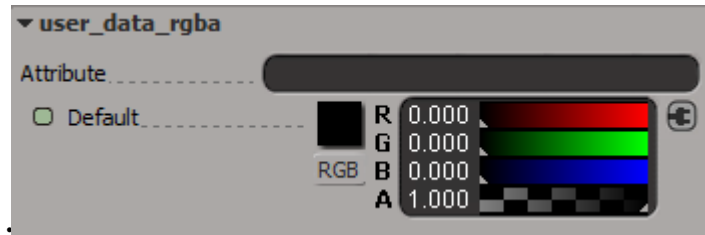
Attribute

Read RGB color from shape user data, at the current shading point on the surface.

Default

Output value to use if user data with the specified name is not available.

User Data RGBA



Attribute

Read RGB color and alpha from shape user data, at the current shading point on the surface.

Default

Output value to use if user data with the specified name is not available.

User Data String



Attribute

Reads the string from shape user data.

Default

Output value to use if user data with the specified name is not available.

-
-
-

Softimage shaders

The following Softimage shaders are supported, and their source code (the right-hand column is the cpp file name) provided in the svn trunk.

..

Illumination

Constant	material_lambert
----------	------------------

Texture ..

Image	txt2d_image_explicit
Scalar Image	txt2d_scalarimage_explicit
Gradient	txt2d_gradient_v2
Color Gradient	sib_color_gradient

Attributes

Boolean	sib_attribute_boolean
Color	sib_attribute_color
Integer	sib_attribute_integer
Scalar	sib_attribute_scalar
Vector	sib_attribute_vector
Transform	sib_attribute_transform

Bump

Bumpmap	sib_zbump
---------	-----------

Color •Channels

Picker	sib_channel_picker
HLSA Combine	sib_hlsa_combine
HSVA Combine	sib_hsva_combine
RGBA Combine	sib_color_combine

Image•Processing

Color Correction	sib_color_correction
Color Balance	sib_color_balance
HLS Adjust	sib_color_hls_adjust
Intensity	mib_color_intensity
Invert	sib_color_invert
RGBA Keyer	rgba_keyer
Scalar Invert	sib_scalar_invert

Mixers

--	--

Mix 2 Colors	sib_color_2mix
Mix 8 Colors	sib_color_8mix

Conversion

Color to Scalar	sib_color_to_scalar
Color to Vector	sib_color_to_vector
Color to Boolean	sib_color_to_boolean
RGB to HSV	sib_color_rgb_to_hsv
HSV to RGB	sib_color_hsv_to_rgb
Integer to Scalar	sib_integer_to_scalar
Scalar to Color	sib_scalar_to_color
Scalar to Integer	sib_scalar_to_integer
Scalars to Vector	sib_scalars_to_vector
Vector to Scalar	sib_vector_to_scalar
Vector Coordinate Convertor	sib_space_conversion
Vector2 To Vector3	Vector2ToVector3
Vector3 To Vector2	Vector3ToVector2

Math

Color Basic	sib_color_math_basic
Color Exponent	sib_color_math_exponent
Color Logic	sib_color_math_logic
Color Unary	sib_color_math_unary
Color Smooth Range	sib_color_smoothrange
Average	mib_color_average
Change Range	sib_interp_linear
In Range	sib_color_inrange
Scalar Basic	sib_scalar_math_basic
Scalar Exponent	sib_scalar_math_exponent
Scalar In Range	sib_scalar_inrange
Scalar Logic	sib_scalar_math_logic
Scalar Unary	sib_scalar_math_unary
Scalar Smooth Range	sib_scalar_smoothrange
Fmod	Math_int_fmod
Boolean Logic	sib_boolean_math_logic
Boolean Negate	sib_boolean_negate
Vector-Scalar	sib_vector_math_scalar

Vector-Vector	sib_vector_math_vector
Linear Falloff	sib_atten_linear
Scalar Curve	sib_scalar_math_curve
Color Curve	sib_color_math_curve

Share

Boolean	sib_boolean_passthrough
Color	sib_color_passthrough
Integer	sib_integer_passthrough
Scalar	sib_scalar_passthrough
Vector	sib_vector_passthrough

Switch

Color Boolean	sib_color_switch
Scalar Boolean	sib_scalar_switch
Vector Boolean	sib_vector_switch
Color Multi-Switch	sib_color_multi_switch
Scalar Multi-Switch	sib_scalar_multi_switch
Vector Multi-Switch	sib_vector_multi_switch
Boolean Array Switch	BooleanSwitch
Integer Array Switch	IntegerSwitch
Scalar Array Switch	ScalarSwitch
Color4 Array Switch	Color4Switch
Vector3 Array Switch	Vector3Switch

Texture Space Generators

Projection Lookup	sib_texproj_lookup
-------------------	--------------------

Map LookUps

Image LookUp	mib_texture_lookup
Map Lookup Boolean	map_lookup_boolean
Map Lookup Color	map_lookup_color
Map Lookup Integer	map_lookup_integer
Map Lookup Scalar	map_lookup_scalar
Map Lookup Vector	map_lookup_vector
Edit_UV	mib_texture_remap
Vertex Color	sib_vertex_color_alpha

Auxiliary Shaders



ImageClip	sib_image_clip
Matte	sib_matte
Texture Layers	sib_texturelayer

-
-

Dll-so shaders

.dll (.so under linux) shaders are those for which only the dll (so) library file is provided, but their ui (either the spdl file or the shaderdef script) is not.

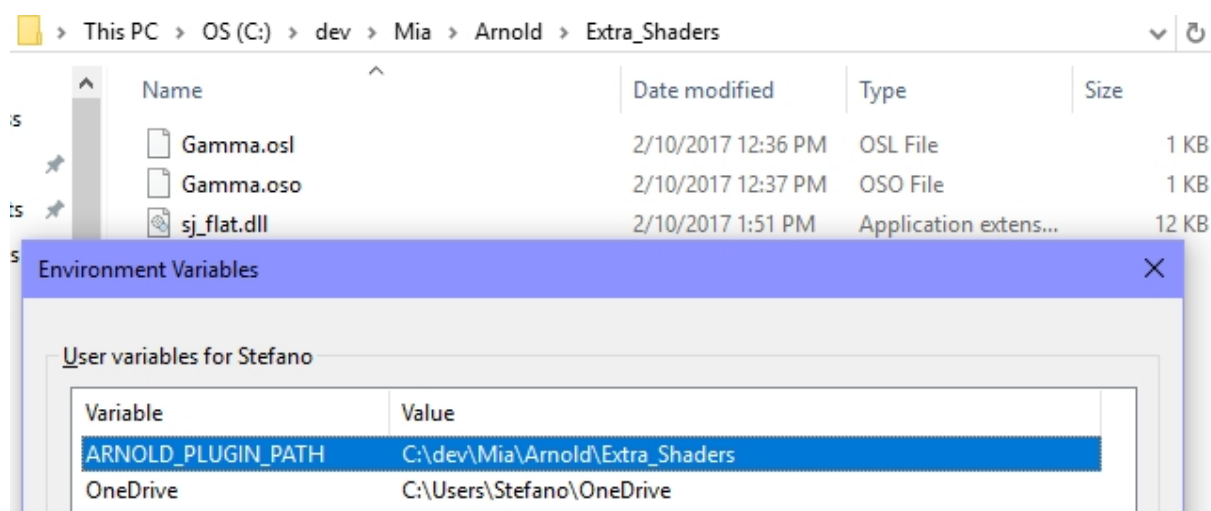
SltoA parses the shaders defined in the library files, and for each it builds on the fly a basic interface with all the parameters set at their default values.

This makes it possible to:

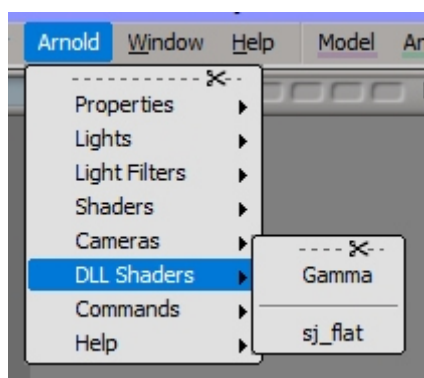
- Use the shaders that come with the MtoA, C4DtoA or HtoA installation.
- Use any 3rd-party shader without needing a dedicated ui file.

The dll/so files must be placed in the same directory as the SltoA dll/so, or under any of the paths pointed by the Plugin SearchPath, or under the directory defined by the ARNOLD_PLUGIN_PATH environment variable.

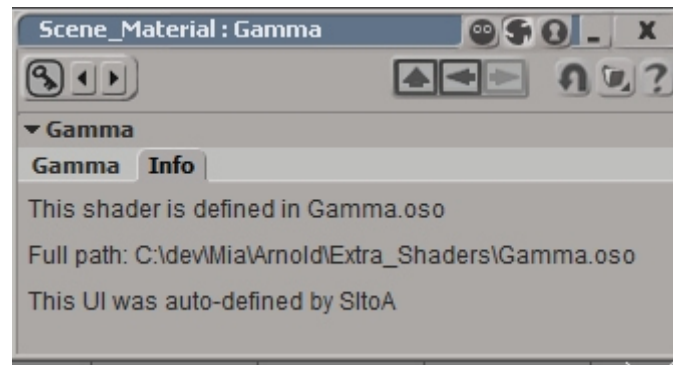
In the example below, I have two shaders, one of which on osl, stored in my ARNOLD_PLUGIN_PATH directory:



On opening Softimage, the shaders are listed under the DLL Shaders category.



For every auto-generated ui, an extra Info tab is also added, to show some additional information on the shader.



-
-



Third Party Shaders

Installation

The recommended way to install third-party shaders and plugins is to use the `ARNOLD_PLUGIN_PATH` environment variable. For example, set the `ARNOLD_PLUGIN_PATH` environment variable like so:

•

```
ARNOLD_PLUGIN_PATH=C:\Program Files\Autodesk\Arnold\maya2020\bin
```

•

Alternatively:

- Copy the content of the bin folder to ...\\Addons\\SltA\\Application\\Plugins\\bin\\nt-x86-64 in windows or ...\\Addons\\SltA\\Application\\Plugins\\bin\\linux inlinux
- That should be sufficient to get the shaders on the Arnold > DLL Shaders menu.
- Copy the SPDLs to ...\\Addons\\SltA\\Application\\spdl
- In Softimage, before you try to use the shaders, generate presets (in the Plug-in Tree, right-click thespdl and click Regenerate Presets)
- In the render tree, you may need to refresh the preset manager to see the shaders.
- If you have already created some instances of the shader and they didn't have the PPGLayout, delete thespdlxsiindexin your \$XSI_USERHOME\\Application folder and start Softimage again.



There are several third-party shader resources available for the Arnold plugins, as listed on the Arnold renderer [website](#).



These third-party shaders, extensions, and integrations are not developed or supported by Autodesk but are provided for your convenience.

Legacy Shaders



These shaders are considered deprecated and will be removed in a future release.

- [Hair](#)
- [Skin](#)
- [Sky](#)
- [Standard](#)
- [Volume Collector](#)

Hair

! This shader is considered deprecated and will be removed in a future release. You should use the Standard Hair shader instead.

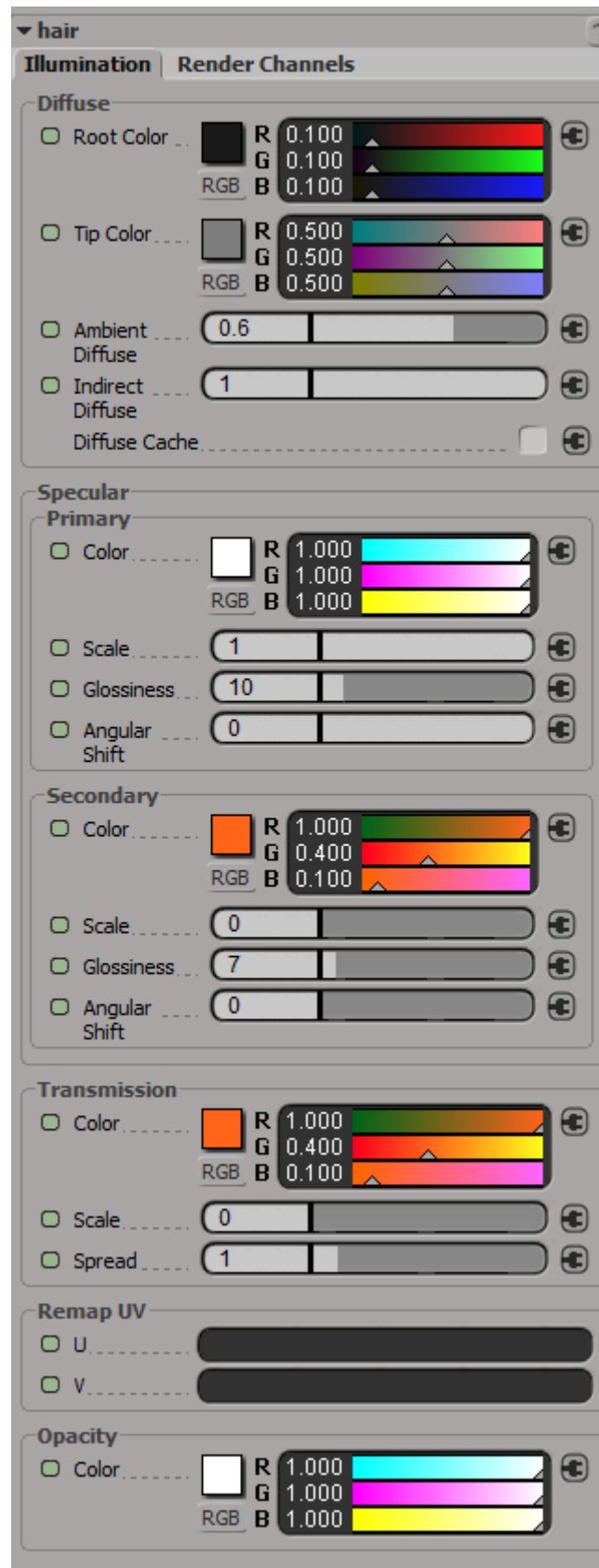


Image credit to Pascal Floerks (rollover image)

The Hair shader is designed to render the look of hair or fur. Note that this shader needs to be used on geometry that has been defined to look like hair or fur (for example using Arnold's `curves` shape node). Other shaders such as Lambert could be used, but the Hair shader has a specular shading model specifically designed for thin hair fibers and lets you define different diffuse colors for the root and tip of the hairs, for example. This shader currently uses a simple Kay-Kajiya shading model.

For further hair options, see [Arnold Curves Parameters](#), and the [Hair Tutorial](#).

Further examples of customer work using the Hair shader can be found [here](#).



- Diffuse

- **Specular**

•

Remap UV

The hair object can be textured, using the classic Softimage workflow.

If you are providing extra UV sets (for example because the hair is being imported as a standin), you can specify in U,V the name of the user data attached to the object that define the U and V coordinates for each point of the curves.

Opacity

Opacity of the hair. This is set to full white by default, which means fully opaque hair. Playing with this value to make the hair less opaque (or in other words, more transparent) can sometimes help make the hair look softer, at the expense of increased render times. If not set to full white, be sure to have the `Opaque` flag off in the `Arnold Parameters` property of the hair object, or else Arnold will bypass any transparency-related calculations to optimize render times.



1



0.5

•

The images below show the separate effect of the root, tip and specular colors. The far right image is the resulting combination of these colors.



Root Color

Tip Color

Specular Color

Beauty

•

Hair - Diffuse

Root Color

Diffuse color at the root of the hair.

Tip Color

Diffuse color at the tip of the hair.



Root Color - purple. Tip Color - blue.

Ambient Diffuse

Ambient Diffuse controls the blending between two different diffuse shading models: fully isotropic and the Kajiya-Kay diffuse BRDF. When set to 0, the diffuse part of the shader will be fully isotropic, which means that the strand will reflect the same amount of light towards the camera no matter which direction the light is coming from. When set to 1, the diffuse part of the shader will follow the Kajiya-Kay diffuse equation, which means that light shining down on the tip of a hair will not brighten the strand as much as light shining in from the sides. Any value between 0 and 1 is also valid, and will just blend between the two models.

Indirect Diffuse

Indirect diffuse controls the amount of indirect diffuse light (GI light rays) reflected by the shader. Setting it to 0 disables indirect lighting and is the fastest to render. The higher the setting, the more that global illumination will affect the final color of the shader. It is strongly recommended to stay in the 0 to 1 range, as values higher than 1 will create energy out of nowhere and therefore break energy conservation and introduce excessive noise. The example below shows an exaggerated value of 5 merely for illustrative purposes.



Diffuse Cache

When enabled, global illumination rays will only see the automatically baked direct diffuse illumination from each control point of the strands rather than a full shader evaluation as they normally would. This can give a big reduction in render times, especially with scenes containing many lights.

-

Hair - Specular

Primary Specular

Color

The color of the primary specular component. For realistic and clean hair, the color should be set to white.



It is also possible to assign texture maps to the specular color attribute. The image on the left is an example of some hair that has been rendered with a brown specular color. On the right is the same hair that has been rendered with a fractal texture connected to the specular color attribute.



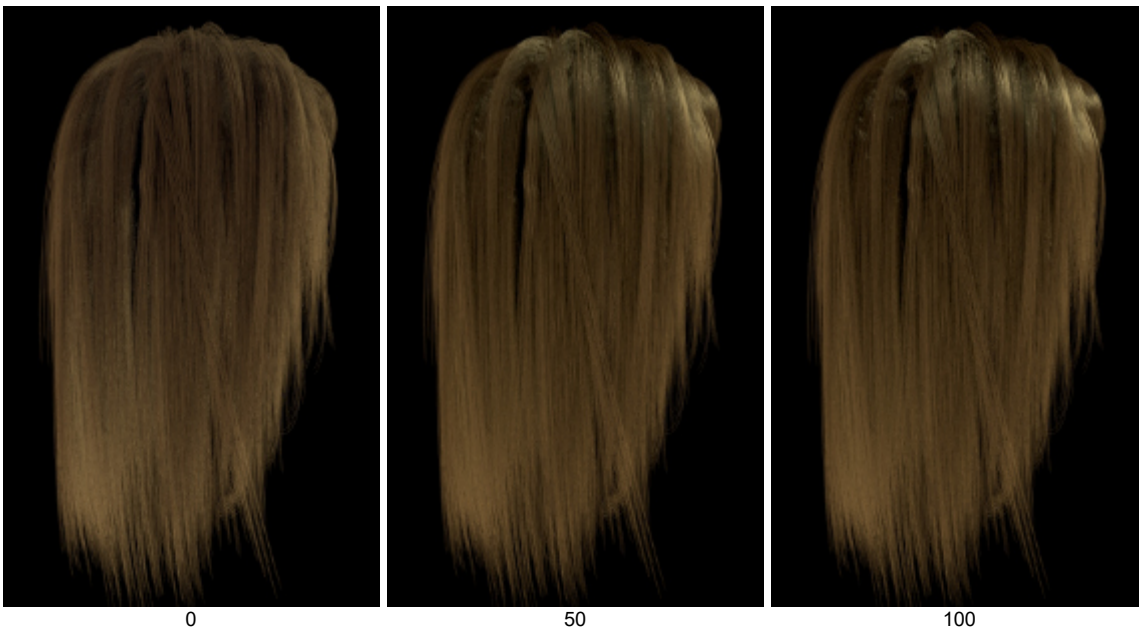
Scale

Scale of the primary specular contribution, which simply multiplies the primary specular color. It is strongly recommended to keep within the 0 to 1 range. In particular, values above 1 will create energy out of nowhere and break energy conservation and possibly introduce more noise than necessary.



Glossy

Glossiness of the primary specular component. The higher the value, the tighter and sharper the specular. Internally, this corresponds to the exponent of a cosine power. For realistic hair, the value should be in the range 5 to 500. When Glossiness is zero, the specular BRDF behaves like a perfect diffuser, reflecting the same amount of light in all directions.



Shift

Angular shift of the primary specular in degrees. It controls the shift of the specular lobes away from the perfect mirror direction and is used to control the offset of the specular highlight along the hair strand. For realistic results, it should take a negative value, typically in the -10 to -5 range.



Specular Angular Shift -1. Secondary Specular Angular Shift 1.5



Specular Angular Shift -10. Secondary Specular Angular Shift 7.5

••

Secondary Specular

The **Hair** shader has a secondary glossy specular lobe with its own Scale, Color, Glossiness and Shift controls. The secondary specular tends to shift towards the root of the hair, which is achieved with a positive value in the Shift control. Even though in real hair the secondary shift is linked to the primary shift with the expression $\text{secondary shift} = \text{primary shift} * -1.5$, the specular shift controls are decoupled for greater artistic control.



The secondary specular usually shifts 1.5 times as much towards the root as the primary specular. It is also wider than the primary specular. The Scale of the secondary specular defaults to 0 to preserve behaviour when using materials from earlier versions.

•





Primary Specular

Secondary Specular

Color

The color of the secondary specular component. For realistic and clean hair, the color is usually non-white, tinted by absorption inside the hair fiber.

Scale

Scale of the secondary specular contribution, which simply multiplies the secondary specular color. It's strongly recommended to stay in the 0 to 1 range. In particular, values above 1 will create energy out of nowhere and break energy conservation and possibly introduce more noise than necessary.

Glossy

Glossiness of the secondary specular component. The higher the value, the tighter and sharper the specular. Internally, this corresponds to the exponent of a cosine power. For realistic hair, the value should be around half the primary specular glossiness, i.e. it's slightly rougher due to scattering when light goes through the hair fiber.

Shift

Angular shift of the secondary specular in degrees. It controls the shift of the specular lobes away from the perfect mirror direction and is used to control the offset of the specular highlight along the hair strand. For realistic results, it should take a positive value. A suggested expression is to link the primary and secondary specular shift values together like: $\text{secondary specular shift} = -1.5 * \text{primary specular shift}$.

Hair - Transmission

An energy conserving, Marschner-style transmission component (also known as TT). Since this models light shining through hair before getting to the camera, the hair must be between the camera and the light source for the transmission effect to be visible. This hair shading component is more important when doing rim lighting on hair or fur.



No Transmission



Transmission

Transmission Color

The color of the transmission component.

Transmission Spread

A multiplier on the transmission effect's spread: a smaller value produces a more focused, brighter effect. The default spread is 1.0, which corresponds to a realistic look for blonde hair. Reasonable values should be within the 0.5 to 5.0 range.



1



3

Skin

! This shader is considered deprecated and will be removed in a future release. You should use the Standard Surface shader instead.



The `skin` shader node is provided to give an easy to use generic skin shader. Further examples of customer work using the Skin shader can be found [here](#).

The parameters are presented in several different groups:

- [SSS Layers](#)
- [Specular Layer](#)
- [Opacity](#)
- [Advanced](#)
- [Skin AOV Names](#)

i A tutorial that covers how to use the Skin shader can be found [here](#).

Skin Layers

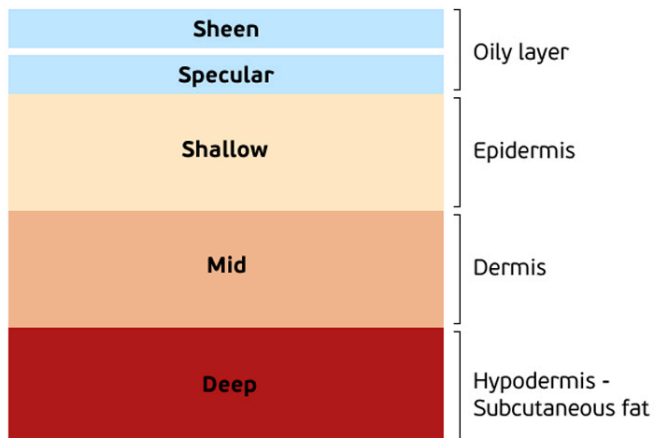
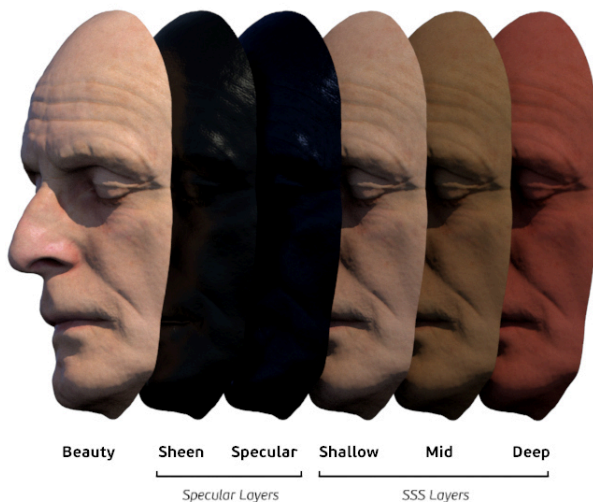


Sheen Layer



Specular

As you can see, the shader is broken up into different SSS layers ('Shallow Scatter', 'Mid Scatter' and 'Deep Scatter'), each one with its own scattering radius, color, and weighting factor. There are also two separate specular/reflection layers.



Presets

The Skin shader can also be used to create a wide variety of materials such as wax, clay, marble, milk and plastics. Below are some more examples of different materials that can also be achieved with the Skin shader using different presets. Click on the images below to view the Skin settings:



Candle Wax

Clay

Jade

Marble



Milk

Cloudy Glass

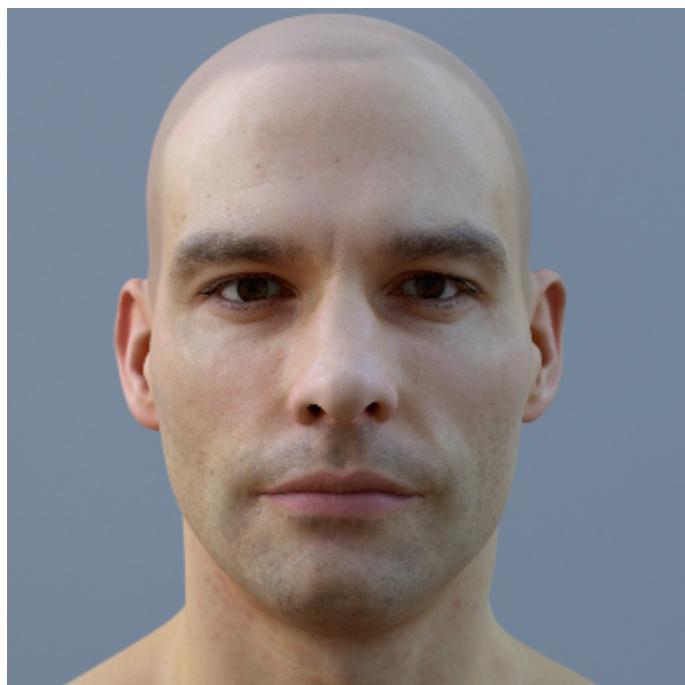
Skin

Wax



Note that **Skin** is not part of the Arnold core, but is an Arnold shader node provided with end-user plug-ins such as SltToA and MtoA to aid usability.

-
-



Further Skin shader examples

SSS Layers

SSS Weight

Specifies the weight of the SSS contribution.**



0



0.5

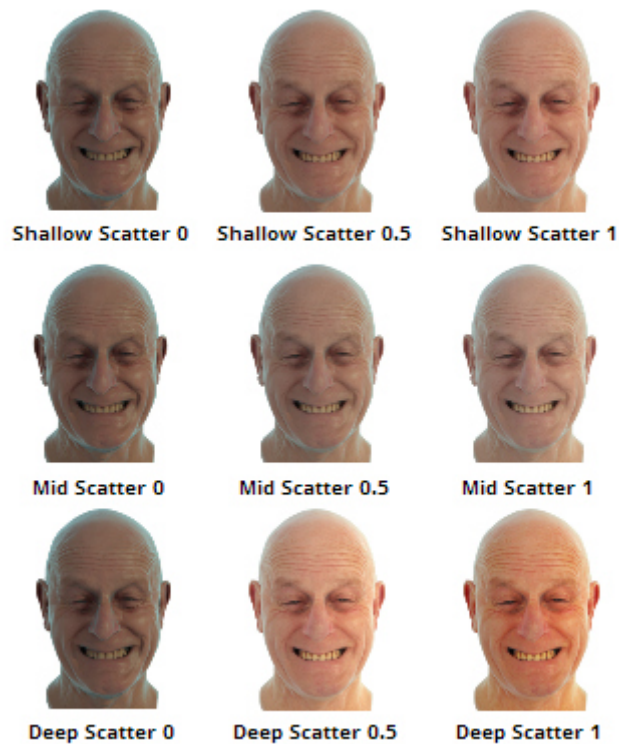
Shallow Scatter, Mid Scatter, Deep Scatter

For each of the three layers, the following controls are provided:

- Color
- Weight
- Radius

This allows you to provide different effects near the surface of the skin, at a deeper layer of skin, and behind the skin (you will see from the default colors for example that a red color for the deep scattering picks up the red color from blood beneath the skin, with more typical flesh tones used for the other two layers), and control how much contribution there is from each.

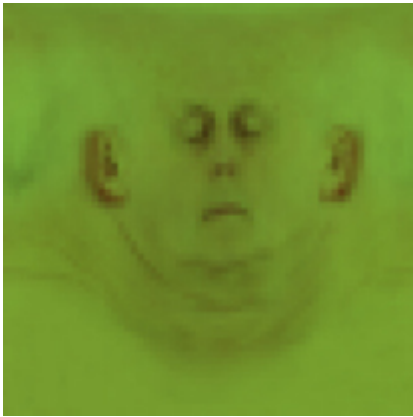
In general you should keep the shallow radius low in order to emulate a diffuse shading effect. The Mid and Deep scatter layers can then be used to add more SSS.



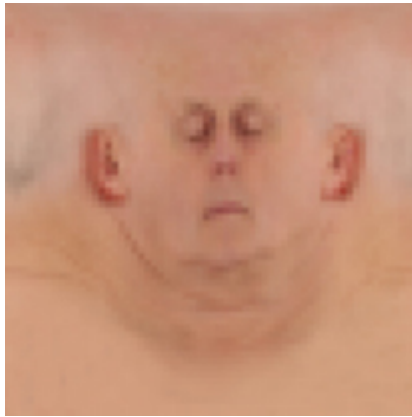
SSS Layers and Texture Maps



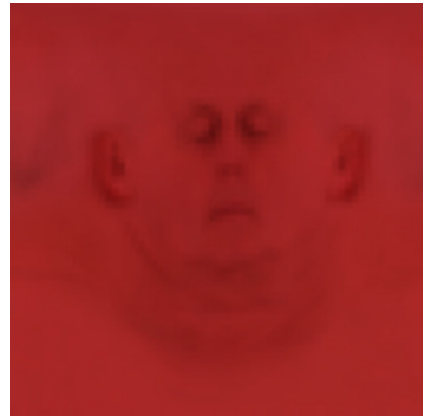
You may also want to experiment with adding some extra textures to the scatter colors. These were made by desaturating and gaining the color to pull out the dark areas, like his eyebrows and stubble. This was multiplied over flat colors of the values from the shader. Obviously a lot more time could be spent here to add a lot more detail in the subsurface and the specular to make him look more realistic. In order to represent the lower dermal skin layer, the Deep Scatter image has been gaussian blurred and has less frequency detail.



Shallow Scatter tinted with green color



Mid Scatter



Deep Scatter has been blurred and tinted with a red color

The•Epidermal SSS• texture is connected to the•Shallow Scatter •attribute.

The•Upper •dermal SSS •texture•is connected to the•Mid Scatter •attribute.

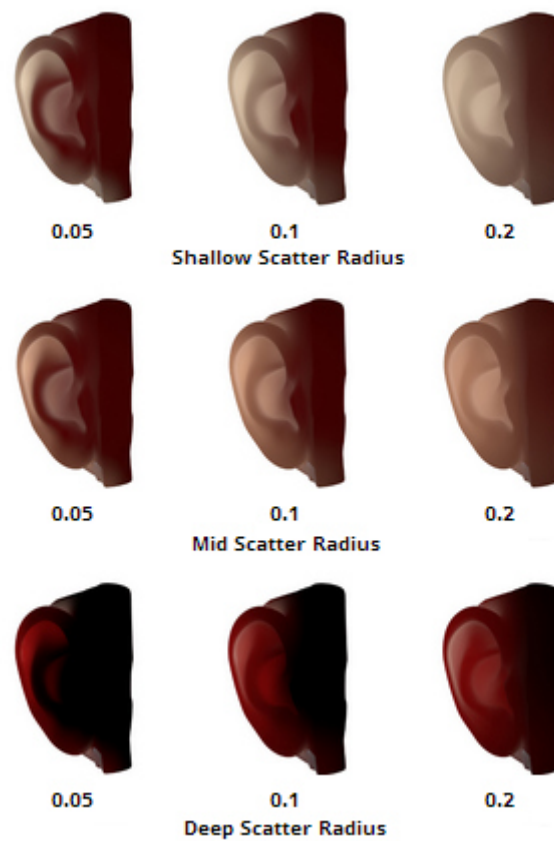
The•Lower dermal SSS •texture•is connected to the•Deep Scatter •attribute.



Modified colors for Shallow, Mid and Deep scatter layers

Scatter Radius

Radius in which light travels through the surface of the skin before exiting. Lower values produce less scattering and the skin can appear more opaque. Higher values give a larger sense of translucency, similar to wax.



The SSS radius is scale dependent and works best with real-world units. You can see the effect that 'scatter radius' has on a mesh with different scale sizes below:



Head mesh scale 0.1

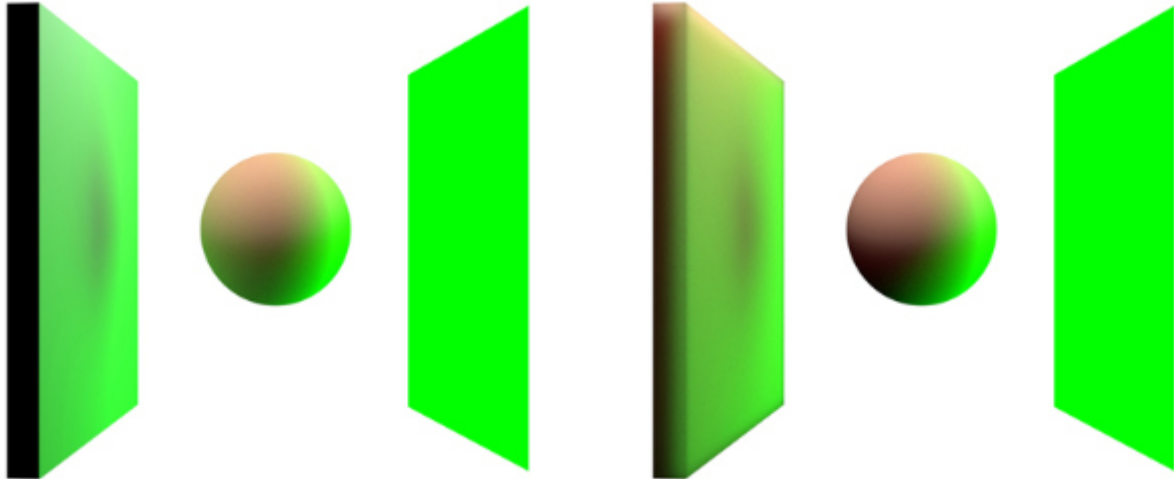


Head mesh scale 10

-
-
- SSS and indirect rays •

SSS is not evaluated in indirect rays traced by the SSS. Indirect rays fired by the SSS engine from one object that ends up hitting another object will return black. This is in part legacy inherited from the point-cloud based SSS engine, and in part a fundamental limitation of diffusion based BSSRDFs (which their inventor, Henrik wann Jensen, always warned were only exact for infinite, semi-infinite media and not for curved/thin surfaces etc). We have investigated how to fix this limitation of BSSRDFs but so far our efforts have not been successful. A workaround is to connect a Standard material to the diffuse attribute of a [Rayswitch](#) shader.

In the example below, you can see that green light is bounced from the cube on the left onto the sphere. However, no light is bounced from the cube when a Skin shader is assigned to it.



Standard shader assigned to cube on left. Green light is bounced onto left side of sphere.

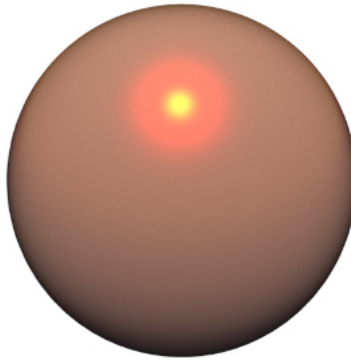
Skin shader assigned to cube on left. No light bounces onto the sphere from the skin.

Specular Layer

The 'Specular' and 'Sheen Layer' settings are the same. It is recommended that you use 'Sheen Layer' for tight specular reflections and 'Specular' for broader specular reflections.

Color

The color the specular reflection will be modulated with.



Sheen Layer (yellow) and Specular Color (red)

Weight

The specular weight.



0



0.5

Roughness

Controls the glossiness of the specular reflections. The lower the value, the sharper the reflection. In the limit, a value of 0 will give you a perfectly sharp mirror reflection, whilst 1.0 will create reflections that are close to a diffuse reflection.



0.1



0.5

Fresnel

IOR

The amount of reflection will be dependent on the viewing angle of the surface following the Fresnel equations (which depends on the IOR value).

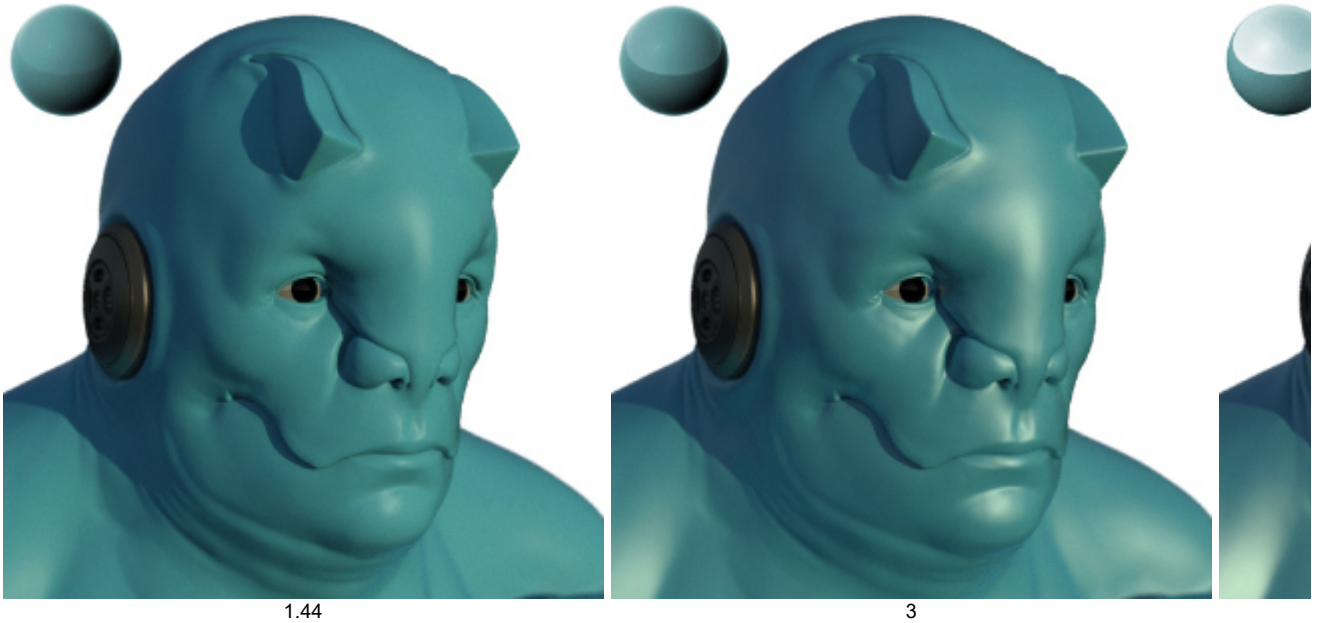


1.44

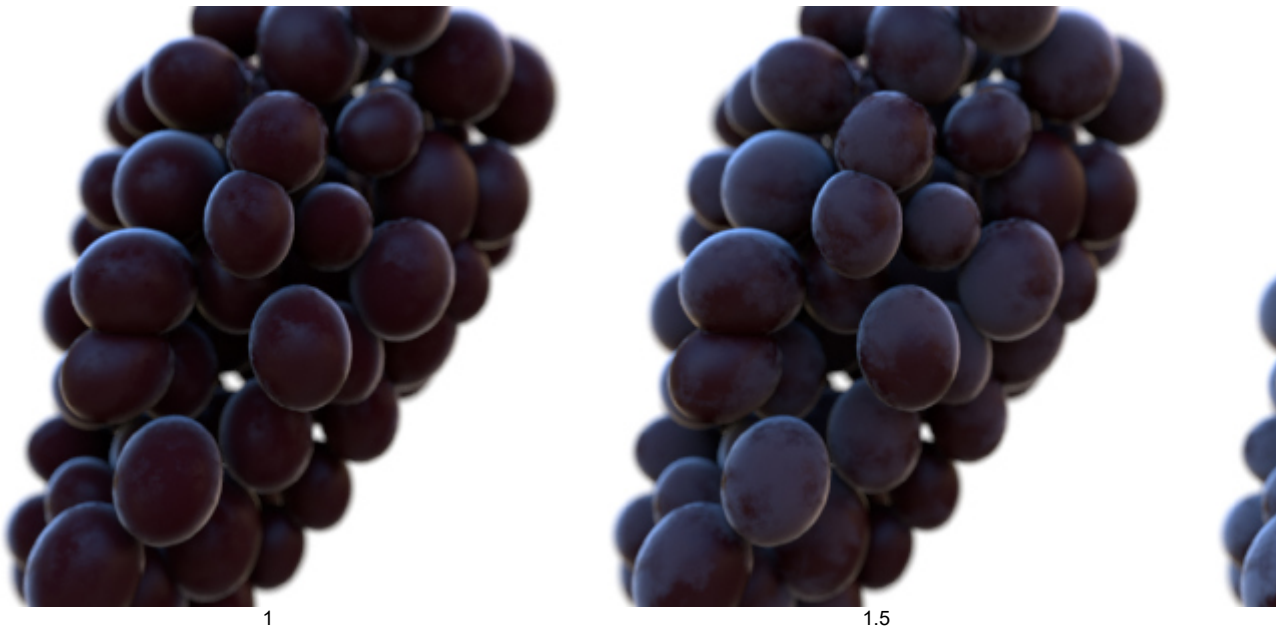


3

The visual effect is that the reflection increases as the viewer's angle of incidence, with respect to the surface normal, approaches 0.



The white powdery look, apparent on the grapes in the images below, is achieved using a high roughness value. Notice the effect that increasing the IOR has on the specular roughness.



There is currently an issue with Specular Fresnel (IOR) which produces a darkening effect around the edges of the Skin shader. This issue is unrelated to bump mapping. However, the effect is more apparent when using bump mapping because it increases the areas where there are grazing angles. This issue is due to be fixed in a future release. A workaround is to disable 'Fresnel Affects SSS' found within the 'Advanced' options of the Skin shader.



With Specular IOR: Darkening around edge of nostril



Without Specular IOR: No darkening visible



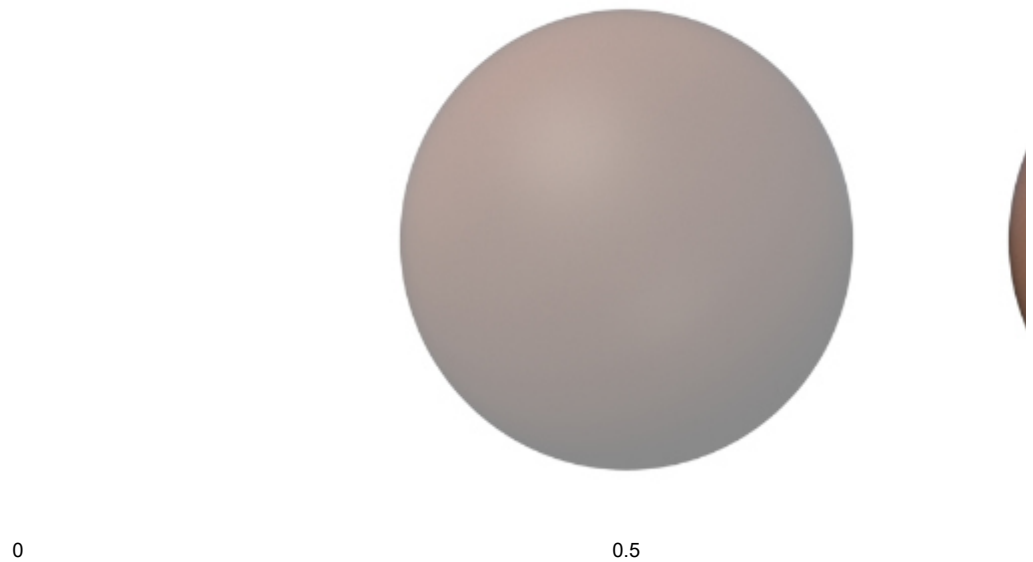
Bump mapping increases darkening effect caused by IOR

-
-

Opacity

Opacity Weight

Sets the opacity of the skin.



Opacity Color

The color or texture used to affect the opacity of the skin. This is set to full white by default, which means fully opaque skin. If not set to full white, be sure to have the **Opaque** flag off in the **Arnold Parameters** property of the skin object, or else Arnold will bypass any transparency-related calculations to optimize render times.

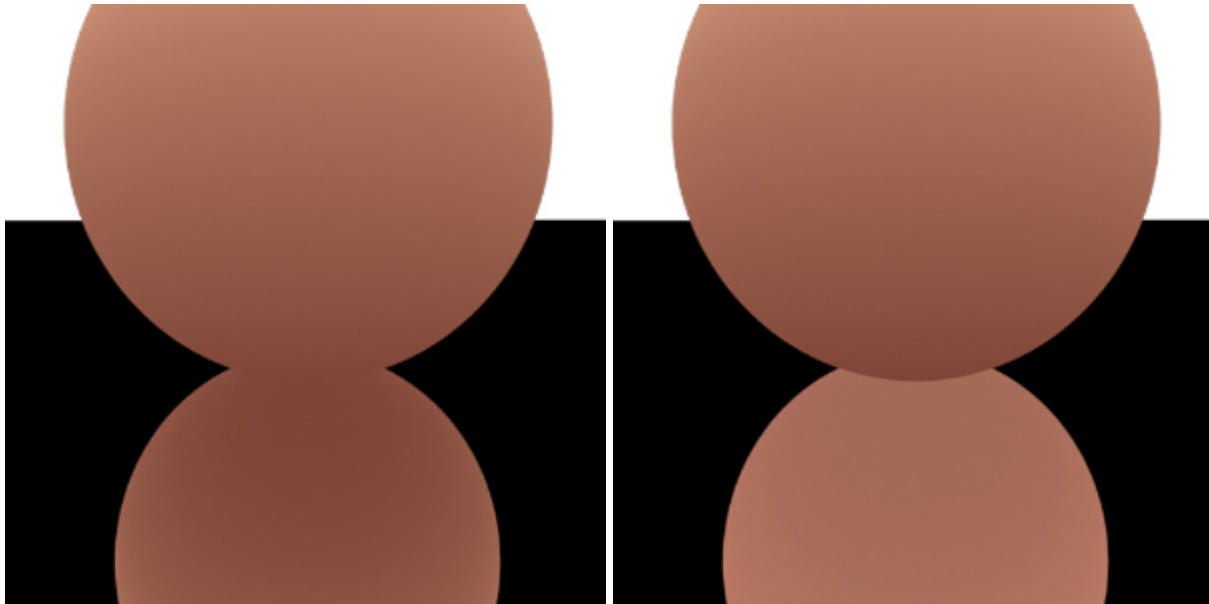


Ramp texture connected to Opacity Color. Black areas of map are non-opaque.

Advanced.

Specular in Secondary Rays

By default 'Specular in Secondary Rays' is disabled, so for diffuse and glossy rays, the Skin shader evaluates the SSS layers only. This makes the shader faster to render. When 'Specular in Secondary Rays' is enabled, the Skin shader evaluates the Specular layers as well as the SSS layers for diffuse and glossy rays.



Sample SSS only in Indirect Specular Rays: Off

Sample SSS only in Indirect Specular Rays: On

Fresnel Affect SSS

Determines whether fresnel computations affect SSS. It is recommended that this should be left on by default.



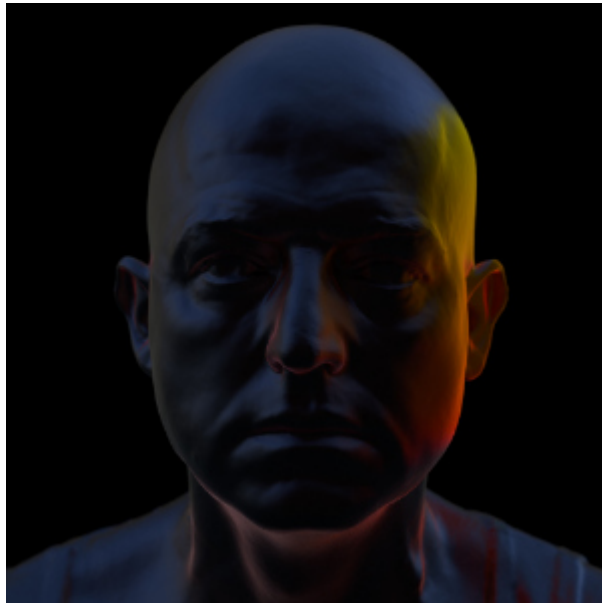
Enabled

Disabled

Skin AOV Names



Beauty



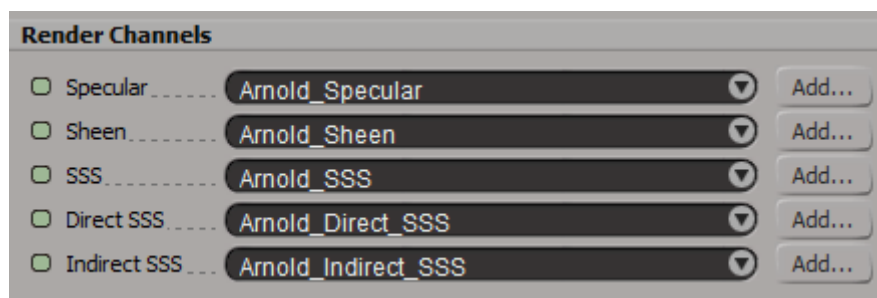
Specular



SSS



Direct SSS



A list of available AOVs associated with the shader. Each entry creates a separate aov render pass for that component. •

- Specular
- Sheen
- SSS
- Direct SSS
- Indirect SSS

•

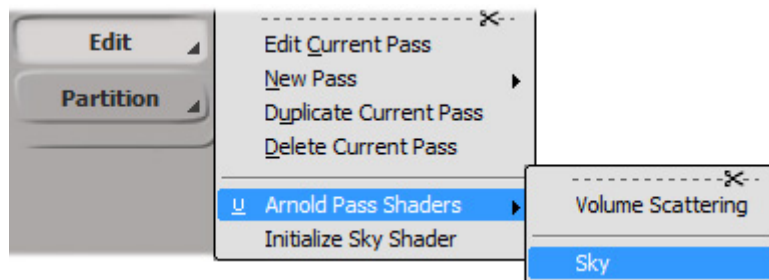
Sky

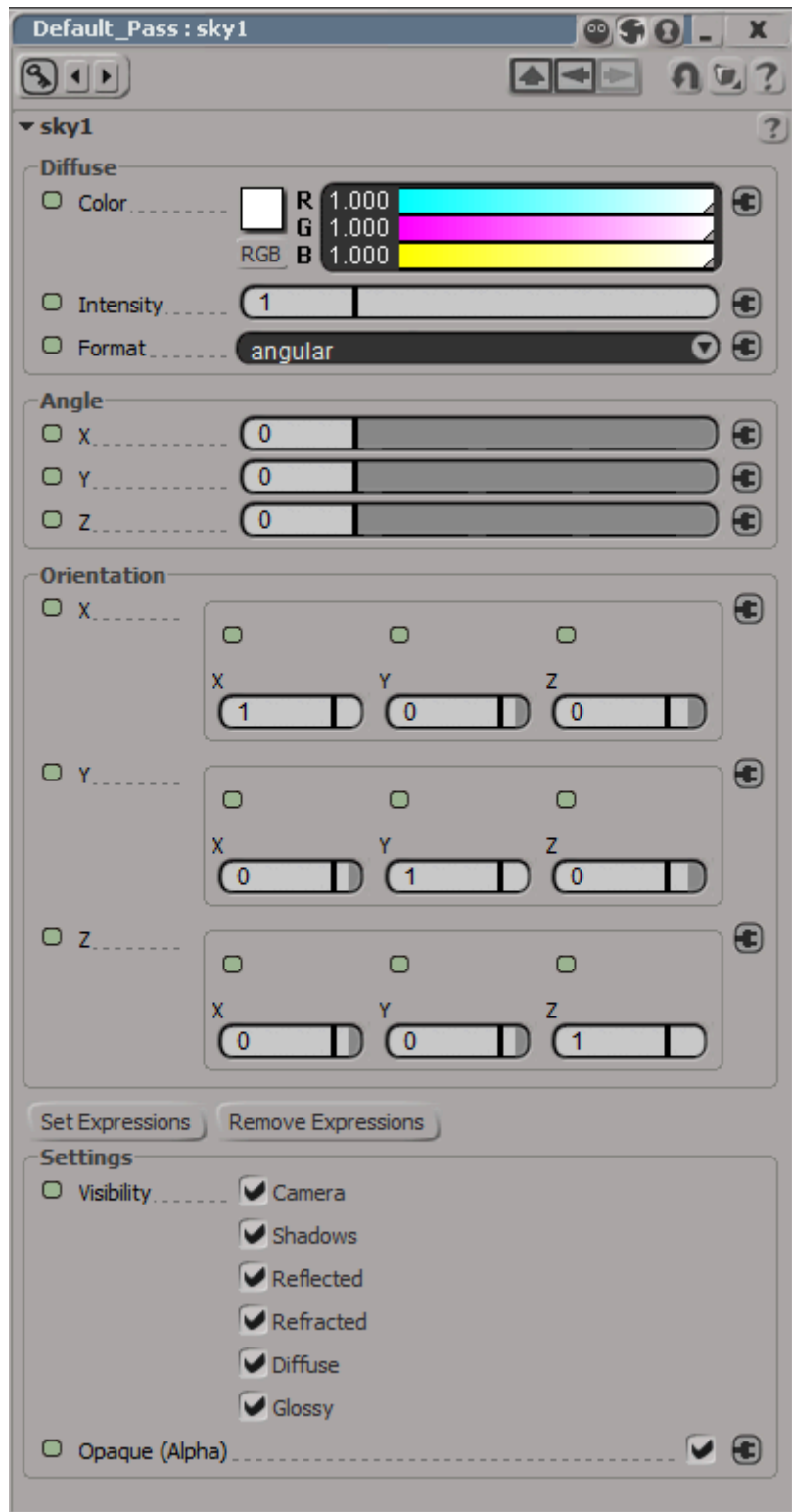
 The Sky shader is considered deprecated and will be removed in a future release. You should instead use the visibility options on the Skydome light instead.

This shader is intended to be used as a global environment shader (or a background shader in Arnold terms), thus affecting all objects in the scene. By default, it has a simple white color and will illuminate the whole scene evenly using this color. For image-based lighting, use the skydome light node instead.

 This shader is deprecated. The skydome light is a better way to provide IBL to your scene, as it supports MIS.

•
One way to set up IBL in Arnold is to simply use the sky shader. • The easiest way to add a sky shader to the current pass is from the "Pass" menu:





Diffuse

Color

The sky color. By default it's white color, but you will probably connect a texture map to it.

Intensity

Scaler of the color

Format

Format of the input texture map (if any). Options are Mirrored Ball, Angular, Lat-Long .

Angle•XYZ

Provides an XYZ orientation for the sky based on euler angles.

Orientation XYZ

The Set Expressions button can be used to link the XYZ axes against an object's orientation. Once picked, the rotation axes of an object is transformed, by a set of expression, into the corresponding axes set.

This way, you can simply rotate the object to interactively adjust the sky orientation.

To unlink the axes from the object, click Remove Expressions .•

Settings

Visibility •

The ray types affected by the shader.

Opaque (Alpha)

Note that all the examples below are rendered with the Sky shader plus one quad area light acting as key light. The Dragon model has around 7 million triangles and it's rendered as it is without any displacement/bump maps. The following sampling settings were used in all the examples:•

- Camera (AA) Samples: 5
- Indirect Diffuse Samples: 3
- Indirect Glossy Samples: 3

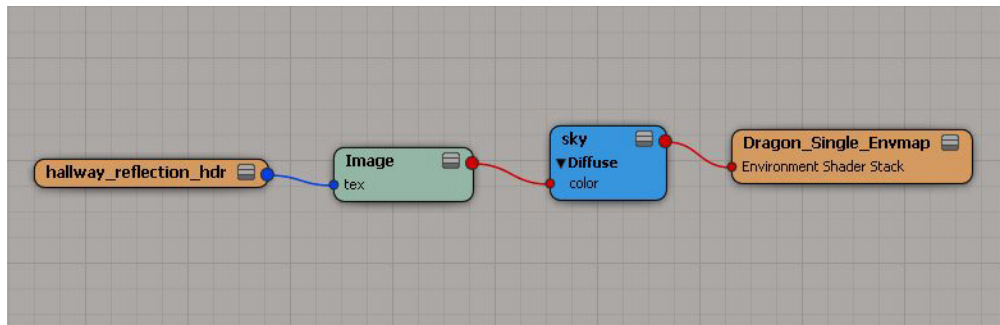
•

Below is a version of the dragon with the sky shader's color set to blue.



Dragon model lit with sky shader set to a flat blue color

If you select the sky shader on the pass and open the rendertree, you can use any image to illuminate the scene by simply hooking up a normal Image node into the Color slot, as shown here:



Rendertree for the sky shader using a HDR image to illuminate the scene

Note that it's important to set the proper mapping format in the sky shader. We will use the following HDR image in latlong format:



Dragon model lit with sky shader using a high-contrast HDR image

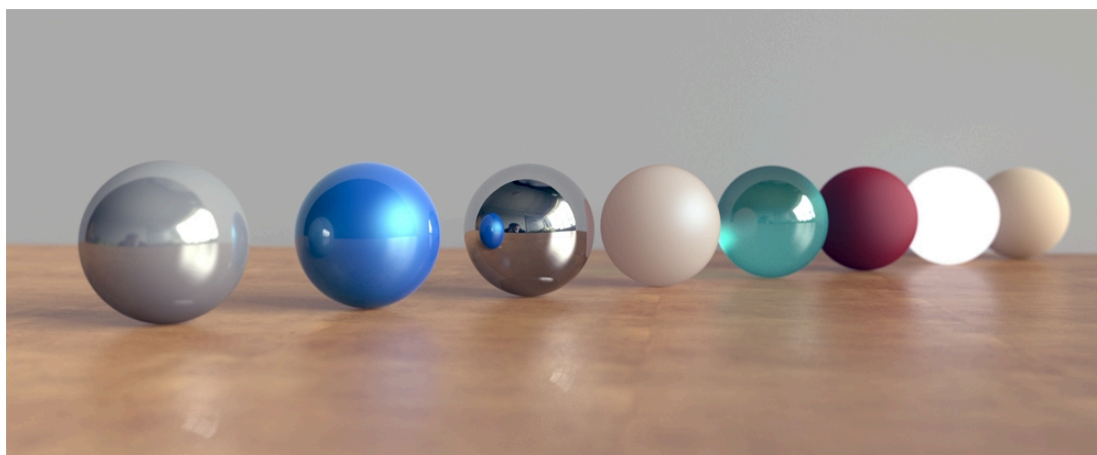
Even though this image is using the exact same GI sampling setting as the previous one, you can see that it has a lot more noise. This is because the environment map has a lot of high contrast areas with bright spots (windows, ceiling lights) and dark shadows, which is difficult to sample.

Standard

! This shader is considered deprecated and will be removed in a future release. You should use the Standard Surface shader instead.

Due to the large number of controls, the Standard shader is presented in SltOa via several tabs, each containing several groups of controls. The individual settings for each group and are described in more detail in the pages below.

- [Diffuse](#)
- [Specular](#)
- [Bounce Factor](#)
- [Reflection](#)
- [Refraction](#)
- [Sub-Surface Scattering](#)
- [Emission](#)
- [Standard AOV Names](#)



The Standard shader is a multi-purpose shader capable of producing all types of materials, from simple plastic, to car paint or skin, with effects like sub-surface scattering or transmittance.

The Standard shader is very powerful, and allows a large number of different sorts of materials to be created, but can be somewhat daunting at first. It's often best to start with a material which has been pre-defined using the Standard shader and make incremental changes to it to get the effect you want, rather than starting from scratch. The Arnold Tutorial manual contains a section of preset material values, with common materials like car paints, chrome, glossy and matte plastic, and others.

Physically Correct Shading

Artists need to create materials that not only 'look good', but are more physically plausible and pay more attention to the physical laws of energy conservation. That is not to say these are hard and fast rules. The key to achieving a believable material is still observation, and being able to convincingly translate what you see into the final shader. However, adhering to these simple rules will give more consistently realistic and believable lighting and shading because a physically based shading model reacts much more like real world lighting.

Energy Conservation

A surface should not return more energy than is being contributed by the incoming light, otherwise the material will no longer be physically accurate. When such overly bright materials bounce too much light it can also lead to 'fireflies'. When light hits an object, the energy is reflected as either the specular (highlight) or diffuse (color) component. The relationship between specular and diffuse is what defines what type of material it is. If 50% of it is diffuse energy then the remaining specular energy must be 50%. If the specular value increases, then the diffuse value must drop and vice versa. For example, chalk has a high diffuse value, with little specular, whereas glass is highly reflective with almost no diffuse. To achieve a photo-

realistic render, the net value of the shaders attributes should not exceed 1. The total energy of reflected light is less than or equal to the energy of the incident light. You should use energy conserving BRDFs and take care to enforce these energy conservation rules when creating materials with the Standard shader.



The term albedo, or reflection coefficient, refers to the diffuse reflectivity or reflecting power of a surface. Albedos (K_d , K_s , K_r , etc.) summing to values above 1 is a bad idea. This means that the Standard shader will not conserve energy, but will instead gain energy with every GI Diffuse bounce. They should not sum above 1 EXCEPT if you are using a Fresnel value that affects both Specular and Diffuse because when set to affect Specular and Diffuse, it scales the Specular effect by a value between 0 and 1 that depends on the viewing angle.

For energy conservation with Diffuse and SSS, you must have $\text{Diffuse} + \text{SSS} \leq 1$

With Specular, it depends on whether or not Fresnel is enabled:

- If Fresnel is disabled, then you must have $\text{SSS} + \text{Specular} + \text{Diffuse} \leq 1$
- If Fresnel is enabled, then you must only have $\text{Diffuse} + \text{SSS} \leq 1$. The shader will mix specular with diffuse and SSS in a way that is energy conserving.

Ok, that is the simple case (without diffuse backlighting and refraction). Now let us add diffuse backlighting and refraction:

For energy conservation with diffuse and SSS, you must have $\text{Diffuse} + \text{SSS} + \text{Backlighting} \leq 1$

With Specular, it depends on whether or not Fresnel is enabled:

- If Fresnel is disabled, then you must have $\text{Specular} + (\text{SSS} + \text{Diffuse} + \text{Backlighting}) \leq 1$
- If Fresnel is enabled, then you must only have $\text{Diffuse} + \text{SSS} + \text{Backlighting} \leq 1$. The specular and refraction weights can be anywhere between 0 and 1, and the shader will mix everything in a way that is energy conserving.

The example below shows the difference when rendering using physically correct values (left) compared to a shader that uses physically incorrect values above 1 (right image. The light leaving the surface is brighter than the light which originally fell upon it). No material bounces back light at 100%, except for a perfect spotless mirror.

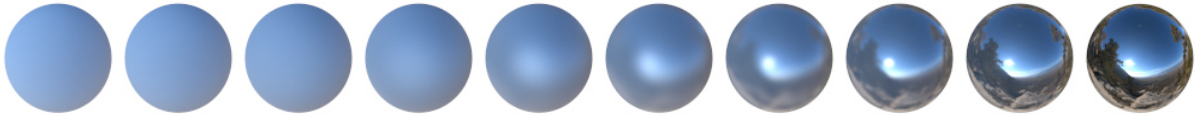


Diffuse Weight: 0.7, Specular Weight: 0.3 (totalling 1 is correct)



Diffuse Weight: 0.7, Specular Weight: 1 (totals above 1. Specular highlight appears blown out)

A Standard shader with a high Diffuse Roughness will reflect dimmer and broader highlights, while smoother and more reflective materials will reflect brighter and tighter highlights.

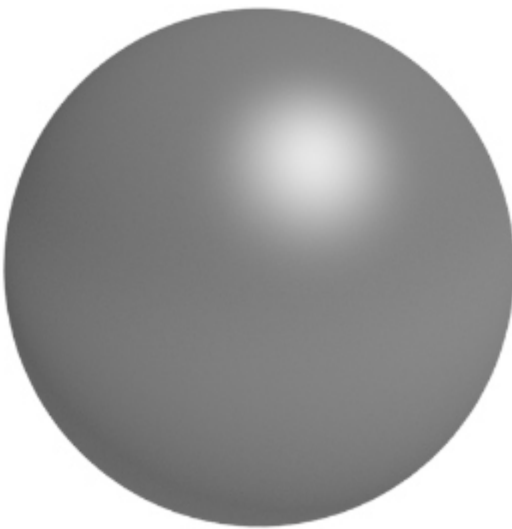


Diffuse and rough (left) to reflective and glossy (right).

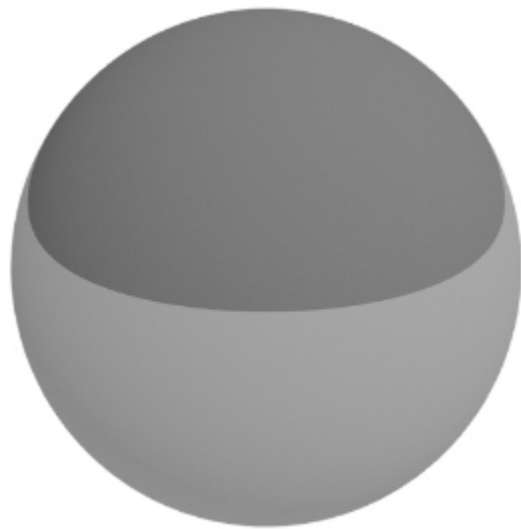
• Specularity vs Reflection

It is generally recommended that you use **Specular** to control glossy specular reflections. The `specular_roughness` parameter of the Standard shader affects the blurriness of the blurry specular reflections. In Arnold, this is called "glossy reflections", or also "specular reflections".

•
The standard shader has a secondary layer of reflection, called simply "**reflection**", which is perfectly sharp and mirror-like. •It is only there for perfect mirrors or glass. Not for blurry effects. •If you want blurry reflections only you should use **Specular**.

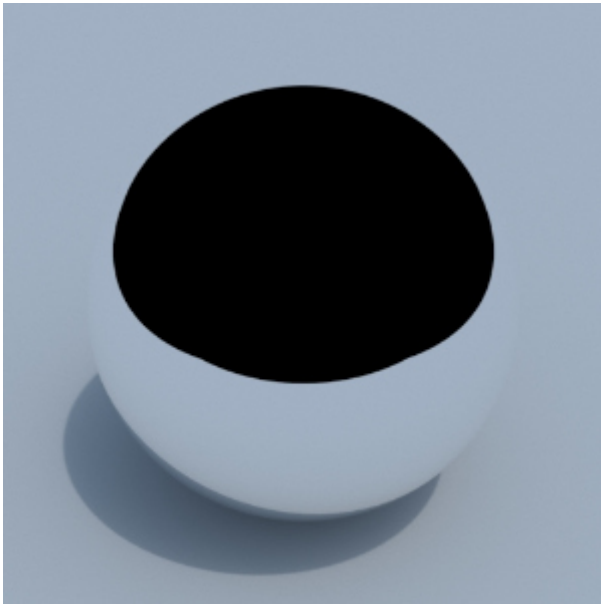


Specular only

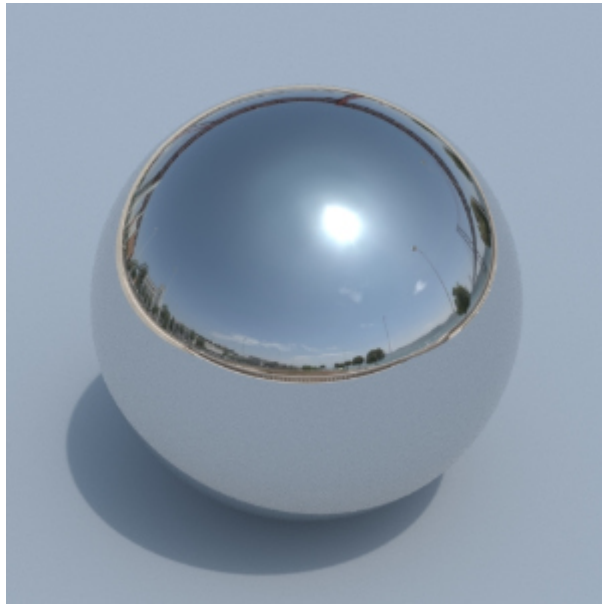


Reflection only

•
The **Sky Dome light** will also not appear in reflections of the **Standard** shader. It is recommended to use **Specular** in combination with the Sky Dome light.



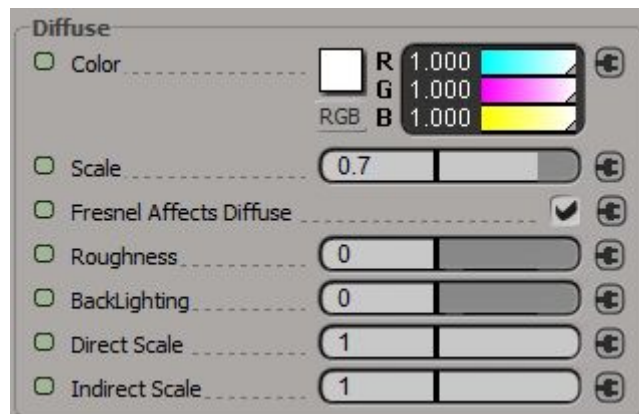
Specular 0. Reflection 1.



Specular 1. Reflection 0.

 Further information about physically based rendering in Arnold can be found [here](#).

Diffuse

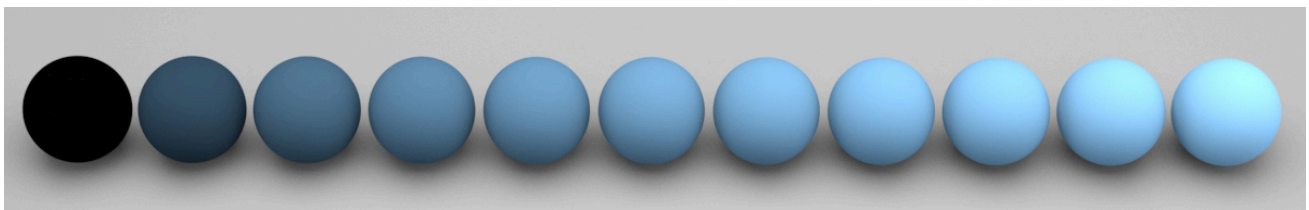


Color

The diffuse color sets how bright the surface is when lit directly with a white light source (intensity at 100%). It defines which percentage for each component of the RGB spectrum which does not get absorbed when light scatters beneath the surface. Metal's normally have a black or very dark diffuse color, however, rusty metal's need some diffuse color. A diffuse map is usually required.

Scale

The diffuse weight.



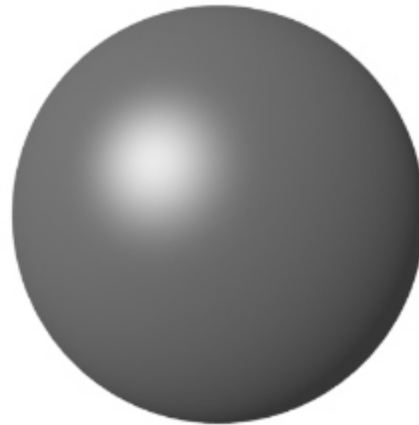
Diffuse 0-1

Fresnel affects Diffuse

Specify whether Fresnel affects the diffuse component.



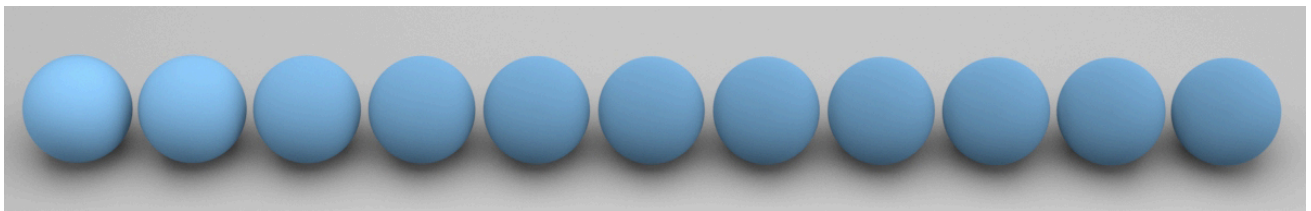
Disabled



Enabled

Roughness

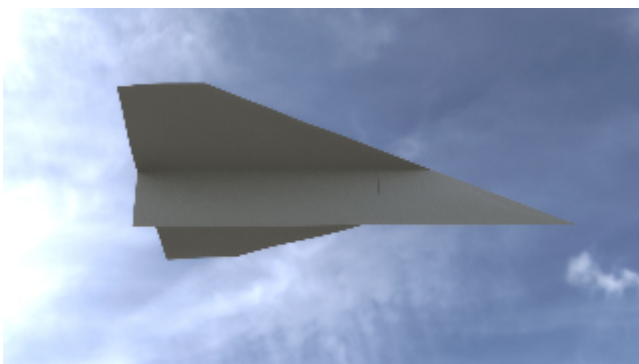
The diffuse component follows an Oren-Nayar reflection model with surface roughness. A value of 0.0 is comparable to a Lambert reflection. Higher values will result in a rougher surface look more suitable for materials like concrete, plaster or sand.



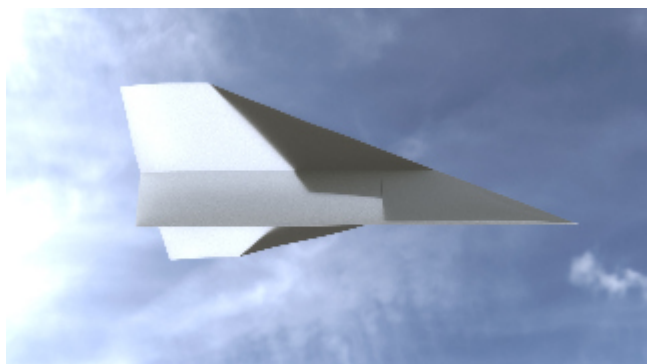
Roughness 0-1

Backlighting

Backlight provides the effect of a translucent object being lit from behind (the shading point is 'lit' by the specified fraction of the light hitting the reverse of the object at that point). It is recommended that this only be used with thin objects (single sided geometry) as objects with thickness may render incorrectly.♦♦

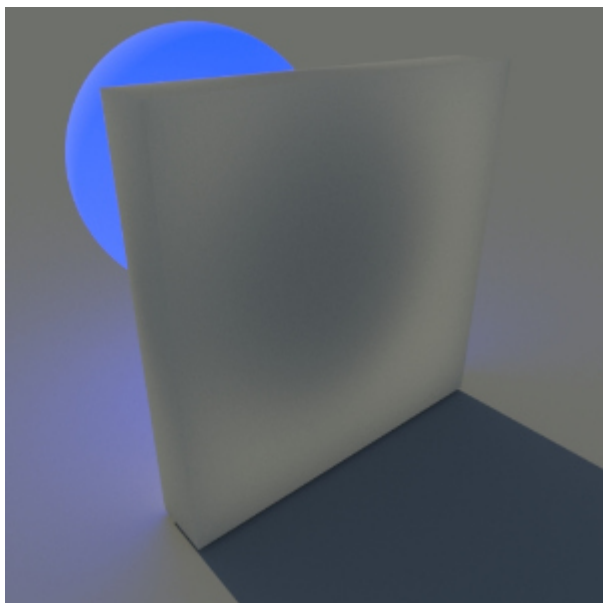


0

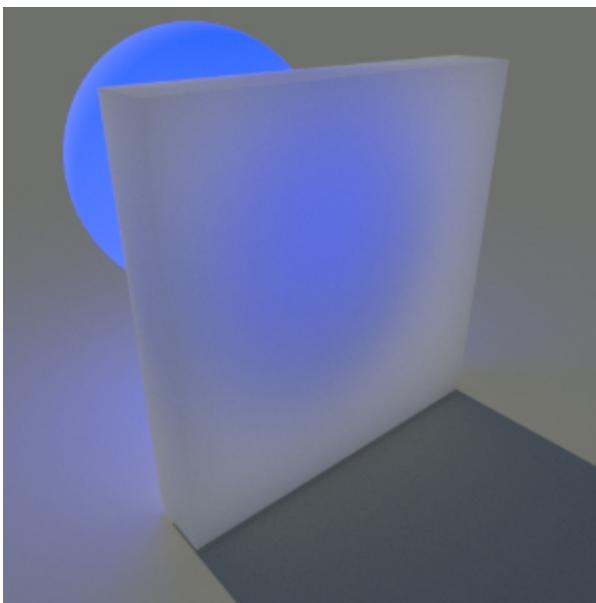


1

In certain situations backlighting may work fine with thickness (ensure that the diffuse ray depth level is above 1).



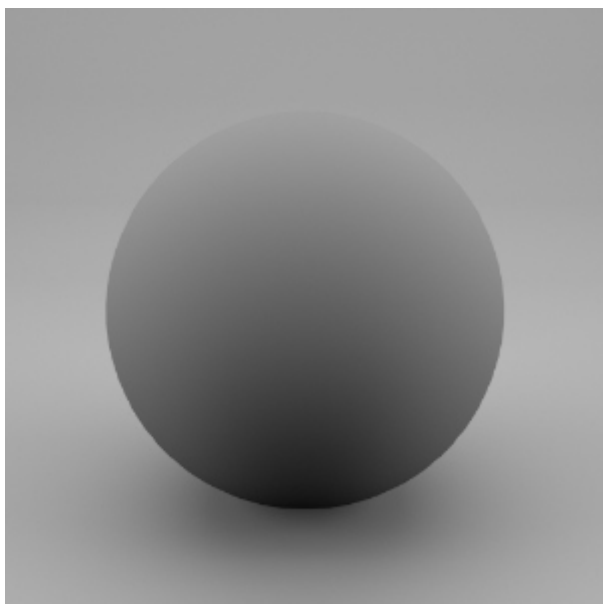
Diffuse Ray Depth 1



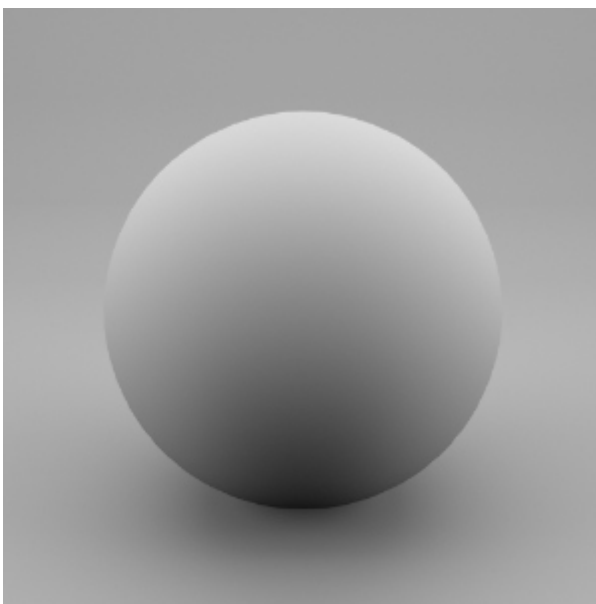
Diffuse Ray Depth 2. Blue emission is visible in backlighting.

Direct Scale

The amount of diffuse light received from direct sources only. •



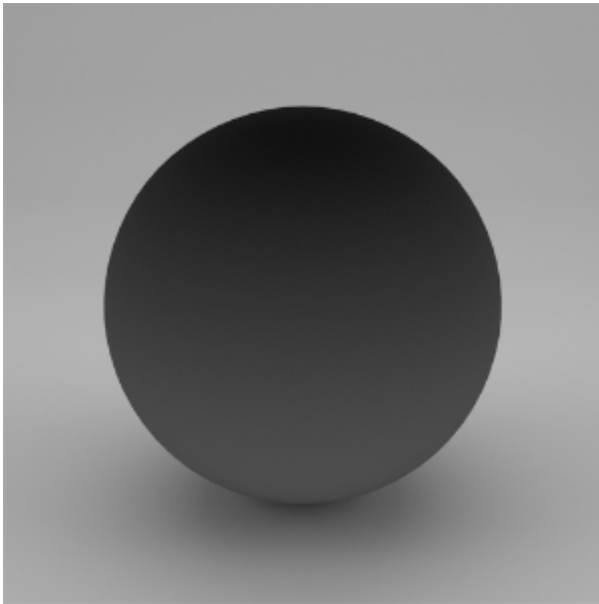
0.5



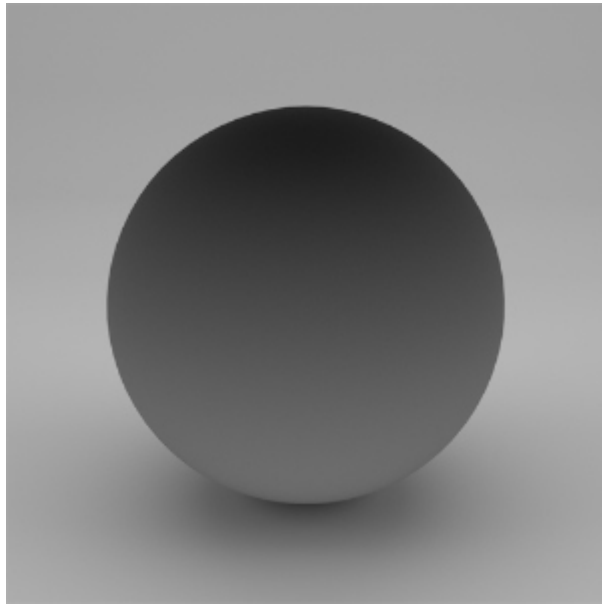
1

Indirect Diffuse

The amount of diffuse light received from indirect sources only. •



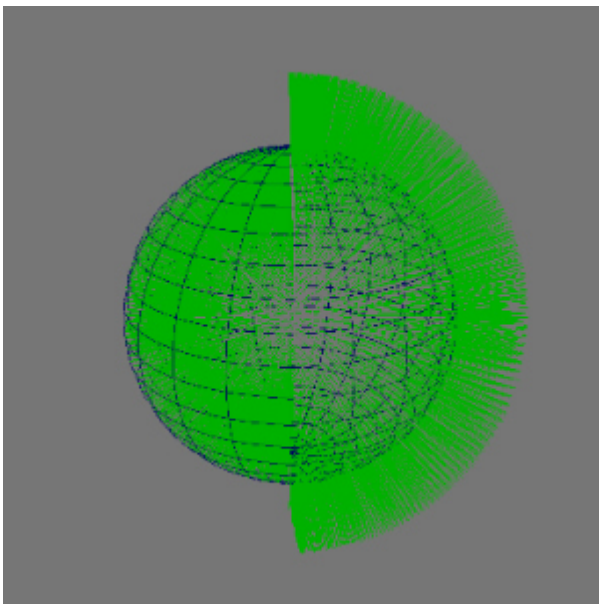
0.5



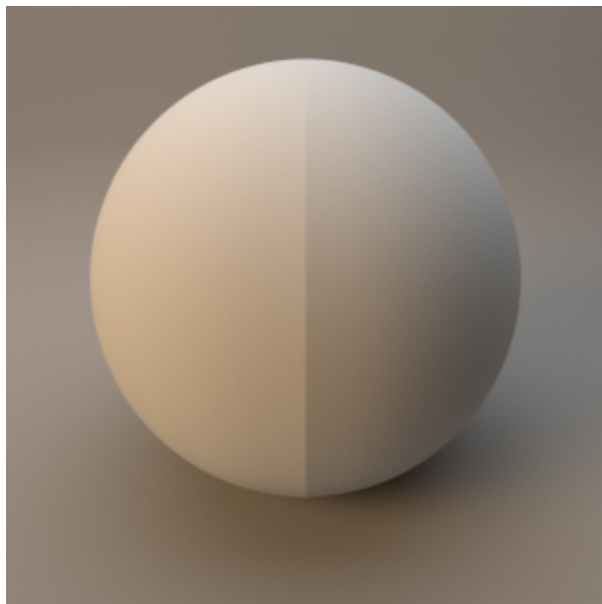
1

Surface normal direction

When rendering diffuse surfaces it is very important that the normals of the geometry face in the right direction. In the example below you can see the difference between normals that are facing inwards in the wrong direction (left) versus those that are facing correctly in the outwards direction (right).



The normals are reversed on the left side of the sphere

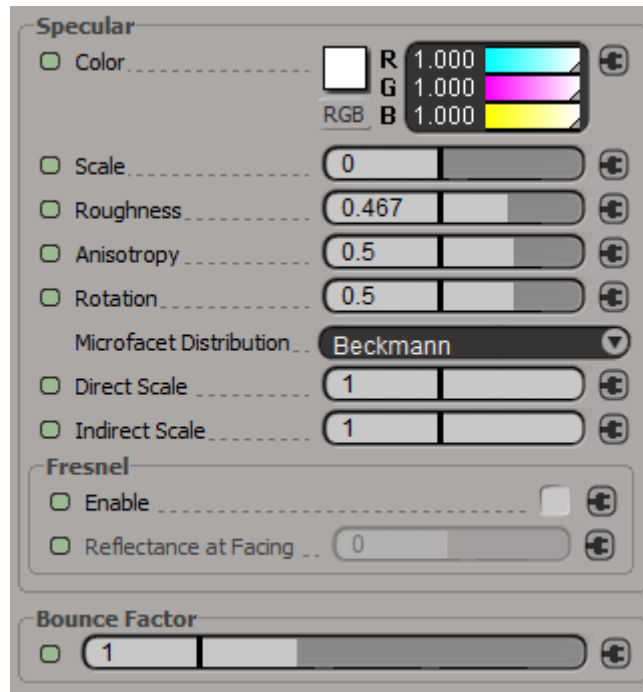


The right side of the sphere renders correctly

Specular.

Specular is direct and indirect reflections which can be made glossy (blurred).

•



Color

The color the specular reflection will be modulated with. Use this color to 'tint' the specular highlight. You should only use colored specular for certain metals, whereas non-metallic surfaces usually have a monochromatic specular color. Non-metallic surfaces normally do not have a colored specular.



Aluminium R:128 G:126 B:121

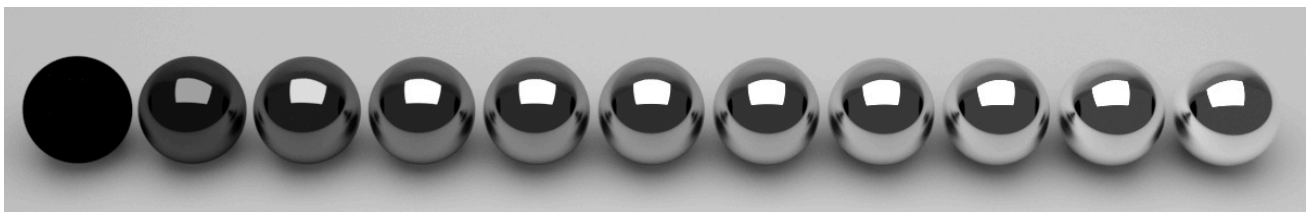
Chrome R:139 G:139 B: 145

Copper R



Scale

The specular weight. Influences the brightness of the specular highlight.



Specular Weight 0 to 1

Roughness

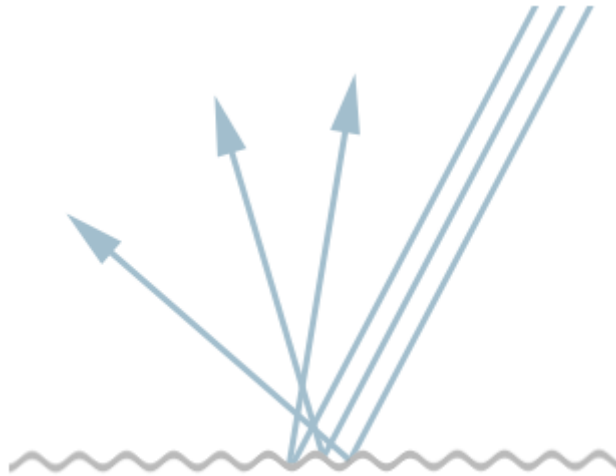
Controls the glossiness of the specular reflections. The lower the value, the sharper the reflection. In the limit, a value of 0 will give you a perfectly sharp mirror reflection, whilst 1.0 will create reflections that are close to a diffuse reflection. You should connect a map here in order to get variation in the specular highlight.



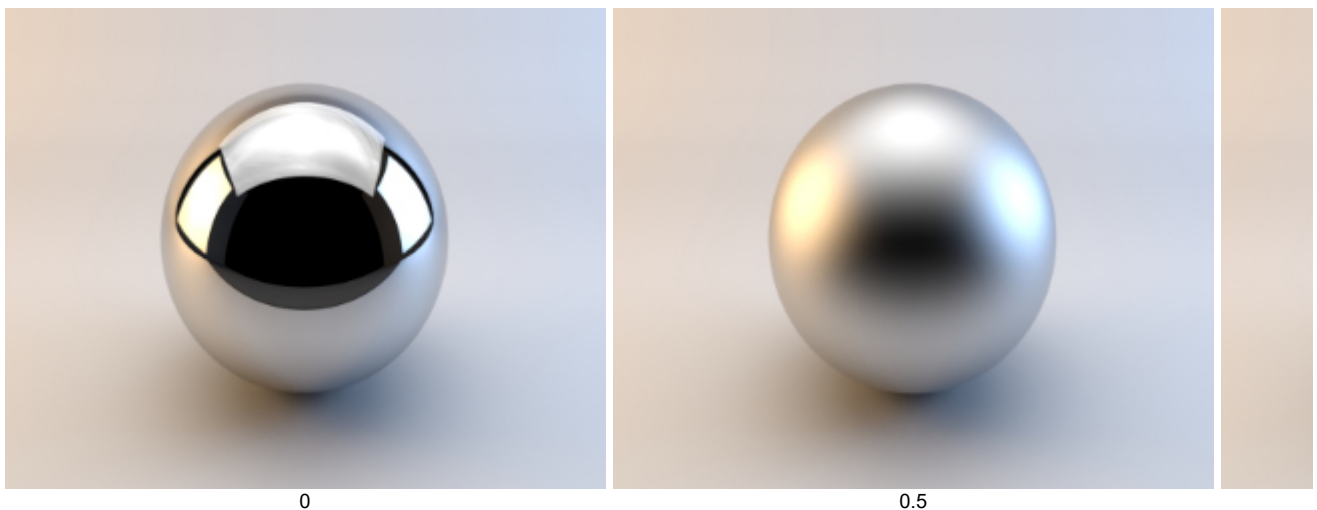
Specular Roughness 0-1

The 'microscopic' features of a surface affect the diffusion and reflection of light. This 'microsurface' detail has the most noticeable effect on specular reflections. In the diagram below, you can view parallel lines of incoming light commence to diverge when reflected from rougher surfaces, when each

ray hits a part of the surface with a different orientation. In summary, the rougher the surface becomes, the more the reflected light will diverge or appear 'blurred'.



The diagram above shows 'microsurface' detail represented as a general measure of roughness (this surface would have a high Specular Roughness value).



The brightness of the Specular highlight is automatically linked to its size due to the Standard shader's energy conserving nature. In the example below, all of the materials are reflecting the same amount of light, but the rougher surface is spreading it out in multiple directions. However, with low amounts of roughness the surface is reflecting a more concentrated amount of light.



0.1

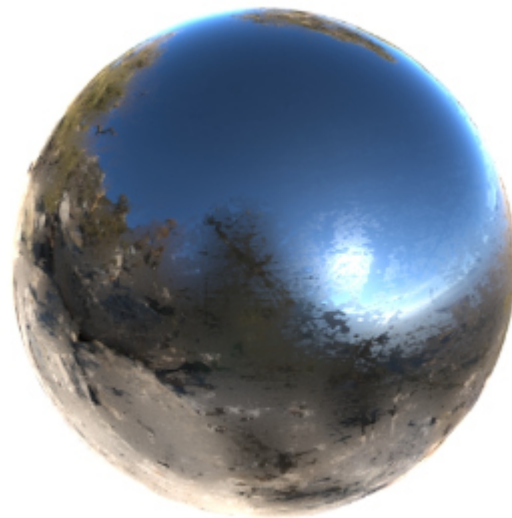


0.2

To get variation in the highlights of the surface, a map should be connected to the Specular Roughness. This will influence not only the brightness of the highlight but also its size and the sharpness of the environmental reflection.



0

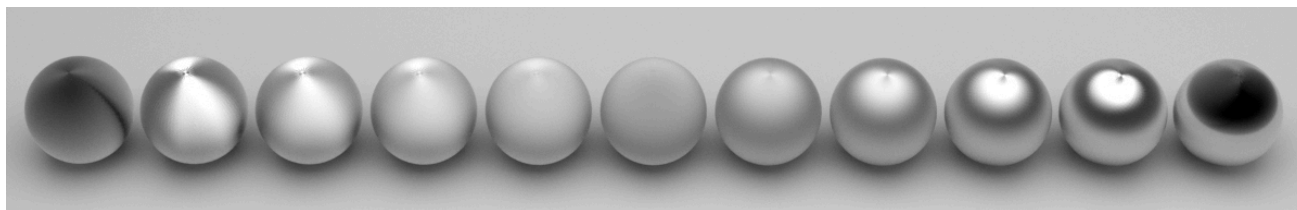


0.5

'Scratches' texture connected to Roughness

Anisotropy

Anisotropy reflects and transmits light with a directional bias and causes materials to appear rougher or glossier in certain directions. The default value for Anisotropy is 0.5, which means 'isotropic'. As you move this control towards 0.0, the surface is made more anisotropic in the U axis, and as you move the control towards 1.0 the surface is made more anisotropic in the V axis.



Anisotropy 0-1

Anisotropy is suitable for materials that have a clear brush direction such as brushed metal which has tiny grooves in which form a 'stretched' anisotropic reflection.



Many small discs form together to create an effect which is the anisotropic highlight

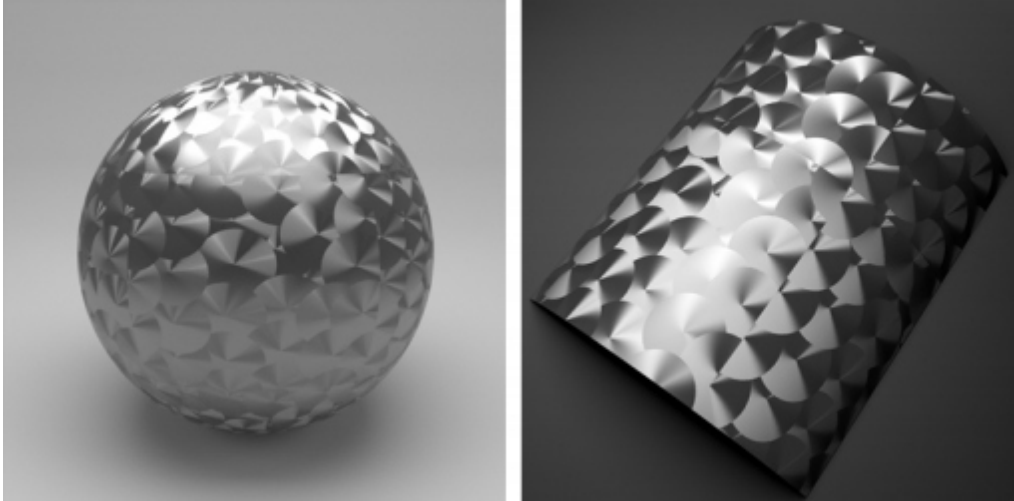


0.3



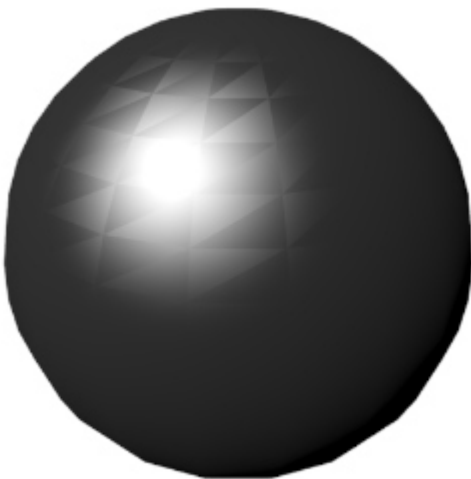
0.6

Anisotropic reflections are suitable for brushed metal effects such as in the example below:•



Texture assigned to Specular Anisotropic Rotation

You may notice faceting appear in specular highlights when using anisotropy. It is possible to remove the faceted appearance by enabling smooth subdivision tangents (via Arnold `subdiv_smooth_derivs` parameter). Take into account this requires a subdivision iteration of at least one in the polymesh to work.

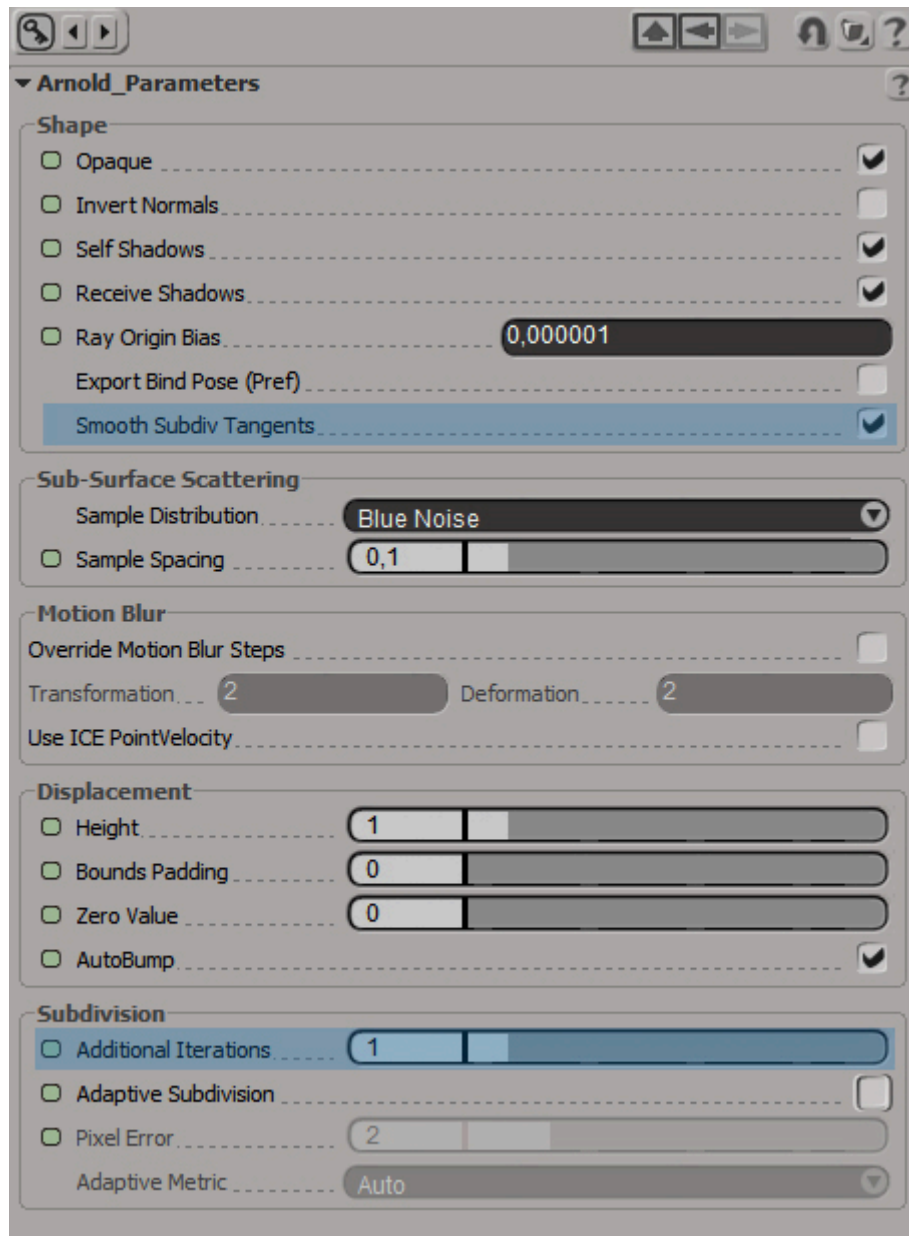


Faceting seen in the anisotropic highlight



Subdivision: enabled. Subdivision Iterations: 1. Smooth Tangents: enabled.

-
-

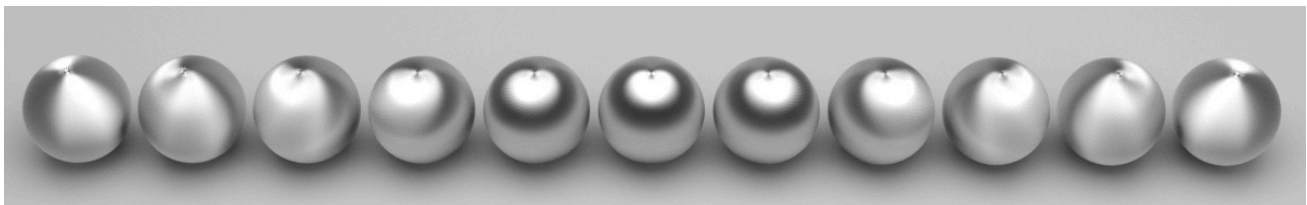


Increase Subdivision Iterations and Smooth Subdivision Tangents to remove anisotropic specular faceting

More information about anisotropy can be found [here](#).

Rotation

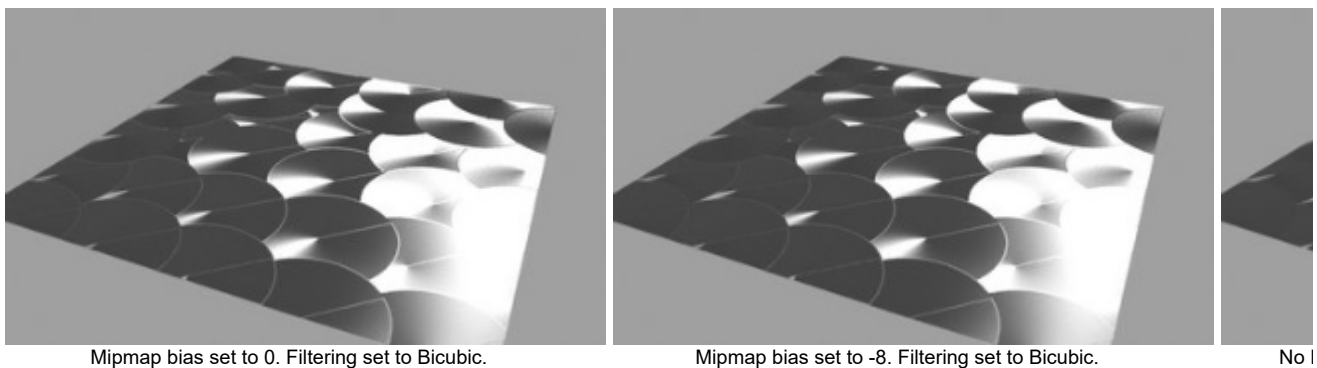
The rotation value changes the orientation of the anisotropic reflectance in UV space. At 0.0, there is no rotation, while at 1.0 the effect is rotated by 180 degrees. For a surface of brushed metal, this controls the angle at which the material was brushed. For metallic surfaces, the anisotropic highlight should stretch out in a direction perpendicular to the brushing direction.



Specular Rotation 0-1

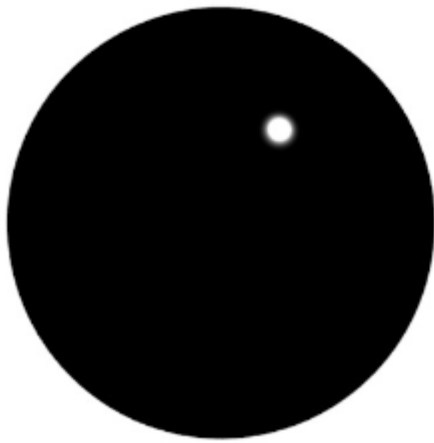


It is possible to assign textures to specular rotation. When doing so it is advisable to avoid texture filtering. This means disabling MIP-mapping and disabling the magnification filter, which by default is set to "smart bicubic". One way is to set the `mipmap_bias` of the image node to a strong negative value, like -8, which means "use 8 MIP levels higher resolution than usual".



Microfacet Distribution

Choose between GGX microfacet distribution or Beckmann (default). GGX is a microfacet distribution. It has a sharper peak and a larger tail than Beckmann. GGX is suitable for modeling light reflection from surfaces more realistically.



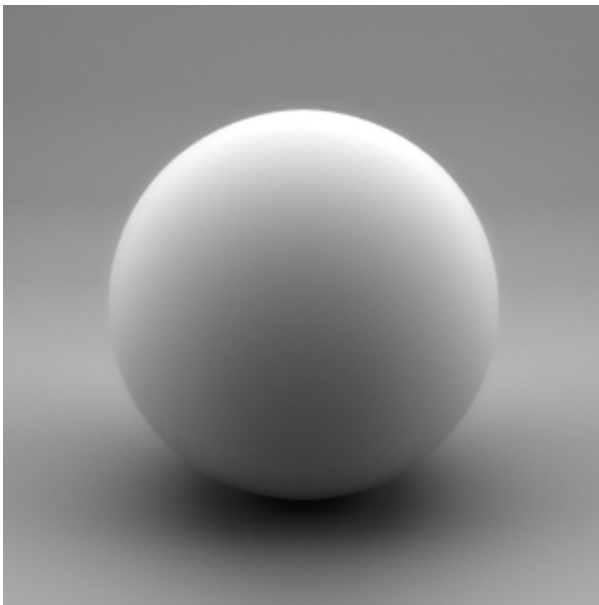
Beckmann



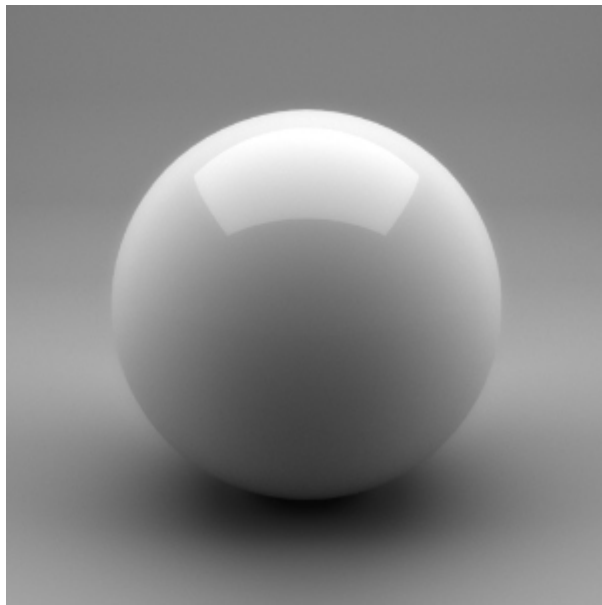
GGX

Direct Scale

The amount of specularity received from direct sources only. Values other than 1.0 will cause the materials to not preserve energy and global illumination may not converge. •



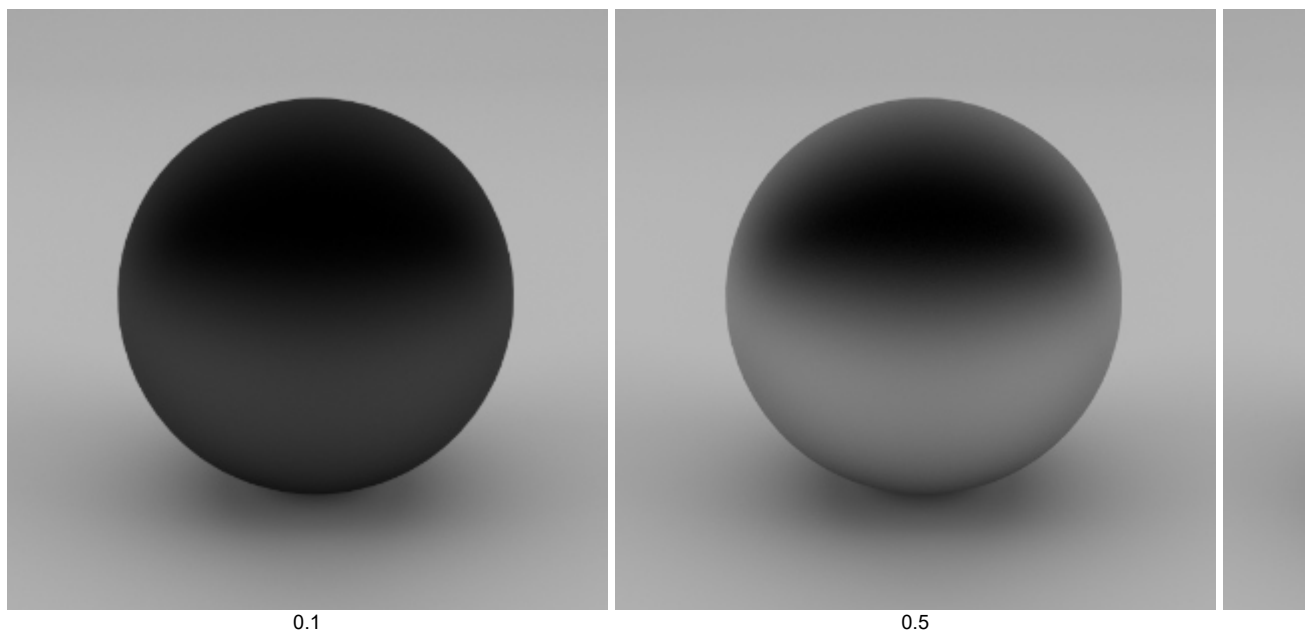
0



0.5

Indirect Scale

The amount of specularity received from indirect sources only. • Values other than 1.0 will cause the materials to not preserve energy and global illumination may not converge.

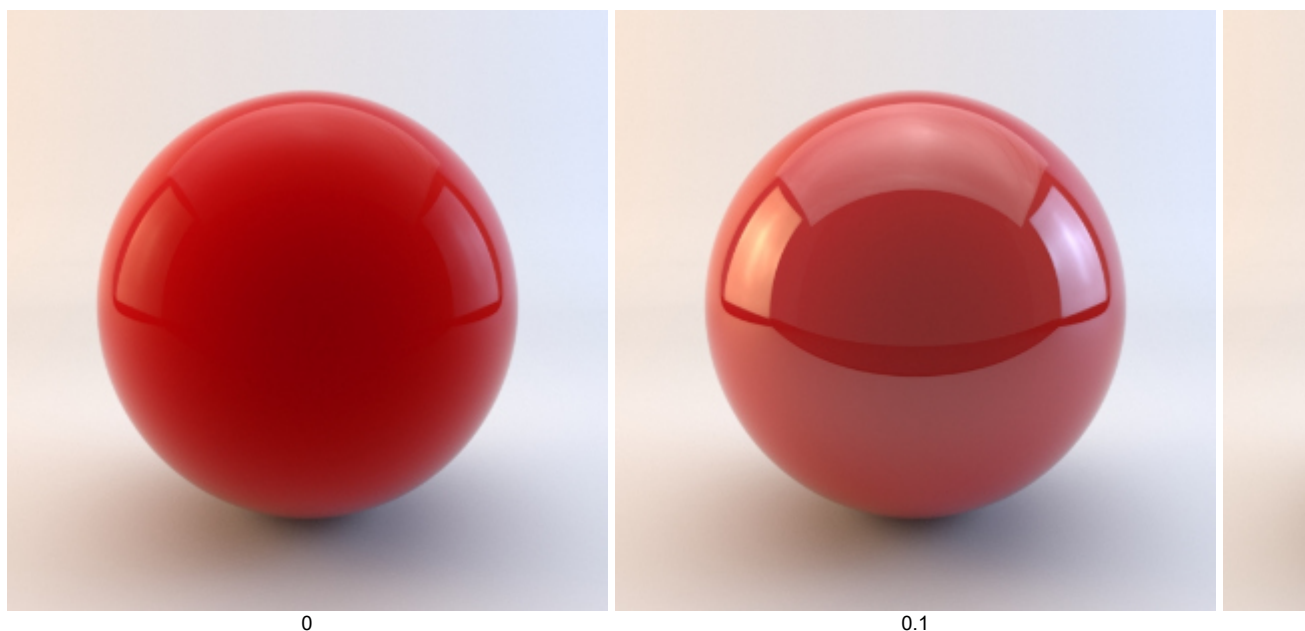


(Fresnel) Enable

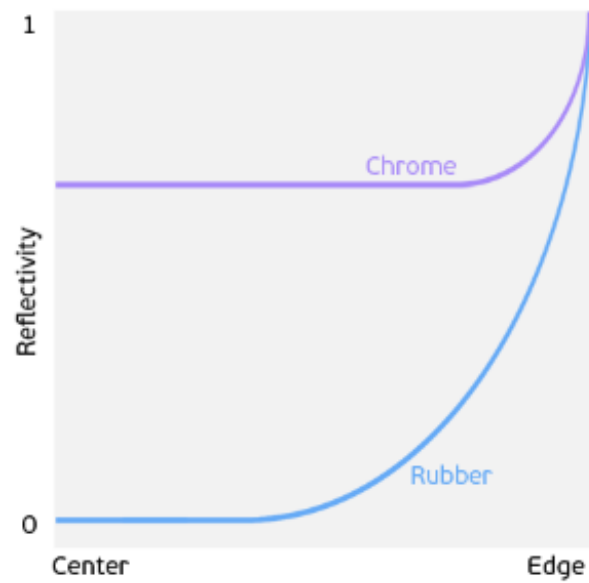
When checked, the reflection level will be dependent on the viewing angle of the surface following the Fresnel equations (which depends on the IOR value). The visual effect is that the reflection increases as the viewer's angle of incidence, with respect to the surface normal, approaches 0. Fresnel has a large effect on almost all materials such as glass, water and smooth coated surfaces, however Fresnel is just as visually important with less shiny materials.

Reflectance at Facing

The Fresnel effect is more noticeable when using lower values. Increasing this value gives the material a more metallic-like specular reflection. Metals have a more uniform reflectance across all angles compared to plastics or dielectrics, which have very little normal reflectance. Note that the Fresnel effect is less evident when a surface become rougher (the unpredictable nature of a rough surface 'scatters' the Fresnel effect, preventing the viewer from being able to clearly see it). As visible in the images below, objects rendered with a correct Fresnel effect will appear to have brighter specular reflections near the edges.



All types of materials become 100% reflective at grazing angles, as evident in the diagram below:



The examples below show various materials which require Reflectance at Normal:



Bubble: Reflectance at Normal 0.03



Car paint: Reflectance at Normal 0.01



For realistic materials, reflectance at normal must be lower than the specular/reflection scale (which controls the reflectance at grazing). Otherwise, you will get a darker reflection at the edges, which is exactly the opposite of the effect seen in nature. It is also not advisable to tint speculars when using Fresnel as this is not physically correct.

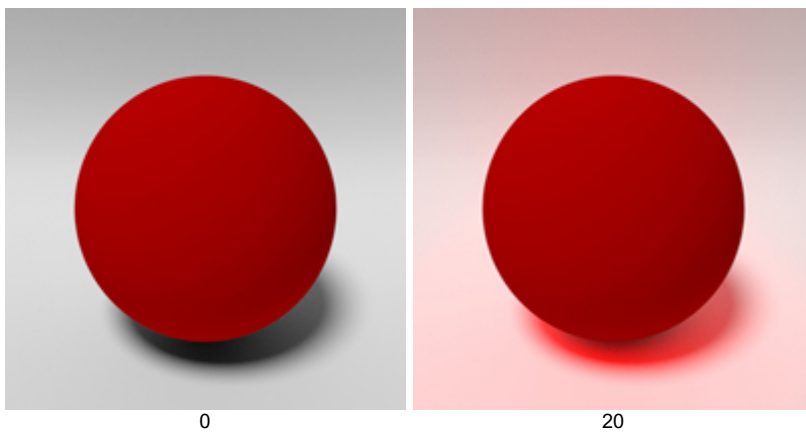
Bounce Factor



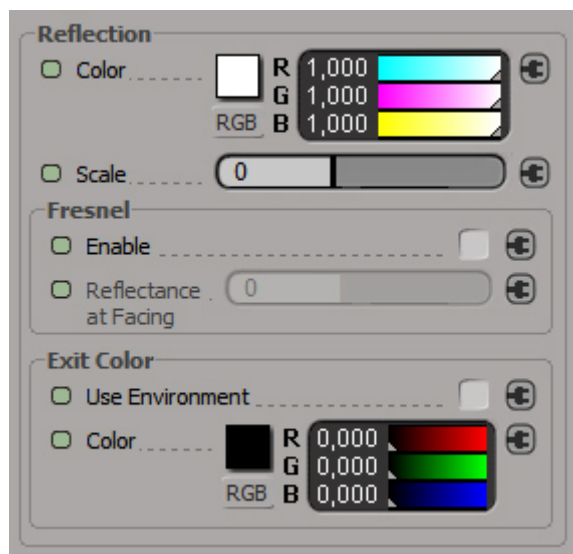
The relative energy loss (or gain) at each bounce. This should be left at its default value of 1.0, which is the only value with meaningful physical sense. Values bigger than 1.0 will make it impossible for GI algorithms to converge to a stable solution, and values smaller than 1.0 will have a reduction in the color bleeding effect.

•

Bounce Factor gives more control over GI bounces on a per shader basis rather than on a global basis via the render settings. In the following example the Bounce Factor has been changed for the shader assigned to the red sphere. When the Bounce Factor of the red sphere has been set to zero, the GI rays traced from the white floor to the sphere will not see it and therefore will not receive any color bleeding.



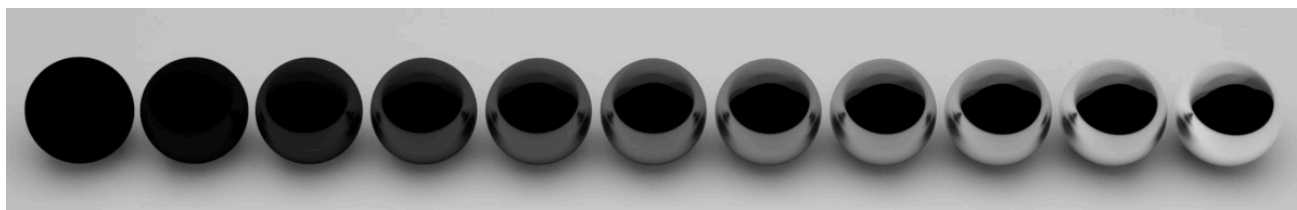
Reflection



The Standard shader has a secondary layer of reflection, called simply **reflection**, which is perfectly sharp and mirror-like. It should only be used to replicate perfect mirrors or glass shaders. If you want blurry reflections only you should use [Specular](#).

Color

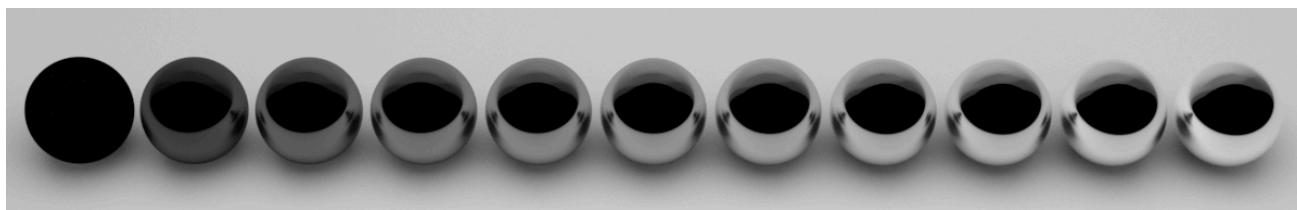
The color of the reflection ray at the current point.



Reflection Color 0-1

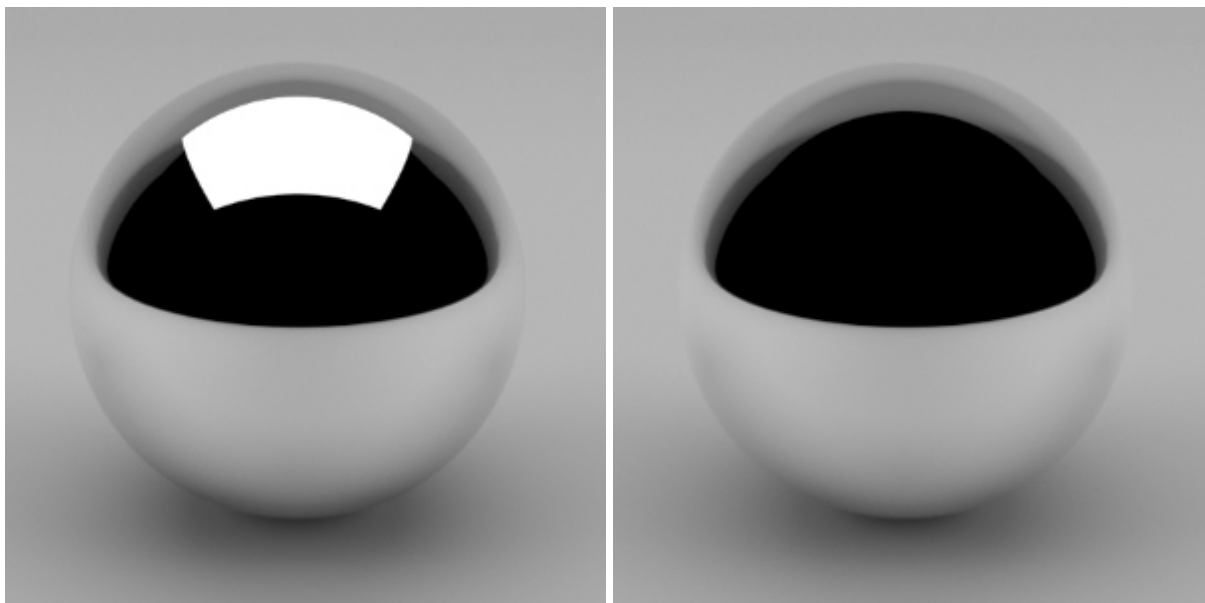
Scale

The contribution from reflection rays (the amount of light that the surface reflects).



Reflection 0-1

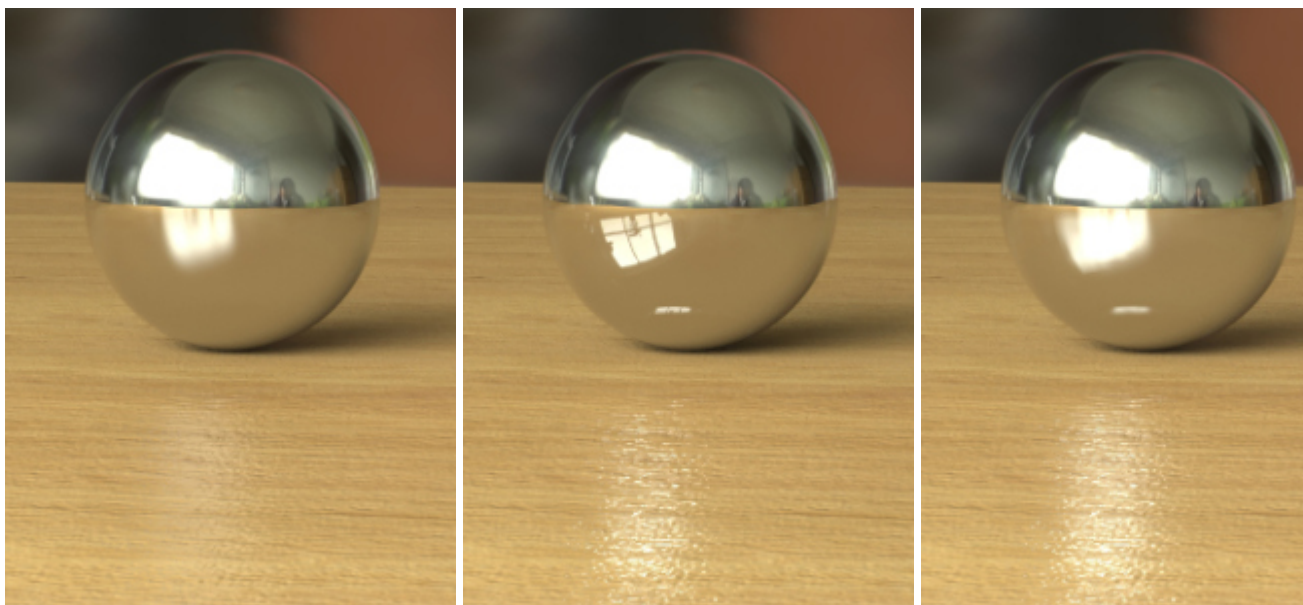
You will notice in the image below, that the light source is not visible in the **reflection**. This is because **reflectivity** does not sample light sources (direct light). Therefore, it is recommended that you use Specularity with glossy materials, unless a pure 'mirror' reflection is required.



Specular 1. Light source is reflected.

Reflection 1. Light source is not reflected.

In the chrome sphere below you can see the difference between the Specular and Reflection attributes when using a SkyDome light. Notice the difference between the Specular and Reflective highlights of the window in the chrome sphere and wood surface.



Specular Only

Reflection Only

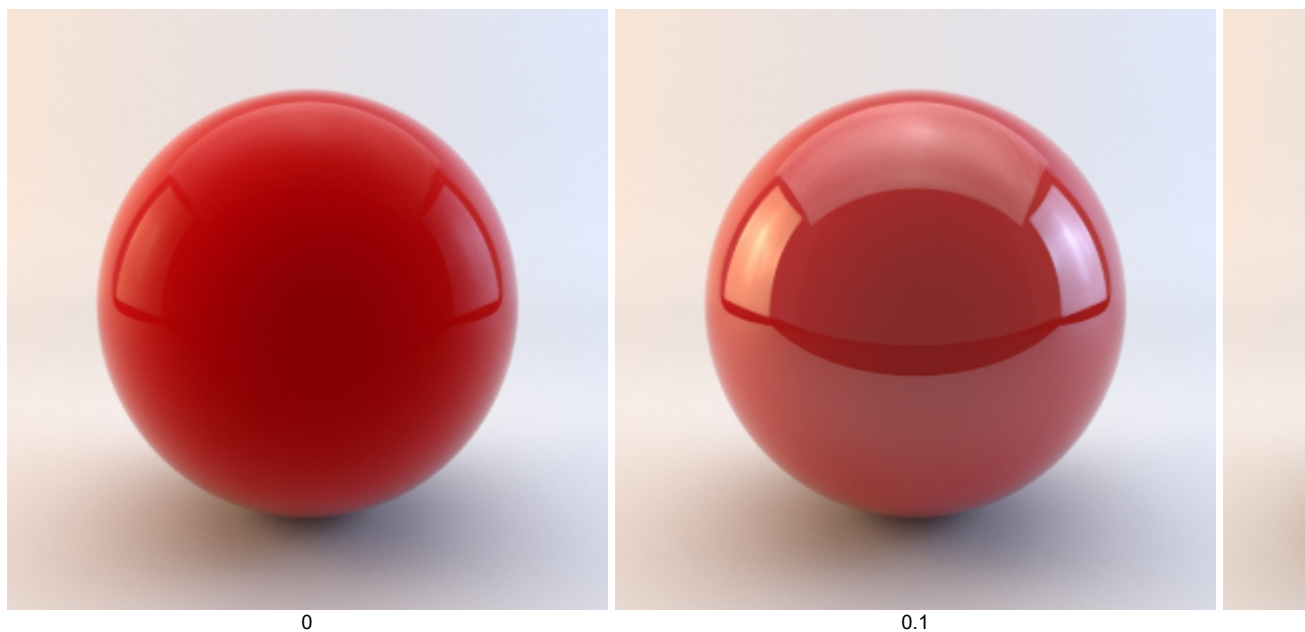
Specular & Reflection

Fresnel (Enable)

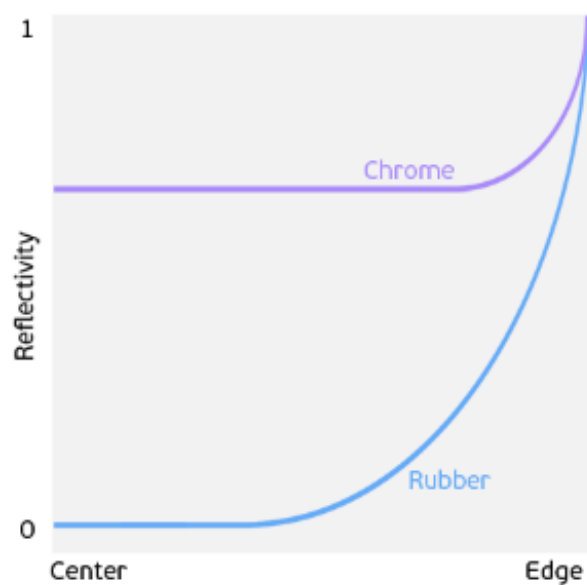
Enables or disables the Fresnel effect.

Reflectance at Facing

The Fresnel effect is more noticeable when using lower values. Increasing this value gives the material a more metallic-like specular reflection. Metals have a more uniform reflectance across all angles compared to plastics or dielectrics, which have very little normal reflectance. •Note that the Fresnel effect is less evident when a surface becomes rougher (the unpredictable nature of a rough surface 'scatters' the Fresnel effect, preventing the viewer from being able to clearly see it). As visible in the images below, objects rendered with a correct Fresnel effect will appear to have brighter specular reflections near the edges. •



All types of materials tend to have full reflectivity at grazing angles, as evident in the diagram below:



The examples below show various materials which require Reflectance at Normal:



Bubble: Reflectance at Normal 0.03

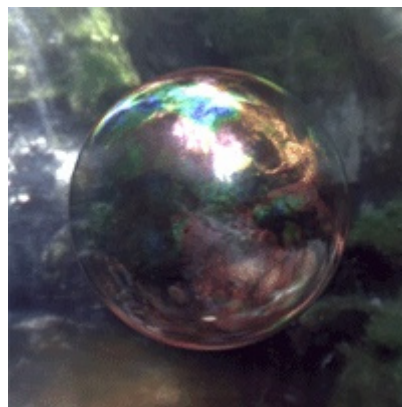


Car paint: Reflectance at Normal 0.01



For realistic materials, reflectance at normal must be lower than the specular/reflection scale (which controls the reflectance at grazing). Otherwise, you will get a darker reflection at the edges, which is exactly the opposite of the effect seen in nature. It is also not advisable to tint speculars when using Fresnel as this is not physically correct.

Note that it is also possible to map a texture to Reflectance at Normal in order to control the intensity of the Fresnel effect:



V ramp texture connected to 'Reflectance at Normal' (1 to 0). Rollover image.

Reflection Exit Color

The color returned when a ray has reached its maximum reflection depth value. •

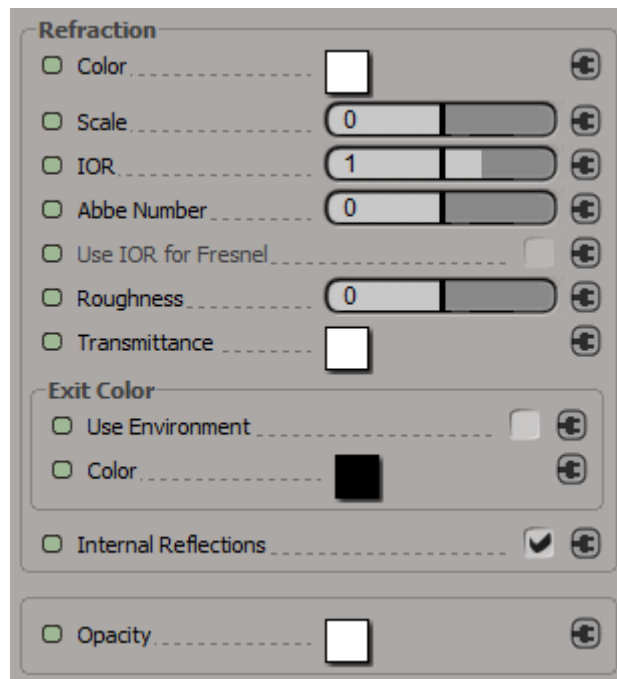


Enabled



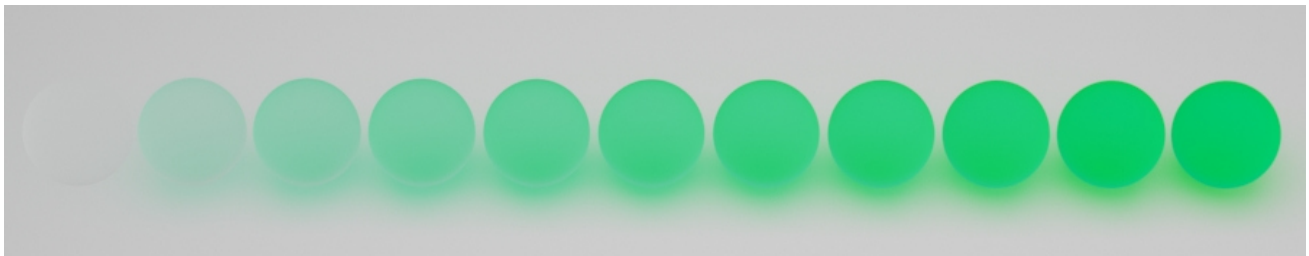
Reflection Exit Color - Red (Reflection ray depth set to 1).

Refraction



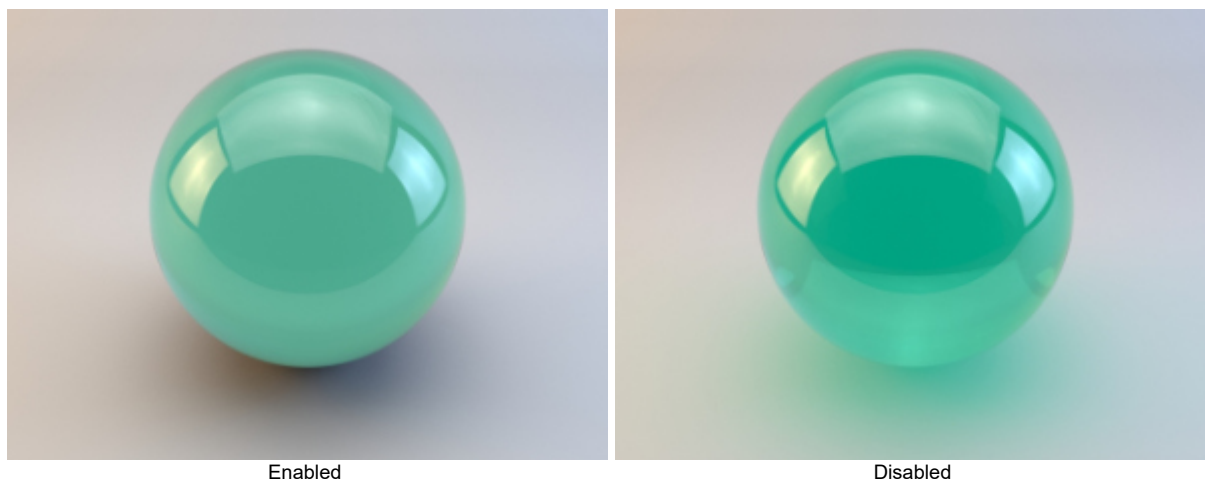
Color

Transparency color multiplies the refracted result by a color. For tinted glass it is best to control the tint color via the Transmittance color since it actually filters the refraction according to the distance traveled by the refracted ray.



Refraction Color Hue 0-1

If this value has a color and shadows tinted with that color are required then disable 'opaque' for the mesh that has been assigned the Standard material. In the example below, you can see that with **Opaque** enabled the rays cannot pass through the sphere. Whereas with **Opaque** disabled, the rays can pass through the sphere and absorb the color set by the transmittance, thereby creating the effect of colored shadows.

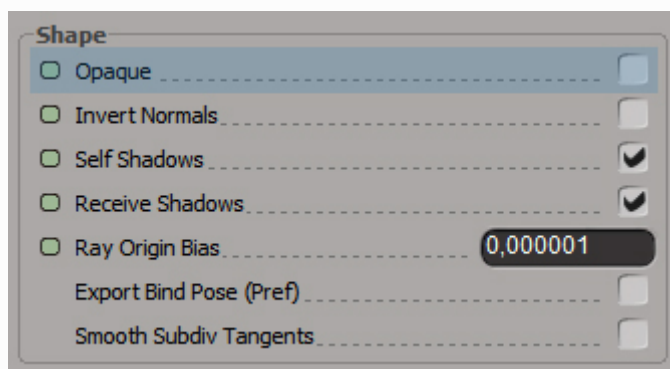


Scale

Transparency allows light to pass through the material.

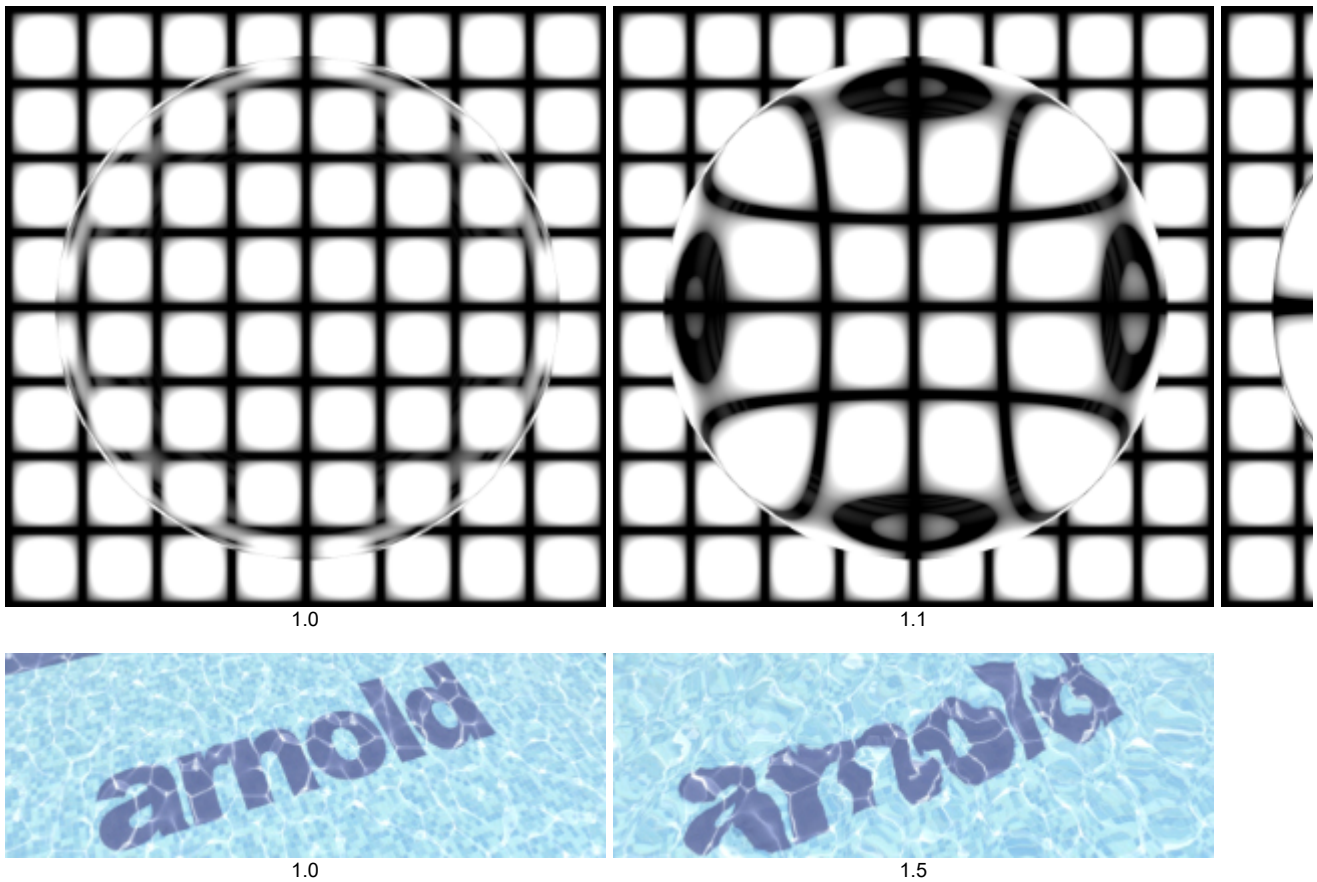


i Note that you must disable 'Opaque' for the mesh that has been assigned the Standard*shader.



IOR

The index of refraction used. The default value of 1.0 is the refractive index of a vacuum, i.e., an object with IOR of 1.0 in empty space will not refract any rays. In simple terms, 1.0 means 'no refraction'. The Standard shader assumes that any geometry has outward facing normals, that objects are embedded in air (IOR 1.0) and that there are no overlapping surfaces. •



When rendering transparent and refractive surfaces it is very important that the normals of the geometry face in the right direction. In the example below you can see the difference between normals that are facing in the right direction (outward) versus those that are facing inwards (incorrect). This is especially important when rendering surfaces with double sided thickness, such as a glass.

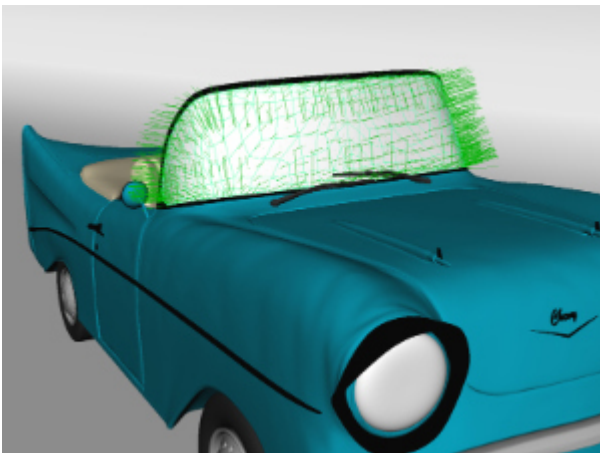


Normals Correct



Normals Incorrect

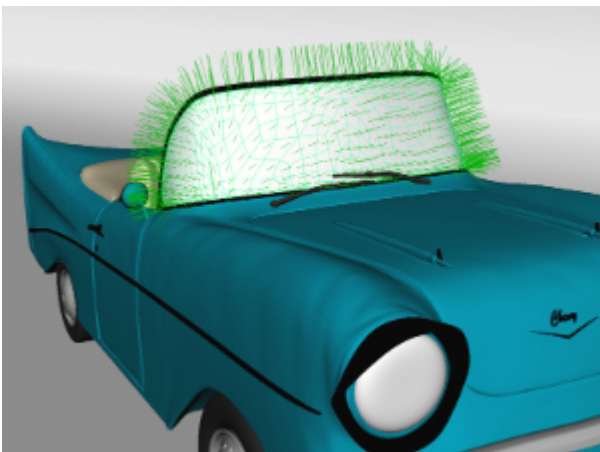
The direction of the normals is equally important when rendering single sided surfaces. The windscreen model below is single sided. The difference is clearly visible when the normal direction is facing in the wrong direction. •



Normals pointing inwards: Incorrect



Windscreen glass looks wrong



Normals pointing outwards: Correct



Windscreen glass renders correctly



If you can only see black where there should be refraction, you may not have a high enough `refraction ray depth` value (found in the `Ray Depth` section in the `Render Settings`). The default value is two.

Abbe Number

Specifies the Abbe number of the material, which describes how much the index of refraction varies across wavelengths. For glass and diamonds this is typically in the range of 10 to 70, with lower numbers giving more dispersion. The default value is 0, which turns off dispersion. The chromatic noise can be reduced by either increasing the global Camera (AA) samples, or the Refraction samples.



10

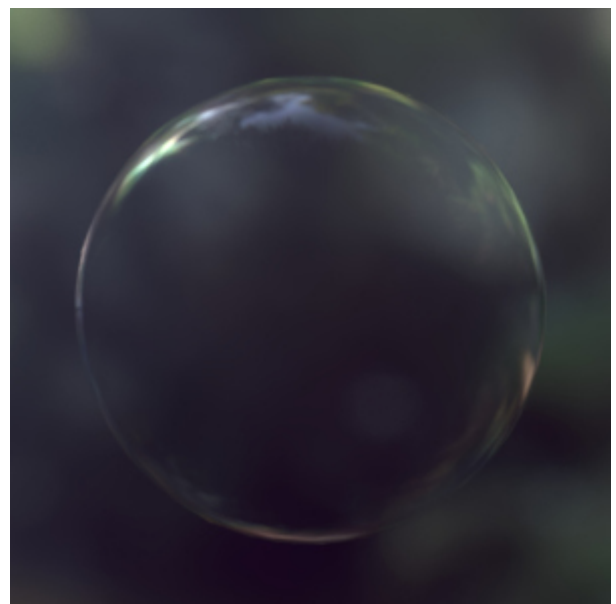


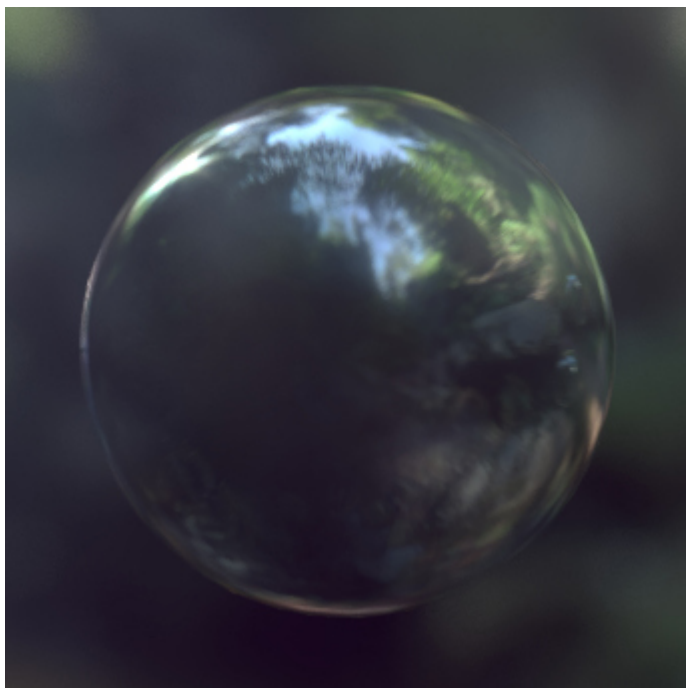
20

Animation [here](#).

Use IOR for Fresnel

Calculates Fresnel reflectance based on the IOR parameter, ignoring the values set in `Krn` and `Ksn`.

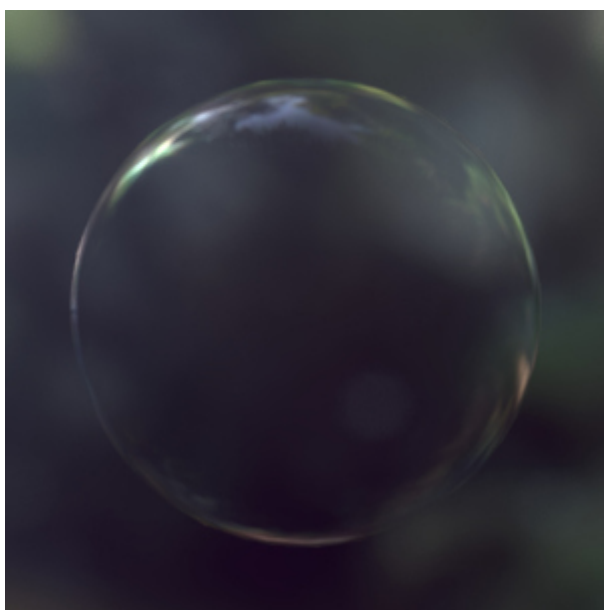




Disabled

Enabled

The images below show the effect that increasing the Index of Refraction has on the Fresnel reflectance.



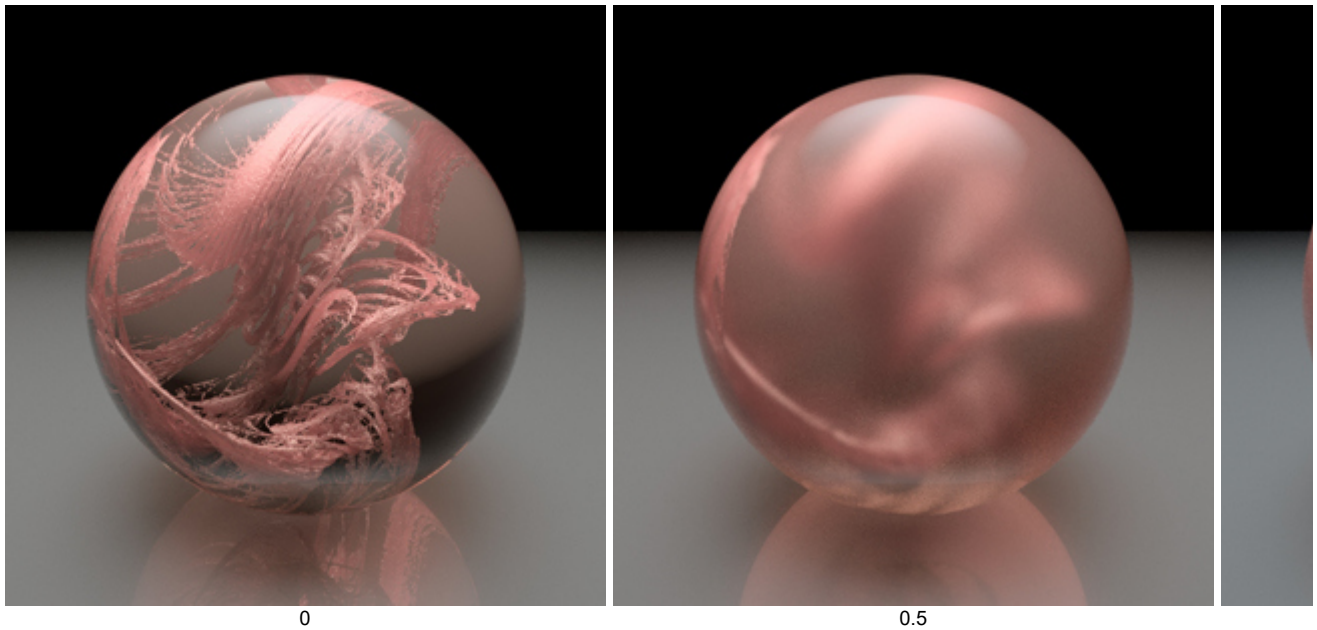
1



1.5

Roughness

Controls the blurriness of a refraction computed with an [isotropic microfacet BTDF](#). The range goes from 0 (no roughness) to 1.



Transmittance

Transmittance filters the refraction according to the distance traveled by the refracted ray. The longer light travels inside a mesh, the more it is affected by the Transmittance color. Therefore green glass gets a deeper green as rays travel through thicker parts. The effect is exponential and computed with [Beer's Law](#). It is recommended to use light, subtle color values.

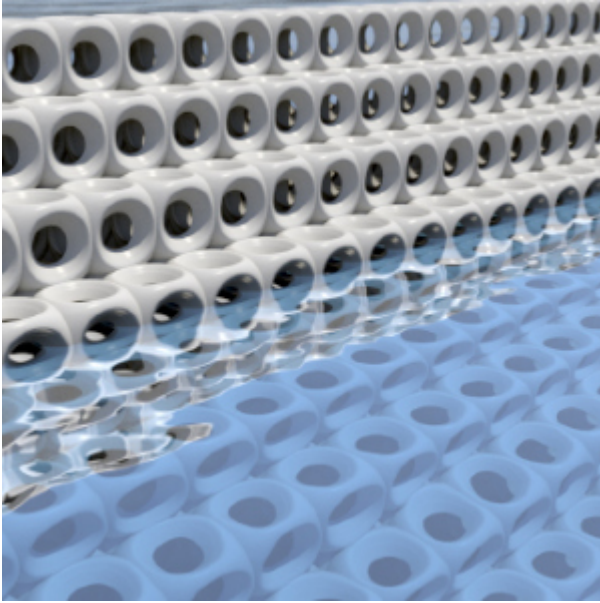
In the example below, you can see this difference in the bottom of the glass. The glass with **Transmittance color** appears more physically accurate than the glass that uses **Refraction color**.



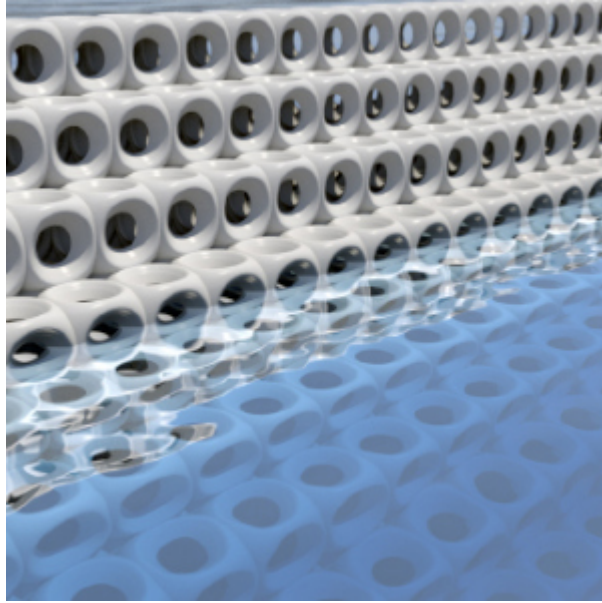
Refraction Color

Transmittance Color

The effect of **Transmittance color** is even more obvious in the example below. You can see that the white shapes become darker as they go deeper below the surface when using Transmittance.



Refraction



Transmittance

The images below show the effect of transmittance between a color value range of 0.8 and 1 (with a green tint).



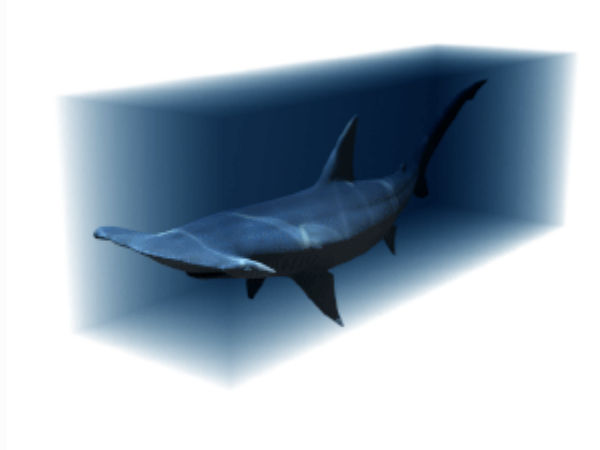
0.8



0.9



Transmittance color is scene scale dependent which can have a dramatic effect on its appearance. If you cannot see the effect of Transmittance color then you may need to check the size of your scene.

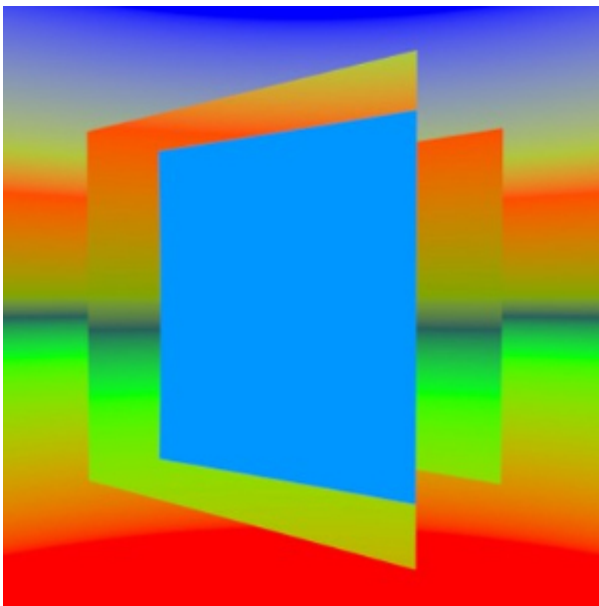


Transmittance effect is more noticeable when cube is scaled up

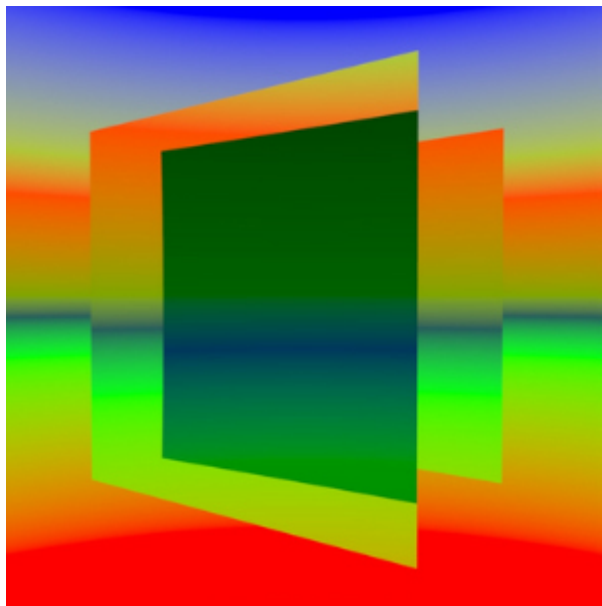
Exit Color

Use Environment

Specify whether to use the environment color for refraction rays where there was insufficient ray depth (true), or the color specified by `refraction_exit_color` (false).•



Disabled



Enabled

Further information can be found in•[Refraction And Opacity](#).

Color

The color returned when a ray has reached its maximum refraction depth value.•



Red



White

Internal Reflections

Unchecking internal reflections will disable indirect specular and mirror perfect reflection computations when ray refraction depth is bigger than zero (when there has been at least one refraction ray traced in the current ray tree).

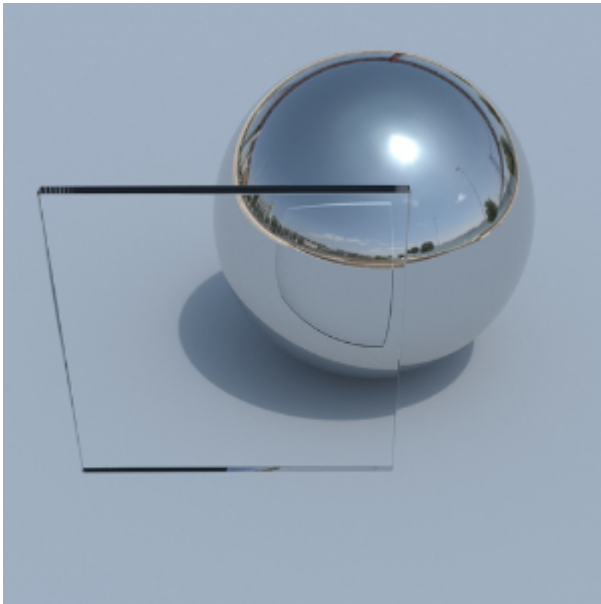


Enabled

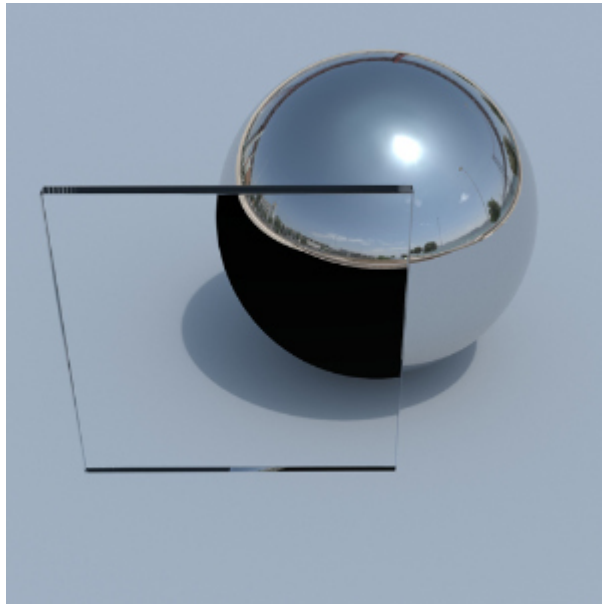


Disabled

In the right image below, the sphere appears black because Enable Internal Reflections is disabled in the Standard shader assigned to the sphere.



Enabled



Disabled



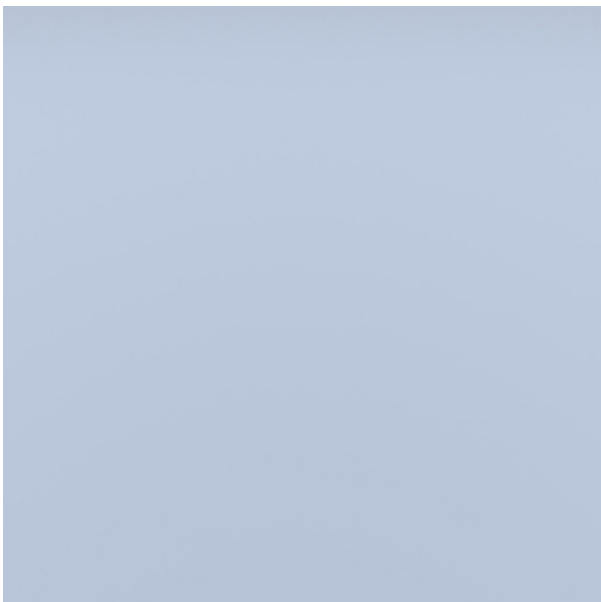
Disabling 'Enable Internal Reflections' can dramatically reduced render times for highly reflective/ refractive scenes.

Opacity

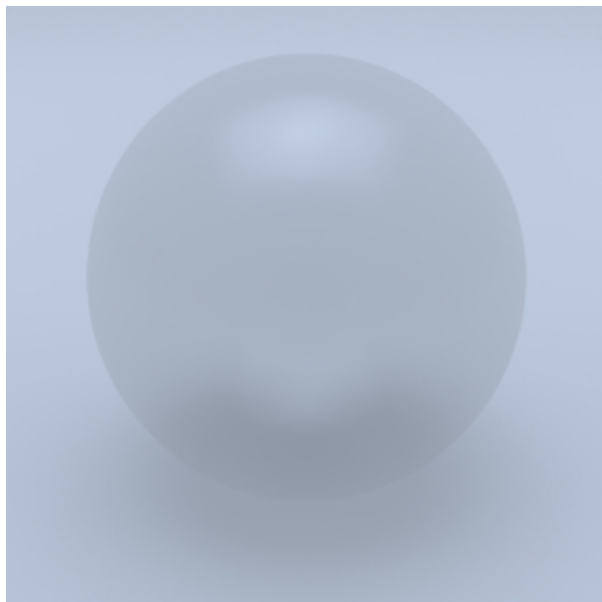
Controls the degree to which light is not allowed to travel through it. Unlike transparency, whereby the material still considers diffuse, specular etc, opacity will affect the entire shader. Useful for retaining the shadow definition of an object, whilst making the object itself invisible to the camera.



You must ensure that 'Opaque' is disabled for the mesh that the Standard shader is assigned to when using 'Opacity'.



0



0.5



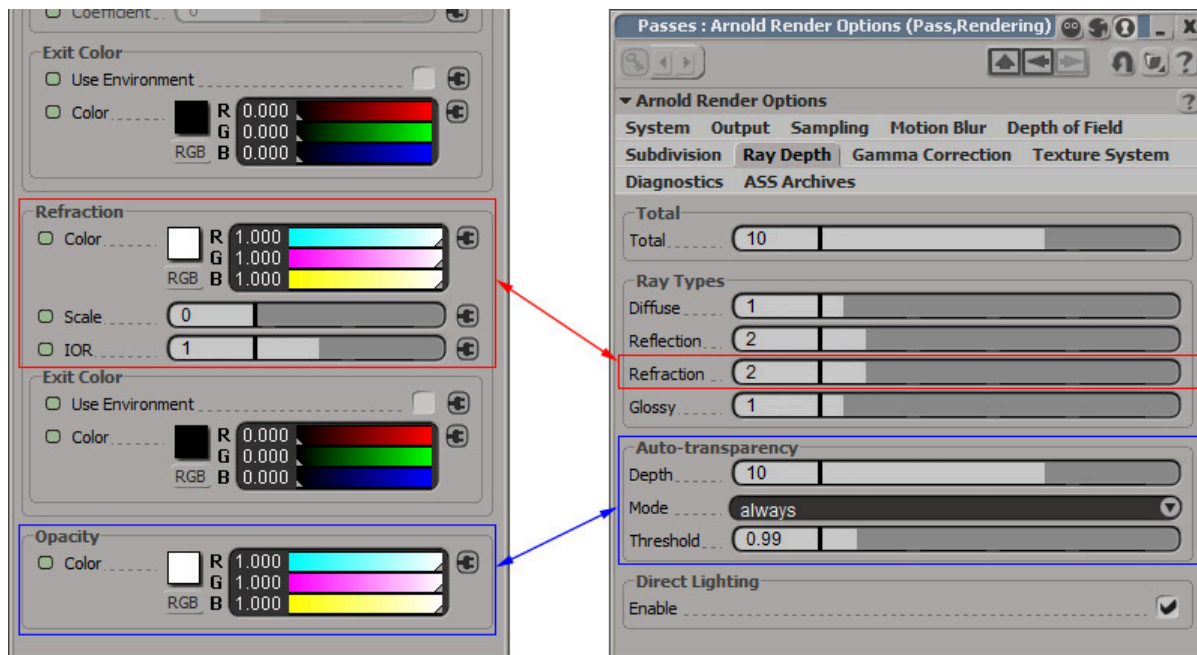
More information about working with Opacity can be found [here](#).

Transparency (Refraction and Opacity)

Refraction and opacity are pretty similar in Arnold compared with other render engines, however, there are a few differences. The purpose of this document is to give users a better understanding of these differences as well as explaining when to use opacity and when to use refraction.

The Standard shader

Arnold has two different ways of calculating transparency, **refraction** and **opacity**. They are different ray types and thus have different controls in the Standard shader as well as in the render options, the image below illustrates how they correspond to each other:



These two ways of calculating transparency have different purposes, they can be used together but most of the time you'll want to use either one or the other.

Usage for Refraction

- Glass, water or other refractive materials.

Usage for Opacity

- Sprite type of effect, such as cutting out the shape of a leaf from a polygon card.
- Making the tips of hair strands transparent.

If you leave IOR at 1.0 both methods can give similar results, however Opacity or "Auto Transparency" as it's called in the render options renders faster, for sprites Opacity will also cut out the shape completely whereas Refraction will leave the reflections/speculars visible even on areas that are completely transparent, here are two images to show the difference when rendering sprites:



Leaf rendered driving the opacity using a mask



Leaf rendered driving the refraction using a mask

This is the texture and mask used for the leaf:



Leaf texture map



Leaf alpha map

Note how when using refraction, the specular/reflections are still visible in the transparent areas. You can of course drive the specular weight using the same mask to fix the problem, but using refraction for this purpose is simply wrong.

Using Refraction and Opacity together

As explained earlier it's usually best not to use Refraction and Opacity together in the same shader as it would cause unnecessary slowdowns in the render, however there are a few situations where combining them can be useful, below are some examples of a simple glass sphere being cut out using a stripe mask:



Sphere using refraction only



Sphere rendered with the mask texture



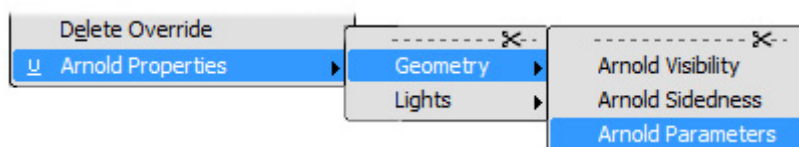
Spher

The "Opaque" switch

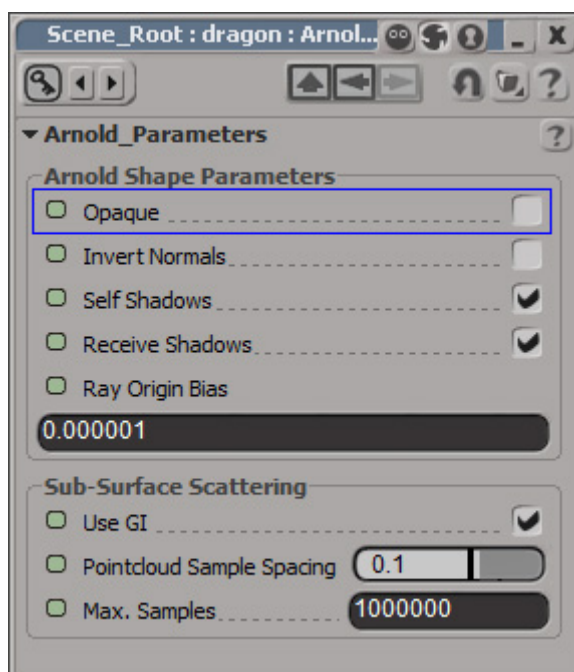
Here is where Arnold differs from most other renderers. By default all objects are flagged as "Opaque", which allows Arnold to take some shortcuts while tracing rays, thus making rendering faster. If an object is flagged as opaque it means Arnold doesn't even need to access the parts of the shader that handles Opacity. The fact that this "Opaque" flag is on by default means two things:

- Opacity will not work at all
- Refraction will work however any shadows cast by the objects will always be solid and not pick up the refraction color or density of the shader

From the "Properties" menu you can apply the "Arnold Parameters" which among other things allows you to toggle the "Opaque" switch:



This property can be applied to objects, partitions and groups and has the following options:



In the following scene you can see the Stanford dragon rendered with the opaque switch On and Off. Note that Opacity is not used at all in this example:



Dragon with 'Opaque' on

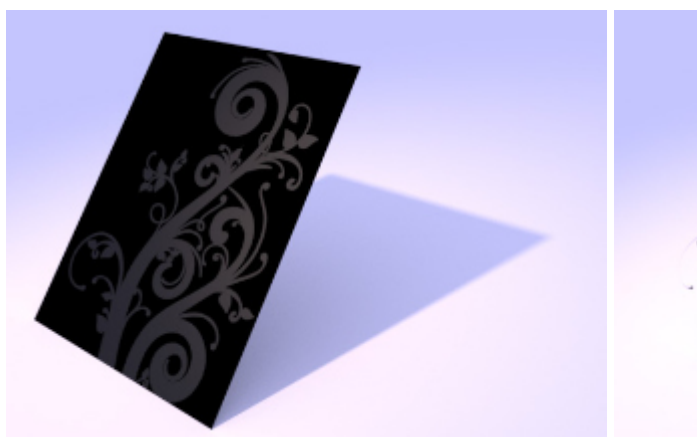


Dragon with 'Opaque' off

Below is another example with Opaque enabled and disabled. In this case only **Opacity** is being used in the **Standard** shader:



Image used to drive the Opacity parameter

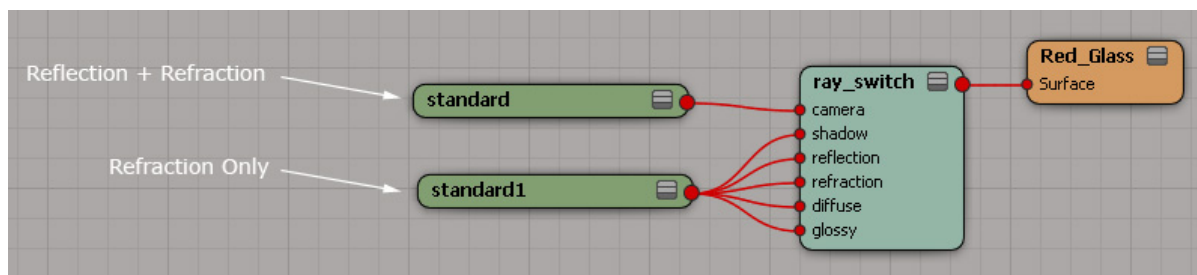


Sprite with 'Opaque' on

Optimization

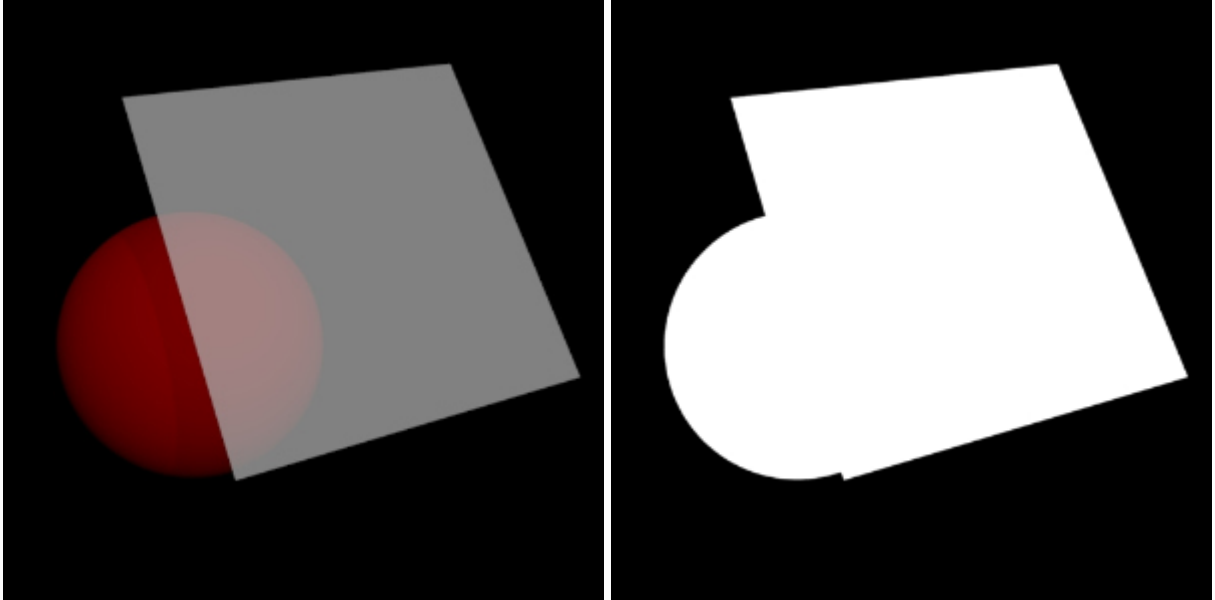
As mentioned earlier using the right method for the right task will give you optimal rendering speed, however there are some things which can further speed things up.

- When using opacity for sprites, make sure that your mask is pure black and white, for example in the black areas there should not be any noise or other imperfections as that will have a negative effect on the render times. In other words it is not a good idea to use jpeg textures for masks as they often have compression artifacts
- When combining Refraction and Reflection such as on the dragon above it can speed things up a lot if you make sure that the refraction part of the shader does not pick up any reflections. In other words, when disabling internal reflections, you can do this using a **Ray_Switch** shader as shown below:



Transparency and alpha •

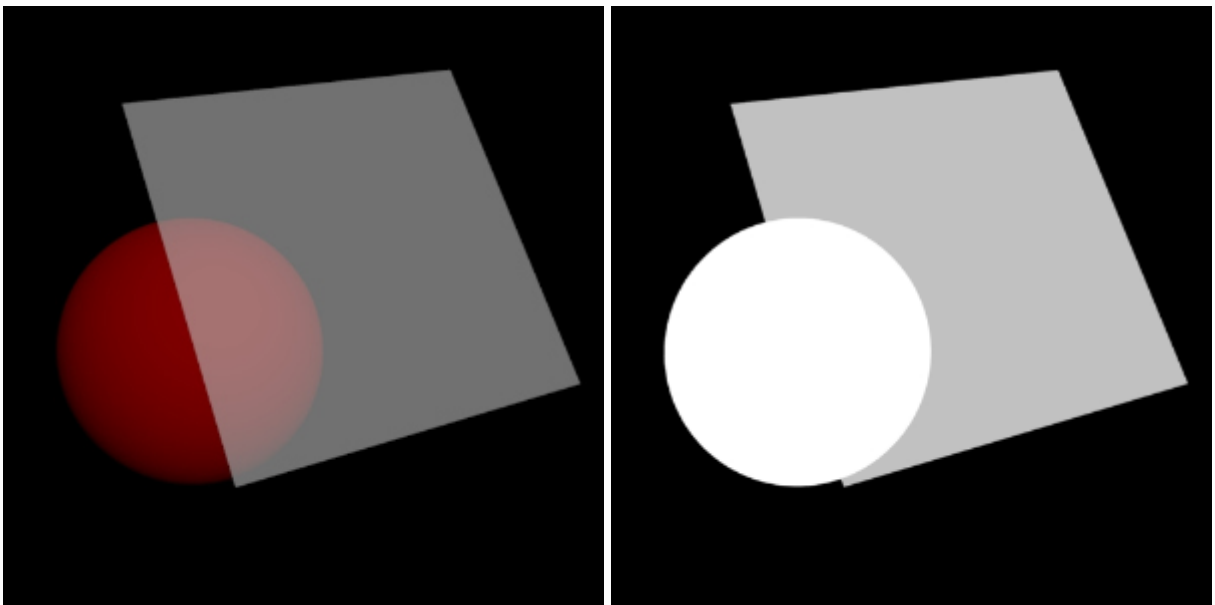
Bear in mind that `Refraction Scale` will not propagate through to the alpha channel. The examples below show the difference in the alpha when using `Opacity` (correct) rather than `Refraction Scale` (incorrect). 'Opaque' has also been disabled for the polygon plane.



Refraction Scale (incorrect alpha)

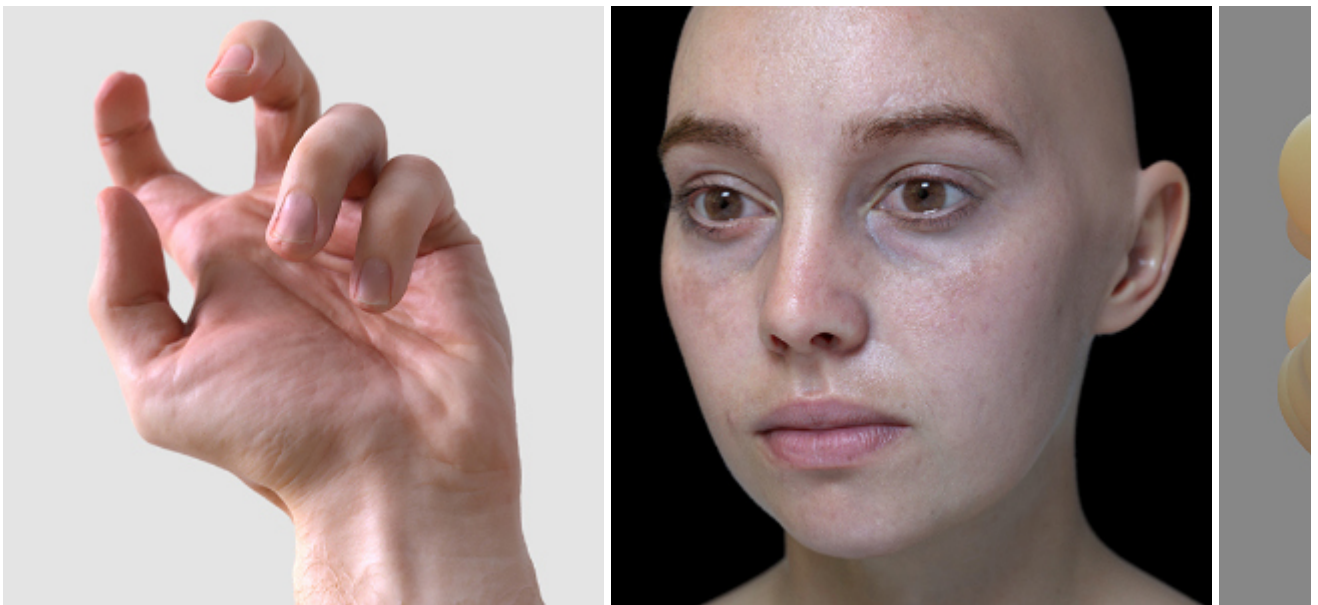
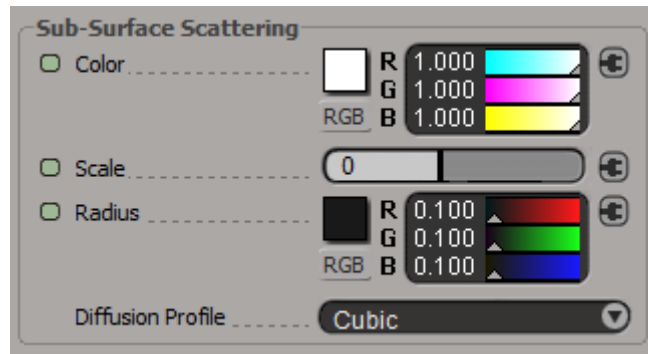
•

If you want to see the alpha channel, use `Opacity` instead of `Refraction Scale` :

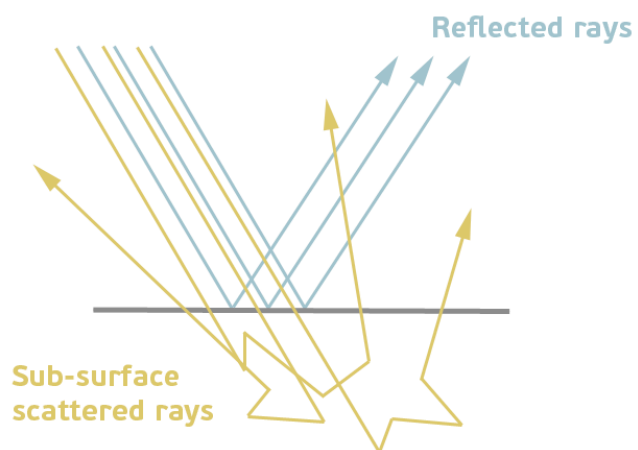


Opacity (correct alpha)

Sub-Surface Scattering



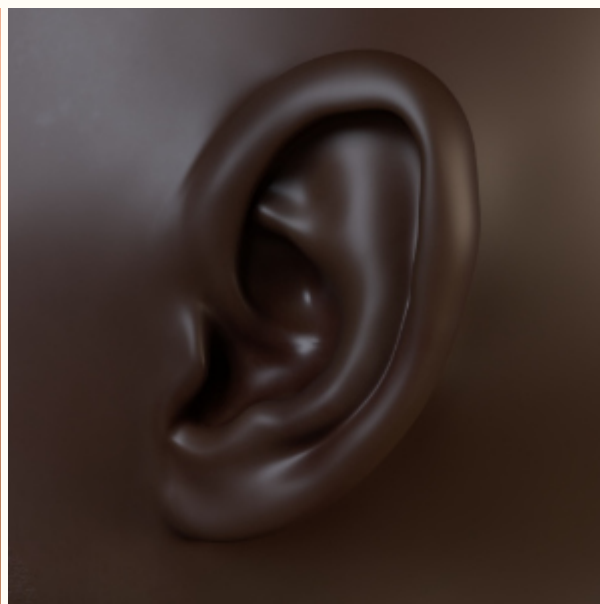
Sub-Surface Scattering (SSS) simulates the effect of light entering an object and scattering beneath its surface. Not all light reflects from a surface. Some of it will penetrate below the surface of an illuminated object. There it will be absorbed by the material and scattered internally. Some of this scattered light will make its way back out of the surface and become visible to the camera. This is known as 'sub-surface scattering' or 'SSS'. SSS is necessary for the realistic rendering of materials such as marble, skin, leaves, wax, and milk. The SSS component in this shader is calculated using a brute-force raytracing method.



You must ensure that the geometry normals are pointing in the correct direction, otherwise, SSS will not render correctly.



Normals pointing outwards (correct)



Normals pointing inwards (incorrect)

SSS is important when replicating realistic materials such as plastics, for example:



SSS disabled



SSS enabled

Color

The color used to determine the sub-surface scattering effect. For example, replicating a skin material, would mean setting this to a fleshy color.

Scale

The amount of sub-surface scattering. Multiplies the SSS Color. Below you can see the effect of adding SSS Weight:



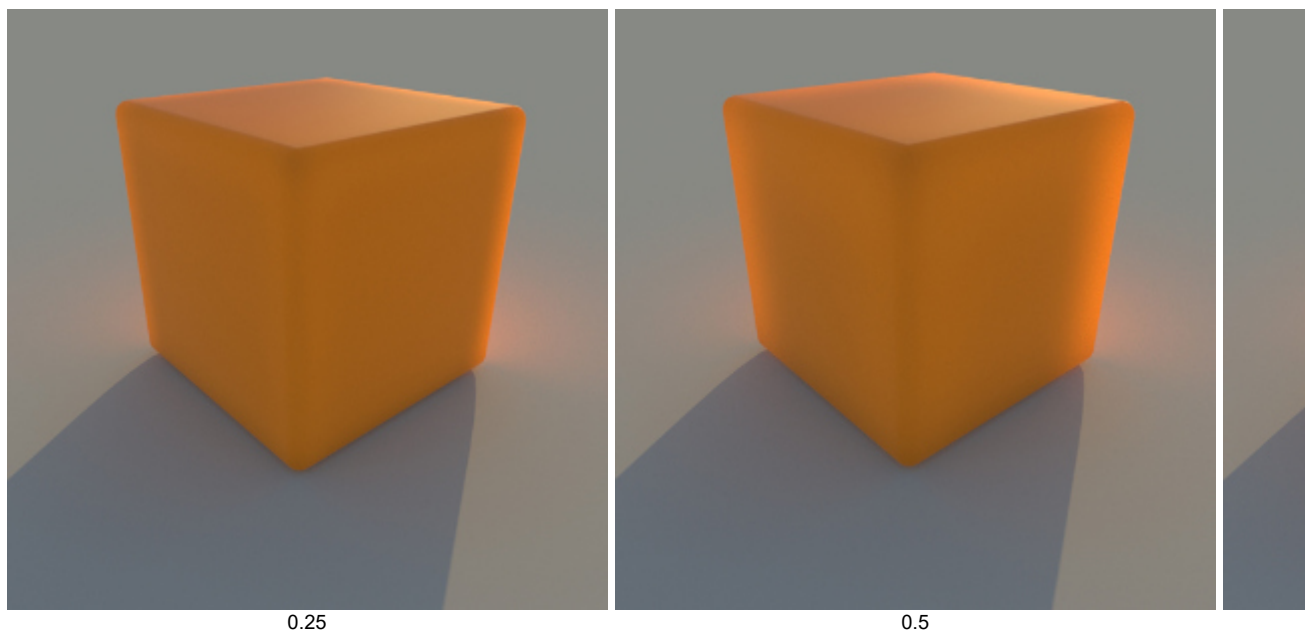
0



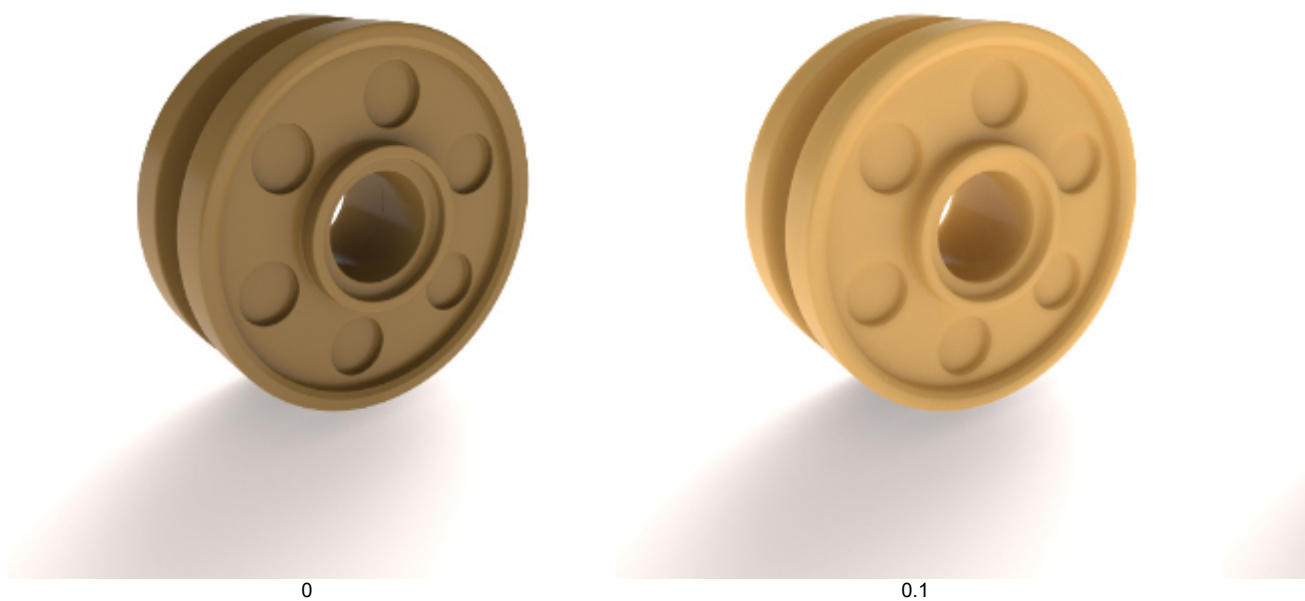
0.5

Radius

The radius of the area each sample affects. Higher values will smooth the appearance of the sub-surface scattering. Results will vary depending on the scale of the object in your scene. Arnold will take into account the shape and thickness of the object being lit. If it is thin enough, the object will often see light scattering out the back side, depending on the radius value (see example images below):



The lighter the color, the more light is scattered. A value of 0 will produce no scattering effect:•



Increasing the radius value can radically change the appearance of the material, from looking like leather to marble.



0.1



1

Instead of distributing all of the colors with the same amount, you can also choose different radius values for each of the RGB colors. For example, a material like clay or skin should have a higher red radius than green and blue.

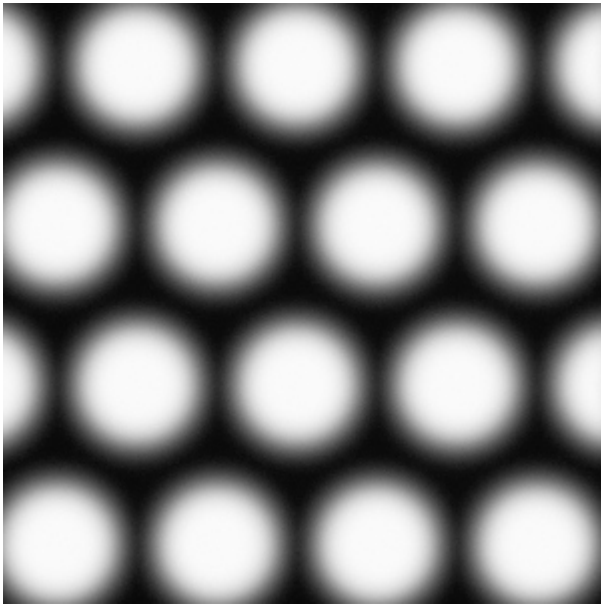


Red

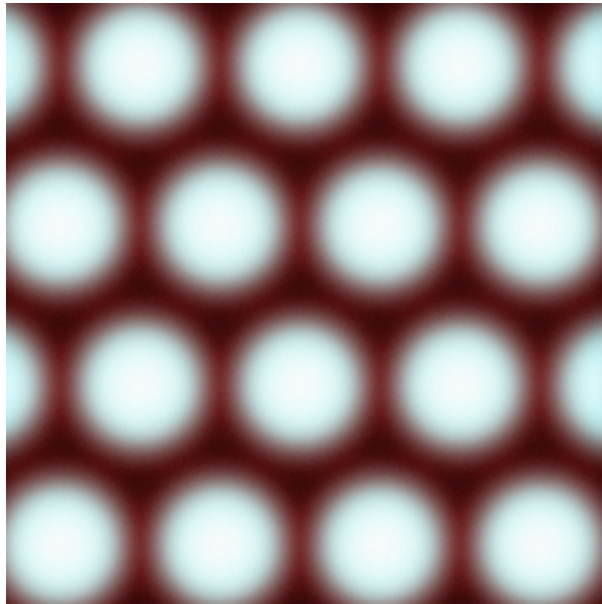


Green

The images below show the effect when increasing the red color of the radius. Notice the colored 'fringing' effect around the edges of the circles. The same effect occurs when gaussian blurring the red channel of the source image in a compositing package.



R:0.5 G:0.5 B:0.5



R:0.1 G:0.5 B:0.5

Diffusion Profile

Choose between empirical or cubic (default). If set to `empirical`, Arnold uses a more physically accurate subsurface scattering profile, that, with a single layer, can capture both surface detail and deep scattering.



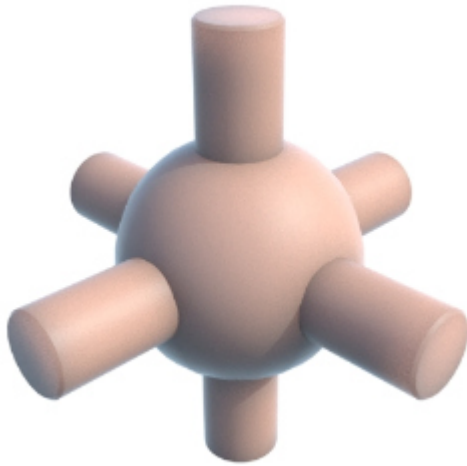
Cubic



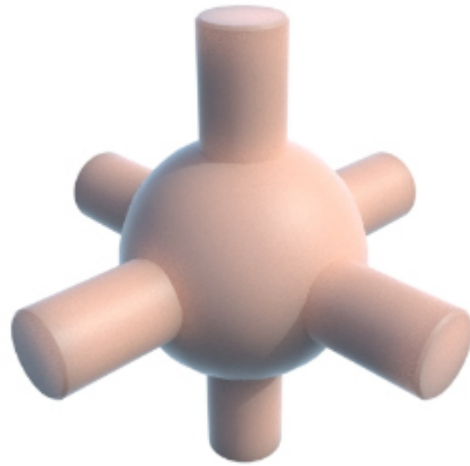
Empirical

SSS Between Objects

It is possible to tag multiple objects as belonging to the same SSS 'set' so that illumination will blur across object boundaries. A common use case might be blurring between teeth and gum geometry. It is enabled by adding the constant `STRING userdata sss_setname` to the same value on the objects in the set.



No Set Name

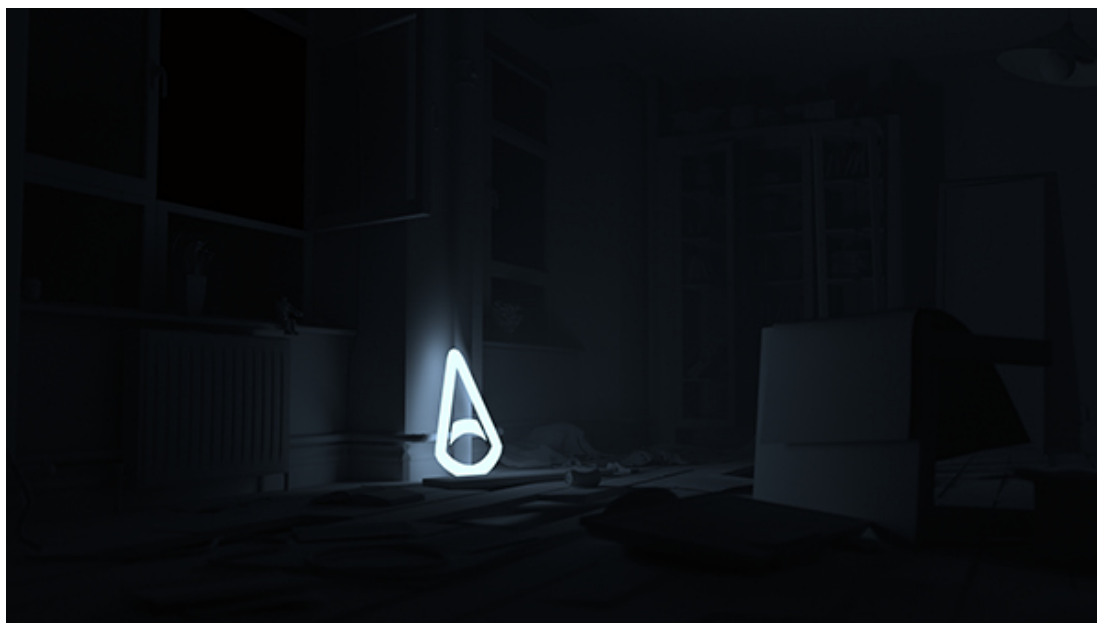


Objects In Same Set

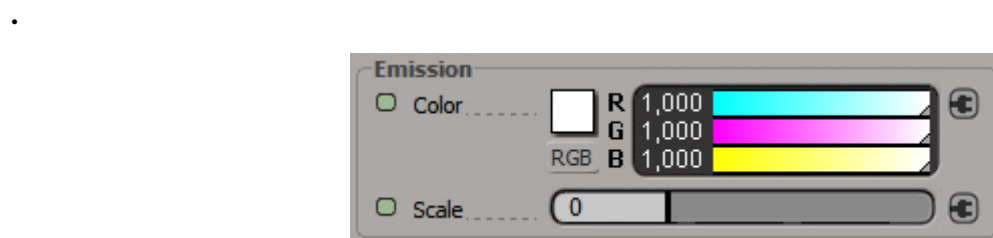


As well as the SSS component of the Standard shader there is also a specific [Skin shader](#).

Emission.



- This parameter gives the appearance that the material is emitting incandescent light.



Color

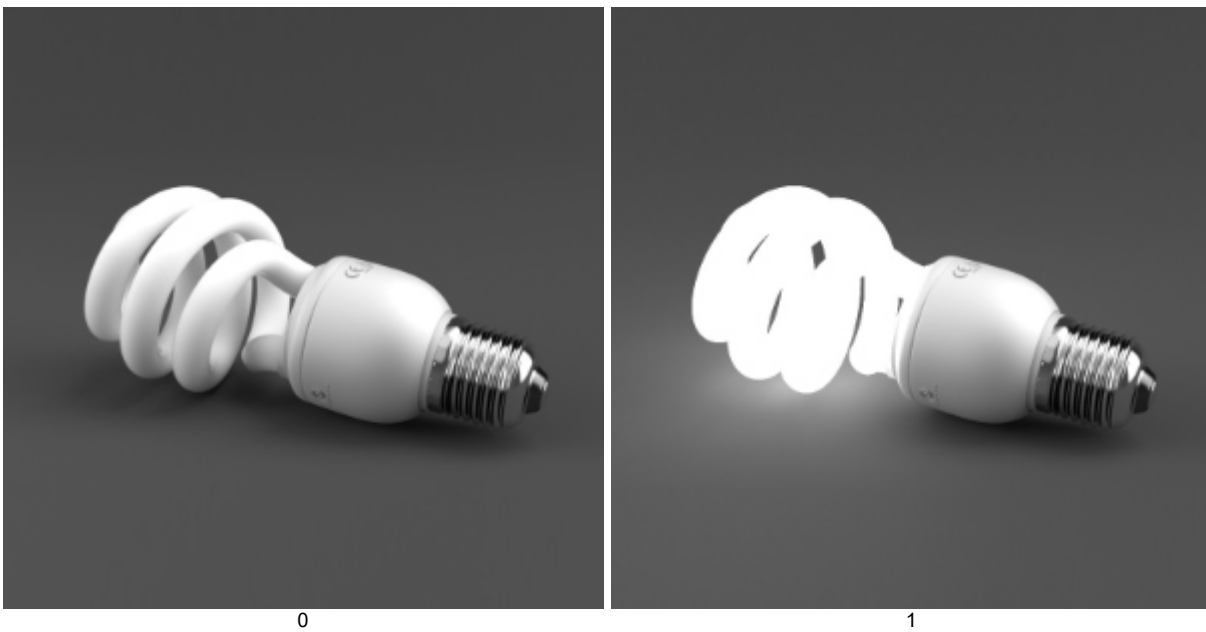
The emitted light color.



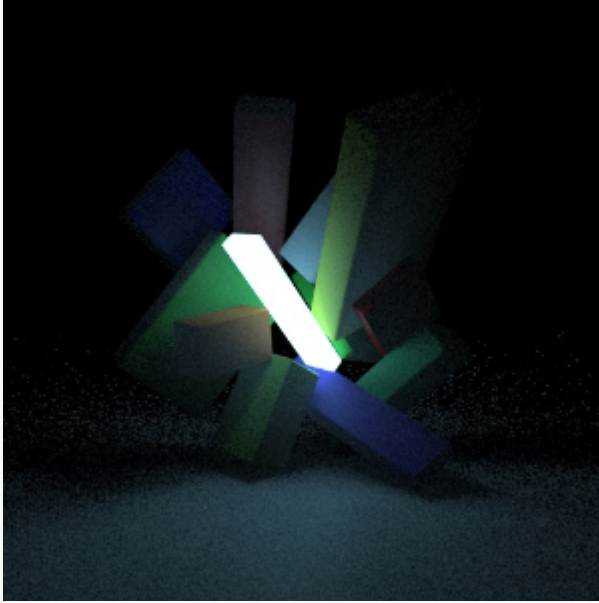
Texture map representing hot lava connected to Emission Color

Scale

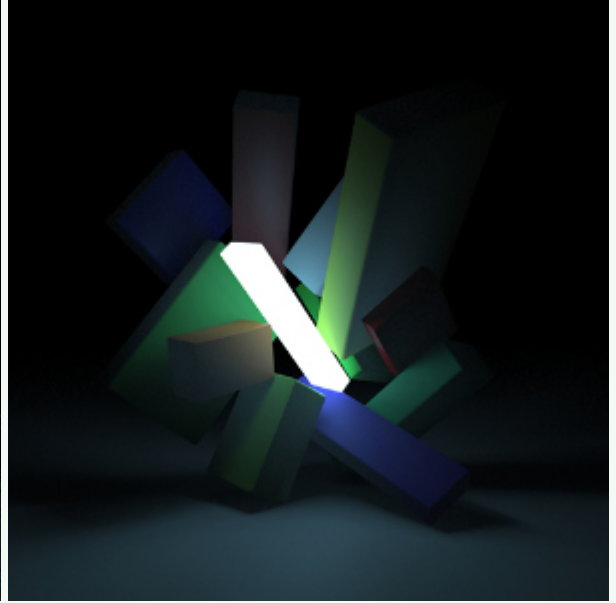
Controls the amount of emitted light. It can create noise, especially if the source of indirect illumination is very small (e.g. light bulb geometry).



Increasing the number of Diffuse samples will help to reduce any noise in dark, indirectly lit areas of a scene when using emission.



Diffuse Samples: 2

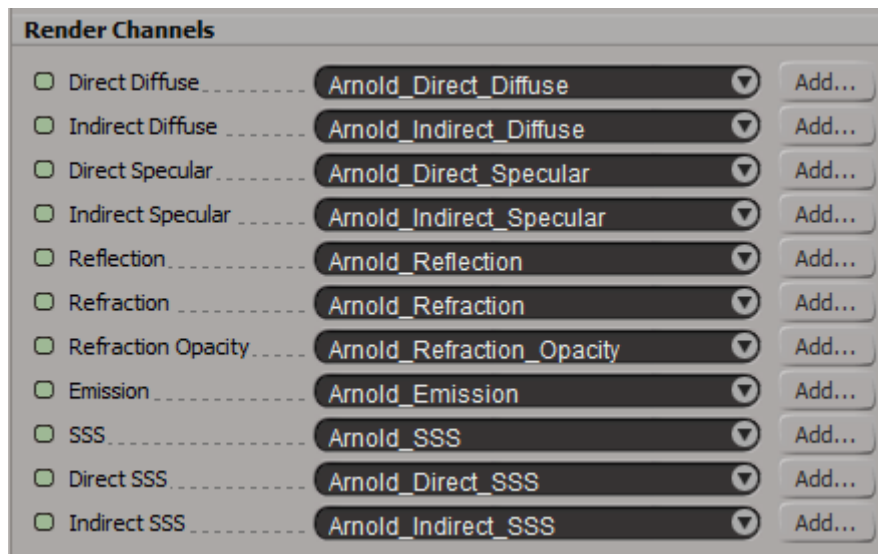


Diffuse Samples: 6



Note that a [Mesh light](#) may work better in a situation where you need an object to emit light that casts realistic ray-traced shadows.

Standard AOV Names



A list of available AOVs associated with the Standard shader. Each option creates a separate AOV render pass for that component.

- Direct Diffuse
- Indirect Diffuse
- Direct Specular
- Indirect Specular
- Reflection
- Refraction
- Refraction Opacity
- Emission
- SSS
- Direct SSS
- Indirect SSS

Volume Collector

! This shader is considered deprecated and will be removed in a future release. You should use the [Standard Volume](#) shader instead.



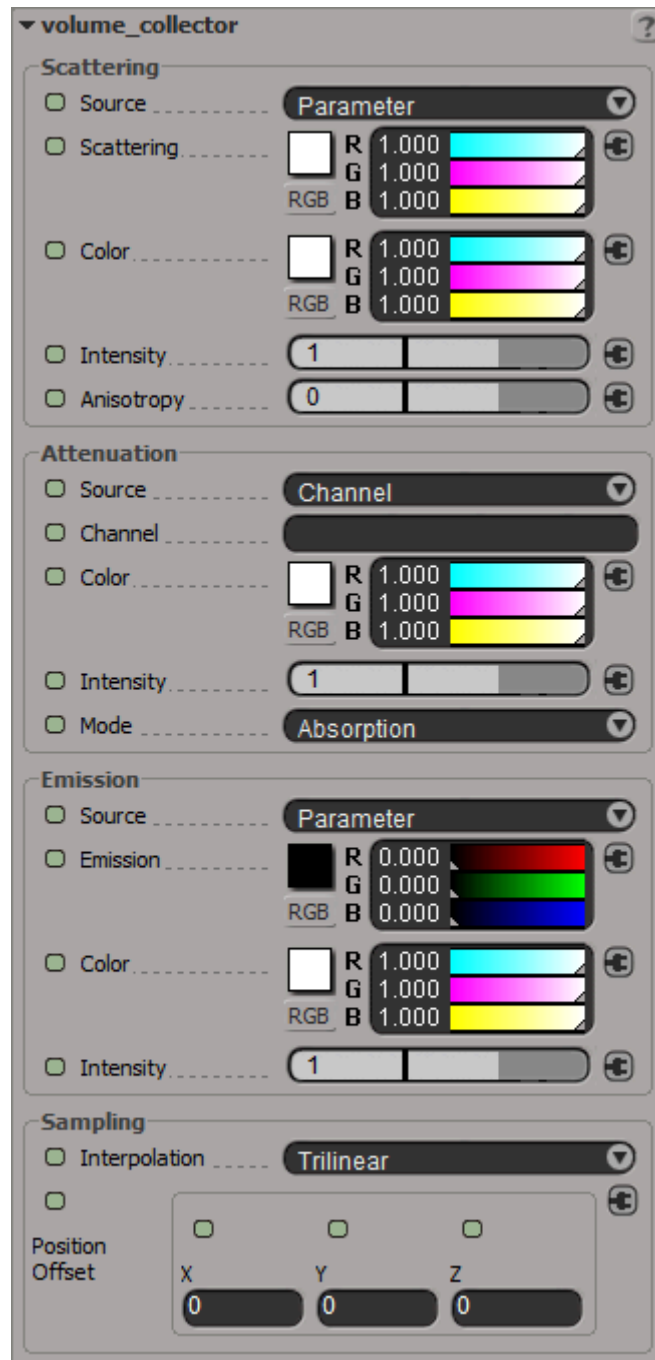
The Volume Collector has three color parameters called **Scattering**, **Attenuation**, and **Emission** which are simply used to set the scattering, attenuation and emission coefficients for the volume at the integration sample inside the volume. The **(Henyey-Greenstein) Anisotropy** parameter is an open interval $(-1, +1)$ where negative values mean backward anisotropic scattering, positive values mean forward anisotropic scattering and zero means isotropic scattering.

Be warned that the evaluation of a shader network for volume rendering is much more expensive than for surface shading, because the shader network is called many times per ray, once per ray march sample. So, in a production environment, it's much better to have a single shader doing all the work, instead of relying on potentially expensive inputs.

Each component (scattering, attenuation and emission) has a source enum, that can be either "parameter" or "channel". On "parameter", the parameter value is used (e.g. scattering). On "channel", the channel (e.g. scattering_channel) is used, its value coming from the density plugin. In addition, the attenuation's source has a third option, called "scattering", in which case the scattering source (parameter or channel) is used. On selecting an enum value (say "channel" for the scattering source), the unused parameter is hidden from the interface. On selecting "parameter", the channel gets hidden. When only channel names are used, the Volume Collector is very similar in functionality to the Arnold core **Density** shader. Note that the Volume Collector is also fully compatible with the old volume API.

i For information about using the Volume Collector with the Volume VDB, click [here](#).

i If you need to further edit the result of the reading of a channel, for instance to color correct it, there are the [Volume Sample RGB](#) and [Volume Sample Float](#) shaders.



The Volume Collector only works with an objects bounding box (and internally bounding sphere).

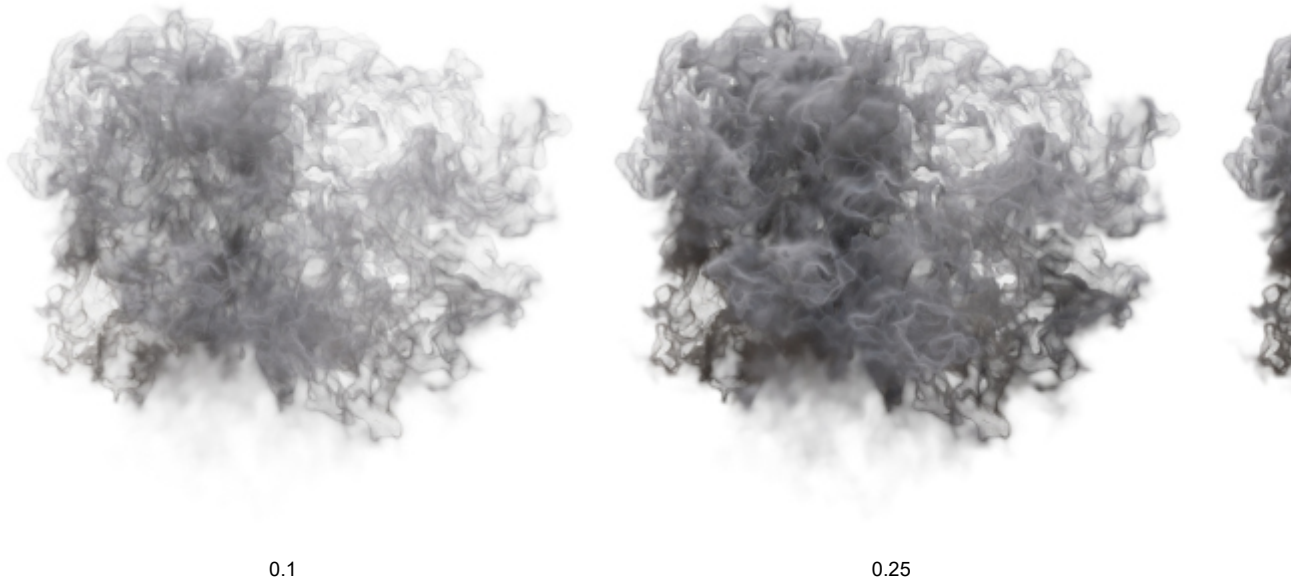


It is recommended that volume shading networks are kept as lean as possible. This is important for render times in the volume context because it is evaluated so often.

Scattering

Scattering•Parameter

This is the rate at which light is scattered (or reflected) at a given point. The greater the rate of scattering, the shorter the average distance a ray of light will travel through a volume before being bounced off of its course. Usually, the volume density is connected to this parameter.



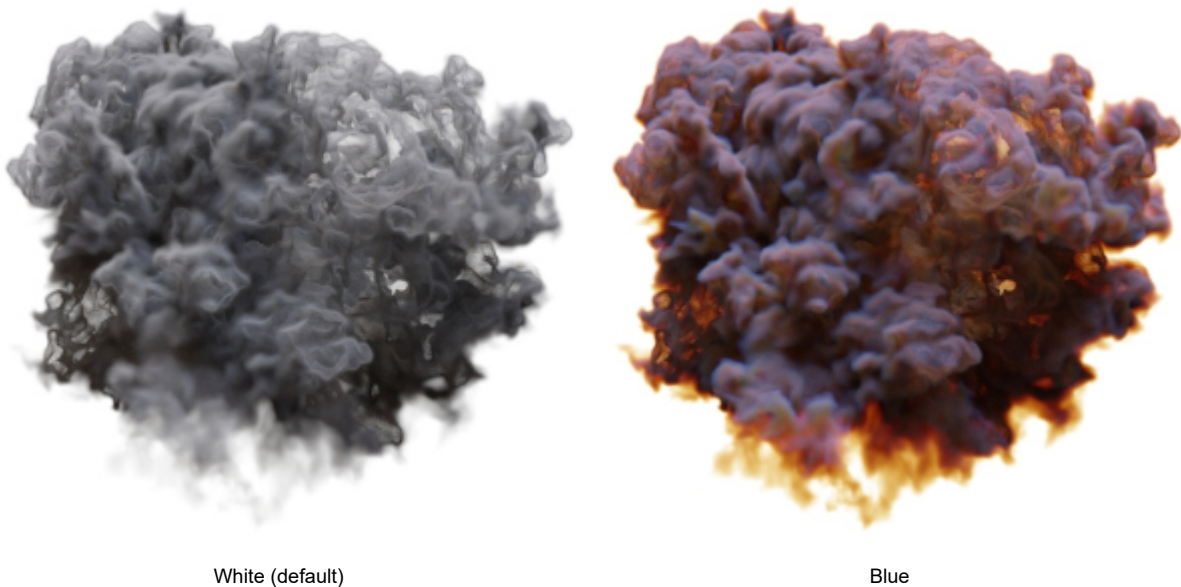
Scattering Channel

When not empty, this channel will be used to sample the scattering values for the volume instead of the Scattering parameter.

The sampling tab has additional parameters to control how data is sampled from volume channels.

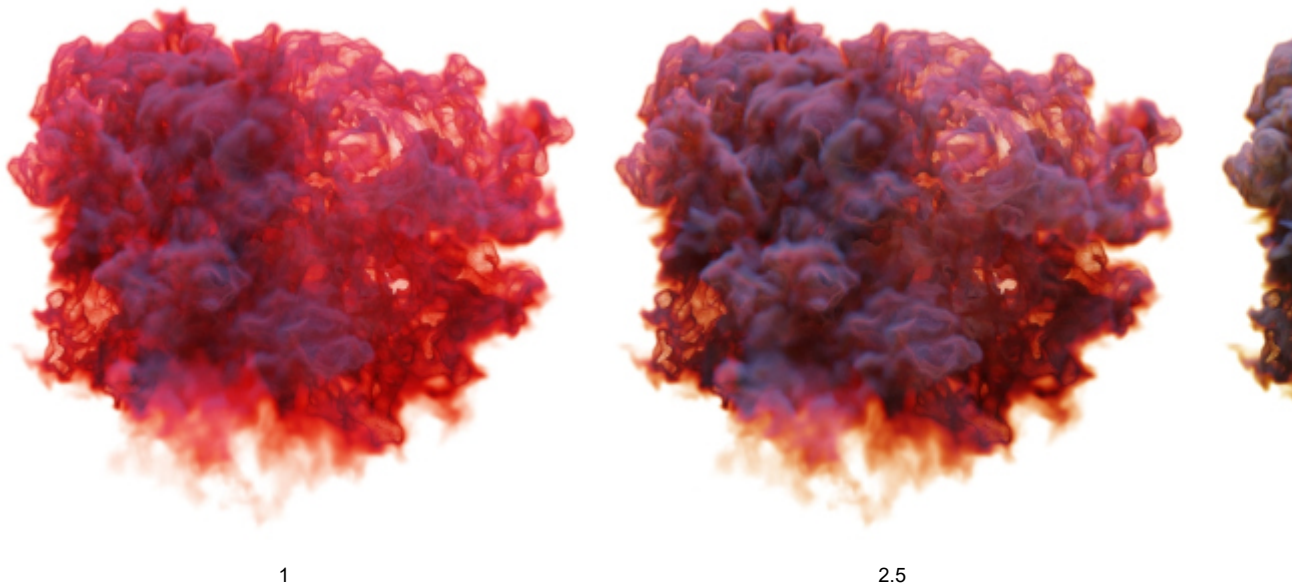
Scattering Color

A color to tint (multiply to) the scattering.



Scattering Intensity

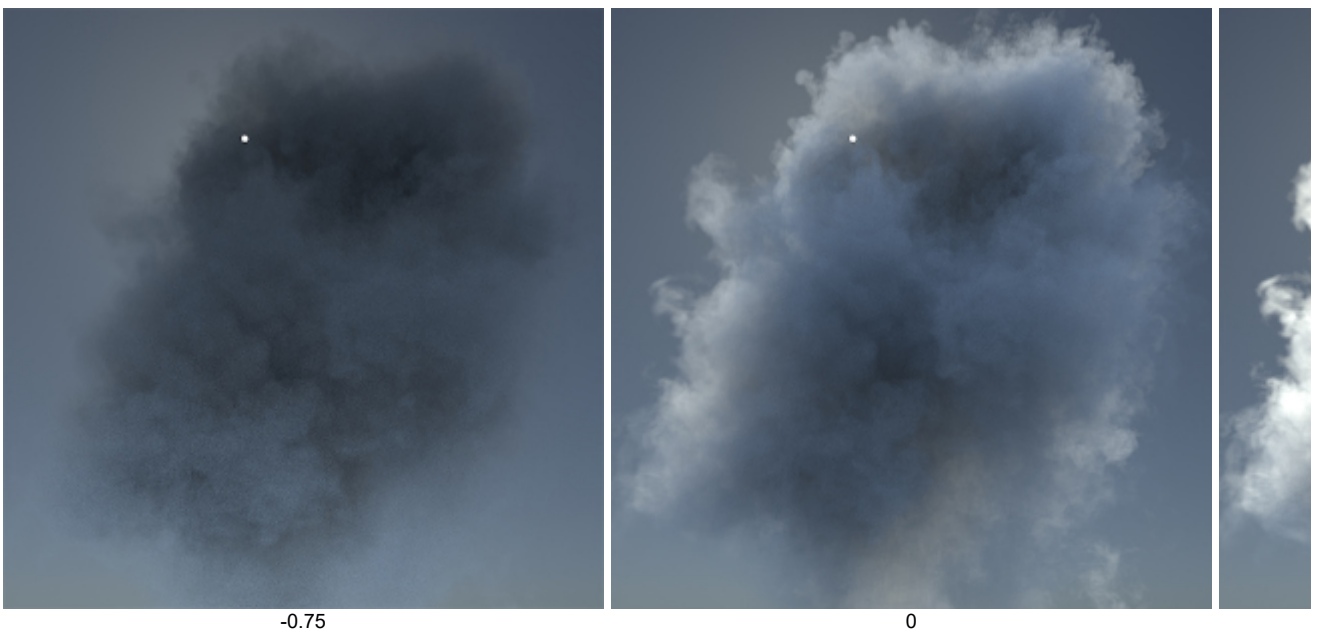
A scale factor to adjust the scattering.



Anisotropy

Heney-Greenstein Anisotropy coefficient between -1 (full back-scatter) and 1 (full forward-scatter). Default is 0 for an isotropic medium.

Note that values very close to 1.0 (above 0.95) or -1.0 (below -0.95) will produce scattering that is so directional that it will not be very visible from most angles, so such values are not recommended.

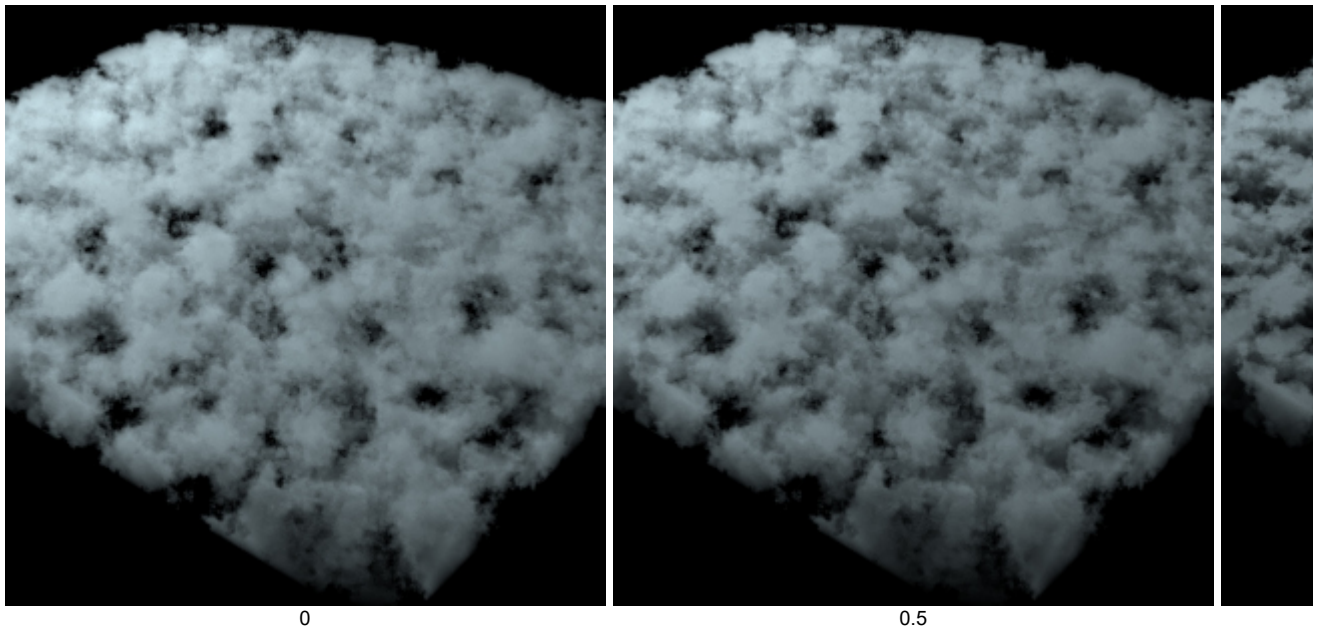


.

Attenuation

Attenuation •Parameter

The rate at which the intensity of a ray traversing a volume is lost.



Attenuation•Channel

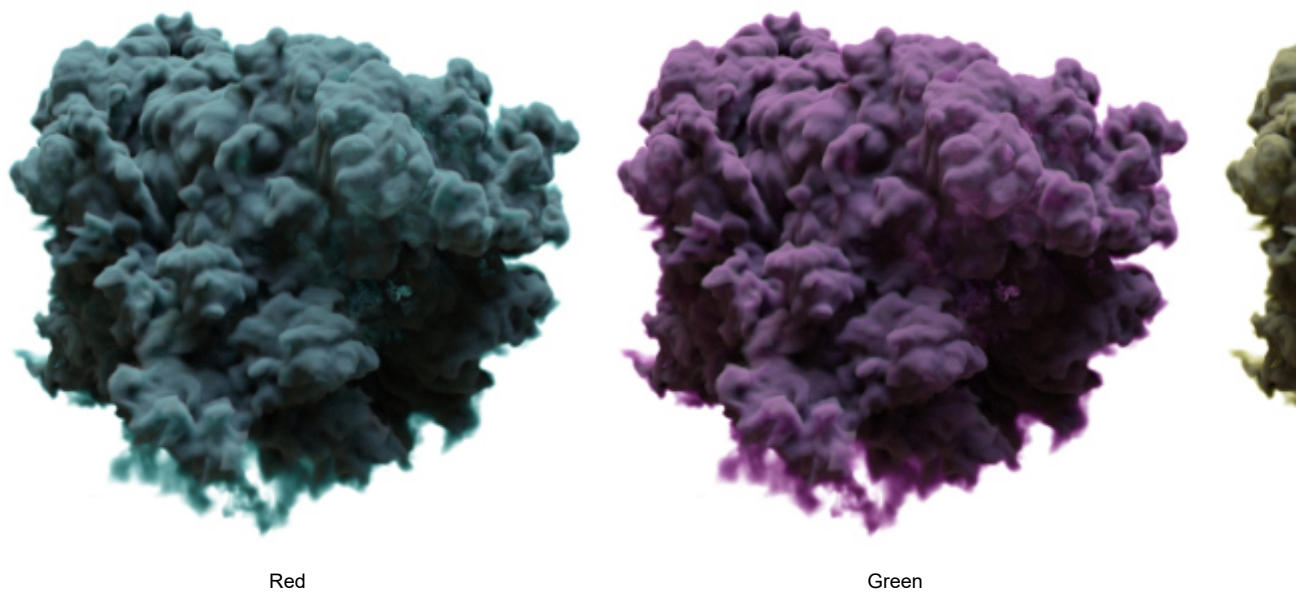
When not empty, this channel will be used to sample the scattering values for the volume instead of the Attenuation parameter. The sampling tab has additional parameters to control how data is sampled from volume channels.

Attenuation Scattering

Very often you want to use the same density field for both scattering and attenuation, modulo a factor. This parameter enables the reuse of the unmodified scattering value for the attenuation, avoiding a costly volume sample. Note that the same effect is achieved if the•Scattering Channel•and•Attenuation Channel•have the same non-empty value.

Attenuation•Color

A color to tint (multiply to) the attenuation.



Attenuation•Intensity

The rate at which the intensity of a ray traversing a volume is lost.



Emission

i Ensure that the emission attribute is driven by a VDB grid. If it is a fixed parameter value you may get artifacts.

Emission•Parameter

Emission is the rate at which a volume emits light at a given point. The light emitted by a volume is visible to GI. It will also be affected by any out-scattering or absorption effects in the volume. A heat (flames) grid would be connected to this parameter.

i Volumes can only receive indirect lighting from other emissive volumes when the [Volume Ray Depth](#) is above 0.

Emission•Channel

When not empty, this channel will be used to sample the emission values for the volume instead of the Emission parameter. The sampling tab has additional parameters to control how data is sampled from volume channels.

Emission•Color

A color to tint (multiply to) the emission.



Color ramp mapped to emission color

Emission•Intensity

A scale factor to adjust the emission.



1



5

Attenuation•Mode

The method of defining the volume attenuation characteristics.

Absorption

The intensity is lost due to absorption events only. The volume is guaranteed to conserve energy so it should work robustly no matter the lighting and GI settings, but can appear overly dark.

Extinction

The intensity is lost due to either scattering or absorption events. It is possible to obtain non-physically real scattering effects in the volume. More light will be scattered than what is actually attenuated by the volume. This is done by setting the attenuation coefficient to a value that is lower than the

scattering coefficient. This can be useful to have more control over the brightness of a volume, but causes energy conservation issues which can become troublesome depending on the lighting setup and GI settings.



Absorption



Extinction

Sampling

Position Offset

An object space offset to apply to the volume sampling position when using named channels to fetch the volume data. This is useful to displace the volume data.



You will need to connect a noise texture to the position offset to see any result.



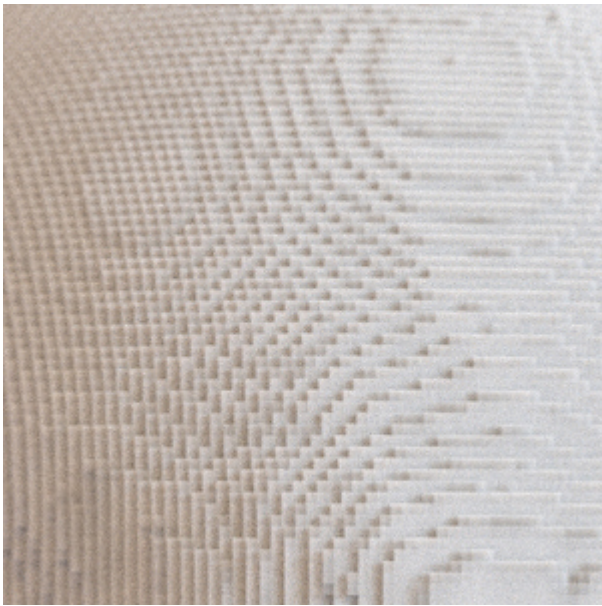
X: 0



X: 50

Interpolation

The voxel interpolation to use when sampling the volume data using named channels.



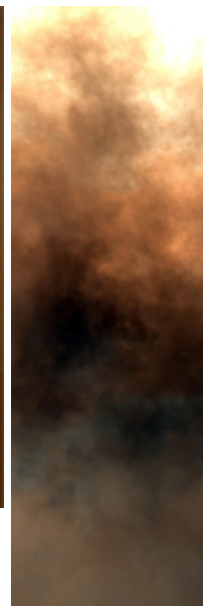
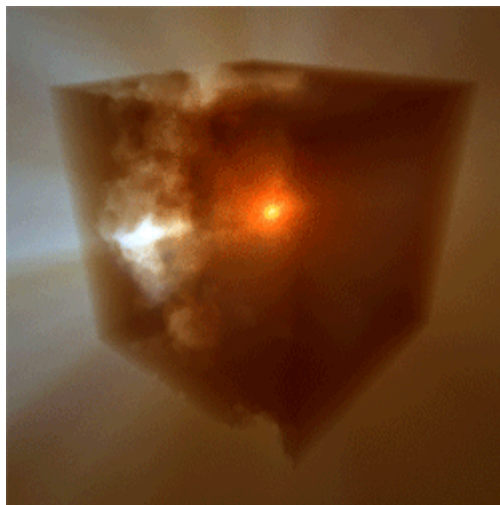
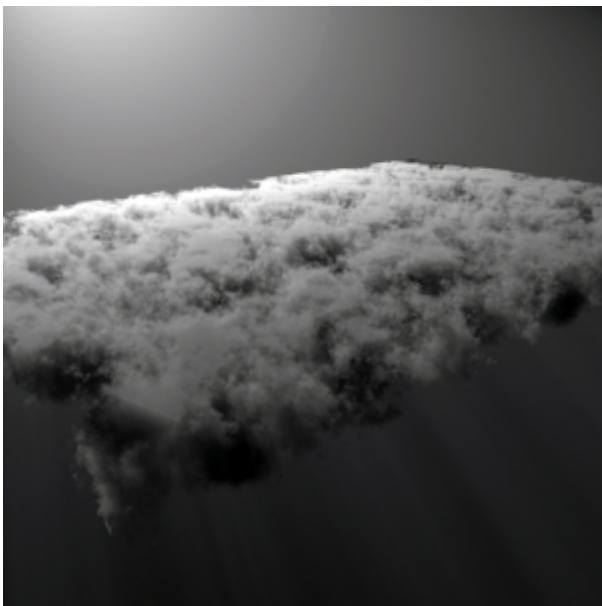
Closest



Trilinear



-
-



Examples using the volume collector shader ('God rays' achieved using Volume Scattering).

-

Textures

- [Maketx](#)
- [Tokens](#)

When using Arnold it is best to use a tiled mipmapped texture format such as .exr or .tx that has been created using maketx.♦♦

.tx textures are:♦

1. Tiled (usually the tiles are 64×64 pixels).
2. Mip-mapped.



If you already have tiled and mipmapped EXR's that have been created by another renderer, you won't need to convert those files to .tx

Due to (1), Arnold's texture system can load one tile at a time, as needed, rather than having to wastefully load the entire texture map in memory. This can result in faster texture load times, as texels that will never be seen in the rendered image will not even be loaded. In addition to the speed improvement, only the most recently used tiles are kept in memory, in a texture cache of default size 512 MB (can be tuned via options. texture_max_memory_MB). Tiles that have not been used for a long time are simply discarded from memory, to make space for new tiles. Arnold will never use more than 512 MB, even if you use hundreds or thousands of 4k and 8k images. But then, if you only use a handful of 1k textures, this will not matter.

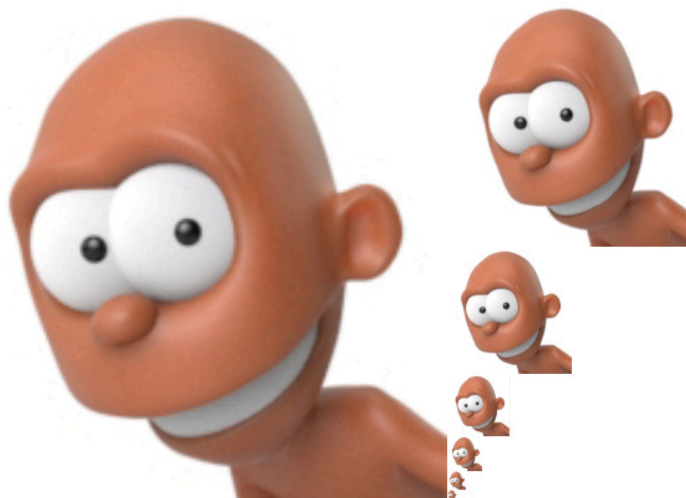
Due to (2), the textures are anti-aliased, even at low AA sample settings. Neither of these is possible with JPEG or other untiled/unmipped formats (unless you tell Arnold to auto-tilde and auto-mip the textures for you, but this is very inefficient because it has to be done per rendered frame, rather than a one-time pre-process with maketx).

It should be worth pointing out that .tx files are basically .exr or .tif files that have been renamed to .tx. This means .tx files can be read by image editors, although you may need to rename the file to .exr or .tif so that the image editor will load it. However, .tx files have a few extra custom attributes set that are not normally included in .exr/.tif files which can make rendering even faster. ♦For instance, they will include a hash so that if you try to load two different files that contain the same data, Arnold only needs to load this data once.

Benefits of Using .TX

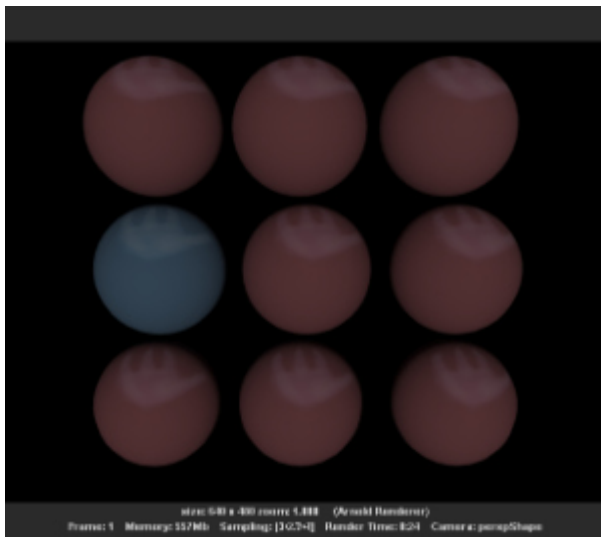
The first benefit is that you are assured of having mipmaps and tiles. These will dramatically improve time to the first pixel, overall render time, and allow using a smaller texture cache. This should be considered mandatory, with the possible exception being don't use .tx for the few images you are actively modifying if you don't want to wait for them to be constantly converted to .tx every time you make a change.

The previous level you could get from non-tx files that you saved with tiles and mipmapping. The second level of benefit is only with .tx files and that involves maketx adding metadata that lets Arnold make more optimizations, such as being able to detect duplicate textures and only loading a single copy into memory or detecting constant color images, such as an all-black [UDIM](#), and instead of storing all those black pixels in memory Arnold can special case that.

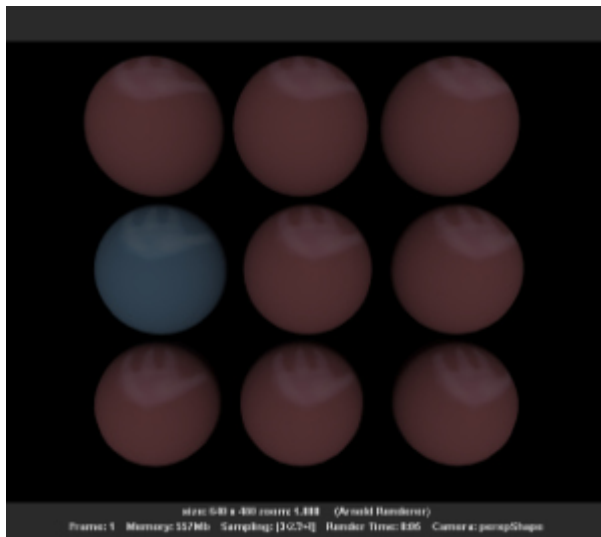


An example of mipmap image storage. The first image on the left is shown with filtered copies reduced in size.

The example below shows a speed increase of seven times when using .tx files compared to .jpg files:



JPEG: 42 seconds



TX: 6 seconds

Manually Generate TX Textures

The process is quite simple. You will need the maketx.exe utility, the texture that needs to be converted, and a dos shell. Below is an example of how you would convert a .tif file to a .tx file.

```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Documents\projects\MakeTx\sourceimages> maketx --oio C:\Users\MakeTx\sourceimages\texture.tif
```

Folder which contains the texture that needs to be converted.
The maketx.exe utility has also been copied to this folder.

Maketx command with the optimize '--oio flag' and the pathname
of the .tif file that needs to be converted.

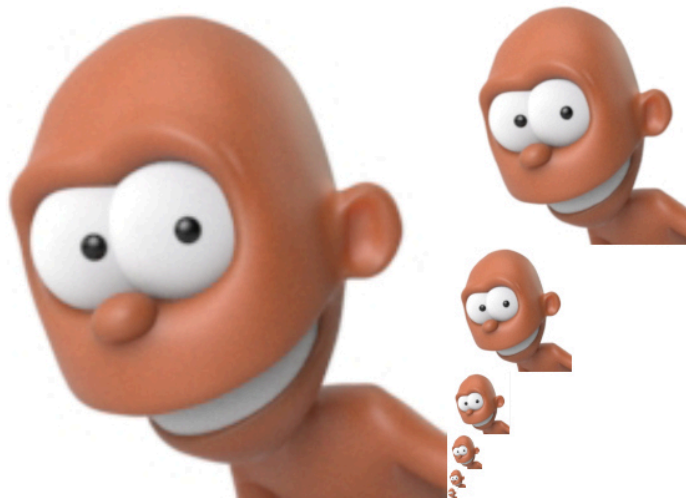
Maketx

Maketx is a command-line utility to convert images to tiled, MIP-mapped textures, similar to `txmake` in Pixar Renderman. It is part of OpenImageIO (<http://www.openimageio.org>) and was developed by Larry Gritz at Sony Pictures Imageworks.

Benefits of Using .TX

The first benefit is that you are assured of having mipmaps and tiles. These will dramatically improve time to the first pixel, overall render time, and allow using a smaller texture cache. This should be considered mandatory, with the possible exception being don't use `.tx` for the few images you are actively modifying if you don't want to wait for them to be constantly converted to `.tx` every time you make a change.

The previous level you could get from non-`tx` files that you saved with tiles and mipmapping. The second level of benefit is only with `.tx` files and that involves maketx adding metadata that lets Arnold make more optimizations, such as being able to detect duplicate textures and only loading a single copy into memory or detecting constant color images, such as an all-black `UDIM`, and instead of storing all those black pixels in memory Arnold can special case that.



An example of mipmap image storage. The first image on the left is shown with filtered copies reduced in size.

```

michael@ws1:~/Documents/HtoA/scenes/textures/tif
File Edit View Search Terminal Help
[michael@ws1 tif]$ maketx -u -v --oio --checknan --filter lanczos3 circle.tif -o circle.tx
Reading file: circle.tif
SHA-1: 54EFA3A63F131E1A4EA747B81D386C0CCF6E6B4
Writing file: circle.tx
Filter "lanczos3" width = 6
Mipmapping...
128x128
64x64
32x32
16x16
8x8
4x4
2x2
1x1
Wrote file: circle.tx
maketx Runtime statistics (seconds):
total runtime: 0.09
file read: 0.00
file write: 0.03
initial resize: 0.00
mip computation: 0.04
unaccounted: 0.01
maketx memory used: 828.0 MB
[michael@ws1 tif]$

```

The maketx command with verbosity, converting from a `tif` to a `tx`

Usage

`maketx [options] file...`

eg: `maketx -v -u --oio --checknan --filter lanczos3 path/to/fileIn.tif -o path/to/fileOut.tx`

.



When pre-processing your texture files with maketx make sure you add the `--oio` flag which will generate TX files optimized for Arnold that can render even faster by forcing the tile size to be optimal for OpenImageIO (64x64). It also stores channels interleaved (RGBARGBA...) making them faster to read and enables constant color detection: if the image is a constant color it will convert it to a tiny single color .tx file. Additional metadata and an image hash (fingerprint) are embedded by default. The fingerprint lets OIO detect duplicate images (not tiles) when opening new files. Generally using maketx will result in better texture I/O performance than using txmake.

Note also that if you need your textures to stay compatible with Renderman, there is a `--prman` switch that will ensure the tile size matches what Renderman expects.

If the output color space is not the same bit depth as input color space, then the data format needs to be set to the proper bit depth using the `-d` option.



The best workflow is to always generate .tx files using maketx as the last step before rendering and use those .tx files just for rendering. This will ensure the best performance. Also, ensure that your textures are in linear color space before converting them with maketx. No information is lost when converting High Dynamic Range images to the .tx format.

Parameters •

flag	description
<code>--help</code>	Print help message
<code>-v</code>	Verbose status messages
<code>-o•%s</code>	Output filename
<code>--threads %d</code>	Number of threads (default: #cores)
<code>-u</code>	Update mode
<code>--format %s</code>	Specify output file format (default: guess from•extension)
<code>--nchannels %d</code>	Specify the number of output image channels.
<code>-d %s</code>	Set the output data format to one of: uint8,•sint8, uint16, sint16, half, float
<code>--tile %d %d</code>	Specify tile size
<code>--separate</code>	Use planarconfig separate (default: contiguous)
<code>--fov %f</code>	Field of view for envcube/shadcube/twofish
<code>--fovcot %f</code>	Override the frame aspect ratio. Default is•width/height.
<code>--wrap %s</code>	Specify wrap mode (black, clamp, periodic,•mirror)
<code>--swrap %s</code>	Specify s wrap mode separately
<code>--twrap %s</code>	Specify t wrap mode separately
<code>--resize</code>	Resize textures to power of 2 (default: no)
<code>--noresize</code>	Do not resize textures to power of 2•(deprecated)
<code>--filter %s</code>	Select filter for resizing (choices: box•triangle gaussian catrom•blackman-harris sinc•lanczos3 radial-lanczos3 mitchell bspline, disk,•default=box)
<code>--nomipmap</code>	Do not make multiple MIP-map levels•
<code>--checknan</code>	Check for NaN/Inf values (abort if found).
<code>--Mcamera %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f</code>	Set the camera matrix
<code>--Mscreen %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f</code>	Set the camera matrix
<code>--hash</code>	Embed SHA-1 hash of pixels in the header (deprecated. hashes are always computed).
<code>--prman-metadata</code>	Add prman specific metadata
<code>--constant-color-detect</code>	Create 1-tile textures from constant color•inputs
<code>--monochrome-detect</code>	Create 1-channel textures from monochrome•inputs
<code>--opaque-detect</code>	Drop alpha channel that is always 1.0
<code>--stats</code>	Print runtime statistics
<code>--mipimage %s</code>	Specify an individual MIP level
Basic modes (default is plain texture)	•

<code>--shadow</code>	Create shadow map
<code>--envlatl</code>	Create lat/long environment map
<code>--envcube</code>	Create cubic env map (file order: px, nx, py,•ny, pz, nz) (UNIMP)
Color Management Options	•
<code>--colorconvert %s %s</code>	Apply a color space conversion to the image. If the output color space is not the same bit depth as input color space, it is your responsibility to set the data format to the proper bit depth using the -d option. (choices: linear, sRGB, Rec709)
<code>--unpremult</code>	Unpremultiply before color conversion, then premultiply after the color conversion. You'll probably want to use this flag if your image contains an alpha channel.
Configuration Presets	•
<code>--oiio</code>	Use OIIO-optimized settings for tile size,•planarconfig, metadata, and constant-color•optimizations.
<code>--prman</code>	Use PRMan-safe settings for tile size, planarconfig, and metadata.

 Do not linearize textures used to drive scalar values as you will lose precision values.

 Maketx supports the following file formats:

-
- BMP
- Cineon
- DDS
- DPX
- FITS
- HDR/RGBE
- ICO
- IFF
- JPEG
- OpenEXR
- PNG
- PNM
- PSD (flattened)
- RLA
- SGI
- Softimage PIC
- Targa
- TIFF
- Zfile



Tokens

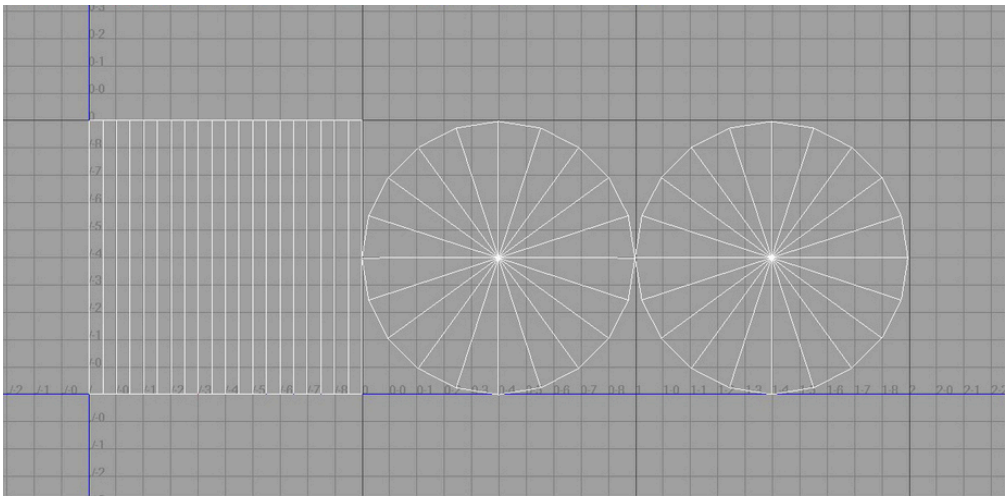
SltoA supports the following textures tokens, including both the•<udim>•(Mari) and•<tile>•(Mudbox)•tokens.•

Examples

Token	Texture Path	Result
<udim>	texture_<udim>.tx	texture_1001.tx, texture_1002.tx, texture_1003.tx, etc.
<tile>	texture_<tile>.tx	texture_v1_u1.tx, texture_v1_u2.tx, texture_v1_u3.tx, etc.

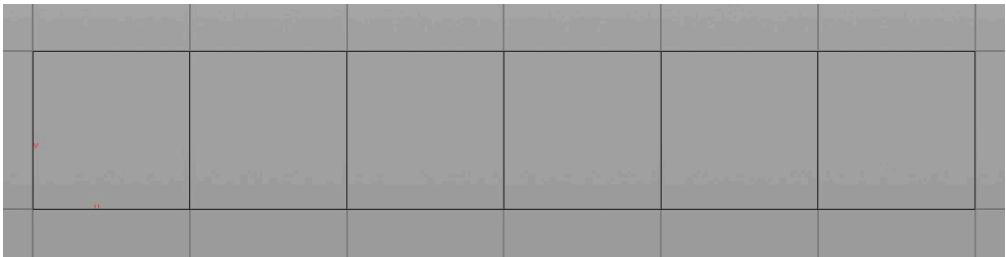
•

<udim>



A cylinder with its side in u0,v0, base in u0,v1, and top in u0,v2

UDIM values are a way of representing the integer coordinates of a texture, from the coordinates of its bottom-left corner in UV space. This way, multiple textures can be used to cover a model, rather than a single, much larger, texture.



Faces of a cube unwrapped in u0,v0, to u0,v5,

•

It is a 4 digit number starting at 1001, created using the formula,•1001 + u + (10 * v)

In the example above the side would be 1001, base 1002, and top 1003.

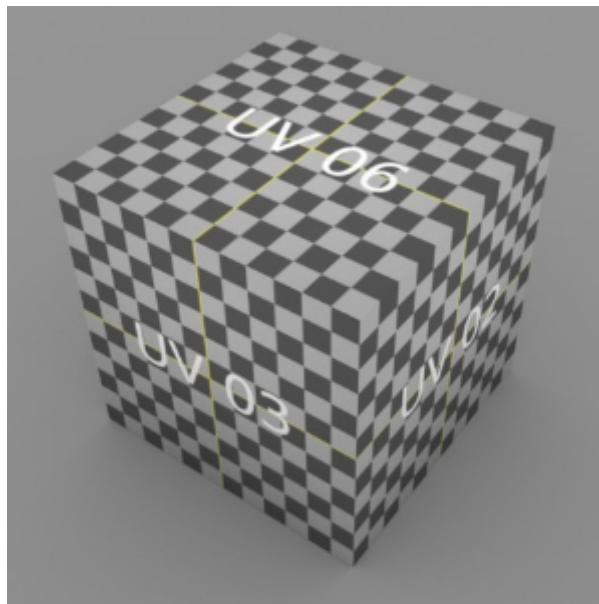
To use this method in your file inputs, use the•<udim>•tag in the file name, which is then replaced with the•appropriate•number.

texture.<udim>.exr•will be read as•texture.1001.exr when looked up for rendering.

Name	
	checkerUV_1001.tx
	checkerUV_1002.tx
	checkerUV_1003.tx
	checkerUV_1004.tx
	checkerUV_1005.tx
	checkerUV_1006.tx

•

texture_<udim>.tx will be read as texture_1001.tx when looked up for rendering.



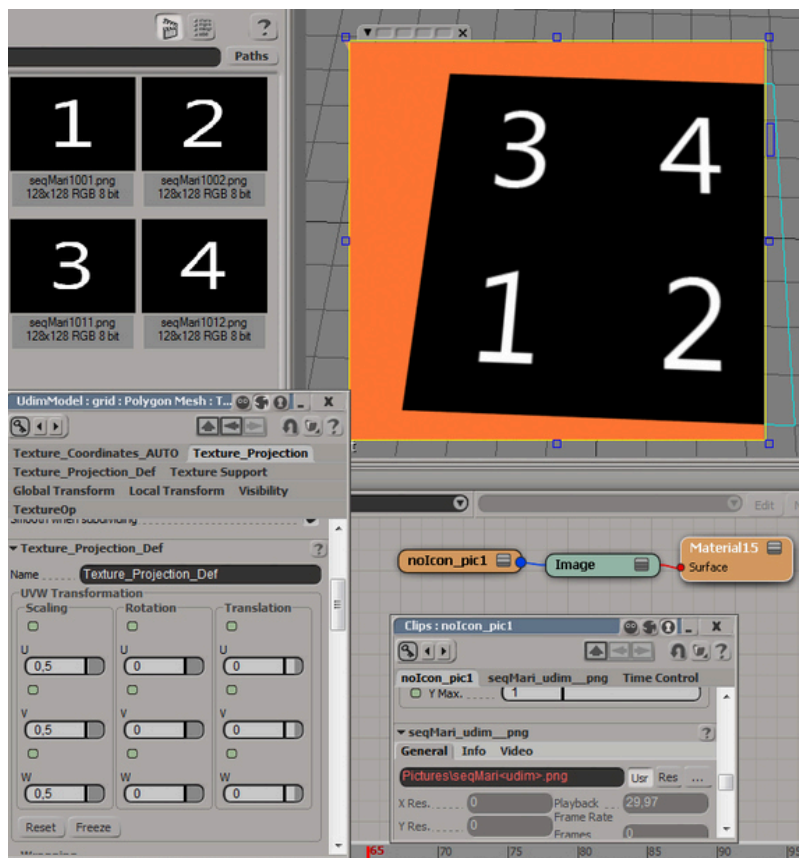
Check out Ben Neall's [page](#) for a clear description of UDIM numbering.

•

When the path to the image used in `sib_image_clip` contains such a token, the actual name of the image to be used is resolved, depending on the current UV coordinates.

Here is one Mari case in action.

Note that, unfortunately, the path turns red when you edit it to replace part of its name with the <udim> token, but there is nothing we can do at the moment.

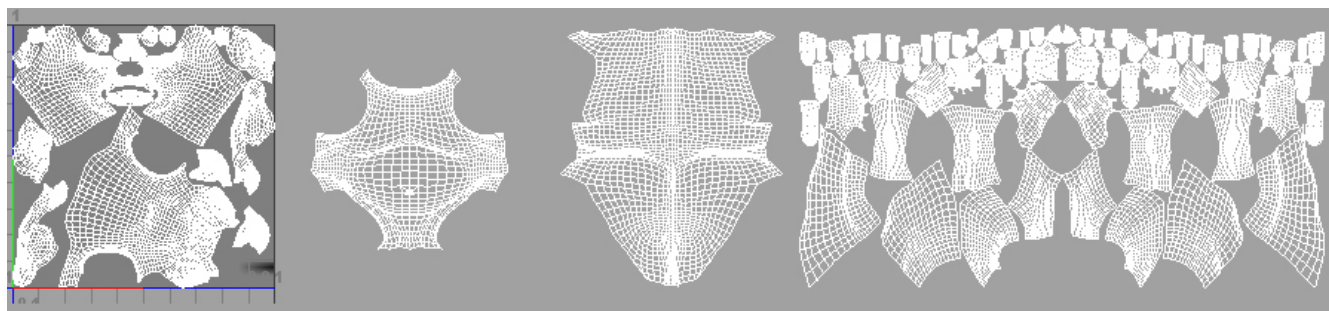


•

<tile>

When using the `<tile>` token, multiple textures can be used to cover a model, rather than a single, much larger, texture. It is the notation at which it is exported in UV texture space. This starts at 1001 (0,0) and continues in U for another 10 offsets. It then progresses onto the next row of V, meaning for every 10 in U it moves 1 in V.

•



UV's should be arranged into separate UV tiles in order to maximize texture space

•

You can use a single `File` texture to open textures composed of multiple images that correspond to the grid tiles in your UV layout. You can, therefore, open and render `high-resolution` textures produced by 3D painting applications such as Mudbox, and is a better alternative to using UVsets.

•

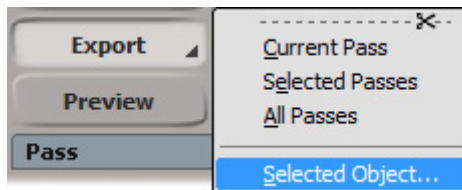
Exporting to file

Exporting assets to .ass archives

The `.ass` extension stands for Arnold Scene Source.

There are three ways to export assets to .ass files:

- 1. Select the objects to export, and pick Export->Selected Object...



You can export single objects as well as a whole hierarchy made out of multiple objects. •



Important note: During export, Arnold will honor its current render settings, for example, whether motion blur is enabled. You should think about whether you will want to use motion blur (and so enable it or not) before you export objects to be used as stand-ins later on.

- 2. From the script editor, with the command

```
SITOA_ExportScene(frameStart, frameEnd, frameStep, createStandIn,  
selectionOnly, filename)
```

- frameStart: start frame of the sequence to be exported.
- frameEnd: end frame.
- frameStep: frame step.
- createStandIn: if true, the options and camera are not exported, and a .asstoc file is also created, hosting the bounding box information of the exported objects. This is usually the option you want to use if you plan to use the exported ass file as a stand-in later.
- selectionOnly: if true, only the selected objects are exported.
- filename: the output filename.

Filename supports tokens. When exporting a sequence, you can use the [Frame] token in the filename, to have it resolved correctly at each frame. •

Example :

```
SITOA_ExportScene(1, 10, 1, false, false, "/usr/tmp/Test.[Frame  
#4].ass");
```

Exports 10 .ass files, named /usr/tmp/Test.0001.ass, /usr/tmp/Test.0002.ass, etc.

- 3. From the rendering options, using the Export Frame or Export Animation buttons in the ASS Archives tab.
-

SltoA Paths

Search • Paths

In the rendering options, you can set the paths where Arnold will look for shaders, procedurals, and textures.

For shaders and procedurals, they are automatically loaded if contained into the directory pointed by the `ARNOLD_PLUGIN_PATH` directory.

Also, you can use the following environment variables that will be read and set as your defaults when creating a new scene:

- `SITOA_SHADERS_PATH`: for shaders

- `SITOA_PROCEDURALS_PATH`: for procedurals

-

The paths can be changed at any time from the Arnold Render Options. If doing so, the following rule applies:

 Except for `ARNOLD_PLUGIN_PATH`, which is Arnold-native, all and only the paths in the search paths are used for searching the shaders (procedurals). If the field is void, the SltoA bin path is used.

-

If you don't need to link to shaders other than the ones coming with SltoA, leave the field void. So, when loading a scene saved with a previous SltoA version, the shaders of your current SltoA version will be loaded.

The rule also means that if you set the shader search path to, say, a workgroup location only, none of the Softimage legacy shaders will be found. So, if you need to have your workgroup's shaders used, be sure to add also the SltoA bin directory to the search path, separated by a ';' character. The order does not matter.

Example: I have some custom shaders in a workgroup located under:

`C:\dev\Arnold\WG`

and my SltoA addon installed under:

`C:\Users\stefano\Autodesk\Softimage_2010_SP1\Addons\SItoA`

Then, to have everything working, I have to set:

`Shaders•Search•Path•==C:\Users\stefano\Autodesk\Softimage_2010_SP1\Addons\SItoA\Application\Plugins\bin\nt-x86-64;C:\dev\Arnold\WG\Application\bin\nt-x86-64`

-

Note that the full path to the shader DLLs is required.

-

Environment Variables

Environment variables are supported for the search paths, through the `[ENV_VAR]` syntax.

Let's reuse the above example, and say that in my `setenv.bat`, I set two env vars:

`set•MY_WG_BIN=C:\dev\Arnold\WG\Application\bin\nt-x86-64`

`set•MY_SITOA_ADDON=C:\Users\stefano\Autodesk\Softimage_2010_SP1\Addons\SItoA`

Then, I can simply set:

`Shaders•Search•Path•==[MY_SITOA_ADDON]\Application\Plugins\bin\nt-x86-64;[MY_WG_BIN]`

An environment variable must point to a single location. Something like:

`set•MY_PATHS•==C:\dev\Arnold\WG\Application\bin\nt-x86-64;C:\Users\stefano\Autodesk\Softimage_2010_SP1\Addons\SItoA`

will not work.

-

Absolute vs Relative paths



If Use Absolute Texture Paths (From the ASS Archives tab of the rendering options) is enabled, the absolute paths is exported and optionally translated by the linktab file if Translate Paths is enabled. Otherwise, the relative paths, with respect to Textures Search Path, is exported.

Say we have to export a texture whose full path is `C:\img\head\eye.jpg`

If you set Textures Search Path to `"C:\img"`, then the texture is exported as `"head\eye.jpg"`,

If set to `"C:\temp"`, the texture is exported as `"..\img\head\eye.jpg"`,

and so on.

•

In order to have the conversion working, the Textures Search Path must exist.

Example: Textures Search Path == `C:\invalid` (where `C:\invalid` does not exist on disk)

then, a warning message is fired, and we export the plain file name, so

Exported path == `eye.jpg`

•

Textures Search Path can be made of more than one path, separated by `';`

Example: Textures Search Path == `C:\img\head;C:\temp`

For multiple search paths, we convert to relative paths with respect to the first of the search paths for which the conversion was successful.

So, if we have 2 search paths, we always try to convert with respect to the first one. If failing, we try the second one, and so on. In the case of failure (none of the search paths worked), we export the plain file name and we log a warning.

This information also applies to procedural search paths, with respect to the Absolute Procedural Path flag.

•

Cross-platform Path Translation

For cross-platform path translations we need to specify an environment variable:

• `SITOA_LINKTAB_LOCATION`

When this variable is declared, the "Translate Paths" parameter in the Arnold Render Options will be enabled. This variable must point to the Softimage linktab.ini file with an absolute path including the filename.

•

Defining the linktab File

The linktab file is used when the render network uses a mix of Windows and other operating systems. Each line in a linktab file contains a Windows-style path and a UNIX-style path, indicating where Softimage, and resources such as textures, are located on both operating systems. This allows the rendering server to find the required files on the clients regardless of the operating system.

•

Most linktab files contain only one line, indicating where Softimage is located on both platforms. Here is a sample linktab file:

```
C:\Softimage...../usr/Softimage
```

• The Windows path must come before the Unix-style path and they must be separated by a tab, not spaces. •

• Linux path names are case-sensitive, but Windows paths are not (they are case-insensitive).

• Do not use slashes (`\` or `/`) at the end of the linktab paths. •

• If you are using textures or memory-mapped images, you must have entries that point to the directories containing them. •

• You can use an exclamation mark to distinguish a mounted volume from the rest of the path. For example, if `\\foobar\users\fred` (Windows) is equivalent to `/home/fred` (Linux) and `F:\foobar\users` is a mounted volume, the line in the linktab file would look like this:

•

```
\\foobar\users!fred • • /home/fred
```



The linktab file must be present on the server and client computers, in the directory specified by the `SI_LINKTAB_LOCATION` environment variable.

•

• `SI_LINKTAB_LOCATION` can define both the path and file name or just the path; if no file name is explicitly set, Softimage assumes the name will be `linktab.ini`.

• When a scene is imported from SOFTIMAGE|3D into Autodesk Softimage, Softimage searches in `SI_LOCATION` for `linktab.ini`. Define `SI_LOCATION` in `setenv.bat` (Windows) or `.xsi_5.0` (Linux) as the directory where Softimage is installed and place a copy of `linktab.ini` there (if you want to reuse the linktab file used by SOFTIMAGE|3D; otherwise, just make `SI_LOCATION` point to the same path and linktab name as defined by `SI_LINKTAB_LOCATION` for Softimage).

•



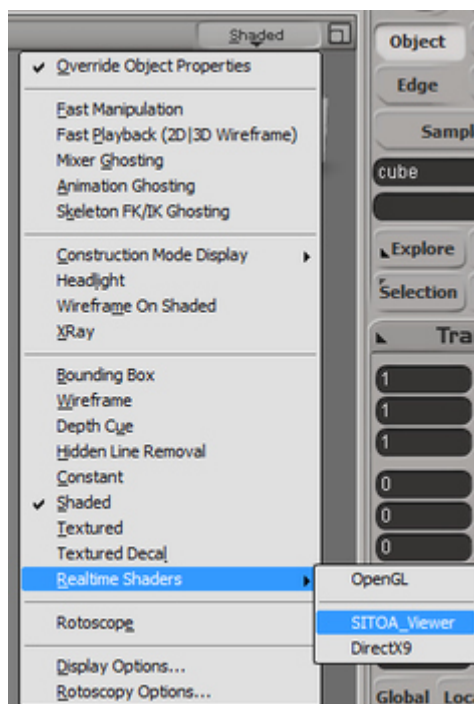
For details on how Softimage uses the `linktab` file, see 'Configuring the `linktab.ini` File' in the Softimage 'Setup & Licensing Guide'.

•

•

The SItOA Viewer

SItOA includes a graphic sequencer, called SItOA_Viewer, allowing you to preview the content of the procedural (also called standin) nodes. From any window, set the display mode to Realtime Shaders -> SItOA_Viewer .



The viewer loads all the procedurals in the scene (both archive-types and compiled so/dll) and draws them in ogl, in either wireframe, points, or bounding box mode.

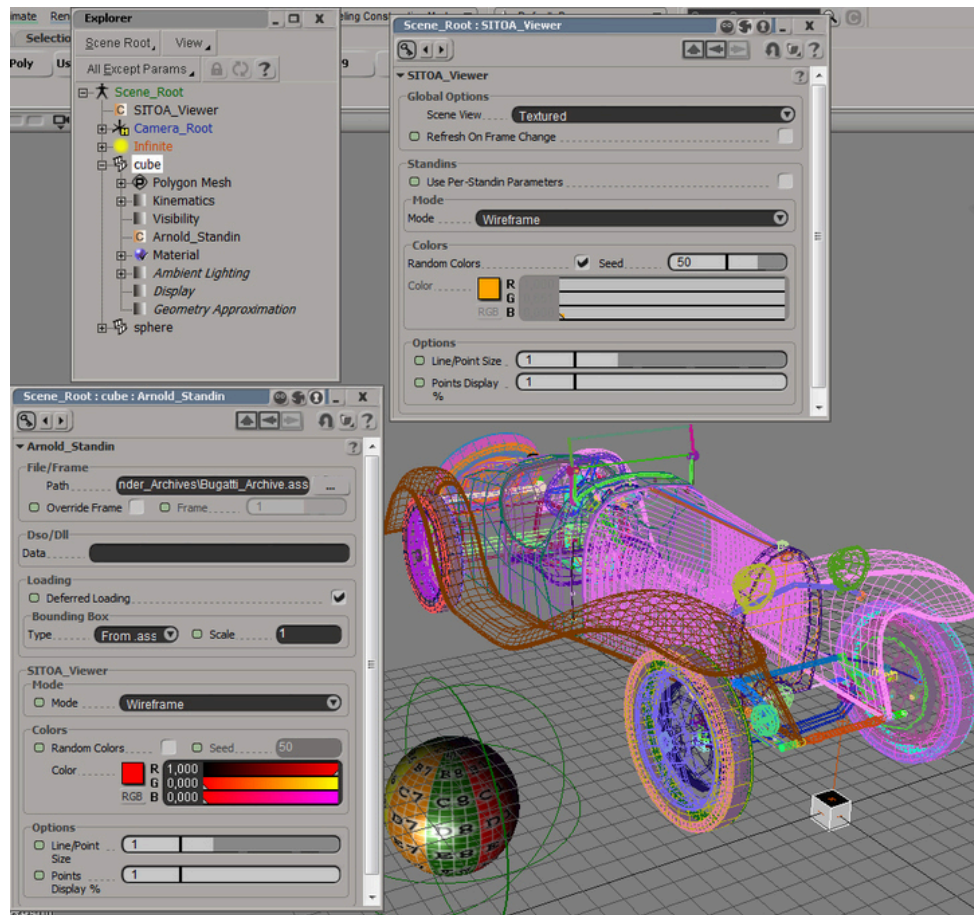
The viewer draws all the polymesh, curves, points and sphere nodes and their ginstances generated by the procedural.

Points and sphere nodes (from ICE or simulation software like Naiad) are always drawn as points. There are two ways you can set the parameters of the viewer:

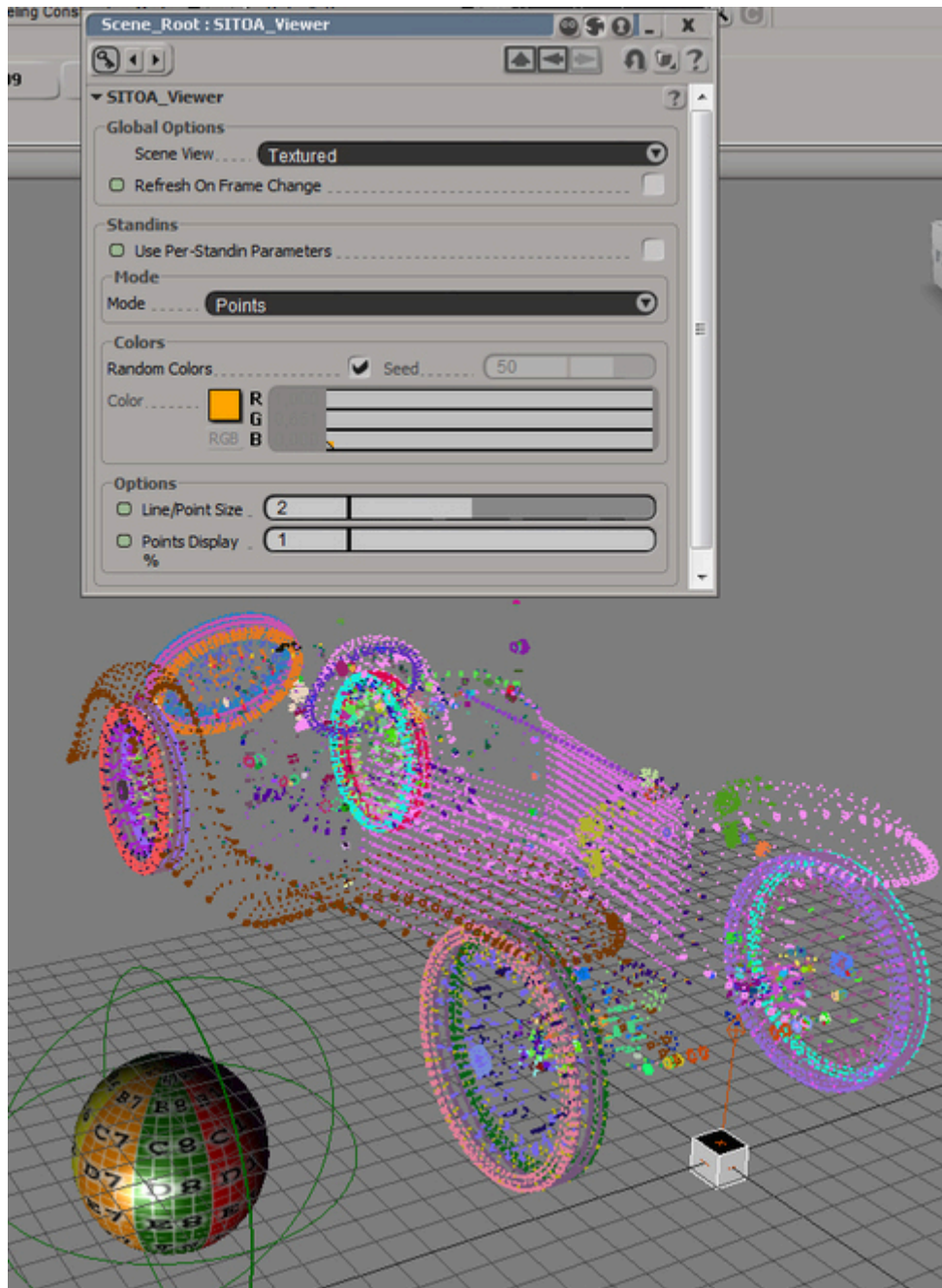
1. Globally: by the SItOA_Viewer property created at the scene root level when you open select the viewer.
2. Locally, per-standin: in the SItOA_Viewer property, turn Use Per-Standin Parameters on. This means that each standin object will drive the way its nodes are drawn, depending on the parameters exposed in the standin property.



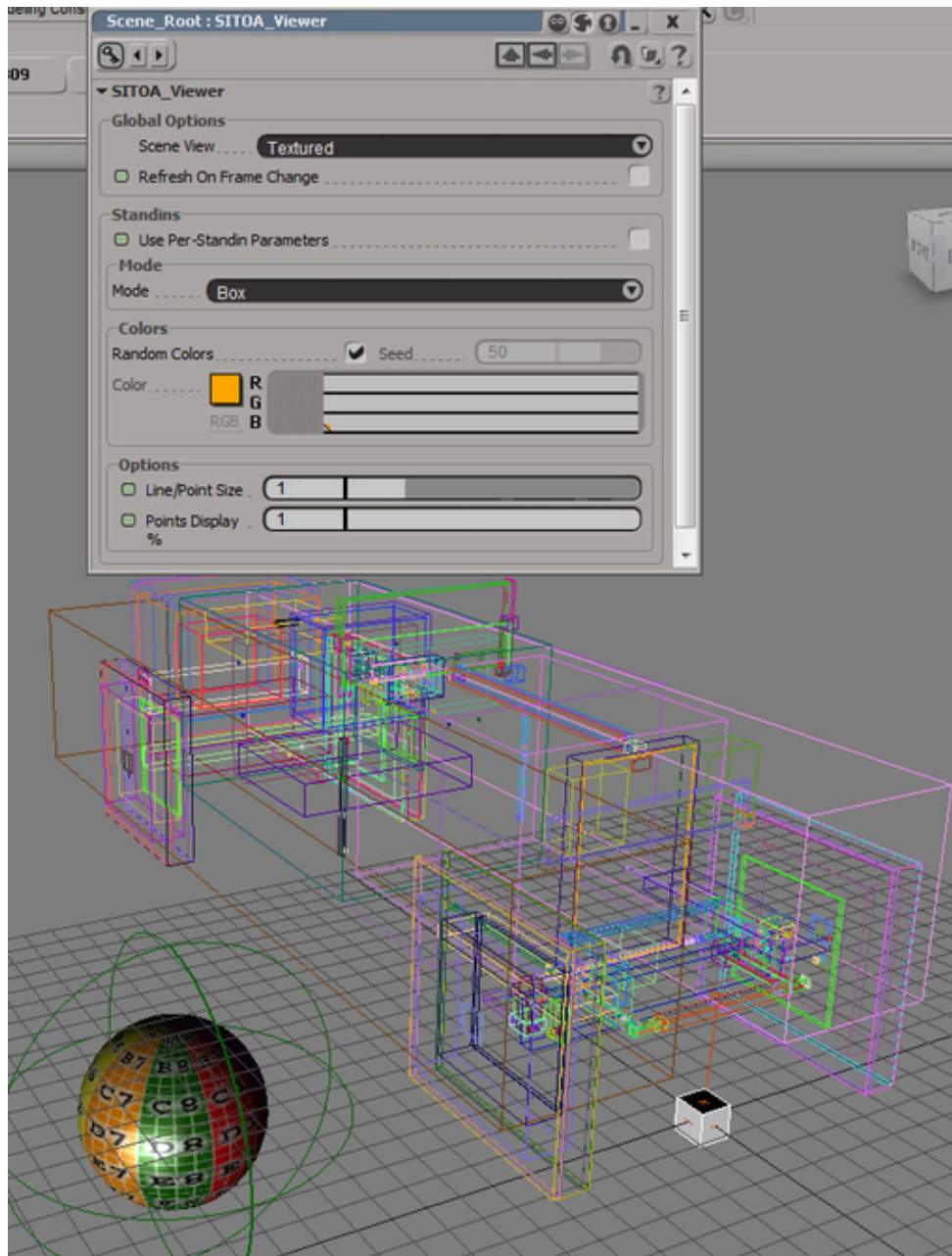
Note that scenes saved with a previous version of the standin property default to the global mode.



You can select the way the rest of the scene is displayed (Scene View) if to use a static color or a random one for each object, and the Size for lines and points. When in points mode, the Points Display % parameter defines the percentage (in the 0..1 range) of points that are actually drawn.

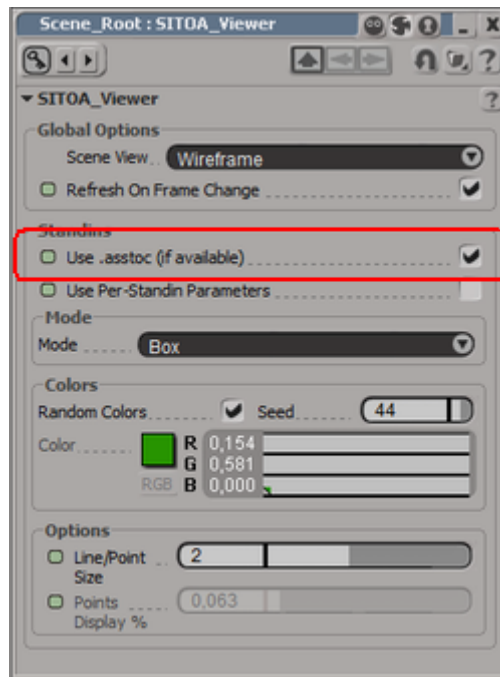


If Refresh On Frame Change is on, the procedurals are reloaded on a frame change. This is handy to preview sequences, but it can be slow with large amounts of data. Also, the graphic sequencer is not very fast itself, so our advice is to use the box mode if you need to move the camera around.



Note that you can also use the viewer to display a whole ass file generated from a scene. You can simply pick a cube, apply the standin property and select the scene ass file as its Path.

From SItoA 2.3 onward, an additional parameter, Use .asstoc (if available) is provided.



•

If enabled, and if a standin path is of type .ass or .ass.gz, and if the corresponding .asstoc file is available, then the standin is always drawn as a box, whose boundaries are the ones listed in the .asstoc file.

This option is only available for the viewer, and not for each individual standin property.

Note that if this control is enabled, but a standin does not have an .asstoc buddy, then it is drawn as before, i.e., according to the Mode of the viewer or of the standin property, depending on Use Per-Standin Parameters.

•

Commands

SltOA provides custom, script-able commands that allow you to manually execute some of the features of the plug-in.

Details of each command are given below. All the listed examples are in JScript.

SITOA_ExportScene

With this command, users can export the Softimage scene into Arnold's native scene definition format. One .ass file will be exported for each frame. These .ass files can later be rendered outside Softimage with the `kick` command-line executable (available on multiple platforms), or with any other custom standalone Arnold Renderer.

Syntax :

```
SITOA_ExportScene(frameStart, frameEnd, frameStep, createStandIn,
selectionOnly, filename);•
```

- frameStart: start frame of the sequence to be exported.
- frameEnd: end frame.
- frameStep: frame step.
- createStandIn: if true, the options and camera are not exported, and a .asstoc file is also created, hosting the bounding box information of the exported objects. This is usually the option you want to use if you plan to use the exported ass file a a standin later.
- selectionOnly: if true, only the selected objects are exported.
- filename: the output filename.

Note:•filename•supports tokens. When exporting a sequence, you can use the [Frame] token in the filename, to have it resolved correctly at each frame.•Example :

```
SITOA_ExportScene(1, 10, 1, false, false, "/usr/tmp/Test.[Frame
#4].ass");
```

Exports 10 .ass files, named /usr/tmp/Test.0001.ass, /usr/tmp/Test.0002.ass, etc.

SITOA_ExportObjects

Same as above, but allowing to export an objects collection.

Syntax :

```
SITOA_ExportObjects(frameStart, frameEnd, frameStep,
createStandIn, objects, recurse, filename);•
```

- frameStart, frameEnd, frameStep, createStandIn: same as above
- objects: collection of objects to be exported (defaulting to selection if void).
- recurse: if true, recursively export also the children of each item in the objects collection.
- filename: the output filename (supporting tokens from 2.6)

SITOA_DestroyScene

This command flushes the current scene loaded in Arnold, which is used for IPR rendering. Because it's difficult for SltOA to detect every single possible user-interface event, users can manually flush the scene, which will force Softimage to resend all the information to Arnold, thus picking up all the latest UI changes. If you find one of these cases, please report it to the developers and we will do our best to fix it.

Hint: we recommend assigning this command to one of the available hotkeys from `File / Keyboard Mapping`. F6 is a popular choice.

SITOA_FlushTextures

This command invalidates any texture maps in memory, forcing them to be reloaded from disk the next time they are accessed. This is very useful in interactive IPR sessions where the textures are being modified by you or somebody else on the network, and avoids an expensive full scene reload via `SITOA_DestroyScene`.

SITOA_ShowVersion

This command prints both the SltOA and Arnold versions of the add-on. This is useful when reporting bugs or suggestions.



Syntax :

```
SITOA_ShowVersion(in_log)
```

- in_log: if true, logs the message in the script editor.
Default value : true

Returned value : a string, containing the SItOA and Arnold versions, separated by a ',' character.

SITOA_CreateRenderChannels

This command creates the following Arnold AOV core layers as Render Channels in the scene, if not there yet:

•

A
albedo
background
cputime
diffuse
diffuse_albedo
diffuse_direct
diffuse_indirect
direct
emission
ID
indirect
motionvector
N
opacity
P
Pref
raycount
shadow_matte
specular
specular_albedo
specular_direct
specular_indirect
sss
sss_albedo
sss_direct
sss_indirect
transmission
transmission_albedo
transmission_direct
transmission_indirect
volume
volume_albedo
volume_direct
volume_indirect
volume_opacity
Z
shadow
shadow_diff
shadow_mask
indirect_diffuse
indirect_specular

SITOA_RegenerateRenderOptions

Regenerates the Arnold Render Options panel, updating the layout, adding new parameters, removing old or non-existent parameters. It will assign automatically to new Render Options all previous values the parameters had. Users will have to execute it in old scenes to update Render Options parameters. Take into account that this command doesn't refresh the Arnold properties already applied to objects.

SITOA_GetMotionBlurKeys

Returns the motion blur keys time samples as a float array

Syntax :

```
SITOA_GetMotionBlurKeys(objName, deformation, frame);
```

- objName: the object to get the motion blur keys from (they can override the global values).
Default value : selection
- deformation: if true, get the deformation motion blur keys, otherwise the transformation ones.
Default value : false
- frame: the frame where the computation is done.
Default value : the current frame

Example : log the deformation key times for object "torus" at frame 1

```
var keys = SITOA_GetMotionBlurKeys("torus", true, 1);
var arr = keys.toArray();
for (var i=0; i<arr.length; i++)
    logMessage(arr[i]);
```

SITOA_OpenVdbGrids

Returns the names of the grids contained in a VDB file, as a single string.

Example : log the grids in "C:\temp\fireball.vdb"

```
var names = SITOA_OpenVdbGrids("C:\\Temp\\fireball.vdb");
LogMessage(names);
```

SITOA_AddVisibilityProperties

Adds the visibility properties to the selected (or input) objects.

SITOA_AddSidednessProperties

Adds the sidedness properties to the selected (or input) objects.

SITOA_AddParametersProperties

Adds the parameters properties to the selected (or input) objects.

SITOA_AddProceduralProperties

Adds the procedural properties to the selected (or input) objects.

SITOA_AddUserOptionsProperties

Adds the user options property to the selected (or input) objects.

SITOA_AddMatteProperties

Adds the matte property to the selected (or input) objects.

SITOA_AddTextureOptionsProperties

Adds the texture options property to the selected (or input) images.

SITOA_AddCameraOptionsProperties

Adds the camera options property to the selected (or input) cameras.

SITOA_AddVolumeProperties

Adds the volume property to the selected (or input) objects.

The commands for applying properties share the same syntax and behavior. Taking the visibility case,

SITOA_AddVisibilityProperties(in_collection, in_inspect)•

- in_collection: the object or collection of objects you want to apply the property to.
Default value : selection
- in_inspect: true if the property must be inspected after being applied.
Default value : false

Returned value : the collection of applied properties, so you can further edit their parameters in your custom script.

Example : assign the visibility property to cube and sphere, and set "shadow" off, without inspecting the ppgs

```
var props = SITOA_AddVisibilityProperties("cube,sphere", false);
for (var i=0; i<props.count; i++)
{
    var prop = props.Item(i);
    prop.Parameters("shadows").Value = false;
}
```

SITOA_AddPointLight

Adds an Arnold point light to the scene.

SITOA_AddDistantLight

Adds an Arnold distant light to the scene.

SITOA_AddSpotLight

Adds an Arnold spot light to the scene.

SITOA_AddSkydomeLight

Adds an Arnold skydome light to the scene.

SITOA_AddDiskLight

Adds an Arnold disk light to the scene.

SITOA_AddQuadLight

Adds an Arnold quad light to the scene.

SITOA_AddMeshLight

Adds an Arnold mesh light to the scene

SITOA_AddPhotometricLight

Adds an Arnold photometric light to the scene

The commands for creating lights, except SITOA_AddMeshLight, share the same syntax and behavior. Taking the point light case,

SITOA_AddPointLight(in_name)

- in_name: the name assigned to the light.
Default value : "Point"

Returned value : the created light.



SITOA_AddMeshLight has an extra mandatory parameter

```
SITOA_AddMeshLight(in_name, in_obj)
```

- in_obj: the object to be used as shape.

Example : create an Arnold point light, set its intensity to 2, and inspect the light

```
var light = SITOA_AddPointLight("Nice_Light");
SetValue(light + ".light.arnold_point_light.intensity", 2.0, null);
InspectObj(light);
```

•

SITOA_AddLightDecayFilter

Adds the light decay filter shader to the selected (or input) lights.

SITOA_AddBarndoorFilter

Adds the barndoor filter shader to the selected (or input) lights.

SITOA_AddGoboFilter

Adds the gobo filter shader to the selected (or input) lights.

SITOA_AddLightBlockerFilter

Adds the light blocker filter shader to the selected (or input) lights.

SITOA_AddPassthroughFilter

Adds the passthrough filter shader to the selected (or input) lights.

The commands for applying filters share the same syntax and behavior. Taking the barndoor case,

```
SITOA_AddBarndoorFilter(in_collection, in_inspect)
```

- in_collection: the light or collection of lights you want to apply the filter to.
Default value : selection
- in_inspect: true if the filter shader must be inspected after being applied.
Default value : false

Returned value : the collection of applied filters, so you can further edit their parameters in your custom script.

Example : apply a decay filter to Point and Spot, enable the Near Attenuation flag, and set the Near Start/End values to 5 and 10, without inspecting the ppgs

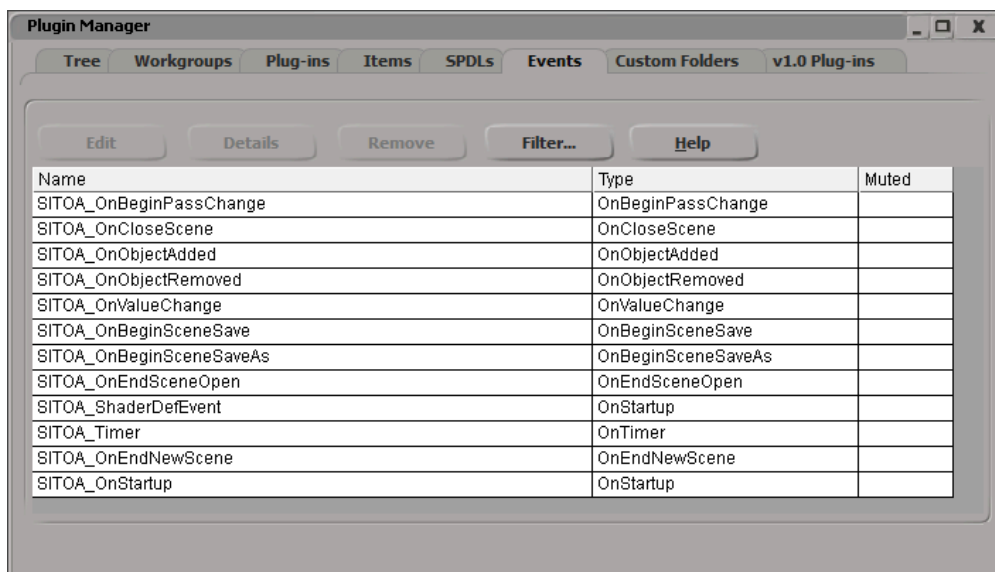
```
var filters = SITOA_AddLightDecayFilter("Point,Spot", false);
for (var i=0; i<filters.count; i++)
{
    var filter = filters.Item(i);
    filter.Parameters("use_near_atten").Value = true;
    filter.Parameters("near_start").Value = 5;
    filter.Parameters("near_end").Value = 10;
}
```




-
-
-

Events

SltOA implements a series of Softimage events. The full list can be displayed in the Events tab of the Plugin Manager. Be sure to enable User Addon in the Filter options.



Name	Type	Muted
SITOA_OnBeginPassChange	OnBeginPassChange	
SITOA_OnCloseScene	OnCloseScene	
SITOA_OnObjectAdded	OnObjectAdded	
SITOA_OnObjectRemoved	OnObjectRemoved	
SITOA_OnValueChanged	OnValueChanged	
SITOA_OnBeginSceneSave	OnBeginSceneSave	
SITOA_OnBeginSceneSaveAs	OnBeginSceneSaveAs	
SITOA_OnEndSceneOpen	OnEndSceneOpen	
SITOA_ShaderDefEvent	OnStartup	
SITOA_Timer	OnTimer	
SITOA_OnEndNewScene	OnEndNewScene	
SITOA_OnStartup	OnStartup	

SITOA_OnBeginPassChange

Fired on a pass change, destroys the Arnold scene.

SITOA_OnCloseScene

Fired on closing a scene, destroys the Arnold scene.

SITOA_OnObjectAdded, SITOA_OnObjectRemoved ••

Notify SltOA that a new object or light was added or removed. In this case, SltOA is able to update the Arnold scene without destroying it.

SITOA_OnValueChanged

Fired when any parameter is changed. SltOA tries to update the current Arnold scene if the change is IPR compatible.

SITOA_OnBeginSceneSave, SITOA_OnBeginSceneSaveAs

On saving a scene, SltOA stored the current plugin version together with it.

SITOA_OnEndSceneOpen

On opening a scene, SltOA reads the plug-in version the scene was saved with. If it's different from the current plug-in version, a warning is logged into the script editor.

SITOA_ShaderDefEvent

Triggers the UI definition for the [dll-so shaders](#).

SITOA_Timer

Every tenth of a second, SltOA prints the messages into the script editor. The messages coming from the several modules of the plug-in are not printed "live". Instead, they are pushed into a queue that is actually printed out by this event. This ensures that only the main thread is responsible for logging messages, as advised by the SDK. If you disable the event, you won't get any message.

SITOA_OnEndNewScene, SITOA_OnStartup

On startup, or on a new scene event, SltOA establishes some basic setup for the scene. For example, the default light is replaced by an Arnold distant light.



.

Naming Convention

This document explains the way we name the exported Arnold nodes after the corresponding Softimage objects.

We try, as much as we can, to have a unique syntax for all the nodes (in older SltoA versions, we had several different ways to compose the names, and different characters to separate the name fields).

This also to give a scripter/developer the ability to retrieve a given node out of the native Softimage object, and vice-versa. For instance, after an .ass file has been exported, a user may want to edit one of its nodes having as a reference the Softimage scene.

As a general rule, the names have the following format:

Softimage Object Full Name + ".SltoA." + something

where "something" is a string depending on the node type and described in more details below.

So, if you need to retrieve the Softimage object out of the Arnold node name, you can simply trim the name string up to the last occurrence of the ".SltoA." substring.

Viceversa, if you need to retrieve an exported node name out of a Softimage object, here are the details, depending on the node type:

Softimage Object	Arnold Node(s)	Arnold Node(s) Name
Mesh	polymesh	Mesh FullName + ".SltoA." + time
Light	point_light, distant_light, etc.	Light FullName + ".SltoA." + time
Camera	camera	Camera FullName + ".SltoA." + time (2.6, was Camera plain Name in 2.5)
Shader	standard, utility, etc.	Shader FullName + ".SltoA." + time + "." + Uniqueld
Standin	procedural	Standin FullName + ".SltoA." + time
Instance	ginstance	Instance Model FullName + " " + master node name
Instance of light	light (cloned)	Instance Model FullName + " " + master node name
Hair	one curves per chunk	Hair FullName + ".SltoA." + time + "." + chunkId
Shape on hair	one polymesh per strand	Hair FullName + ".SltoA." + time + "." + strandId + " " + master node name

ICE:

PointCloud Shape	Arnold Node(s)	Arnold Node(s) Name
Points	points (mode=disk)	PointCloud FullName + ".SltoA.Disk." + time
Sphere	points (mode=sphere)	PointCloud FullName + ".SltoA.Sphere." + time
Disc	one disk per point	PointCloud FullName + ".SltoA.Disc." + time + "." + discId
Box	one polymesh with six quads per point	PointCloud FullName + ".SltoA.Box." + time
Rectangle	one polymesh with one quad per point	PointCloud FullName + ".SltoA.Rectangle." + time
Cylinder	one cylinder per point	PointCloud FullName + ".SltoA.Cylinder." + time + "." + cylinderId
Cone	one cone per point	PointCloud FullName + ".SltoA.Cone." + time + "." + coneId
Instance	one ginstance per point	PointCloud FullName + ".SltoA.Instance." + time + "." + instanceId + " " + master node name
Lights	one light (cloned) per point	PointCloud FullName + ".SltoA.Instance." + time + "." + instanceId + " " + master node name
Strands	curves	PointCloud FullName + ".SltoA.Strands." + time
Shape on strands	one polymesh per strand	PointCloud FullName + ".SltoA.InstanceStrands." + time + "." + strandId

where time is the Softimage frame multiplied by 1000 and cast to integer. In cpp that's:

```
(LONG)floor(in_frame * 1000.0f + 0.5f);
```

We use spaces in just one case, that is instances, for IPR reasons. For example, if you have a cube under a Model at frame 35, the cube is exported as:

```
polymesh
{
  name Model.cube.SItoA.35000
  ...
}
```

If this model is instanced by Model_Instance, then the instanced cube is exported as

```
ginstance
{
  name "Model_Instance Model.cube.SItoA.35000"
  node "Model.cube.SItoA.35000"
  ...
}
```

So, also for ginstance nodes, it is easy to retrieve the master node and its Softimage object.

The various ID's (chunkId, disclId, etc.) are unique integers, needed when several Arnold nodes derive from a unique Softimage object. For instance, say you have a pointcloud setting the particle shape to a cone. A unique Softimage object (the point cloud) is exported as a collection of Arnold cone nodes, and each must have a unique name, as required by Arnold. This is what the .ass file looks like at frame 35:

```
cone
{
  name pointcloud.SItoA.Cone.35000.0
  ...
  shader "Sources.Materials.DefaultLib.Scene_Material.Phong.SItoA.35000.1"
  id 949
}

cone
{
  name pointcloud.SItoA.Cone.35000.1
  ...
  shader "Sources.Materials.DefaultLib.Scene_Material.Phong.SItoA.35000.1"
  id 949
}

cone
{
  name pointcloud.SItoA.Cone.35000.2
  ...
  shader "Sources.Materials.DefaultLib.Scene_Material.Phong.SItoA.35000.1"
  id 949
}
```

```
}
```

```
...
```

```
.
```

```
.
```


FAQ

These issues are specific to the use of the SItOA plug-in, and do not relate to features of the Arnold renderer itself. For issues to do with rendering with Arnold in general, see the Arnold Rendering FAQ.

Sometimes the render-region doesn't update when I change various settings in my scene, how can I force it to refresh?

You can adjust the Scene Rebuild Mode from Auto to Always. There is also a command to flush the scene `SITOA_DestroyScene()` and `textures SITOA_FlushTextures()`. For more information about SItOA-specific commands please see the [SItOACommands](#).

Can I use the Architectural shader?

No, the Architectural shader isn't supported yet but Arnold comes with its own general-purpose shader called "Standard" which is quite similar. For a list of supported Softimage shaders please see the [Shaders section](#).

Where can I find Arnold specific shaders?

All the shaders SItOA comes with are accessible from the bottom of the "Nodes" menu in the rendertree, they are also available by going to Get > Material > Arnold Shaders. See also the [Shaders section](#).

How can I hide polygons from the render?

Currently, SItOA only supports clusters with visibility property (which you can create from Get > Property > Polygon Cluster Visibility).

How can I do glossy reflections using the Arnold standard shader?

In the specular section, increase the "Scale" parameter. The "Roughness" parameter will control how blurry your reflections are. The lower the value the sharper the reflection. In the limit, a value of 0 will give you perfectly sharp mirror reflection. You also have separate controls to adjust the intensity of the direct and indirect specular reflections. Direct specular are reflections coming from regular light sources (spot/point/area/distant lights), and indirect specular are reflections coming from another object or an environment map.

How can I setup Image Based Lighting

The skydome_light will be most of the time the best and fastest way of achieving IBL, in both speed and noise than other methods like using a sky shader.

For a more detailed description of the subject please see the IBL page.

Can I use the normal visibility properties on my objects?

Yes, Arnold will read the visibility options Softimage already has. Take into account the checkbox for "Reflection Visible" controls normal/sharp reflections, and "Reflection Caster" controls glossy visibility. It's a bit of a hack but it's as good as it can be done. In any case, you can also apply an Arnold visibility property to an object, partition or group which has all the visibility options that Arnold natively supports in it. Keep in mind that doing this will override your objects normal visibility settings.

How can I avoid my procedural textures from swimming?

There are a few ways in which 3D procedural textures can "swim" in animation:

1. Your object is undergoing a rigid SRT matrix transformation
2. Your object is deforming, the vertices moving around freely - for example, a walk cycle
3. Your object is neither moving nor deforming, but it has displacement mapping (a lesser form of deformation), and for whatever reason, you'd like to texture based on the un-displaced position

Can I use Softimages' built-in color management for linear workflow?

The color management in Softimage will at the moment only affect the display in SItOA, for textures the only supported workflow is to have your linear images stored in an HDR format (.exr, .tiff-32, .hdr) and your non-linear, sRGB images in a LDR format (.jpg, .tif-8, .png etc). Then, in the global render options, set your Input Gamma values in the Gamma Correction tab to 2.2. Arnold will consider floating point images as linear and will only apply the de-gamma calculations to LDR images.

Why do my shadows stay solid black even when they are cast by a transparent object?

For optimal render speed, Arnold flags all objects "opaque" by default. To get shadows to respect transparency you need to disable the "Opaque" flag on the object(s) in question, you can do so by applying the "Arnold Parameters" property from the menu: Property->Arnold Properties->Geometry->Arnold Parameters.



Do I need to disable opaque on a point cloud if I want the instanced particles to be transparent, or only on the "master" object?

You only need to put the Arnold properties in the point cloud if you want the instanced particles to be transparent. If you apply to the "master" object, it will only apply the properties to that copy.

Why does DOF work in Preview but not in Render Region?

At the moment, SltOA is ignoring the [multi] : Render Region Options, so if you want to have DOF in your Render Region you need to use the [Per View] : Render Region Options available from the Down Arrow of the Render Region Window. It's advised that one define its Render Options at the Pass level and tick the Use Current Pass Options checkbox in the Render Region Options.

How to do lookups in a shader to different UV shells•

Arnold does support mapping out of the usual 0,1 range. You can use this behavior to apply different mapping shells and do lookups to different image nodes (one per shell) composited with a mix shader in additive mode. The Idea is that for each shading point only one lookup to the different images will give the expected value (the one associated with the shell the face that is being shaded is in) and the others will return black (so they won't add to the final result).

To specify for each image the shell it belongs to, you can use the UV remap fields of the advanced tab from image node. What you want to do is to "offset" the corresponding shell to 0,1, adding or subtracting the needed value, both in U and V. If you want to offset a shell located in the range U=2, 3 and V=1,2, then in UV remap you need to set U to -2,-1 and V to -1,0 . With a shell in the negative ranges then you need to put positive numbers in the remap. To remap a shell that is in the range U=-3,-2 and U=-1,0 the remap values would be U=3,4 and V=1,2. You can download an example here [Download](#).

Render Region freezes when changing values in Pass shaders•

This is a known problem. The PPG logic inside the Pass spdl is doing a lot of scene changes that cause unnecessary render calls. When the user does changes in the PPG, the scene will be rendered more than once while the UI doesn't respond. It's not really crashing, and eventually, the render appears and the UI responds. This has been improved in Softimage 2012 (but it is still not 100% fixed).

What's the best way of handling ICE Instances in Arnold?•

You can use the InstanceShape node on particles, or you can try instancing Arnold Standins.

Does SltOA support Object and Children Hierarchy Mode for ICE instances? ••

Yes, if you put them in a model. If you want several models, put them inside a group in branch mode.

Can't see the Arnold specific render channels in the Pass Options Menu•

If the default renderer in the preferences is set to Arnold, the standard Arnold Channels will be automatically created when creating a new scene. But you can always manually call the SltOA_CreateRenderChannels() command.

Bump mapping is not working when connecting images to a Bump2D node•

Bump2D need a consistent uv set applied.

Bump mapping is not working when connecting images to a Bump3D node•

Bump3D works by evaluating the bump shader (an image in this case) at three different points: $P + \epsilon_x$, $P + \epsilon_y$, $P + \epsilon_z$. Since the only thing displaced is the point, the UV coordinates will be the same in the three different lookups. These will give the same texel in the image and result in no perturbation of the normal. You should instead use the Bump2D shader for images.

How do I load .tx files made by maketx? Softimage doesn't recognize them.

There is an option called "Use Existing Tiled Textures" in the Tiling Group of the Texture System tab from the renderer options, which lets you automatically use .tx files if they exist on disk. So, if your scene has one .jpg and one .tx in the same directory, with the same filename, you can use the .jpg in Softimage and SltOA will automatically swap it out for the .tx at render time. This issue has been reported to the Softimage team (so it understands tiled+mipmapped tx files natively).

I have a mesh that subdivides and renders fine in mental ray, but Arnold refuses to render it subdivided - what's going on?

Arnold is a little more particular about mesh integrity than Mental Ray. What you probably have is a bit of bad geometry somewhere that needs to be cleaned up before Arnold can subdivide the mesh for rendering. When you draw a render region or render a preview, you will probably see a yellow warning pop up at the bottom of your viewport that says something like:

[subdiv]•Face•3707•Edge•402,•1552•used•more•than•twice

You will also get a warning in your log (assuming you're using Detailed logging) that says something to the effect of:

```
WARNING:[arnold]•00:00:00•377mb•[subdiv]•polymesh•(no_subdividey.map_marker_1000)•is•not•manifold,•skipping•subdivision
```

To find the offending polygons/edges, just type their Face/Edge numbers into the search box in the MCP (while in the appropriate selection mode - face/edge) and SI will select them for you. Realize that Arnold will only give the warning once per scene refresh - so if you missed the Face number, you'll need to do a SceneDestroy and another render region to get the error to show up again. Once you know where the problems are, just correct the bad geometry (do a SceneDestroy for good measure), and the mesh should render.

My scene that looks fine in Softimage's default rendering is too dark in Arnold

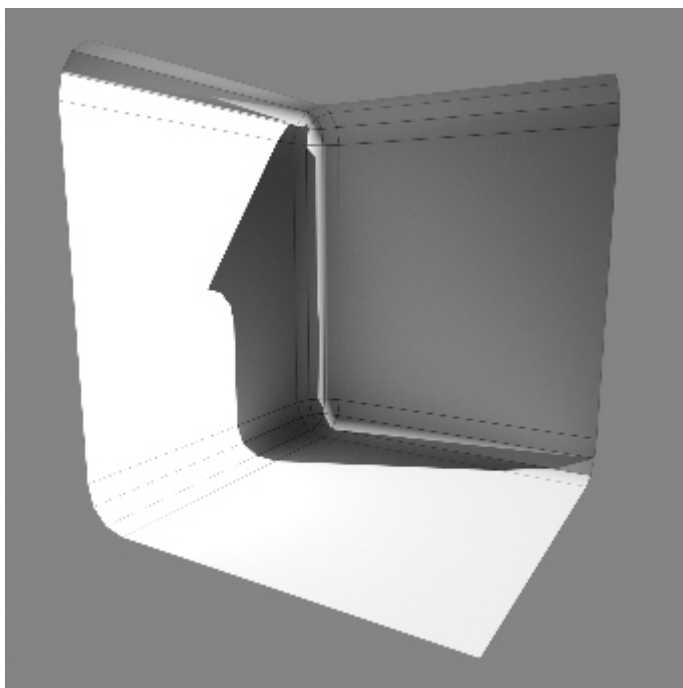
It's important to remember that Arnold uses the inverse square law for light decay (quadratic decay) by default. The approach taken in Arnold is to be physically accurate whenever possible as this makes many things simpler and better - the approach we recommend is to light and model everything using accurate real world values whenever possible. You can if you wish change the light decay to be "constant" (no decay) by altering the Arnold light node's settings, but in the long term it will be better to embrace a physically based approach.

Shadow terminator effect

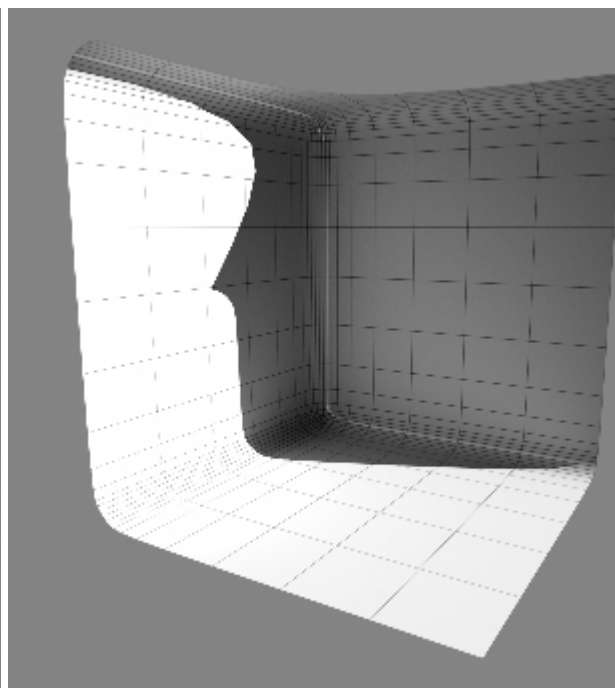
This is a self-shadowing error that can result from low tessellation or when objects do not have thickness. It usually occurs in concave areas or when the light is behind the polygon mesh. In these situations, it can introduce light leaking.

There are some workarounds:

1. Increase the number of subdivided polygons (the artifacts will become smaller).
2. Give the object some thickness so that it is not a single sheet of polygons.



Low poly model - Shadow terminator effect is visible



Increasing the polygon subdivisions reduces the problem

Arnold 5.0

Below are some commonly asked questions with regards to changes in Arnold 5.

Where is the Standard shader?

The old `standard` shader has been replaced with the `standard_surface` shader. The old standard shader is still shipped with Arnold as a legacy shader, for compatibility with old scenes. However, it will be removed in a future release.

Is the Standard Surface shader energy conserving?

If all of the individual weights and colors are less than or equal to 1.0, then the `standard_surface` shader is energy conserving. Unlike the old Standard shader, you don't need to worry about the sum of weights being less than 1.0 or manually enable Fresnel, which is always enabled.

Where is 'Refraction Roughness' in the Standard Surface shader?

The specular_roughness affects both specular reflection and refraction. There is also an `extra_roughness` parameter in `transmission` to add some additional roughness for refraction. You can, however, use `coat` to create a rough reflection layer over a sharp refraction.

Where is the specular GGX microfacet distribution?

GGX is now the default model for the `standard_surface` shader.

How can I control the index of refraction for transparent materials?

You can use the Specular IOR attribute in the same way to handle transparent materials such as glass, diamond, etc.

How can I control Backlighting/Translucency in the Standard Surface shader?

Enabling `thin_walled` and setting `subsurface_weight` to, for example, 0.5 to have half the light reflected and half transmitted.

IOR and Metalness appear to do the same thing. Why use one over the other?

Using very high IOR values can look quite similar to metalness. It looks the same if you set base color to the specular color and the specular color to black. The difference is that you get an extra reflection at the edges, with the specular color controlling the edge tint. The metal Fresnel works the same as in the `Complex IOR` shader, with the artistic parameters.

Where is the Skin shader?

You should use the `subsurface` parameter of the Standard Surface shader. It uses the high-quality empirical SSS profile, which preserves sharp geometric and bump detail without the need for three layers of varying depth, as the old skin shader did.

Where is the Hair shader?

Use the new `standard_hair` shader instead. The old hair shader is shipped with Arnold as a legacy shader, for compatibility with old scenes. However, it will be removed in a future release.

Where have the UV parameters gone?

The `standard_hair` shader does not have a "uvset" parameter at all. Rather all image and procedural texture nodes have a "uvset" parameter instead so that they can pick up the appropriate texture coordinates there.

Where is the Volume Collector shader?

Use the new `standard_volume` shader instead. The old Volume Collector shader is shipped with Arnold as a legacy shader, for compatibility with old scenes. However, it will be removed in a future release.

How can I use OSL shaders?

Information about OSL shaders can be found [here](#).

Where is the Shadow Catcher shader?

The `shadow_catcher` shader is now called the `shadow_matte` shader.



ASS Files

Although the syntax of the file format hasn't changed at all, .ass files generated with Arnold 4 may not render correctly with Arnold 5 and so should be avoided. Likewise, .ass files generated with Arnold 5 may not render correctly with Arnold 4.

Sky Shader

The sky shader is deprecated and will be removed from a future release. You should instead use the visibility settings in the `Skydome` Light.

Skydome Light



The camera visibility in the skydome light replaces the need for creating a sky shader and carefully setting its visibility attributes to avoid double counting. The Skydome light shows up in the indirect AOVs, not the direct. It can be changed to the direct light AOV by setting the light's AOV to "default" to place it in the default light group, or another name for some other light group.

Why are there two versions of some shaders?

The mix, switch and ray_switch shaders have been split up into two shaders. One for switching between shaders and the other for switching between texture colors respectively.



PDF Archive / Updates

User Guide Archive

- [SitoA•for Arnold 4](#)

•

Recent Updates

Activity Stream