Production Volume Rendering

Jerry Tessendorf
Principal Graphics Scientist
Rhythm and Hues Studios
jerryt@rhythm.com
November 2008
In Production

Rhythm & Hues Studios

Land of the Lost

Be Afraid, Be Sort of Afraid

They Came From Upstairs

Cirque Du Freak: A Living Nightmare

Night at the Museum

The Fast and the Furious 4
In Production

Volume Rendering in heavy use
Outline for this week
Outline for this week

- Volume rendering in film production
- Rendering equation/numeric algorithm: without lights
- Gridded Volumes: Voxels

Day 1
Outline for this week

- Volume rendering in film production
- Rendering equation/numeric algorithm: without lights
- Gridded Volumes: Voxels

Day 1

- Rendering equation/numeric algorithm: with lights
- Methods to fill a volume with interesting density

Day 2
Volume Rendered Smoke
Volume Elements
Volume Rendering
Volume Rendering

• Accumulate opacity along light of sight.
Volume Rendering

• Accumulate opacity along light of sight.

• Accumulate color along line of sight, weighted by accumulated opacity and light source.
Volume Rendering

- Accumulate opacity along light of sight.
- Accumulate color along line of sight, weighted by accumulated opacity and light source.
Volume Rendering

• Accumulate opacity along light of sight.

• Accumulate color along line of sight, weighted by accumulated opacity and light source.
Volume Rendering

- Accumulate opacity along light of sight.
- Accumulate color along line of sight, weighted by accumulated opacity and light source.
Volume Rendering

• Accumulate opacity along light of sight.

• Accumulate color along line of sight, weighted by accumulated opacity and light source.
Volume Rendering

• Accumulate opacity along light of sight.

• Accumulate color along line of sight, weighted by accumulated opacity and light source.
Volume Rendering

- Accumulate opacity along light of sight.
- Accumulate color along line of sight, weighted by accumulated opacity and light source.
Volume Rendering

- Accumulate opacity along light of sight.
- Accumulate color along line of sight, weighted by accumulated opacity and light source.
Volume Rendering

- Accumulate opacity along light of sight.
- Accumulate color along line of sight, weighted by accumulated opacity and light source.
Volume Rendering

- Accumulate opacity along light of sight.
- Accumulate color along line of sight, weighted by accumulated opacity and light source.
Volume Rendering

- Accumulate opacity along light of sight.
- Accumulate color along line of sight, weighted by accumulated opacity and light source.
Volume Rendering

• Accumulate opacity along light of sight.

• Accumulate color along line of sight, weighted by accumulated opacity and light source.
Volume Rendering

• Accumulate opacity along light of sight.

• Accumulate color along line of sight, weighted by accumulated opacity and light source.
Volume Rendering

- Accumulate opacity along light of sight.
- Accumulate color along line of sight, weighted by accumulated opacity and light source.
Volume Rendering

- Accumulate opacity along light of sight.
- Accumulate color along line of sight, weighted by accumulated opacity and light source.
Volume Rendering

- Accumulate opacity along light of sight.
- Accumulate color along line of sight, weighted by accumulated opacity and light source.
Volume Rendering

- Accumulate opacity along light of sight.
- Accumulate color along line of sight, weighted by accumulated opacity and light source.
Volume Rendering

- Accumulate opacity along light of sight.
- Accumulate color along line of sight, weighted by accumulated opacity and light source.
Volume Rendering

- Accumulate opacity along light of sight.
- Accumulate color along line of sight, weighted by accumulated opacity and light source.
Avalanche Sequence

The Mummy: Tomb of the Dragon Emperor
Avalanche Sequence

The Mummy: Tomb of the Dragon Emperor
Density & Color Fields

- Density is a scalar as a function of position
  \[ \rho(x) = \begin{cases} 
  > 0 & \text{inside material} \\
  0 & \text{everywhere else} 
\end{cases} \]

- Material color is a triplet as a function of position
  \[ \vec{c}(x) = (r(x), g(x), b(x)) \]

- Density and color are the fundamental inputs
Soft White Sphere

- White color: $\vec{c}(\mathbf{x}) = (1, 1, 1)$
- Formula for sphere density

$$
\rho(\mathbf{x}) = \begin{cases} 
1 - \frac{|\mathbf{x}|^2}{r^2} & 1 - \frac{|\mathbf{x}|^2}{r^2} > 0 \\
0 & 1 - \frac{|\mathbf{x}|^2}{r^2} \leq 0 
\end{cases}
$$

- Sphere has soft edges because density tapers at edges
Volume Rendered Soft White Sphere
(no lights)
Accumulating Color

- Camera located at $x_c$
- Pixel looks in direction $\hat{n}$
- Pixel sees color accumulated along the line $x_c + \hat{n} s$
- Accumulated color $\tilde{C}(x_c, \hat{n})$ has mathematical form

$$\tilde{C}(x_c, \hat{n}) = \int_0^\infty ds \; \tilde{G}(x_c, \hat{n}, s)$$
Proportional to material color and density

\[ \vec{G}(x_c, \hat{n}, s) = \vec{c}(x_c + \hat{n}s) \rho(x_c + \hat{n}s) \ T(x_c, \hat{n}, s) \]
Transmissivity & Opacity

- Transmissivity gives fraction of light passing through volume to reach camera

\[ T(x_c, \hat{n}, s) = \exp \left\{ -\kappa \int_0^s ds' \rho(x_c + \hat{n}s') \right\} \]

- Opacity is the complement of transmissivity

\[ O(x_c, \hat{n}, s) = 1 - T(x_c, \hat{n}, s) \]
Rendering Equation: No Lights

\[ \bar{C}(x_c, \hat{n}) = \int_{0}^{\infty} ds \ \rho(x_c + \hat{n}s) \ \bar{c}(x + \hat{n}s) \ \exp \left\{-\kappa \int_{0}^{s} ds' \ \rho(x_c + \hat{n}s')\right\} \]
Discretize Integration

- Reduce integral over $s$ to a discrete sum evaluated at evenly space points on ray line

$$x_i = x_c + \hat{n} \Delta s i, \quad i = 0, \ldots, \infty$$

- Full sum looks like

$$\tilde{C}(x_c, \hat{n}) = \sum_{i=0}^{\infty} \Delta s \rho(x_i) \tilde{c}(x_i) T(x_c, \hat{n}, i\Delta s)$$
Ray Marching

- Iterative version of this is a march along a line from the camera into the volume.
- Initialize $\vec{C} = (0, 0, 0)$ $x_i = x_c$
- Proceed iteratively to update position and color

$$x_i + = \hat{n} \Delta s$$

$$\vec{C} + = \Delta s \rho(x_i) \vec{c}(x_i) T(x_c, \hat{n}, i\Delta s)$$
At start of march, initialize \( T = 1 \)

As march proceeds, update transmissivity as

\[
T \ast = \exp \left\{ -\kappa \Delta s \, \rho(x_i) \right\}
\]
Full Iterative Ray March

\[ \mathbf{x}_i \ + \ = \ \hat{n} \ \Delta s \]

\[ T \ * \ = \ \exp \{-\kappa \Delta s \ \rho(\mathbf{x}_i)\} \]

\[ \mathbf{C} \ + \ = \ \Delta s \ \rho(\mathbf{x}_i) \ \mathbf{c}(\mathbf{x}_i) \ T \]
Opacity Problem

- When composited into a scene, there is sometimes a black fringe around volume edge.
- Problem lies in how transmissivity is integrated in ray march.

Image from: Antoine Bouthors, Interactive multiple anisotropic multiple scattering in clouds, ACM Symposium on Interactive 3D Graphics and Games (I3D), 2008
Solution

- Instead of

\[ \vec{C} + = \Delta s \rho(x_i) \vec{c}(x_i) T \]

- A better representation of the integral is

\[ \vec{C} + = \frac{1 - e^{-\kappa \Delta s \rho(x_i)}}{\kappa} \vec{c}(x_i) T \]
Complete Ray March (No Lights)

\[ \mathbf{x}_i + = \hat{n} \Delta s \]

\[ \Delta T = \exp \{-\kappa \Delta s \rho(\mathbf{x}_i)\} \]

\[ T \ast = \Delta T \]

\[ \mathbf{C} + = \frac{1 - \Delta T}{\kappa} \mathbf{c}(\mathbf{x}_i) \ T \]
Containers

- More efficient to ray march only where there actually is density
- Can use closed geometry as a bounding container
- Finer container definition improves volume render efficiency
Containers

- More efficient to ray march only where there actually is density
- Can use closed geometry as a bounding container
- Finer container definition improves volume render efficiency
Containers

• More efficient to ray march only where there actually is density

• Can use closed geometry as a bounding container

• Finer container definition improves volume render efficiency

camera
Containers

• More efficient to ray march only where there actually is density

• Can use closed geometry as a bounding container

• Finer container definition improves volume render efficiency

camera
Containers

- More efficient to ray march only where there actually is density
- Can use closed geometry as a bounding container
- Finer container definition improves volume render efficiency
Containers

- More efficient to ray march only where there actually is density
- Can use closed geometry as a bounding container
- Finer container definition improves volume render efficiency
Containers

- More efficient to ray march only where there actually is density
- Can use closed geometry as a bounding container
- Finer container definition improves volume render efficiency
Containers

- More efficient to ray march only where there actually is density
- Can use closed geometry as a bounding container
- Finer container definition improves volume render efficiency
Containers

- More efficient to ray march only where there actually is density
- Can use closed geometry as a bounding container
- Finer container definition improves volume render efficiency
Containers

- More efficient to ray march only where there actually is density
- Can use closed geometry as a bounding container
- Finer container definition improves volume render efficiency

camera
Containers

- More efficient to ray march only where there actually is density
- Can use closed geometry as a bounding container
- Finer container definition improves volume render efficiency
Containers

- More efficient to ray march only where there actually is density

- Can use closed geometry as a bounding container

- Finer container definition improves volume render efficiency
Why Use Gridded Volumes

- Some algorithms for density & color work better with grids
- Even if density & color can be written as equations, they may be so slow to evaluate that a gridded sample is better
- Can store gridded values on disk for later use.
Gridded Volumes: Voxels

- Rectangular mesh of points \( x_{ijk} \) at the center of voxels labelled \( ijk \)
- \( i,j,k = 1, \ldots, M \)
- At each voxel center, density and color have values \( \rho_{ijk} \) \( \vec{c}_{ijk} \)

\[
\rho(x_{ijk}) = \rho_{ijk} \\
\vec{c}(x_{ijk}) = \vec{c}_{ijk}
\]
\( \vec{F} = (F_r, F_g, F_b) \)

\( \vec{c} = (r, g, b) \)

\( \vec{c} \odot \vec{F} = (r F_r, g F_g, b F_b) \)

\( \rho_{ijk} = \rho_{ij}^{ijk} \)

\( x_{ijk} \)

\( x_{ij} \)

\( x_{i+1j} \)

\( x_{ij+1k} \)

\( x_{i+1j+1k} \)
\[ \mathbf{c} = (r, g, b) \]

\[ \mathbf{c} \cdot \mathbf{F} = (rF, gF, bF) \]

\[ \rho_{ijk} = \rho_{ij}(x_{ijk}) \]

\[ x_{ijk+1} = x_{ij+1k} \]

\[ \mathbf{c}_{ijk} \cdot \mathbf{c}_{ij+1k} = \mathbf{c}_{ij+1j+1k} \]
Trilinear Interpolation

- Can express ray march position as a weighted sum of voxel positions

\[
x_i = \sum_{a=i}^{i+1} \sum_{b=j}^{j+1} \sum_{c=k}^{k+1} x_{abc} \omega_{abc}
\]

- Weights are positive and normalized

\[
\sum_{a=i}^{i+1} \sum_{b=j}^{j+1} \sum_{c=k}^{k+1} \omega_{abc} = 1
\]

- Use weights for density & color interpolation

\[
\rho(x_i) = \sum_{a=i}^{i+1} \sum_{b=j}^{j+1} \sum_{c=k}^{k+1} \rho_{abc} \omega_{abc}
\]

\[
\bar{c}(x_i) = \sum_{a=i}^{i+1} \sum_{b=j}^{j+1} \sum_{c=k}^{k+1} \bar{c}_{abc} \omega_{abc}
\]
Interpolation Weights

\[ \mathbf{x}_i = (x_i, y_i, z_i) \quad \mathbf{x}_{abc} = (x_{abc}, y_{abc}, z_{abc}) \]

\[ \omega^x_{abc} = \frac{\Delta x - |x_i - x_{abc}|}{\Delta x} \]

\[ \omega^y_{abc} = \frac{\Delta y - |y_i - y_{abc}|}{\Delta y} \]

\[ \omega^z_{abc} = \frac{\Delta z - |z_i - z_{abc}|}{\Delta z} \]

\[ \omega_{abc} = \omega^x_{abc} \omega^y_{abc} \omega^z_{abc} \]
Motion Tests with Particles
Motion Tests with Gridded Volumes
Light Color Transmission

- Lights modify this ray march procedure

- The color of the material is “multiplied” by the color of the light.

- The color of the light is attenuated by the volume material between the light and the ray march points.
Color Triplet Product

- Color triplet \( \vec{c} = (r, g, b) \)

- Color triplet \( \vec{F} = (F_r, F_g, F_b) \)

- Component-wise product is a color triplet

\[
\vec{c} \odot \vec{F} = (r F_r, g F_g, b F_b)
\]
Point Light

- Position of light: \( \mathbf{X}_l \)
- Light intensity in a vacuum: \( \vec{F} = (F_r, F_g, F_b) \)
- In volumetric medium, intensity depends on how much material density exists between the light and the ray march point.
Light Rays

- Same form of transmissivity as for ray march, but in the direction between the light and the ray march point.

\[ Q(x_c, \hat{n}, s, x_l) = \exp \left\{ -\kappa \int_0^D ds' \rho(x_c + \hat{n}s + \hat{N}s') \right\} \]

\[ D = |x_l - x_c - \hat{n}s| \]

\[ \hat{N} = \frac{x_l - x_c - \hat{n}s}{|x_l - x_c - \hat{n}s|} \]
Rendering Equation: Lights

\[
\vec{C}(x_c, \hat{n}) = \int_0^\infty ds \, \rho(x_c + \hat{n}s) \, (\vec{c}(x + \hat{n}s) \odot \vec{F}) \, \exp \left\{ -\kappa \int_0^s ds' \, \rho(x_c + \hat{n}s') \right\} \, Q(x_c, \hat{n}, s, x_l)
\]
Ray March with Lights

\[ x_i + = \hat{n} \Delta s \]

\[ \Delta T = \exp \{ -\kappa \Delta s \rho(x_i) \} \]

\[ T * = \Delta T \]

\[ \mathbf{C} + = \frac{1 - \Delta T}{\kappa} (\mathbf{C}(x_i) \odot \tilde{F}) \] \[ T \] \[ Q(x_c, \hat{n}, \Delta s \iota, x_l) \]
Precomputed Light Transmissivity

- Compute $Q$ to each voxel center
- Store $Q_{ijk}$ at each voxel.

$$Q_{ijk} = \exp \left\{ -\kappa \int_0^D ds' \rho(x_{ijk} + \hat{N}_{ijk}s') \right\}$$

$$\hat{N}_{ijk} = \frac{x_l - x_{ijk}}{|x_l - x_{ijk}|}$$

- Use trilinear interpolation for points off of voxels centers.
Full Ray March with Lights

\[ \mathbf{x}_i + = \hat{n} \Delta s \]

\[ \Delta T = \exp \{-\kappa \Delta s \, \rho(\mathbf{x}_i)\} \]

\[ T * = \Delta T \]

\[ \mathbf{C} + = \frac{1 - \Delta T}{\kappa} \left( \mathbf{c}(\mathbf{x}_i) \odot \mathbf{F} \right) T \mathcal{Q}(\mathbf{x}_i) \]
Some methods to fill volumes

- Levelsets (implicit functions) of geometry
- Pyroclastic voxels
- Antialiased point “baking”
- Wisps
- Issues with grid memory usage
Implicit Functions

- Implicit functions define a surface geometry implicitly

\[ f(x) = 0 \]

- Examples:
  - sphere
    \[ 1 - \frac{|x|^2}{r^2} = 0 \]
  - torus
    \[ 4R^2(x^2 + z^2) - (|x|^2 + R^2 - r^2)^2 = 0 \]
  - cone
    \[ \cos^{-1} \left( \frac{x \cdot \hat{a}}{|x|} \right) - \theta = 0 \]
One type of implicit function is a levelset: the function is defined at values sampled on a grid, along with interpolation.

Geometry can be converted to levelsets via the “Fast Marching Method”. The levelset is a signed distance function.

Introduction to Implicit Surfaces
Density from Implicit Function

\[
\rho(x) = \begin{cases} 
  \frac{f(x)}{f_{\text{max}}} & f(x) > 0 \text{ (inside)} \\
  0 & f(x) \leq 0 \text{ (outside)} 
\end{cases}
\]
Sphere (1998)
F-Rep Implicit Functions

The Making of Black-Hole and Nebula Clouds for the Motion Picture "Sphere" with Volumetric Rendering and the F-Rep of Solids,
Gokhan Kisacikoglu, Siggraph 1998
Noises

- Many types of noise are employed to generate volumes
  - Pseudo random number generators
  - Perlin noise
  - Perlin noise with octaves
- Quick introduction to them
Pseudo random number generators

- Functions that produce a sequence of numbers that are statistically independent and effectively random.

- The sequence is not truly random, but passes various statistical tests of randomness.

- Controllable via a seed parameter so that you can repeatedly start sampling the sequence at a known place.
rand() generates a “random” number between 0 and RAND_MAX. The algorithm has noticeable patterns in the sequence and repeats after around $2^{31}$ values.
drand48()

- Produces a sequence of values between 0 and 1.
- Higher quality than rand() - fewer patterns in the sequence
- Longer sequence - repeats after about $2^{48}$ values.
Mersenne Twister

- Produces a sequence between 0 and 1
- Extremely high quality
- HUGE sequence length - repeats after $2^{19937}-1$ values.
Perlin Noise

- A procedural texture with a random appearance
- Produces a spatial pattern in 1, 2, 3, or 4 dimensions.
- See Wikipedia for details and code.

*Textures & Modeling: A Procedural Approach*,
Ebert, Musgrave, Peachy, Perlin, & Worley

code: [http://cobweb.ecn.purdue.edu/~ebertd纹理](http://cobweb.ecn.purdue.edu/~ebertd/texture/)

Perlin(x,y)
Perlin Noise with Octaves
(fractal Brownian motion - fBm)

- Fractal sum scaling of multiple copies of Perlin noise.
- Control noise appearance via amplitude $A$ and scale $f$.

\[
fBm(x) = \sum_{\ell=1}^{L} A^{\ell-1} \text{Perlin}(x \ f^{\ell})
\]
Pyroclastic Puff

- Implicit function for a sphere:
  \[ 1 - \frac{|x - x_s|^2}{r^2} \]

- Use fBm of perlin noise to displace boundary

\[ |fBm\left(\frac{x - x_s}{|x - x_s|}\right)| + a - \frac{|x - x_s|^2}{r^2} \]

- Update density of each voxel inside this implicit function
Baking anti-aliased dots

- Some algorithms generate tiny dots of density & color
- Bake many of them into a grid one by one.
- Since they are tiny, baking has to be done with anti-aliasing, smearing dot across eight neighboring voxels.
- This is a very powerful & flexible technique
\[\dot{X} = \exp\left\{-\kappa \int_{D} \rho(x_{ijk} + \hat{N}_{ijk}s')\right\}\]

\[\hat{N}_{ijk} = x_{l} - x_{ijk}\]

\[f_{Bm}(x) = L \sum_{\ell=1}^{\infty} A_{\ell} - 1 \perlin(x_{f}\ell)\]

\[Q_{ijk} = \frac{1}{2}\left(x_{ijk} - x\right)\cdot \frac{1}{r}\]

\[\rho(x) = \begin{cases} 1 - \frac{1}{2} \frac{1}{r}^2 & \text{if } 0 < \frac{1}{2} \frac{1}{r}^2 \\ 0 & \text{if } \frac{1}{2} \frac{1}{r}^2 \leq 0 \end{cases}\]

\[\dot{X}, \rho, \dot{\rho}, \dot{C}, \dot{\vec{c}}\]
\[ Q_{ijk} = \exp \left\{ -\kappa \int_{D_0} \rho(x_{ijk} + \hat{N}_{ijk}s') ds' \right\} \]

\[ \hat{N}_{ijk} = x_l - x_{ijk} \]

\[ f_{Bm}(x) = \sum_{\ell=1}^{L} A_{\ell} - 1 \]

\[ f_{Bm}(x) = \sum_{\ell=1}^{L} A_{\ell} - 1 \]

\[ \|x - x_s\|^2 \]

\[ \|x - x_s\|^2 \]

\[ \|\vec{c}(x) = (1, 1, 1)\| \]
Bake dot by updating voxels

- Dot located at $\mathbf{x}_{\text{dot}}$ with density $\rho_{\text{dot}}$ and color $\mathbf{c}_{\text{dot}}$

- 8 nearest voxels are
  
  $ijk, i+1jk, ij+1k, ijk+1, i+1j+1k, i+1jk+1, ij+1k+1, i+1j+1k+1$

- Use trilinear interpolation weights $\omega_{abc}$

- Update density & color at the 8 nearest voxels
  
  $\rho_{abc} + = \rho_{\text{dot}} \omega_{abc}$

  $\mathbf{c}_{abc} + = \mathbf{c}_{\text{dot}} \omega_{abc}$
Steps to Grow Point Wisps
Steps to Grow Point Wisps

1. Distribute points randomly in space around the guide point
   - correlated random walk
2. Hundreds or thousands of points
Steps to Grow Point Wisps

- Distribute points randomly in space around the guide point
  - correlated random walk
    - hundreds or thousands of points
- Move them to the unit sphere
Steps to Grow Point Wisps

- Distribute points randomly in space around the guide point
  - correlated random walk
  - hundreds or thousands of points
- Move them to the unit sphere
- Use fractal perlin noise to displace radially from unit sphere
Steps to Grow Point Wisps

- Distribute points randomly in space around the guide point
  - correlated random walk
  - hundreds or thousands of points
- Move them to the unit sphere
- Use fractal perlin noise to displace radially from unit sphere
- Use vector fractal perlin noise to displace in 3D
Steps to Grow Point Wisps

- Distribute points randomly in space around the guide point
  - correlated random walk
  - hundreds or thousands of points
- Move them to the unit sphere
- Use fractal perlin noise to displace radially from unit sphere
- Use vector fractal perlin noise to displace in 3D
- "Bake" to a voxel grid as antialiased dots
- Smear dots to emulate motion blur
Wisp algorithm 0:
Set up a Guide Particle Position & Size

( xparticle, yparticle, zparticle ),
particle_size,
particle_density,
particle_color
Wisp algorithm 1:
Generate a new position for a dot

// random position between -1 and 1
x = 2.0*drand48() - 1;
y = 2.0*drand48() - 1;
z = 2.0*drand48() - 1;
Wisp algorithm 2: Move to Unit Sphere

// move position to unit sphere
radius = sqrt( x*x + y*y + z*z );
xsphere = x/radius;
ysphere = y/radius;
zsphere = z/radius;
Wisp algorithm 3:
Displace Radially from Sphere

// displace radially from sphere using fractal sum
radial_disp = pow(fabs(fBm( x, y, z )), clump );
xsphere *= radial_disp;
ysphere *= radial Disp;
zsphere *= radial Disp;
Wisp algorithm 4:
Map to Guide Particle Coordinates

// map to guide particle coordinate
xdot = xparticle + xsphere * particle_size;
ydot = yparticle + ysphere * particle_size;
zdot = zparticle + zsphere * particle_size;
Wisp algorithm 5:
Displace by vector noise

// displace again with 3D fractal sum noise
xfsn = fBm( xsphere, ysphere, zsphere );
yfsn = fBm( xsphere + 0.1, ysphere + 0.1, zsphere + 0.1 );
zfsn = fBm( xsphere - 0.1, ysphere - 0.1, zsphere - 0.1 );

xdot += xfsn;
ydot += yfsn;
zdot += zfsn;
Wisp algorithm 6: Bake and Repeat

• Bake particle_color and particle_density for anti-aliased dot at (xdot, ydot, zdot)

• Repeat at step 1 for another dot.

• When you have enough dots for this guide particle, repeat entire process for another guide particle.
for( loop over particles ){
    // set xparticle, yparticle, zparticle
    // set particle_size, particle_color, particle_density
    for( loop over dots for this particle ){
        // random position between -1 and 1
        x = 2.0*drand48() - 1;
        y = 2.0*drand48() - 1;
        z = 2.0*drand48() - 1;

        // move position to unit sphere
        radius = sqrt( x*x + y*y + z*z );
        xsphere = x/radius;
        ysphere = y/radius;
        zsphere = z/radius;

        // displace radially from sphere using fractal sum
        radial_disp = pow(fabs(fBm( x, y, z )), clump );
        xsphere *= radial_disp;
        ysphere *= radial_disp;
        zsphere *= radial_disp;

        // map to guide particle coordinate
        xdot = xparticle + xsphere * particle_size;
        ydot = yparticle + ysphere * particle_size;
        zdot = zparticle + zsphere * particle_size;

        // displace again with 3D fractal sum noise
        xfsn = fBm( xsphere, ysphere, zsphere );
        yfsn = fBm( xsphere + 0.1, ysphere + 0.1, zsphere + 0.1 );
        zfsn = fBm( xsphere - 0.1, ysphere - 0.1, zsphere - 0.1 );
        xdot += xfsn;
        ydot += yfsn;
        zdot += zfsn;

        // Now ready to bake a dot into the volume at (xdot, ydot, zdot)
        BakeDot( xdot, ydot, zdot, particle_density, particle_color );
    }
}
Guide particles
Avalanche Sequence
The Mummy: Tomb of the Dragon Emperor
Memory Issues

- Avalanche Grid Dimensions: 1 mile X 1 mile X 1 mile
- Resolution: 6 inches
- Grid size: over 40,000 X 40,000 X 40,000
- Implied memory for Avalanche grid: > 200 TB / frame
Memory Solution

- 16 bit floats are usually sufficient for density
- There is a lot of empty space in the avalanche
- Do not allocate memory to voxels that are empty
- Actual memory for Avalanche: around 200 MB/frame