Softimage Explosia FX

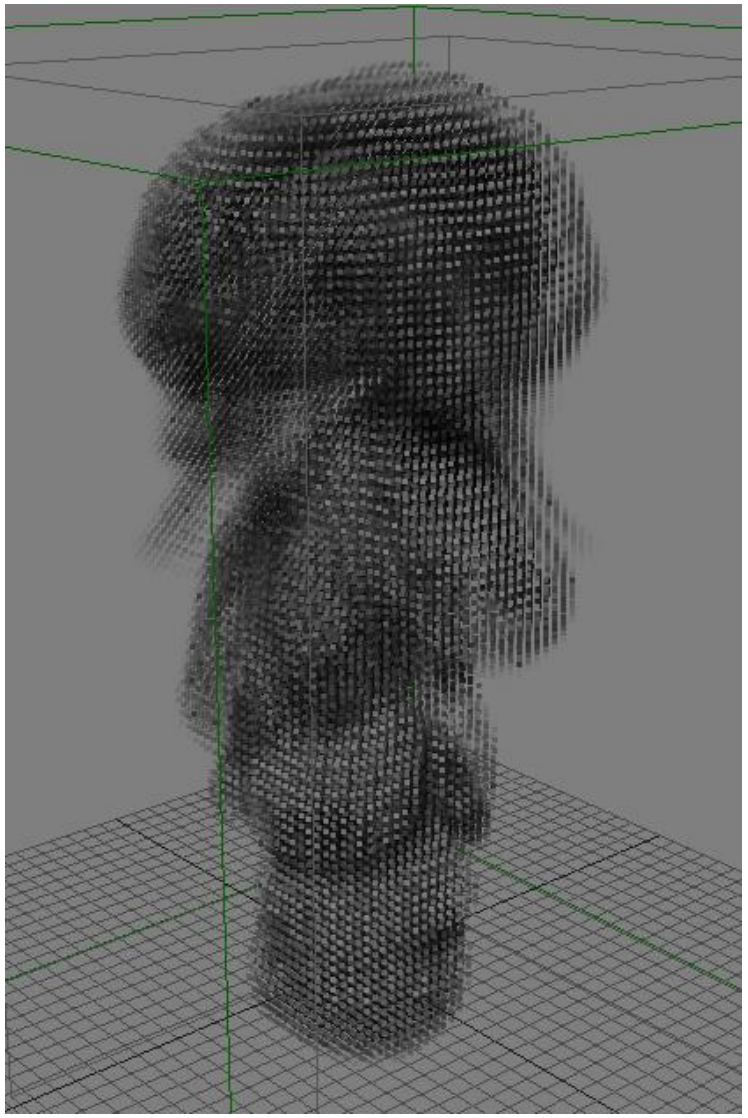
Overview

Voxel based gas fluid simulator

- -Dense voxel fields
- -Fully CPU based
- -Aimed to SolidAngle Arnold render
- -NVidia MentalRay as fallback
- -General data exchange via OpenVDB
- -Field manipulation via nodes (ICE)
- -Shader assembling via nodes(ICE,RT)
- -Fast preview, consistent with Arnold

Simulation Primitive

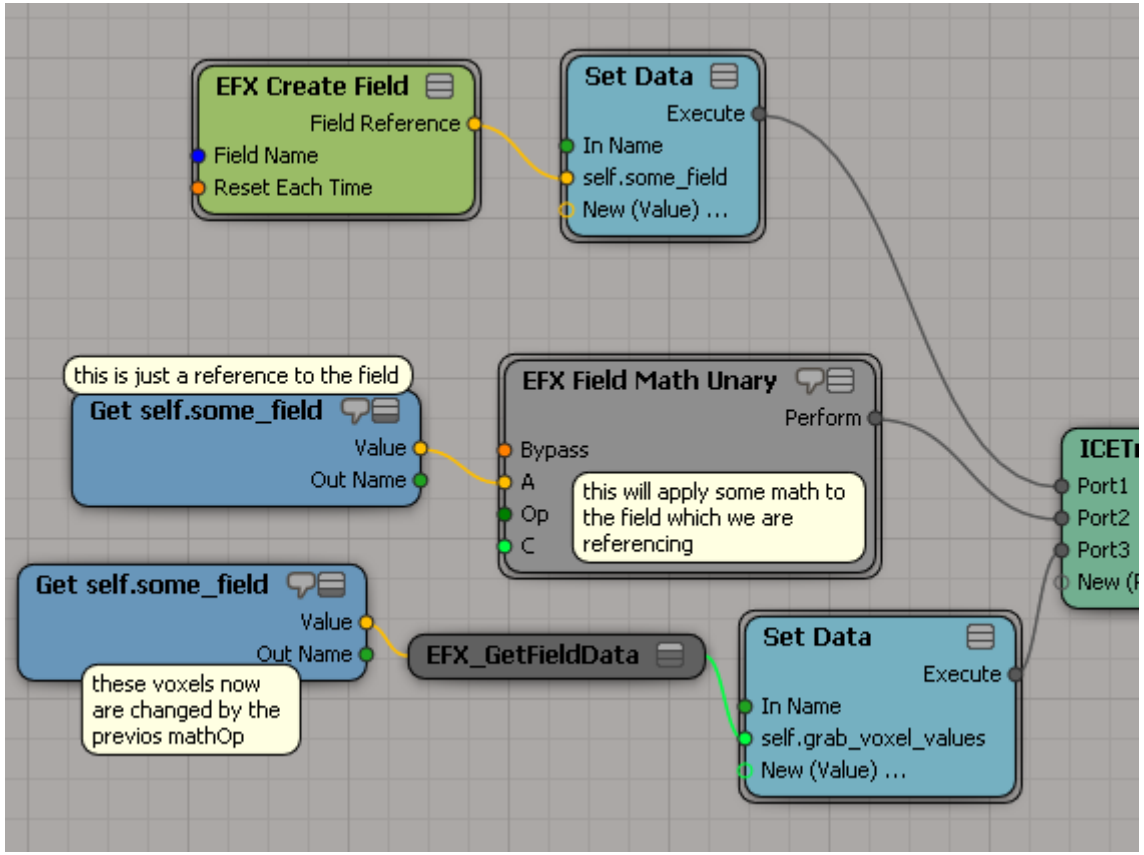
- Rectangular box filled with voxels - field
- Field carries data, kind of per voxel context
- Only scalar fields, 1 voxel = 1 scalar value
- Vector3 data formed as 3 scalar fields
- Same for color(RGB) data



Field

- Always have a name and resolution
- Work with fields differs from typical ICE workflow
- Field holder node is capable to output **field reference** only
- All subsequent processing done with the same field by its reference
- Except few utility nodes such as copy-field nodes
- Hence no redundant copies and no extra memory consumed
- But need to make explicit copies via copy-field-node in some cases

Important to understand

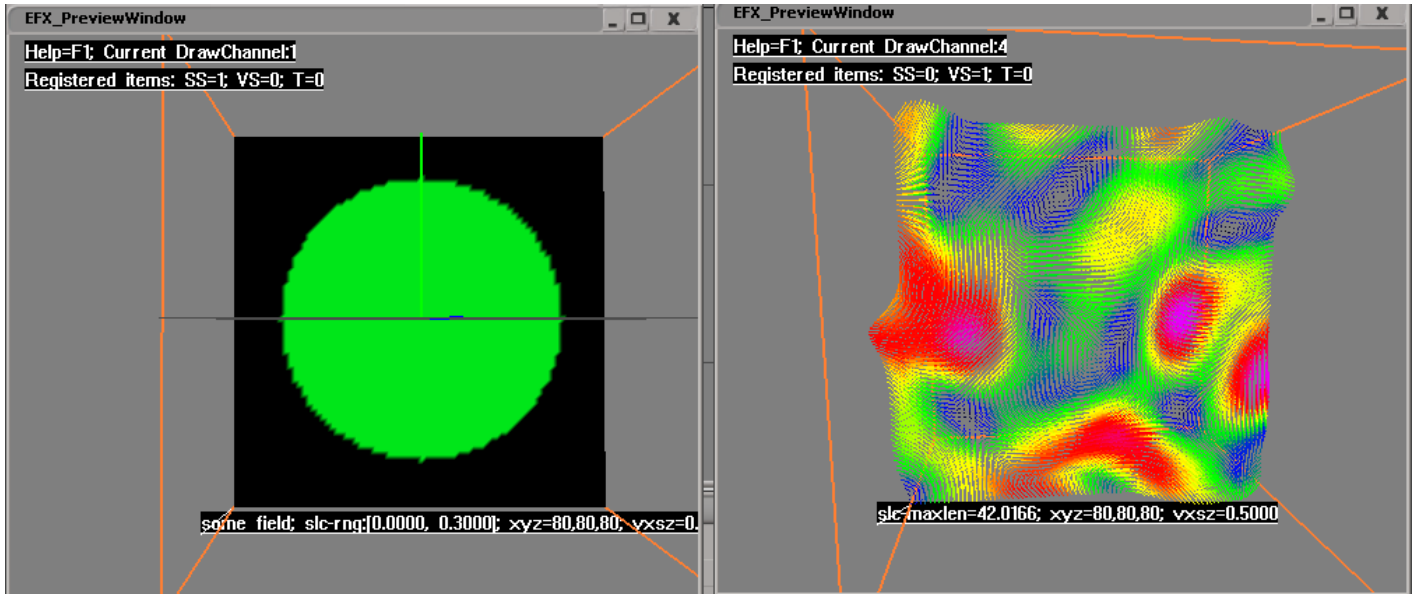


Data Debug Visualization

- Field is not native primitive for Softimage
- Requires special care on visualization
- ExplosiaFX provides dedicated preview OpenGL viewport
- And introduces three drawing primitives:
  - Scalar field slice, draws field content in slice using remapping RGB gradient
  - Vector field slice, draws 3 scalar fields slice as array of lines(vectors)
  - SDF Isosurface, draws triangle array of SDF(obstacles, will discuss later)
- Has 10 drawchannels
- Each drawprim binds to any channel
- Switching between channels by hotkeys 0-9 respectively

Debug Visualization Slices

- Picked at specific place and time, no time depended on field state
- Colorized with RGB gradient, also has min-max values statistic
- Pick position defined in either local (0-1) or world (-INF,+INF) cords



SDF

- Stands for Signed-distance-field, we actually have only DF
- Internally is just a set of scalar fields, 1 per non-overlapping polygon island
- Contains distance from current voxel to closest point on island mesh
- Used as a data emitter or/and as an obstacle for solver
- Voxelizes and solves for distance any closed mesh directly
- Applies shell to open meshes (with holes)
- Can transfer any scalar or vector3 attribute within per-point or single-object context from input polygonal mesh

SDF

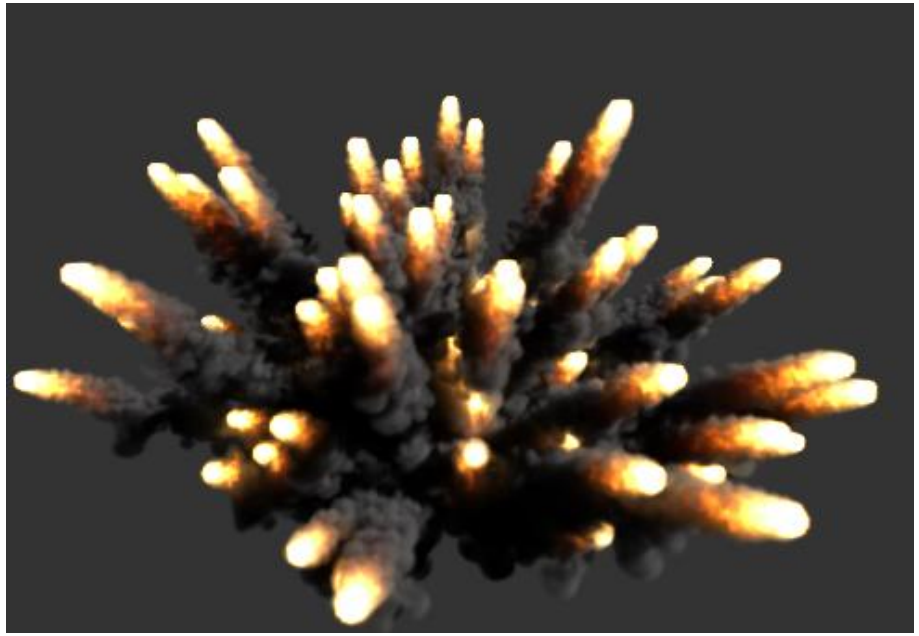
- Input mesh treated as closed physically, meaning two equivalent vertices (v1pos = v2pos) define connection, easy way to transfer per node attribute (such as UV, texture etc.) is to disconnect all triangles
- Object velocity tracked automatically by default, stored in “velocity” vector3 attribute, but only for static topology
- Inner/outer iso defines a chain(or surface) of voxels with values=iso
- Possible to shift collision zone and use simultaneously as emitter and obstacle

Stamping (Emission)

- Data emission from source to field
  - From particles
  - From SDF
  - From field-to-field (strictly speaking not stamping)
- Data stamping has several math application modes(set,add,sub,etc)
- Smoke, heat, fuel are bounded to [0,1] range and applied as set/max operation
- Pressure (artificial divergence, negative value explodes, positive tends to collapse) is unbounded and usually applies as set op
- Velocity is either set or add op (to account for existing velocity there)

Particle stamping

- Each particle has position, velocity, radius and user stamping data: scalar and vec3 types
- Sometimes useful to expand radius on velocity stamp, keeping it small for smoke
- Fast moving particles require more substeps to avoid sparsity between last and current positions, particle trailing as an solution



SDF Stamping

- Values at surface(upper voxel band) and core(deepest voxels) of emitter can be different, so mid is interpolated
- Collision zone should starts at the surface of stamping zone
- Stamp of user-attributes is by name
- Built-in attributes:
  - vec3: velocity, sdfgradient, worldposition(used for UVW);
  - scalar: sdf;

Simulation Framework

- Simulation processed on an orthogonal polycube which bounds active domain – simulation container
- Simcontainer has a minimal set of fields and base parameters (voxelsize, cache path, etc.)
- Any configuration has at least 9 fields: 3 velocity, 1 artificial pressure and 5 auxiliary
- Possible to include fuel, heat, smoke and UVW
- Basic fluid motion requires only velocity (i.e. no smoke, fuel, heat)
- User-specified fields (e.g. colorRGB, age, mask) possible as well

Simulation Framework

- Simcontainer has 3 ICETrees: initialization, simulation, postsimulation
- The init tree setups base parameters and creates framework fields
  - User specified fields initialization goes here as well
- The simulation tree performs stamping, simulation, fields export
  - Tune ICE subsamples count here
- The postsim tree setups volume shader, exports multiple scattered light and reads exported fields after the simulation for preview purposes
  - Any changes made here will have an impact on the next frame of simulation(because of referencing)
- Solver (in general sense) has 3 stages:
  - External forces injection (including burn solver)
  - Incompressibility enforcement (well known pressure solver)
  - Quantities and velocity transport (also known as advection or convection)

External Forces

- Buoyant forces, vorticity amplification, user-defined forces
- Burn solver decrements fuel, increments heat, smoke and pressure
  - Big pressure (gas expansion) and rapid movement requires some air buffer around the flame to prevent shape distortions
- Velocity damp, quantities dissipation
- Smoke, fuel and heat are restricted to [0;1] range
  - Much easier to tune since no real physics are simulated in any CG fluid solver
- Use scalar/vector slices to see and debug channels data

Incompressibility enforcement

- After force injection we have invalid(compressed) velocity
- Pressure solver solves for correcting solution
- Solution correctness defined only by visual pleasure
  - True for gases, water sims requires also mathematical error-free solution
- Modern multigrid solver gives nice result after only 2 iterations
  - In most cases, extreme pressures and very hi-res grids sometimes need more
- Unnatural shape distortions can be resolved by increasing air buffer zone
- Artificial positive and negative pressure causes shrink and expansion
  - Strictly speaking we are talking about artificial divergence rather pressure

Advection

- Any advection tends to smooth out eddies in flow over time
- Use “smart” advection type to minimize diffusion
  - Smart makes use of +3 additional fields internally, not preferable to use on hi-res sims
- When dealing with colorRGB advection it may be useful to stamp with wider stencil to avoid desaturated shape borders caused by advection diffusive errors
- Avoid big maxspeed per step parameter values, 5-10 is preferable
  - Big values result in “rubber” flames and unnatural elongated flow features

Fields Input-Output

- Native format is uncompressed, no empty regions skipping
  - Fast
- OpenVDB format is a standard for volumetric data exchanging
  - Slow, but greatly reduces the final file size, up to tens times
- Writes selected fields each frame at the last ICE substep
- Reads on the timeslider dragging back
  - Simcontainer polycube is updated on this read
- Multiplescattered light fields stored after the simulation, in the postsim ICEtree
  - Actually need to play each frame to evaluate the postsim tree (xsi arch nuance)

Fields tools

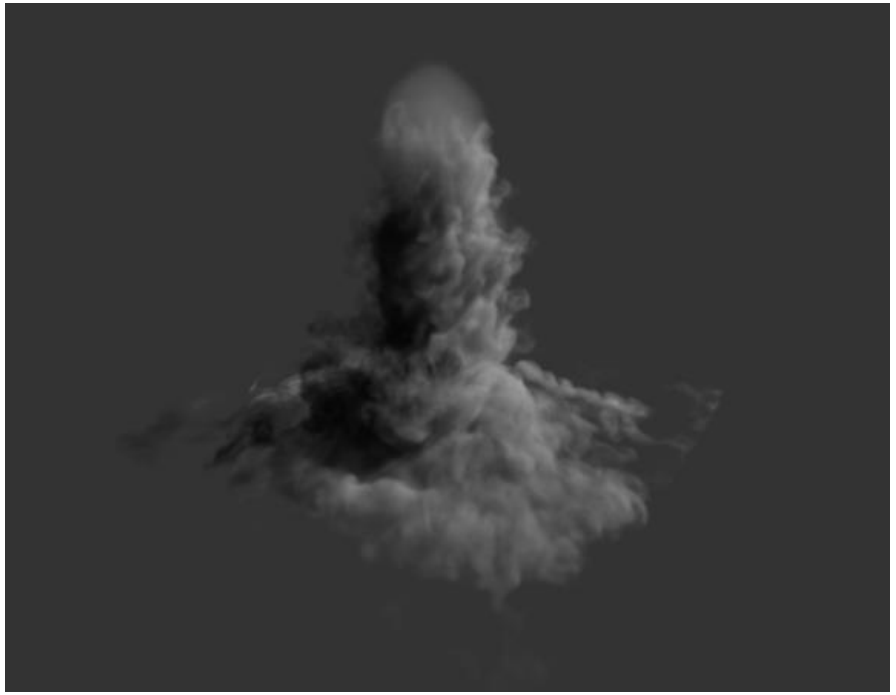
- Several basic math operations on scalar field, on scalar pairs, on vector/vector pairs fields
  - Result is stored to one of input fields
- More sophisticated ops available: divergence, gradient, curl etc.
  - Mainly used for mask generation
- Turbulence (based on vector curl noise) and vorticity amplification
  - Able to mask and/or multiply by another field
  - Operates at user-specified scale
  - Can be used multiply times as cascade with different scales
- Distinct diffusion solver, advection solver
- Particle scatter in field
- Field sampling by position (array/per-point contexts)

Volume shading

- Standard physical model, raymarching
- 3 main light properties: absorption + scattering + emission
- Material is assembled and exported along with fields in the postICEtree
  - Still possible to assemble/extend in RT
- Volume material building blocks(shaders) called mediums
- Three presets for fire, smoke and fire+smoke setups
- Multiple scattered light is evaluated in the postICEtree, saved to field and then used at final rendering
- Preview rendering is consistent with both Arnold and Mray, but uses light map as the main light solver, while final renderers use ray-tracing

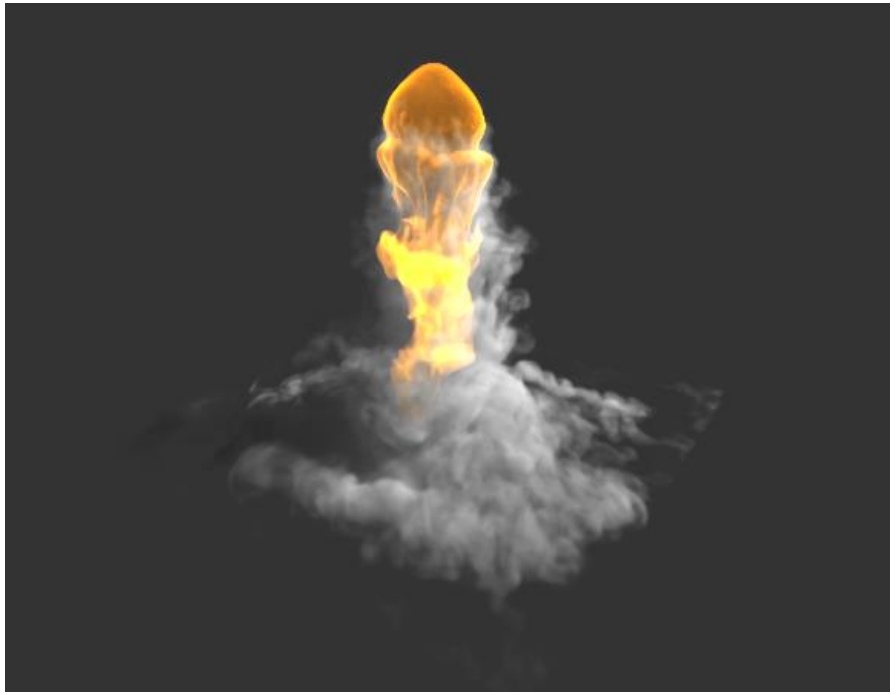
Absorption

Absorption + direct scattering



Absorption + direct scattering + multiple scattering

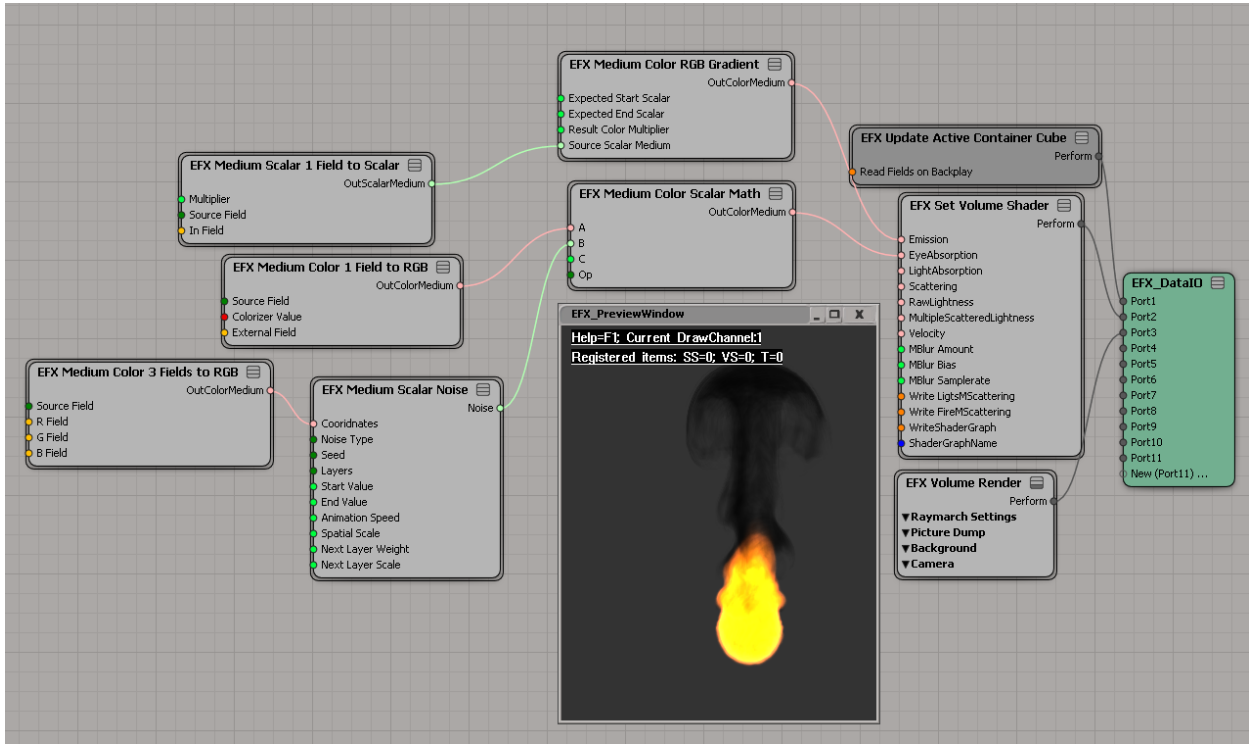
Absorption + direct scattering + multiple scattering + emission



Absorption + direct scattering + total multiple scattering + emission

Volume medium assembling

- Similar to RT approach with material root node
- Mediums math similar to fields math
- Extra tools such as blackbody radiator and 3d noise



Limitations and future goals

- Dynamic obstacles can produce artifacts within multigrid solver
- Support of expressions will allow complex math in one-two line of code instead of dozens of nodes
- Good GPU acceleration usually imposes 10x speedup (pure difference of RAM bandwidth at CPU and GPU)
- Dedicated volume render can be pretty effective in many cases
- What next? –less memory, more speed, improved user interaction and relaxed coupling with Softimage to wider DCC support

For more information visit:

http://blackcore.technology/softimageefx
https://vimeo.com/groups/efxfluid
https://groups.google.com/forum/?#!forum/explosiafx