Softimage ExplosiaFX

Documentation

2015

Table of contents:

Installation

Requirements

The ExplosiaFX Add-On for Autodesk Softimage requires:

– Windows 7 SP1 and up 64 bit or Centos 6.2 and up compliant 64 bit Linux
– Autodesk Softimage 2013 and up

Workgroup connection

To install the Add-On, simply use inside Softimage
File -> Add-On -> Install -> Restart Softimage
Then choose the corresponding add-on for your operating system and Softimage version. It is strongly recommended to install the add-on into a work-group rather a user-folder.
The EFX package ships in two versions: with AVX support and a legacy build. The AVX build requires CPU with AVX support. Please refer to your CPU manufacturer to check out for AVX support. For Intel, it is i5-i7xxxx (starting from i7-2600 Sandy bridge family), for AMD it is FX-xxxx (starting from Bulldozer family).

Arnold connection

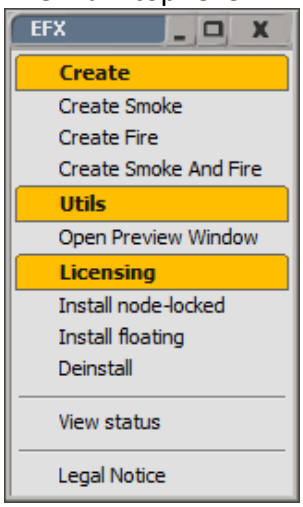
The workgroup contains Mentalray and Arnold shader set in addition to the plugin itself. Mentalray shaders connection does not require any additional steps, while Arnold third party shader installation procedure can be found here:
https://support.solidangle.com/display/SItoAUG/SItoA+Paths
Go to File -> Preferences -> Arnold Render -> System(tab) -> Search Path where you have to specify paths to the Arnold bin dir and to the EFX workgroup separated by ‘;’. As example:

C:\Users\USERNAME\Autodesk\Softimage_2015\Addons\SItoA\Application\Plugins\bin\nt-x86-64;D: \XSI_EFX_Workgroup\Application\Plugins\bin\nt-x86-64

ExplosiaFX (EFX hereafter) usually uses the latest arnold core libs, builds within previous cores can be compiled upon request.

Quick access menu

The main top-level EFX menu provides quick creation of basic setups and license management.



-Create Smoke – creates smoke setup with disabled fire component;
-Create Fire – creates fire setup with disabled smoke component;
-Create Smoke and Fire – creates full setup involving simulation of fire which burns out and produces smoke.

-Open Preview Window – open an instance of the EFX viewport. Press F5 to toggle between OpenGL schematic view and the realistic Raymarcher renderer. You may bind the *EFX_openwin()* command to any free hotkey to quickly open that viewport.

-Install node-locked – install single-seat license key generated for current workstation(where EFX is running) MAC address, this can be also a wi-fi USB stick;
-Install floating – set IP-address and port of the License server host workstation in order to connect and obtain permission for commercial work. License key must be installed on that host workstation. See the section “Floating license server” for more details.

-View status – shows the current license state and the current workstation IP/MAC address.

Licensing

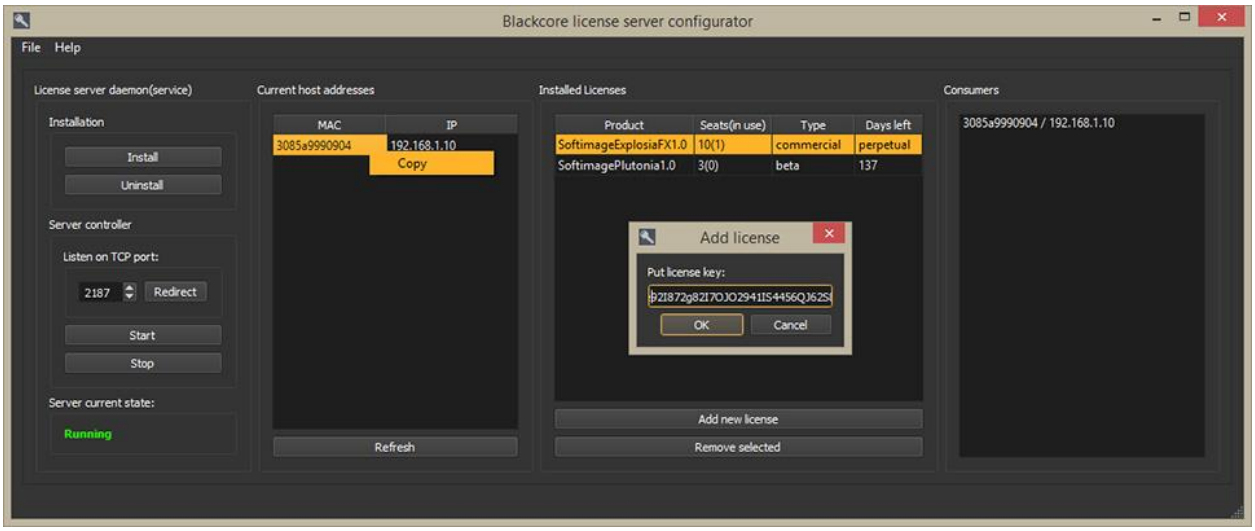
At this moment, you can use two types of licensing: node-locked which is what most single-users usually choose, or floating to distribute multiple seats across network.

Node-locked

Node-locked license is generated from the target workstation MAC address(or a wi-fi USB stick) and must be used only with that MAC device. Open the EFX Menu->View status and copy the MAC address.
Go to http://blackcore.technology/softimage_efx to get a license key (either trial or commercial), then call EFX Menu->Install node-locked and put key. You should see "[EFXTools]: Node-locked license was installed successfully!" message in the history.

Floating

Download the license server http://blackcore.technology/lictools for your OS, then call LICENSE_SERVER_LAUNCHER file.

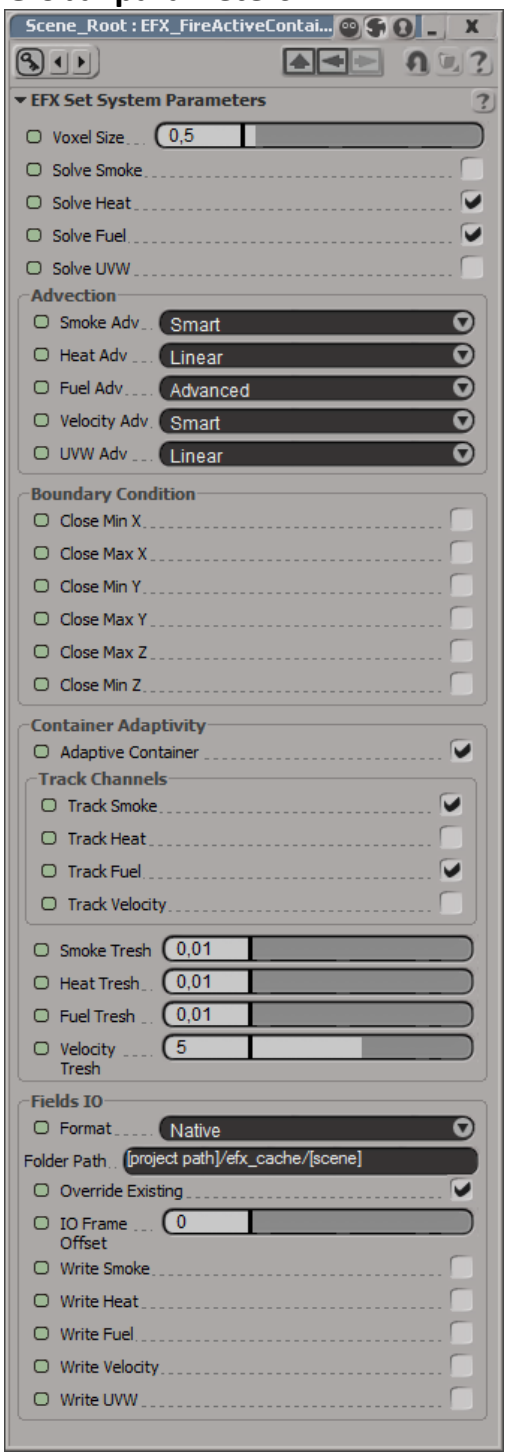


Press "Install" button to mount the server in OS. Once done, copy the MAC address and get a key http://blackcore.technology/softimage_efx and put it after pressing "Add new license" button, then call on each workstation "EFX Menu -> Install floating" and set the license server IP address and port. You should see "[EFXTools]: Floating license accessor was installed successfully!".

System initialization tree

The usual way to get started a new setup with EFX is to call one of the presets in the EFX menu. After creation, there will be a polybox as primary rendering entity .This polybox has three main ICE tree:
-Initialization;
-Simulation;
-Post;
The initialization tree serves data allocation and fundamental parameters assignation.
The EFX Set System Parameters defines all factory fields, paths and primary container attributes line voxelsize and walls behavior.

Global parameters



-Voxelsize – size of basic simulation element, less size = more details and memory/CPU consumption, this value can be manually overrided in the simulation tree(set self.efx_vxsz attribute);
-Solve xxx – if not checked then selected channel is not allocated and all subsequent stuff based on that channel is ignored;
-Advection – the mathematical method to transport channel data at each iteration.

Linear is most fastest and blurred method, very well suited for Heat channel transport (when Fuel channel is used for fire shading).

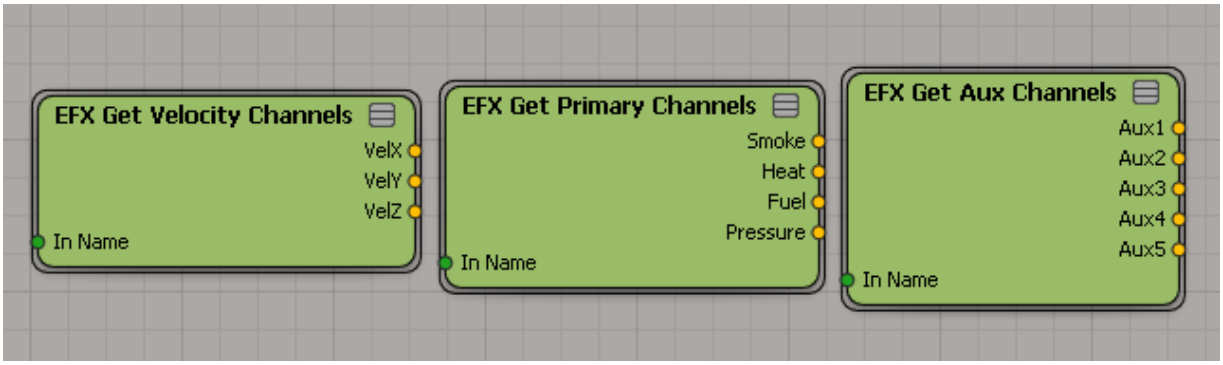
Advanced requires more CPU time but gives less blurred result. Similar to Fumefx's Advanced and Houdini maccormac/BFECC schemes.

Smart is most details-preserved method but it quadruples memory consumption (for each smart field a new 3 additional fields allocated).

-Boundary conditions – defines the walls behavior, set closed to insert wall at the specified side of the bounding container
-Adaptive container – the active container will change it's size depending on it's content
-Track channels – select channels which content will define the size of adaptive container
-xxx Tresh – minimal channel value in the voxel to consider on adaptive container size change
-Fields IO – controls the caching sub-system. Specify folder path where all selected channels will be saved out. IO frame offset is used to shift numeration in the cache names in relation of the XSI timeline. Note, to get motion blur you need to cache out the velocity channels. Limitation: Render Tree does not support tokens so you need to set the full path to the cache folder in shaders.

Factory fields

The typical fire+smoke setup contains 12 factory fields:



Smoke, Fuel, Head + three components of velocity field: VelocityX Y Z + 5 auxilary channels which can be freely used for any data manipulation except the very end of the Post tree (where light map is stored to that fields) + pressure channel (which is actually divergence so treat it as negated pressure). You can access the underlying data by the EFX_GetFieldData and EFX_GetFieldDataAtPosition nodes.

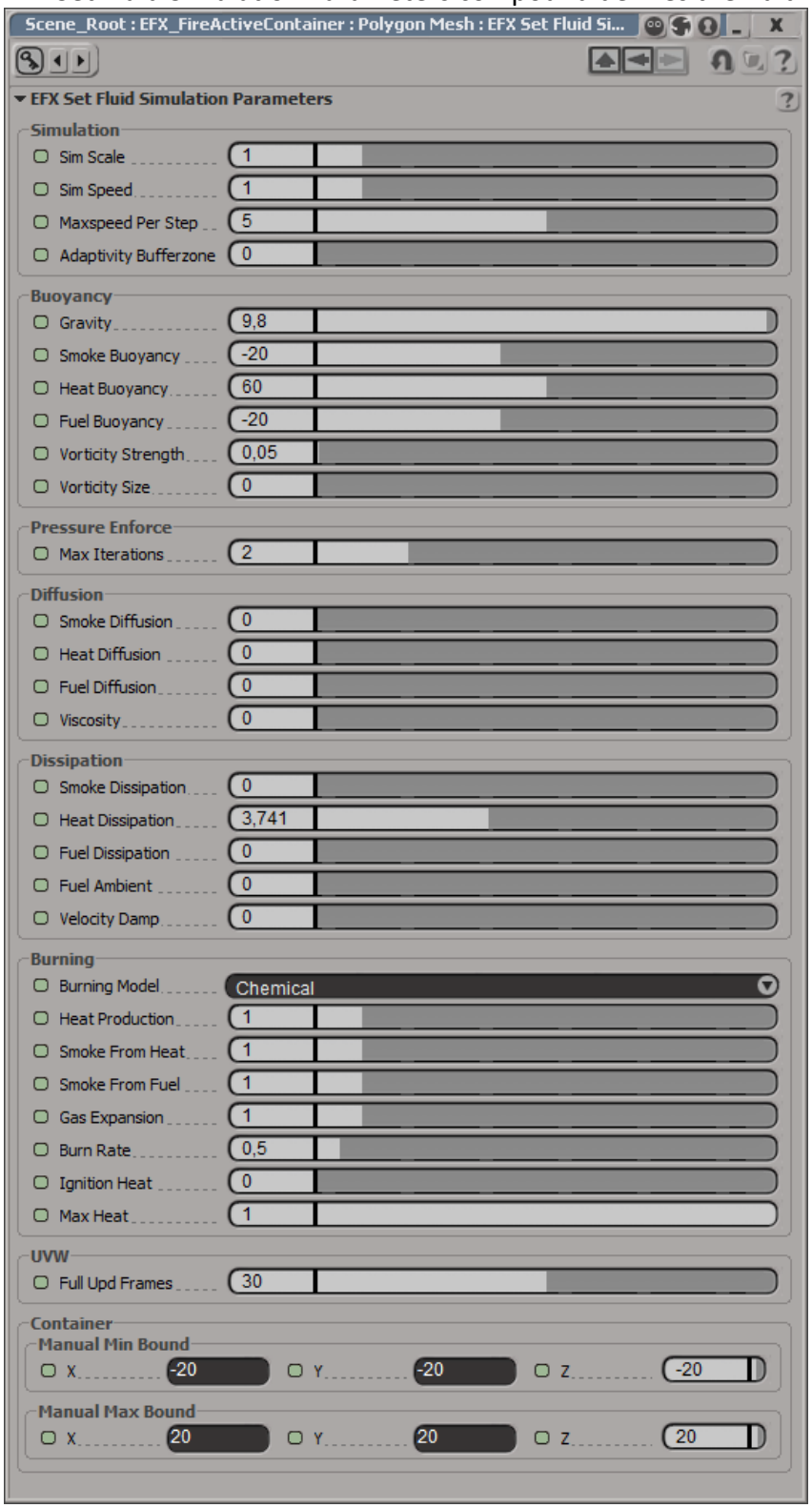
User fields

By default, all factory content-fields are registered in the EFX system and automatically got advected and cached (if enabled). To enable such functionality for user-defined fields, there is special compound "EFX Set user Field Parameters" which registers a user-field in the system.

User can access this field via the “EFX Get User Field” compound. Registration must be done in the Initialization tree, after the “EFX Set System Parameters” compound.

Simulation tree

The simulation tree manages fluid advancing in time and fields IO (input-output). All parameters can be animated over time.
EFX Set Fluid Simulation Parameters compound defines the fluid behavior.



-Sims scale – adjust the global scaling of the system units, it is not recommended to have it in range [0.2;5.0] for numerical precision stability;
-Sim speed – adjust speed of fluid advancing;
-Maxspeed per step – if the current maximum speed in any point of fluid will exceed this value then additional sub-step is performed to avoid artifacts;
-Adaptivity buffer zone – define the air bandwidth between the fluid content and walls of the active container;
-Gravity – world –Y gravity strength;
-xxx buoyancy – how much gravity affects to specified channel, result value is multiplication of channel voxel value, gravity and the buo value;
-Vorticity strength – amplify the "curly" features in the fluid;
-Vorticity size – defines the size of "curly" feature which to amplify;
-Pressure iterations – pressure sub-solver iterations, 2 should be enough for most of setups of any voxelsize, but in case of rapid-movements like explosion may be increased;
-Diffusion – defines the diffusion rate over time for specified channel;
-Viscosity –diffusion for velocity;
-xxxx dissipation – decreases the channel value over time;
-Fuel ambient – min fuel value in the voxel which is subject to dissipate;
-Velocity damp – reduce velocity over time;

-Burning model – define the burning scheme which has direct influence on how many heat, smoke and pressure generated and dissipated. Simple – the most simple scheme, linear dissipation of fuel and proportional generation of smoke (if enabled) and heat; Chemical mimics real chemical processes inside the burning zone, used as a default but in some cases may produce unexpectable results. Chemical macro can be more realistic for macro fire simulations but has the side effect – not able to generate smoke.

-Heat production – how many heat generated in the voxel during burning;
-Smoke from xxxx – how many smoke generated in the voxel during burning, the both of values are multiplied before the actual smoke generation, it is recommended either use 1/1 or connect the EFX FCurve Scalar compound to modulate this value by spline control, treat the remaining fuel value in the voxel as the argument to FCurve;
-Gas expansion – how many pressure generated in the voxel during burning;
-Burn rate – how many fuel consumed in the voxel during burning;
-Ignition heat – threshold above which the burning occurs;
-Max heat – max heat value at the voxel;

-UVW Full upd Frames – the duration over which the UVW fields regenerate uvw coordinates;

-Container bounds – defines the maximum simulation domain size.

Post tree

The post tree is dedicated to provide:
-Backplay load of cached data;
-Setup volume shader for both the preview window and external render;
-Display data in the XSI viewport;
-Bake multiple scattered light to additional fields;

Container size update

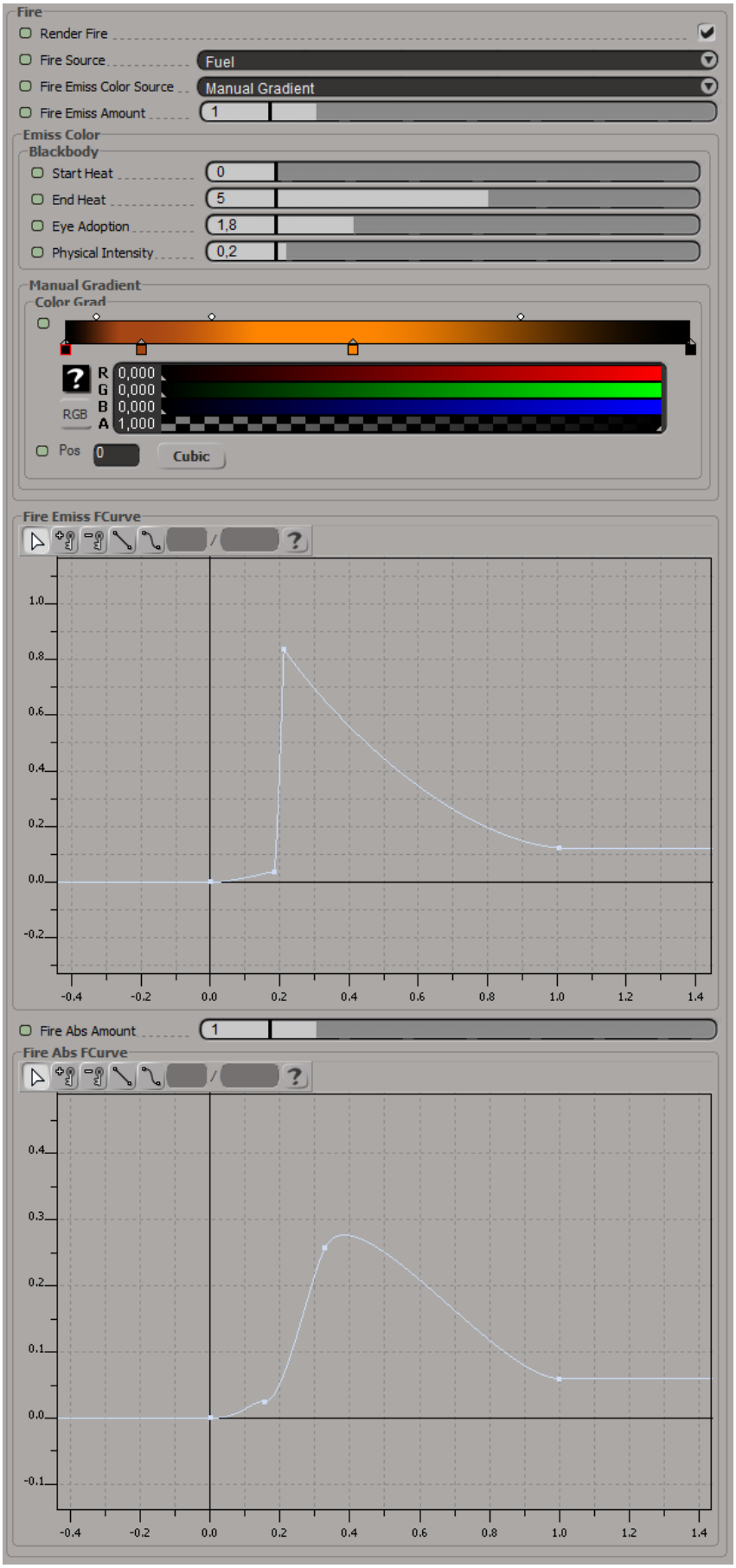
The **EFX Update Active Container Cube** compound must be connected to automatically resize the active container during playback.
The next compound can be one of three options (smoke shader, fire shader, giga (fire+smoke) shader), depending on the selected preset setup.

Volume shading

Custom shaders/mediums assembling chapter will be available soon.

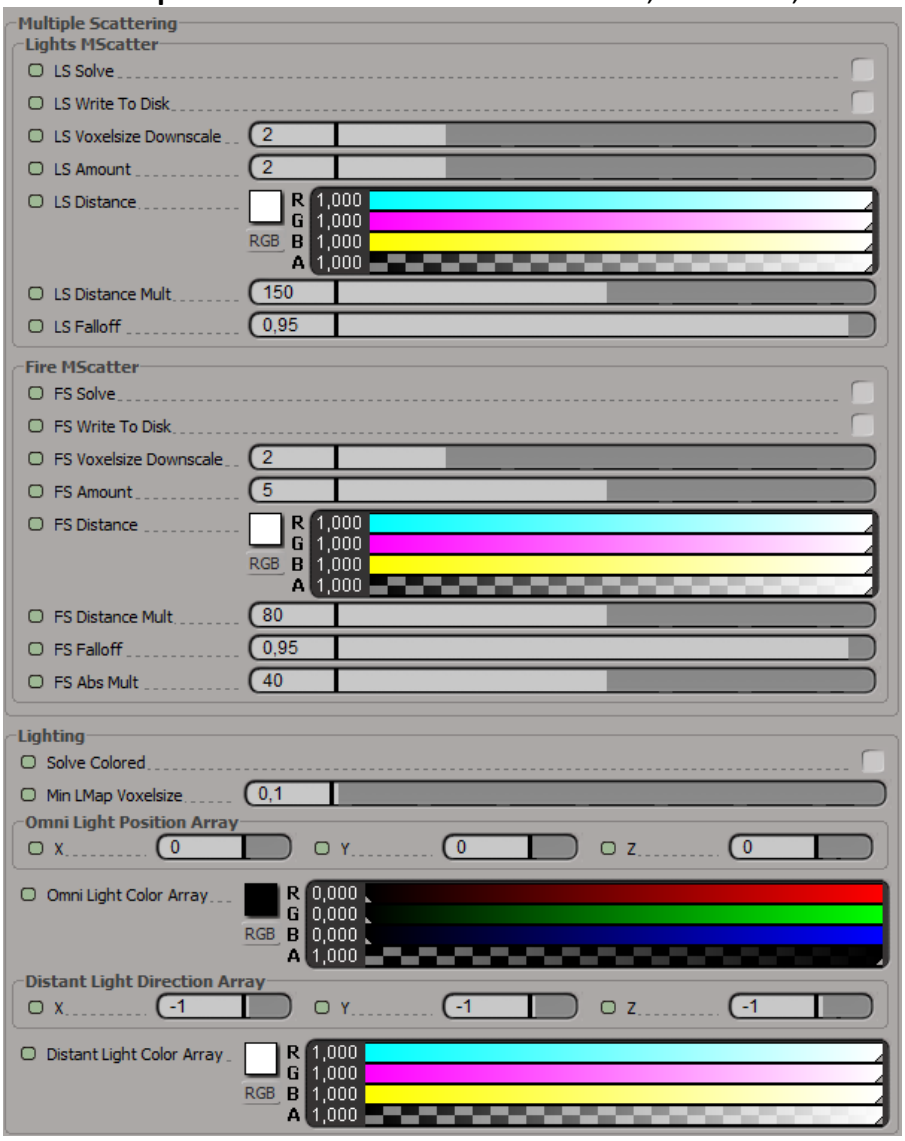
-Supress smoke at fire – remove smoke scattering and absorption where fuel is, this can help to smooth out any sharp transitions between smoke and fire;
-Smoke source – use smoke channel, another channel or any external color medium connected to the input port;
-Abs Color / amount – how transparent smoke is for camera and light rays;
-Single scattering– how many light are scattered by smoke (think about it as diffusion parameter);

-Scattering source – use any channels (see sample color smoke scene for example), custom color medium or a constant color.



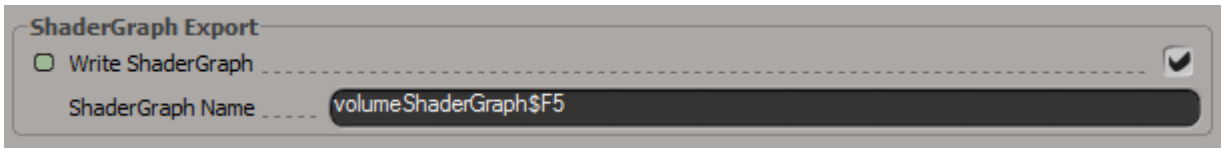
-Fire emiss color source – choose between manual RGB gradient and physically-correct black-body radiation formula, the latter option should be used with the heat channel as fire source.
-Fire emiss FCurve – multiplier for emission strength based on fire source value as the agrument;
-Fire Abs amount – how transparent fire for camera and light rays, this value is added to the smoke absorption;
-Fire Abs FCurve – multiplier for absorption strength based on fire source value as the argument.

-Mblur amount – defines the strength of motion blurring, units are in world-space velocity fraction;
-Mblur Bias – move blurring to start/end of camera shutter open state;
-Mblur samplerate – how accurate motion blur is, be careful, the units are in world-space!



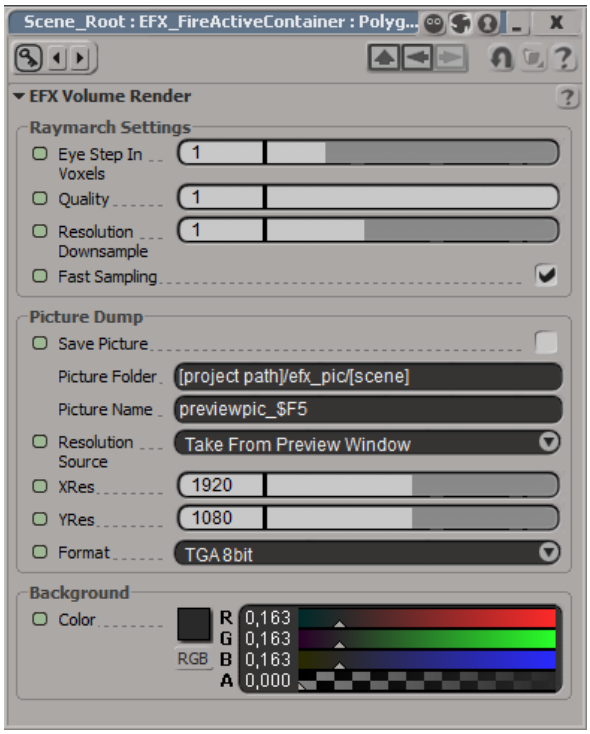
-LS solve – stands for lights scattering. Enable to get that source of light to be multiple scattered in volumes;
-LS Write to disk – write the solver LS to field and write this field to disk for rendering in the external render, should be enabled after complete volume shader tuning. To save all frames just play the desired timeline (enforce the evaluation of the post-tree at each required frame);
-LS Voxelsize downsample – reduce the LS field resolution to accelerate the computation, may produce flickering artifacts on big values;
-LS Amount – global multiplayer of strength;
-LS Distance mult – how far scattering goes;
-LS Falloff – defines the light damp (when = 0 the scattered light is equally bright on all Scattering distance);
The FS is the same as light scattering expect the light source is fire now.
-LS Abs Mult – how many light will be damped by smoke absorption (in relation with the smoke value in the voxel)

-Solve colored – solver the light propagation for RGB channels instead of single luminance(red channel by default). Not recommended to use except the cases when you have colored lights in the scene and have to solve the multiple scattering in according to these colors;
-Min LMap voxelsize – minimum allowed voxelsize for light map after which the map will not be resized to more detailed representation;
-Light color/position/direction arrays – used to generate the arrays of directed and omni lights. Lights with black color are ignored.



-Write shader graph – true by default, stores the current volume shader to disk and hooked up on the external rendering later. It is recommended to not change anything here.

Volume render

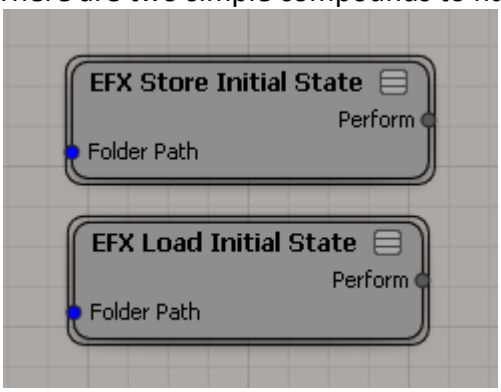


-Eye step in voxels – define sampling rate during raymarch, this is the most important quality parameter, recommended range is [0.2:1.0], less values are important for fire rendering;
-Quality – defines the increasing rate for the raymarch step size during march, 1=no increasing;
-Resolution downsample – reduce ray/pixel ratio, when=0 then 1ray per 1pixel used;
-Fast sampling – heuristic to accelerate the raymarch, disable if there are visible artifacts around the fluid border;

-Save picture – grab the raymarched render result and store to disk in specified format by specified path.

Initial state

There are two simple compounds to handle initial state:



The EFX Store Initial State is connected at the very end of the post-tree, at the desired frame. This will make a snapshot of all system registered content fields (including user-fields if any), so shortly after the store you have to disconnect this compound. Now, connect the EFX Load Initial State at the very end of the init-tree and you get your initial state loaded to all registered fields.

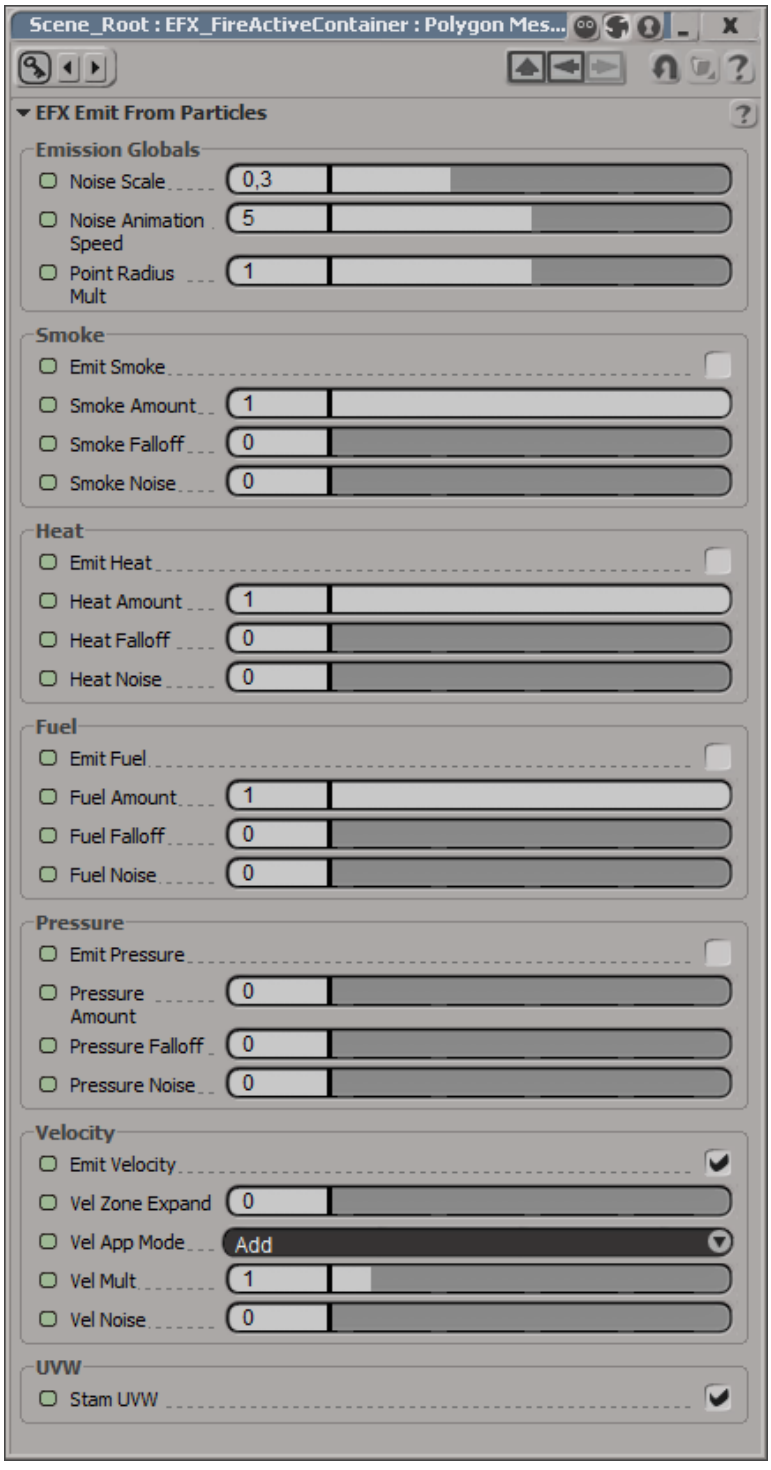
Emission

Fields content can be modified in four ways:

1) Rasterizing arbitrary point cloud onto grid
2) Mapping voxelized geometry (SDF) to grid
3) Mapping from one grid to another
4) Direct write to voxel buffer

From particles

The top-level compound for particle stamping is **EFX Emit from particles**:



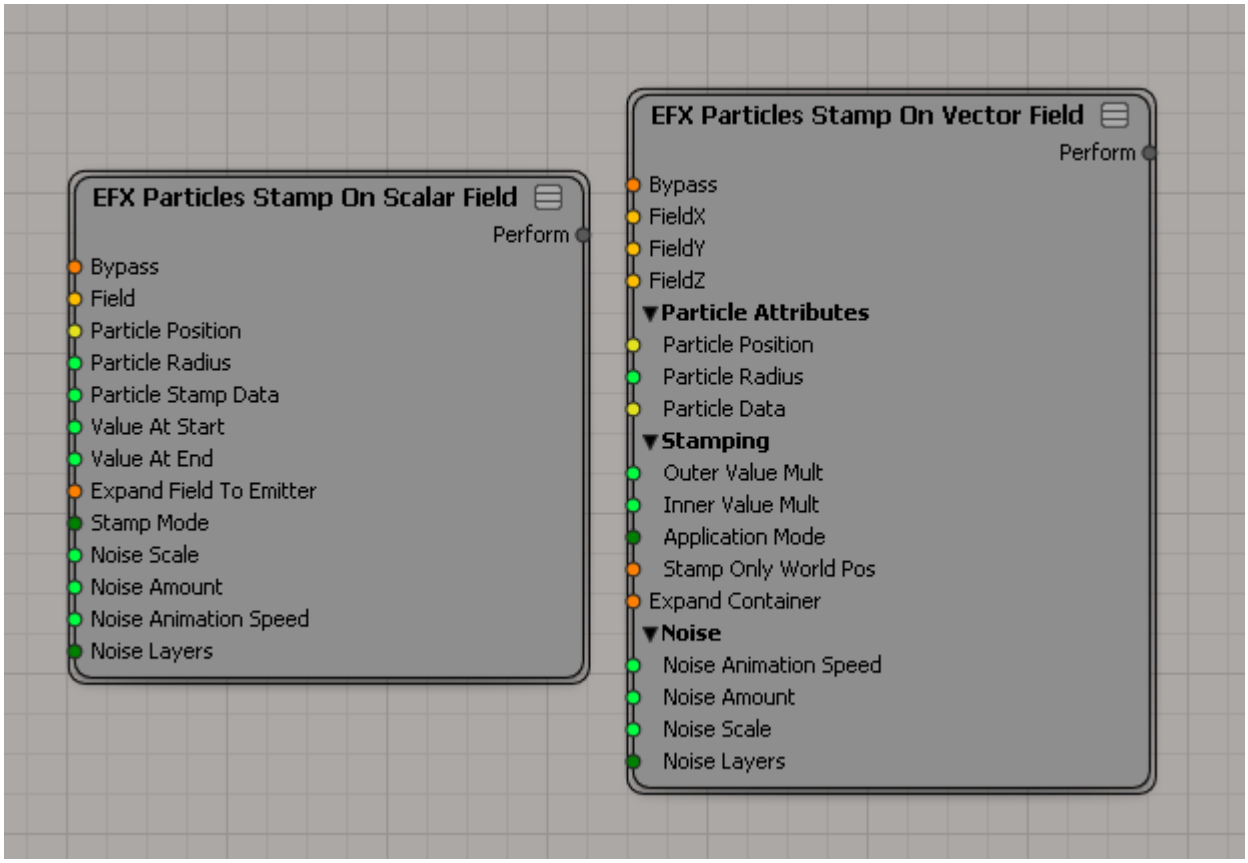
This is a general-purpose emission compound which handles emission to the factory fields such as velocity, fuel, temperature and smoke.
-Noise scale – define simplex noise scale when stamping randomized values;
-Noise animation speed – how fast simplex noise will vary over time;
-Point radius mult – global multiplier for particles radius;

-Emit xxx – enable/disable emission for specific field;
-xxx Amount – value to be stamped on specific field;
-xxx Falloff – defines how much of the Amount will be dropped to zero while approach to the border of a particle;
-xxx Noise – amount of noise multiplied with the target value;

-Vel zone Expand – additionally increase the particle radius when stamping velocity (can help to force the rest of stamped data to conform the main velocity stream injected by particles) ;
-Vel App mode – **set** mode replaces any previous data in voxels, **add** mode sums up the emission value with the old values;

-Stamp UVW – emits the world position coordinates to the U, V, W fields where particles are if the UVW is marked as solvable in the initialization tree.

To use particles as emission source for arbitrary field you have to use the 2 base compounds:



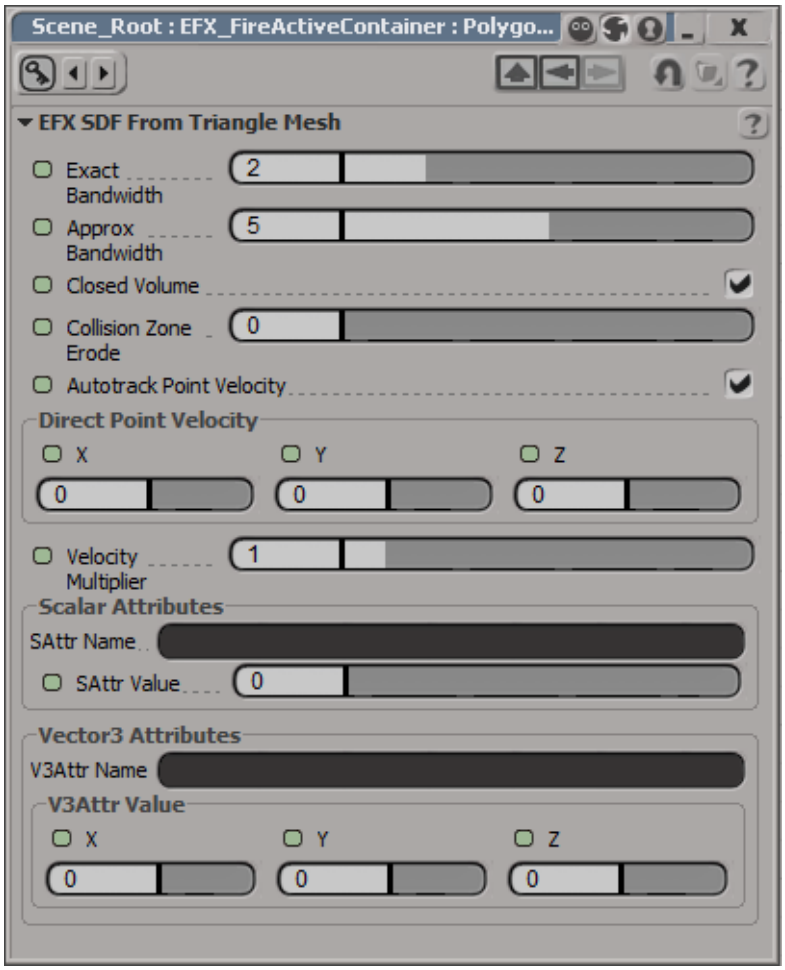
-Particle Position/ Particle Radius/ Particle Stamp Data – the arrays (or per-point attribute, or array per-point attribute) with respective data from the pointcloud-emitter. To access these and other possible attributes in handy way there is **EFX Get Pointcloud Attributes** compound;

-Value At Start/ Value At End – define the stamping values at the border/ core of a particle;
-Expand field to emitter – resize the target field to **the merged** bounding box of bounding boxes of the emission pointcloud and the current field;
-Stamp mode – define mathematical op for the emission value and the voxels current value;
-Noise scale – simplex noise scale on value randomization;
-Noise amount – noise amount;
-Noise speed – noise variation speed over time;
-Noise layers – number of mixed noises.

From Geometry (SDF)

Each polygonal mesh in EFX should be voxelized first. This is done via generation of so-called Signed Distance Field object (SDF). SDF object is represented as a regular scalar fields list inside of SDF data-type instance in the ICE, one field per polygon island typically. Note that if bounding boxes of polyislands are overlapped then the SDF field will be generated in the entire bbox of all overlapped boxes. Each voxel in that SDF field contains distance value to the closest point on the input mesh. This distance has different sign: + inside of the mesh and – outside. This property is useful for both obstacles processing in the solver internals and for emission from polygons too.

The main voxelization compound is **EFX SDF From Triangle Mesh**:

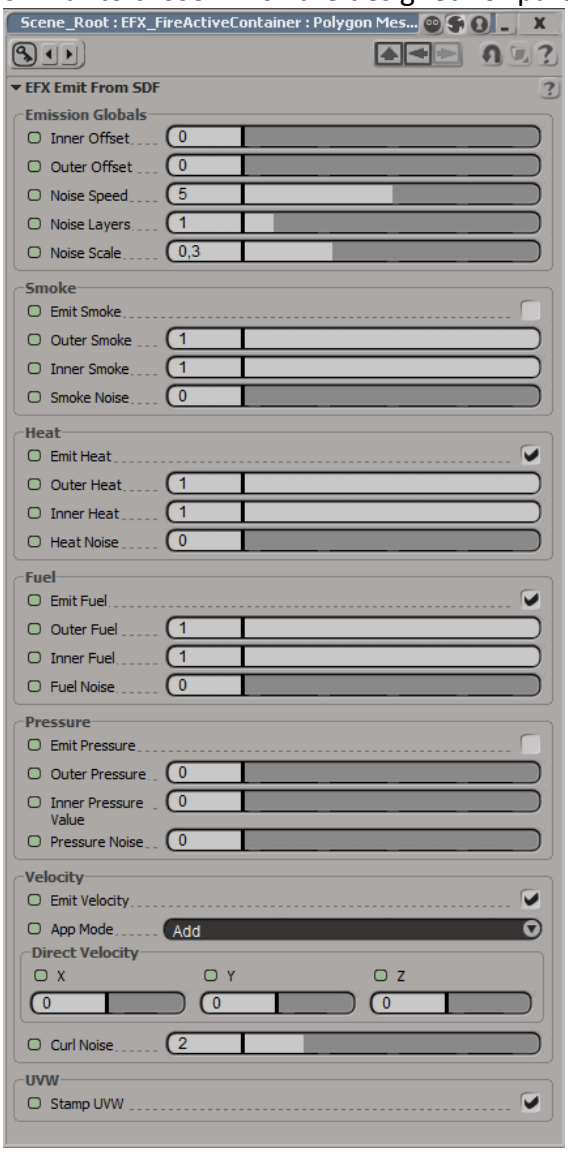


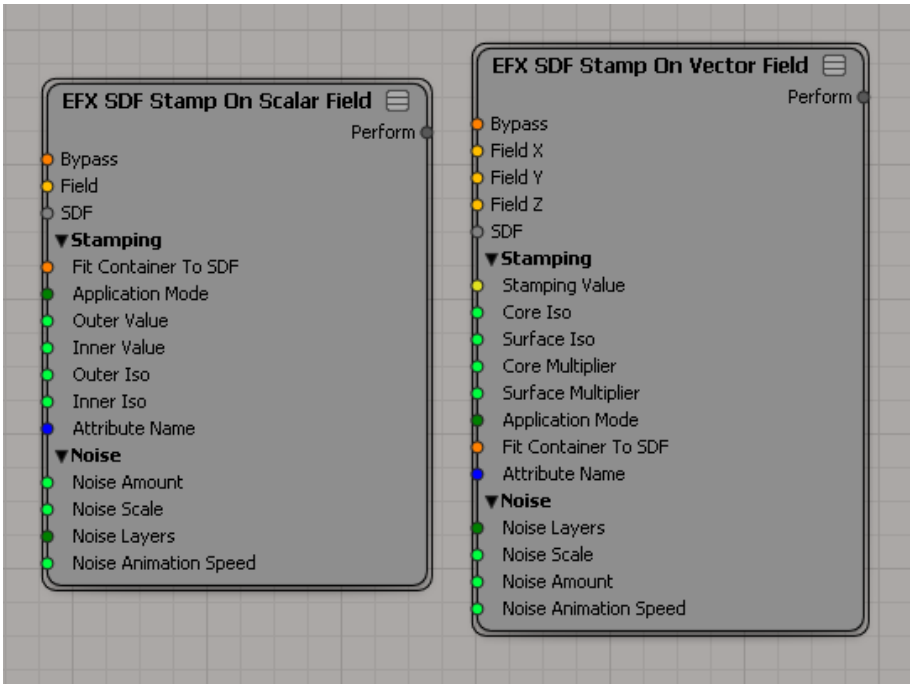
-Exact bandwidth – distance in world units on which the exact SDF value will be calculated for the voxels in that range;
-Approx bandwidth – downsampled SDF calculation (thus faster) for voxels in that range;

-Closed volume – set this as true when you have a proper mesh without holes and selfintersections (intersected but unconnected islands are allowed as well as fully disconnected triangles which preserve physical match of adjacent vertices), otherwise the SDF generator will apply shell op first and then will voxelize the result within the **exact bandwidth** distance;
-Collision Zone Erode – (**DEPRECATED**) defines the erosion distance when the same SDF used as an obstacle (ignoring this value for emission);
-Autotrack Point velocity – calculate velocity values for each vertex over time (only in case of constant topology);
-Direct Point Velocity – uniform velocity value added when using the SDF with general-purpose **EFX Emit From SDF** compound;

-SAttr Name /value – attach user-defines either per-point (per-point of the source mesh) or singleton scalar attribute to this SDF;
-V3Attr Name /value – attach user-defines either per-point (per-point of the source mesh) or singleton vector3 (you can use it as RGB in case of color) attribute to this SDF;

Once you have generated your SDF you can use it as a source for the emission compounds very similar to those which are designed for particle emission:





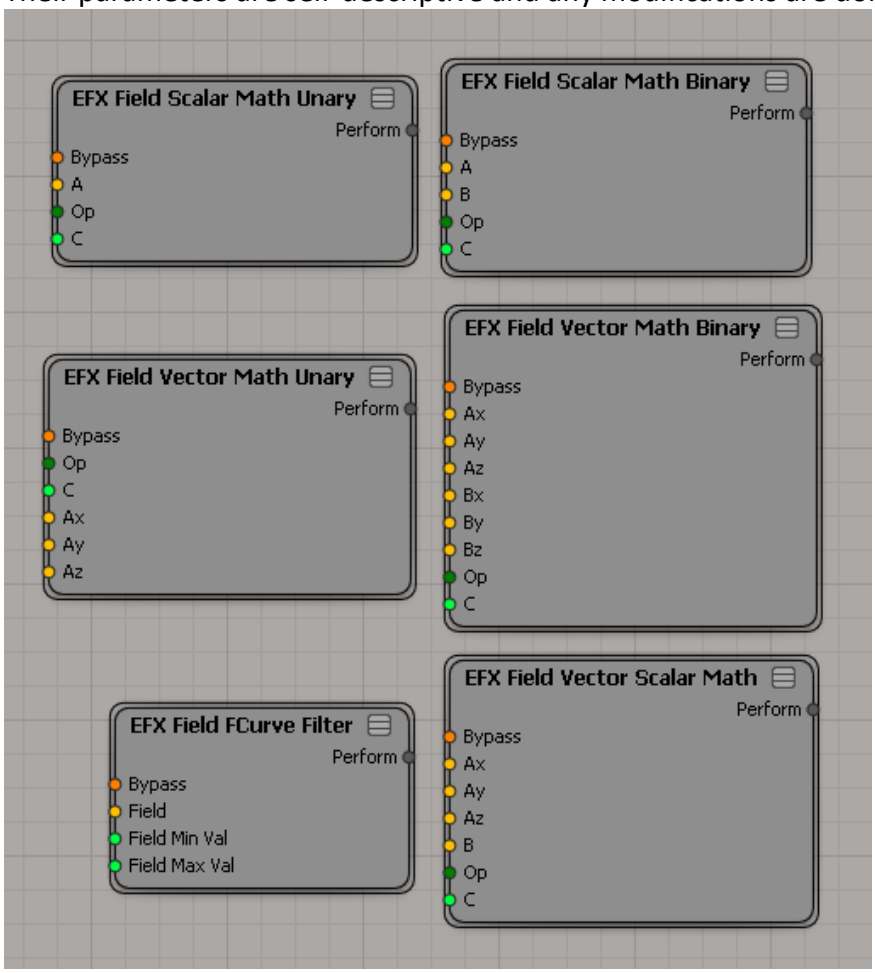
Each SDF has a set of built-in attributes which you can specify in the respective parameters in these compounds:

Scalar:
"**sdf**" – closest distance for evaluated voxel;

Vector3:
"**sdfgradient**" – normalized vector3 value which defines the direction to the closest point;
"**worldposition**" – current voxel world position, mainly used as UVW emission source;
"**velocity**" – total velocity for the current voxel transferred from the source mesh points.

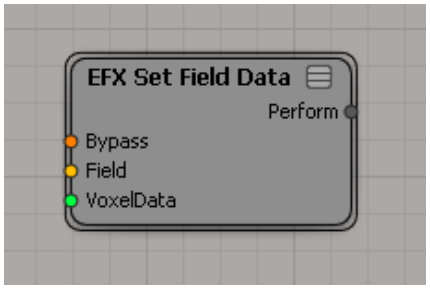
From another field

This is possible to transfer voxels data from one field to another via set of mathepatical compounds dedicated for cross-field interaction.
Please see the complete list under the ICETree Editor-> Task-> Explosia FX -> Tools :
EFX Field Scalar/ Vector Math Unary/ Binary
Their parameters are self-descriptive and any modifications are done in-place.



Direct field modification

The most low-level data modification across voxels is a manual setting via array of scalars:



Note that the length of array must comply with the total voxels count in the field.

Obstacles (emission)

Obstacles can be added to the simulation from the same SDF generators as used for emission, via the **EFX Add Obstacles compound**: