Production Volume Rendering

Jerry Tessendorf

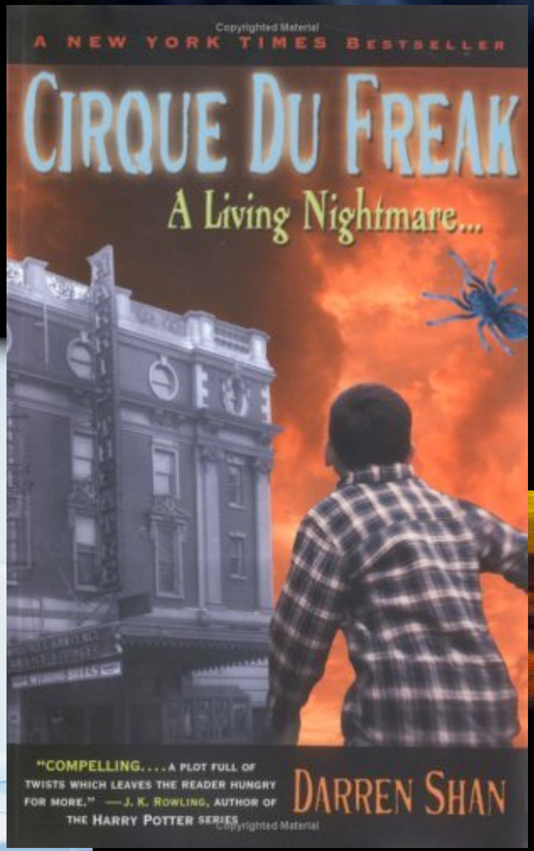
Principal Graphics Scientist

Rhythm and Hues Studios

jerryt@rhythm.com

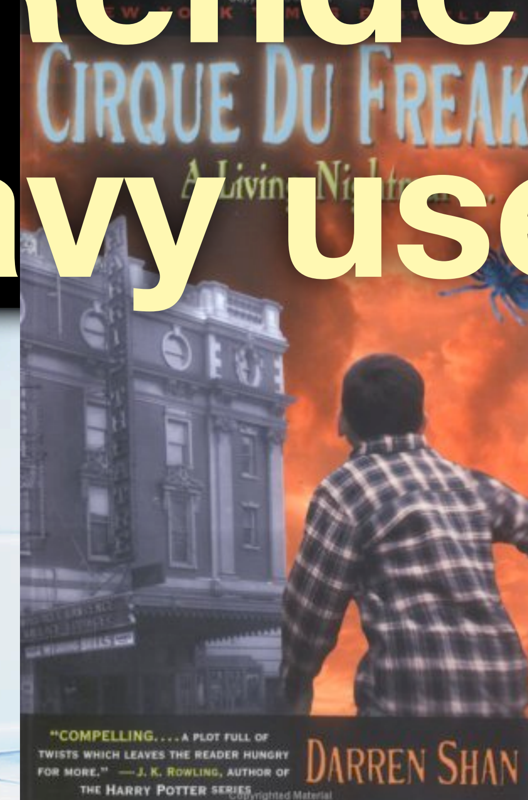
November 2008

In Production



In Production

Volume Rendering in heavy use



Rhythm & Hues Studios

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

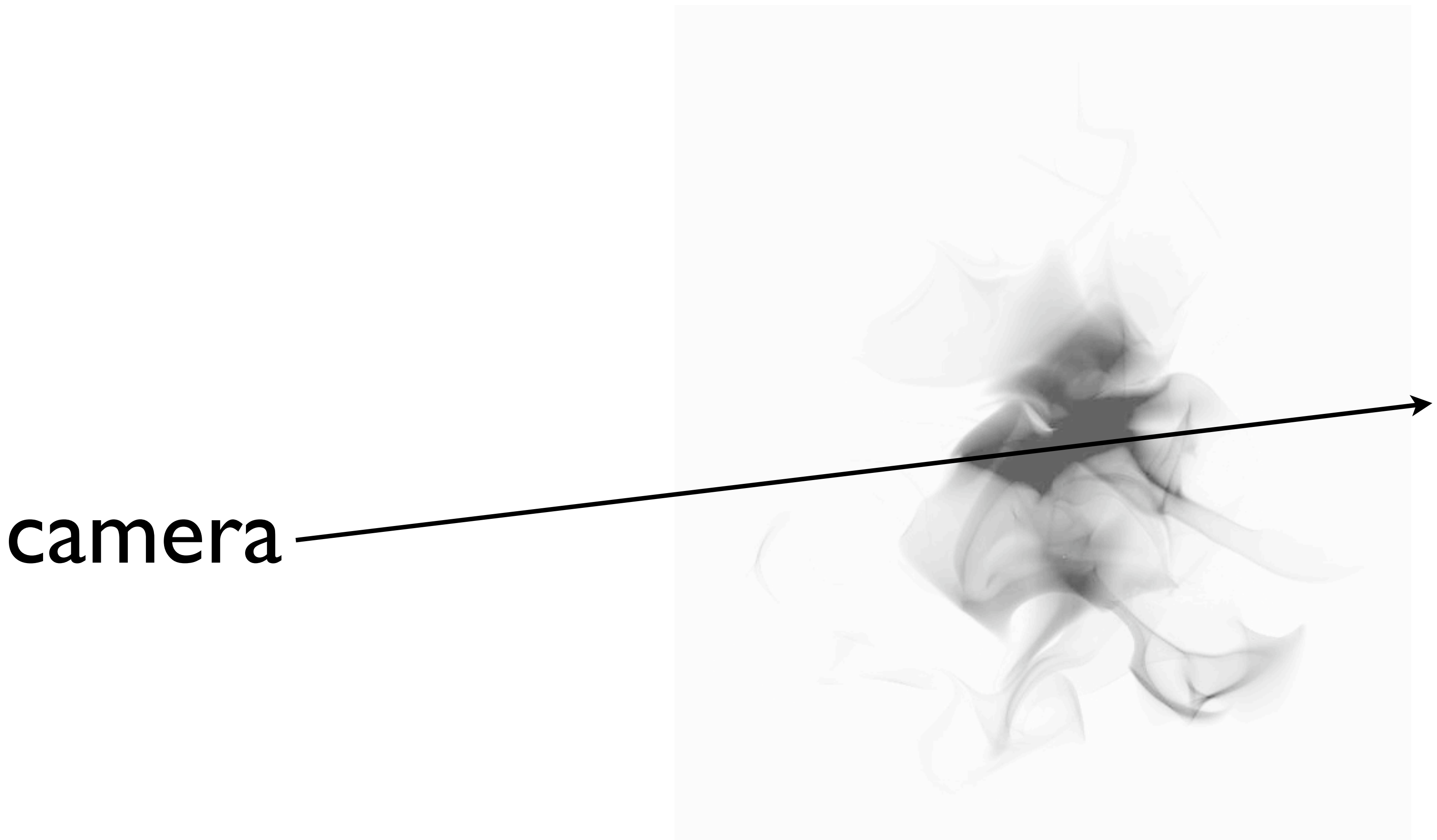
- Accumulate opacity along light of sight.
- Accumulate color along line of sight, weighted by accumulated opacity and light source.

camera

Rhythm&Hues STUDIOS

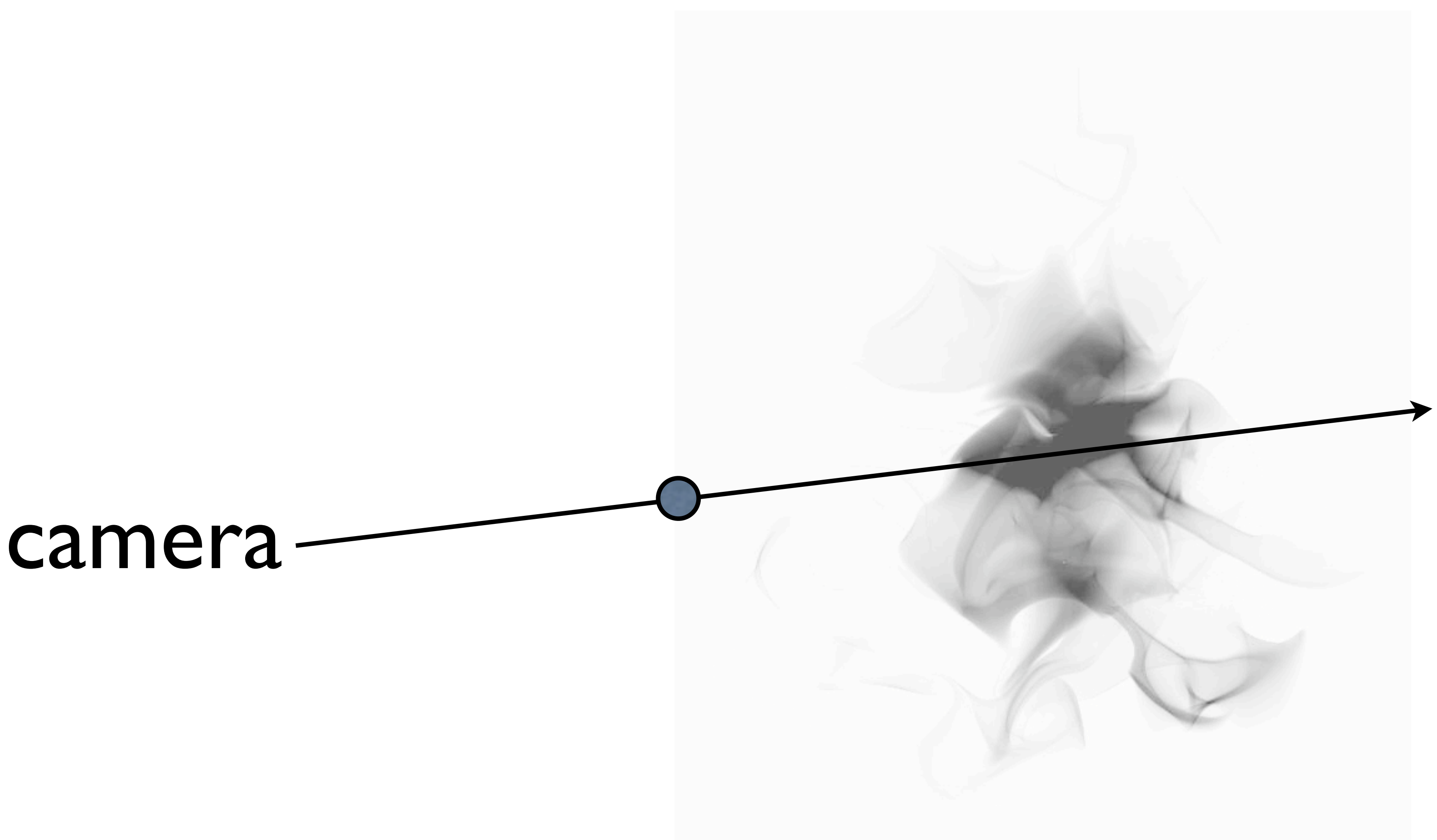
Volume Rendering

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



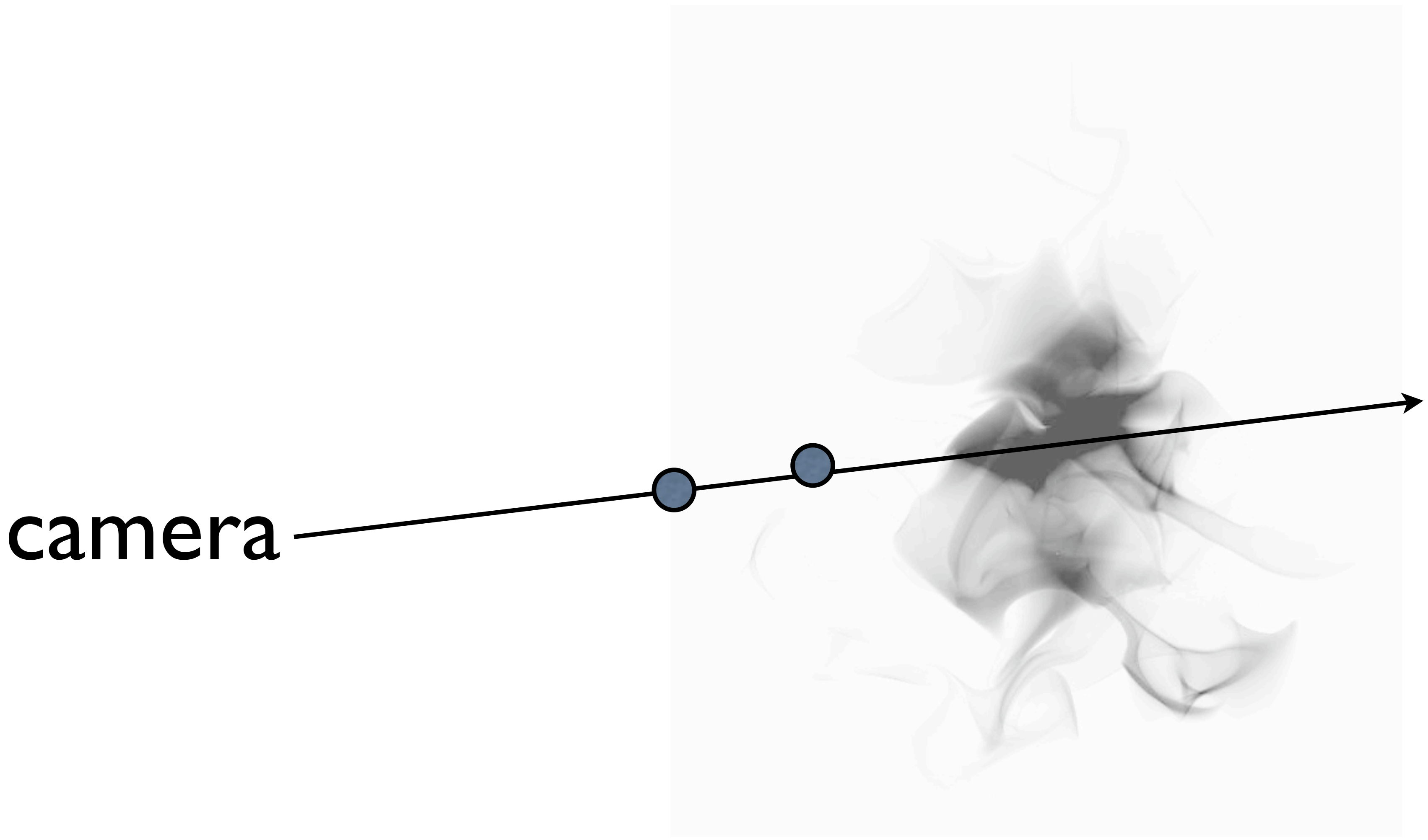
Volume Rendering

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



Volume Rendering

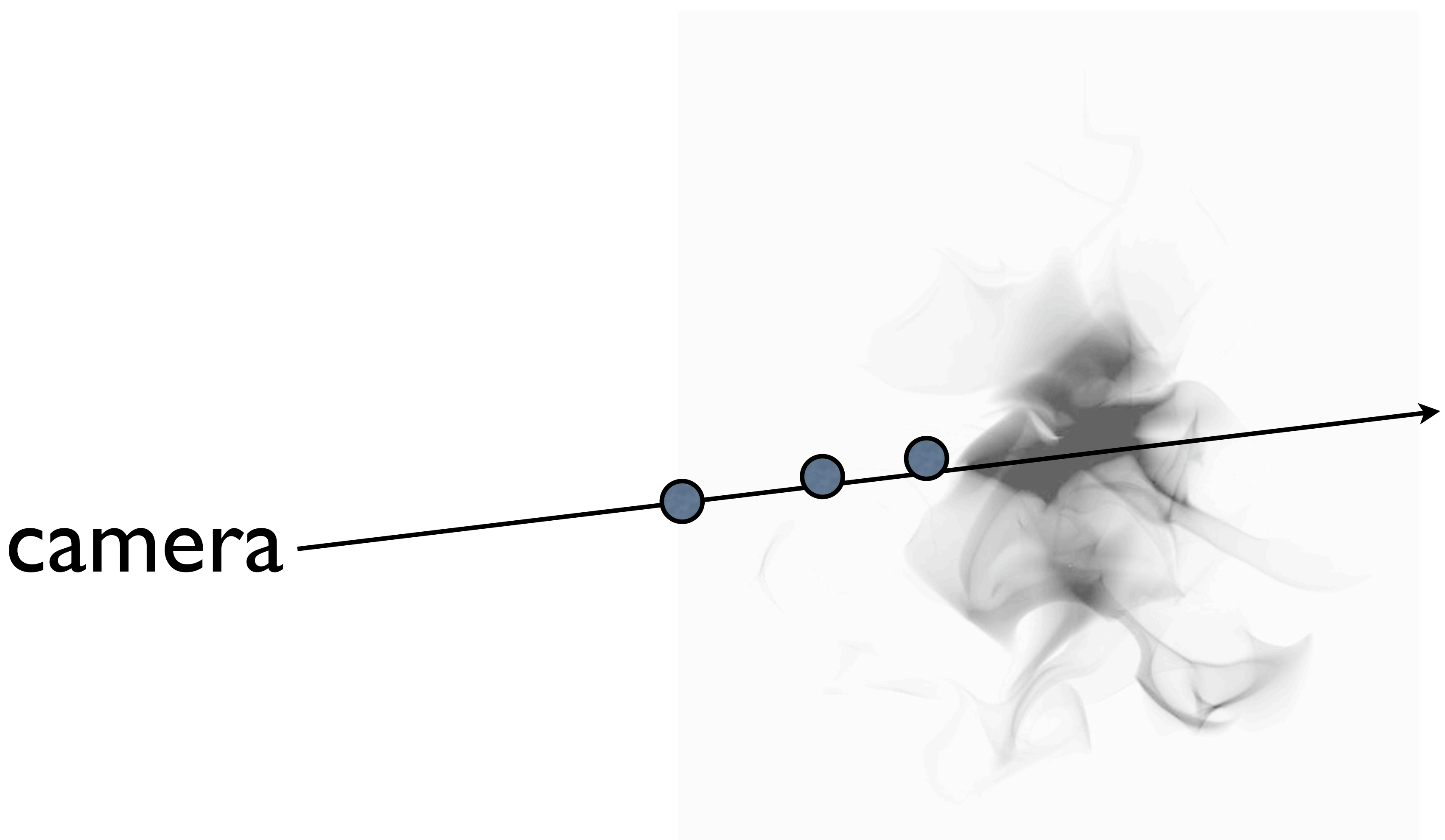
- Accumulate opacity along light of sight.
- Accumulate color along line of sight, weighted by accumulated opacity and light source.



Rhythm&Hues STUDIOS

Volume Rendering

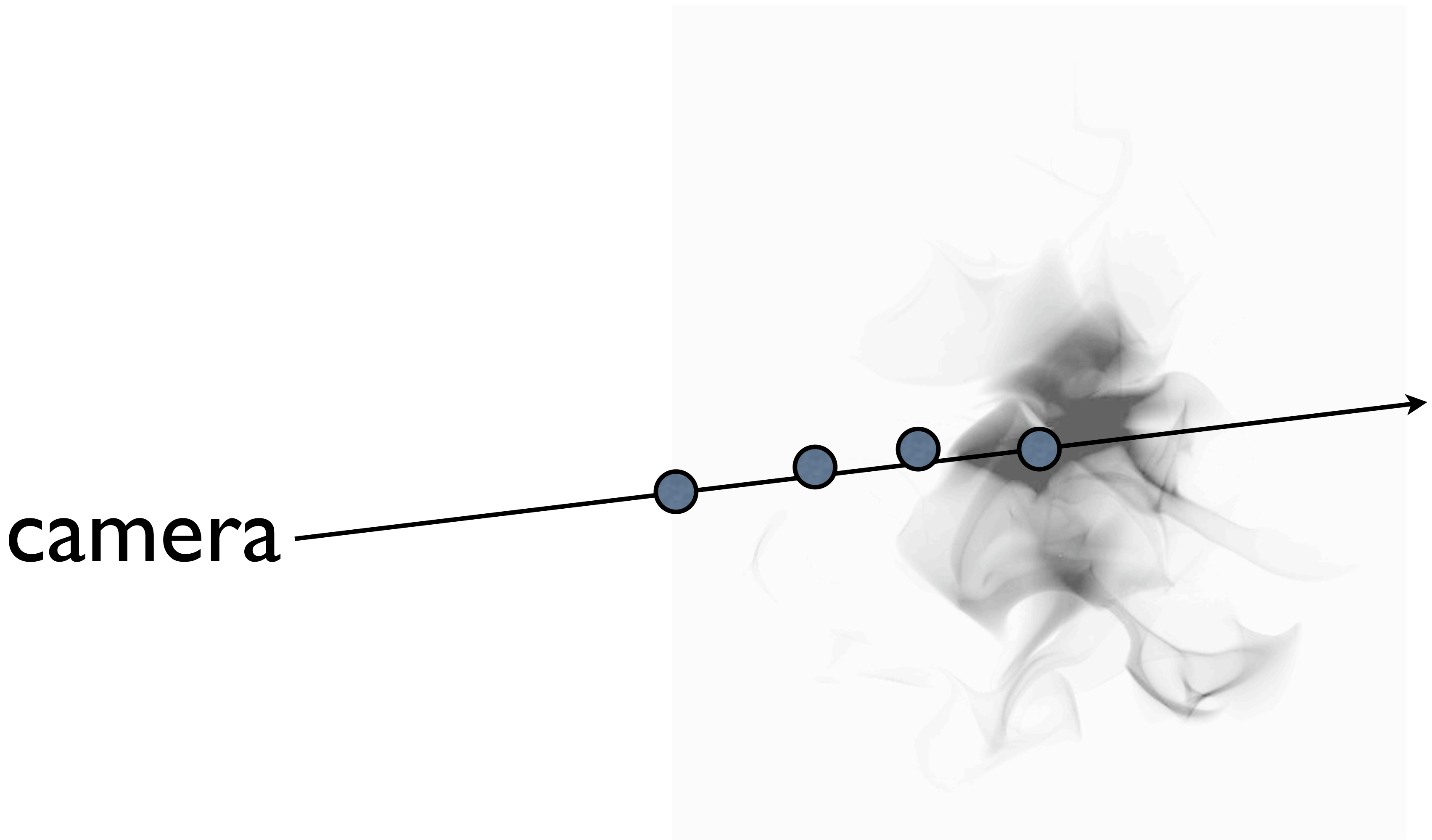
- Accumulate opacity along light of sight.
- Accumulate color along line of sight, weighted by accumulated opacity and light source.



Rhythm&Hues STUDIOS

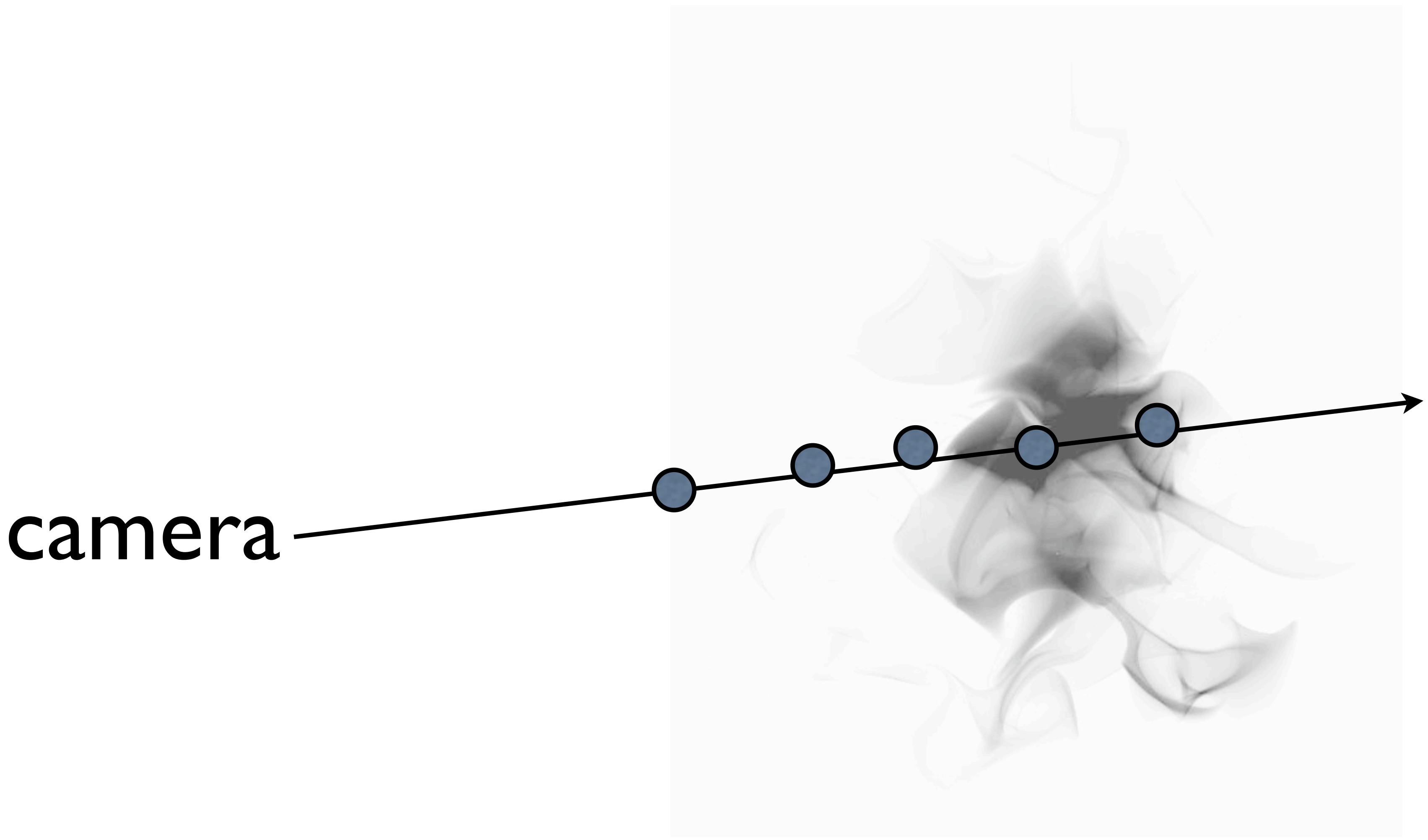
Volume Rendering

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



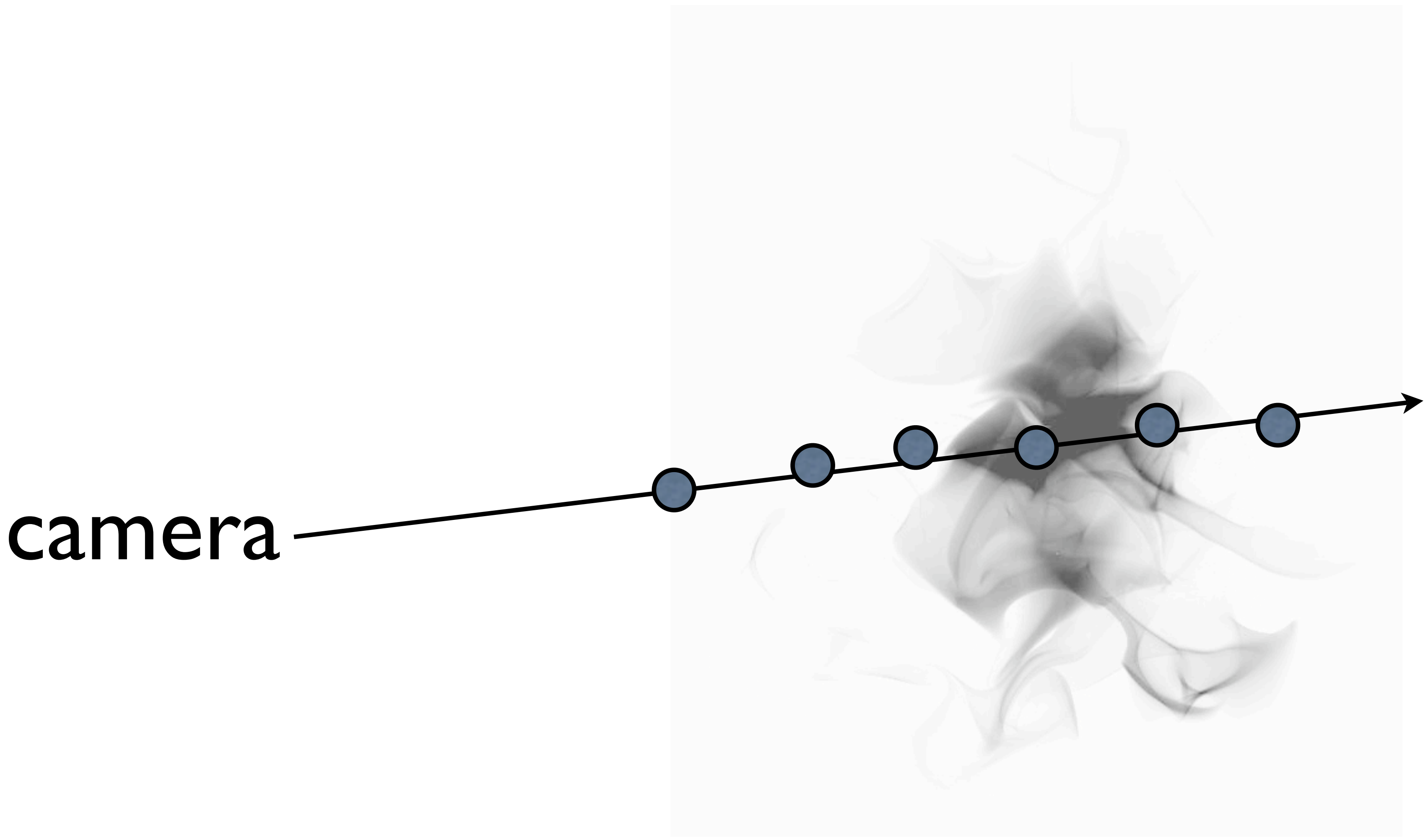
Volume Rendering

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



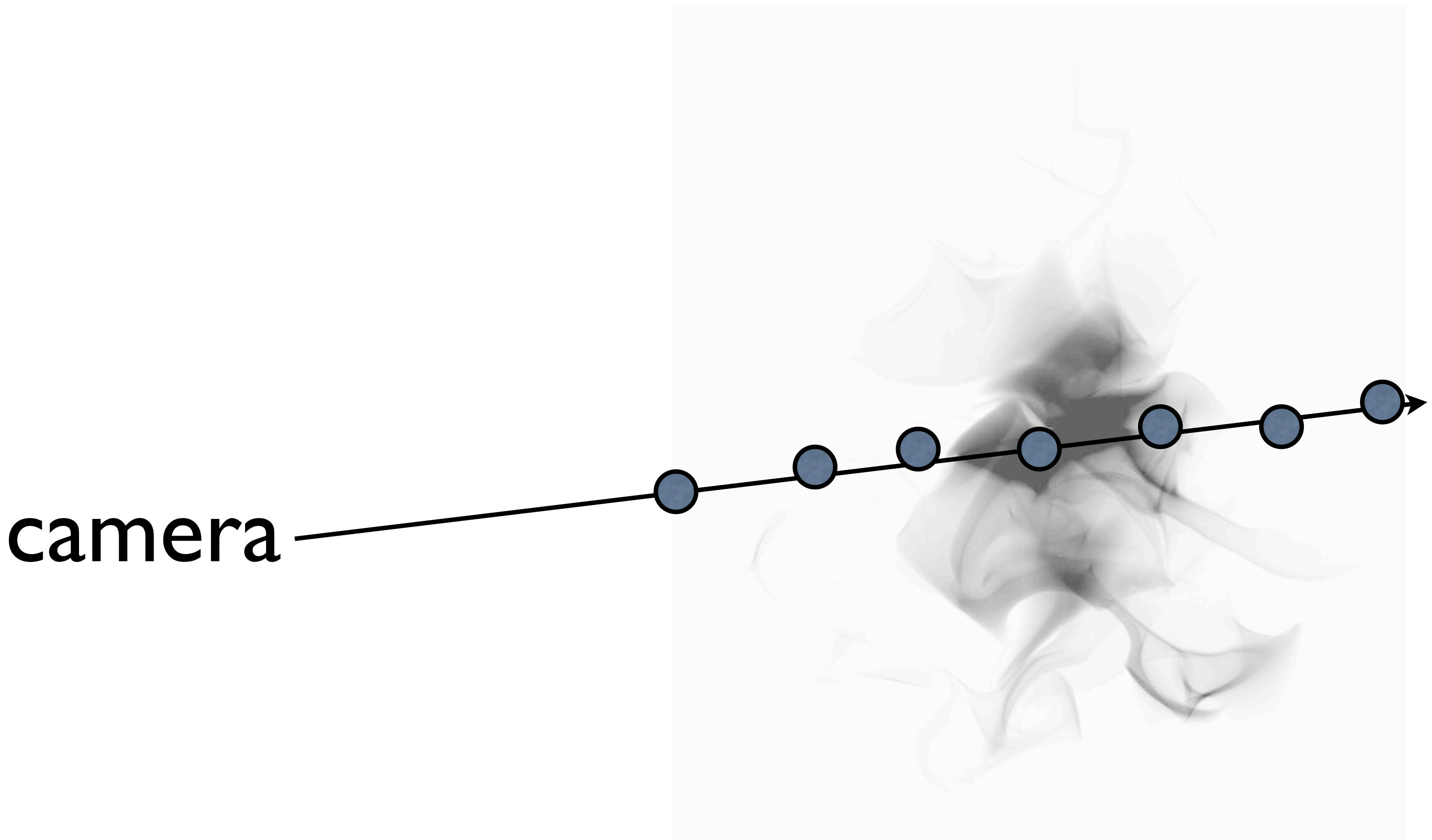
Volume Rendering

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



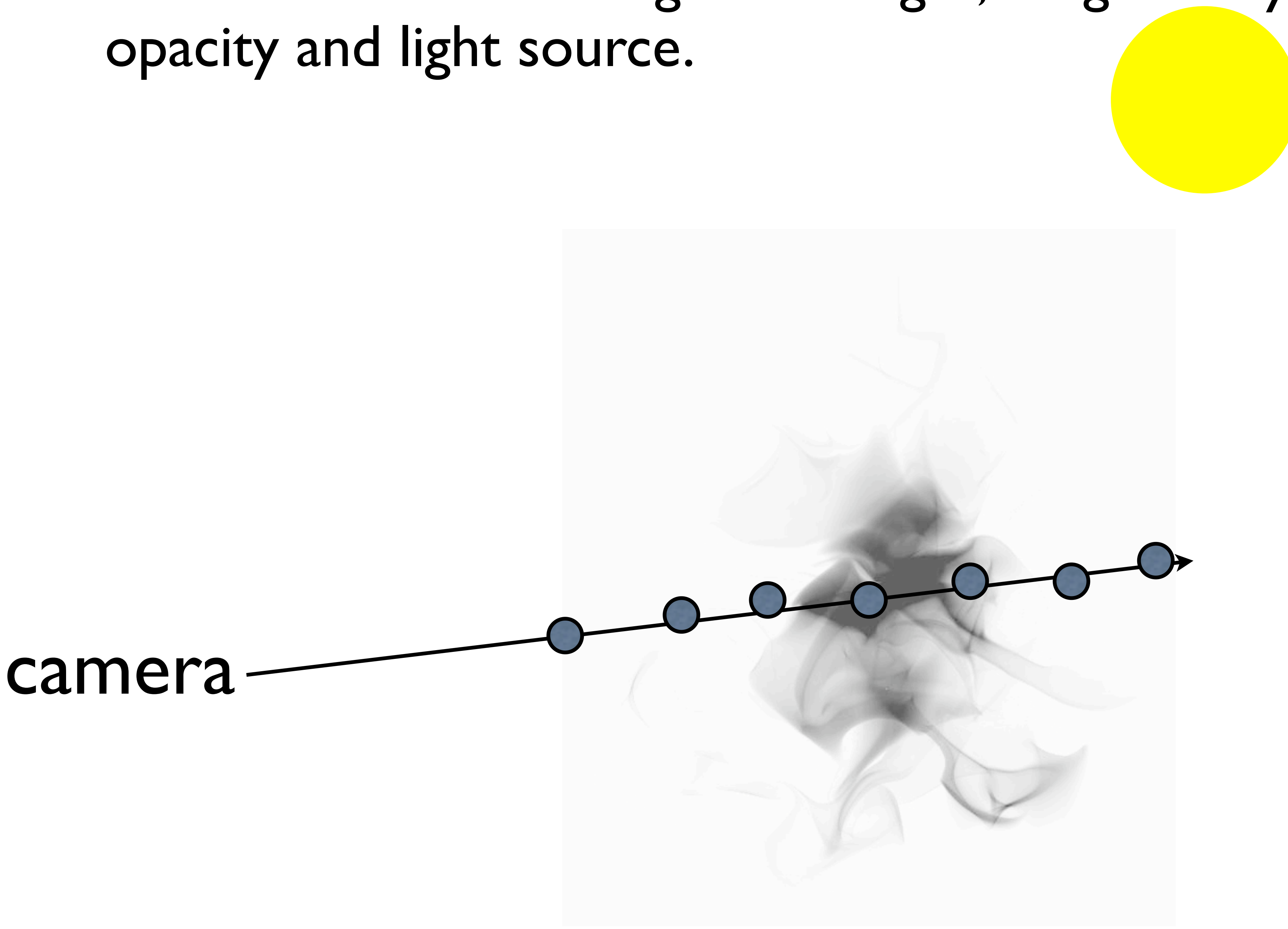
Volume Rendering

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



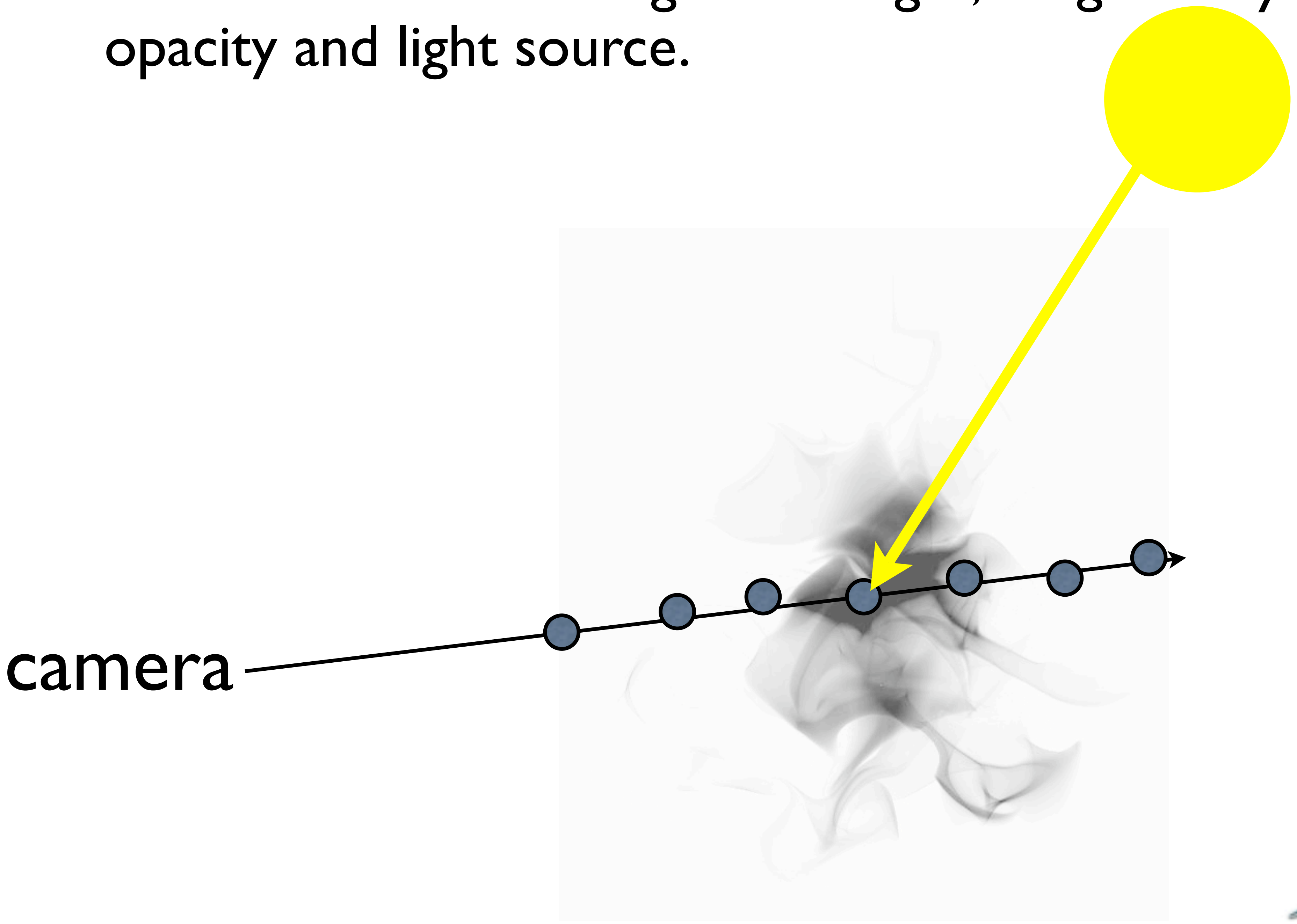
Volume Rendering

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



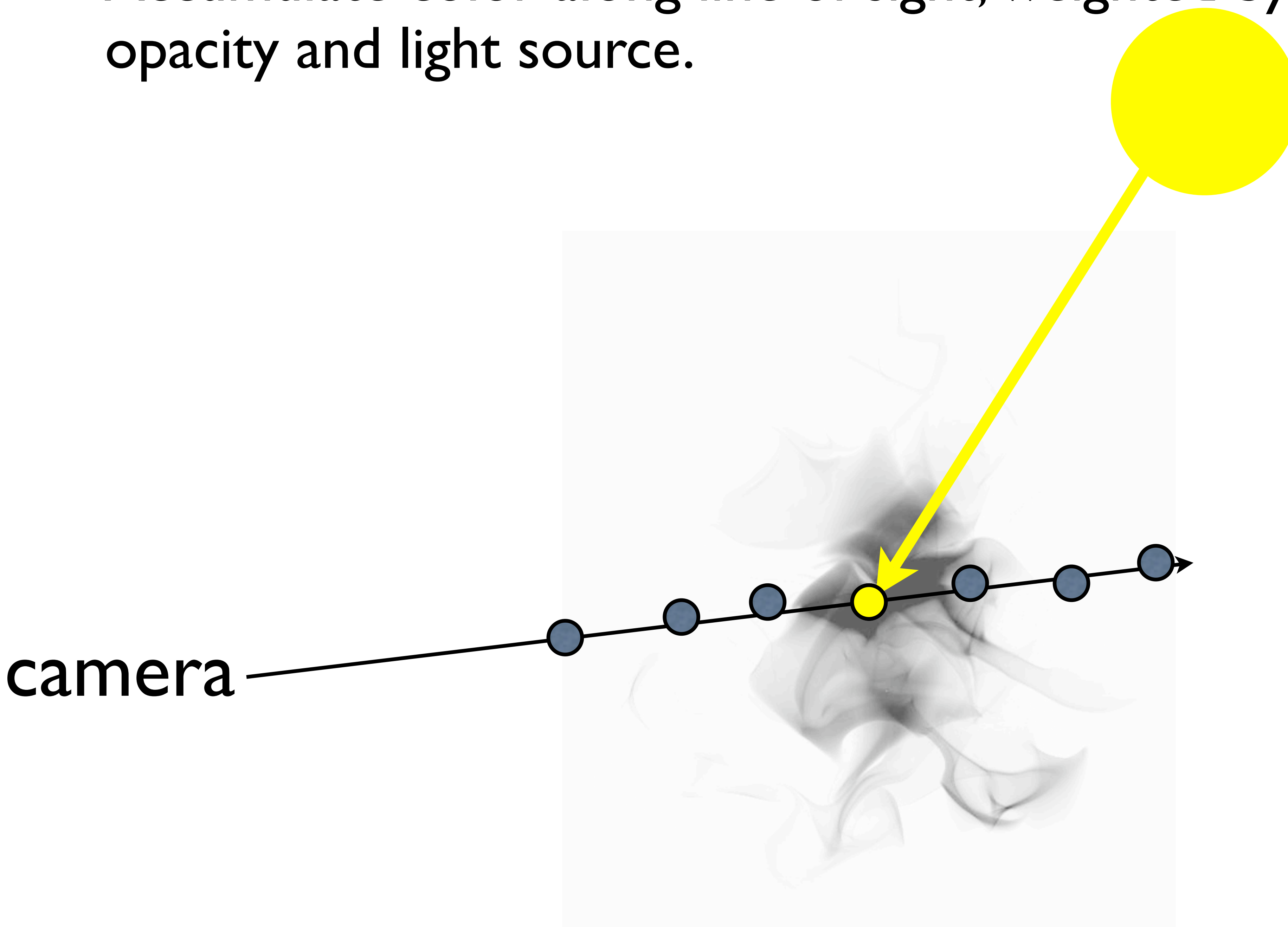
Volume Rendering

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



Volume Rendering

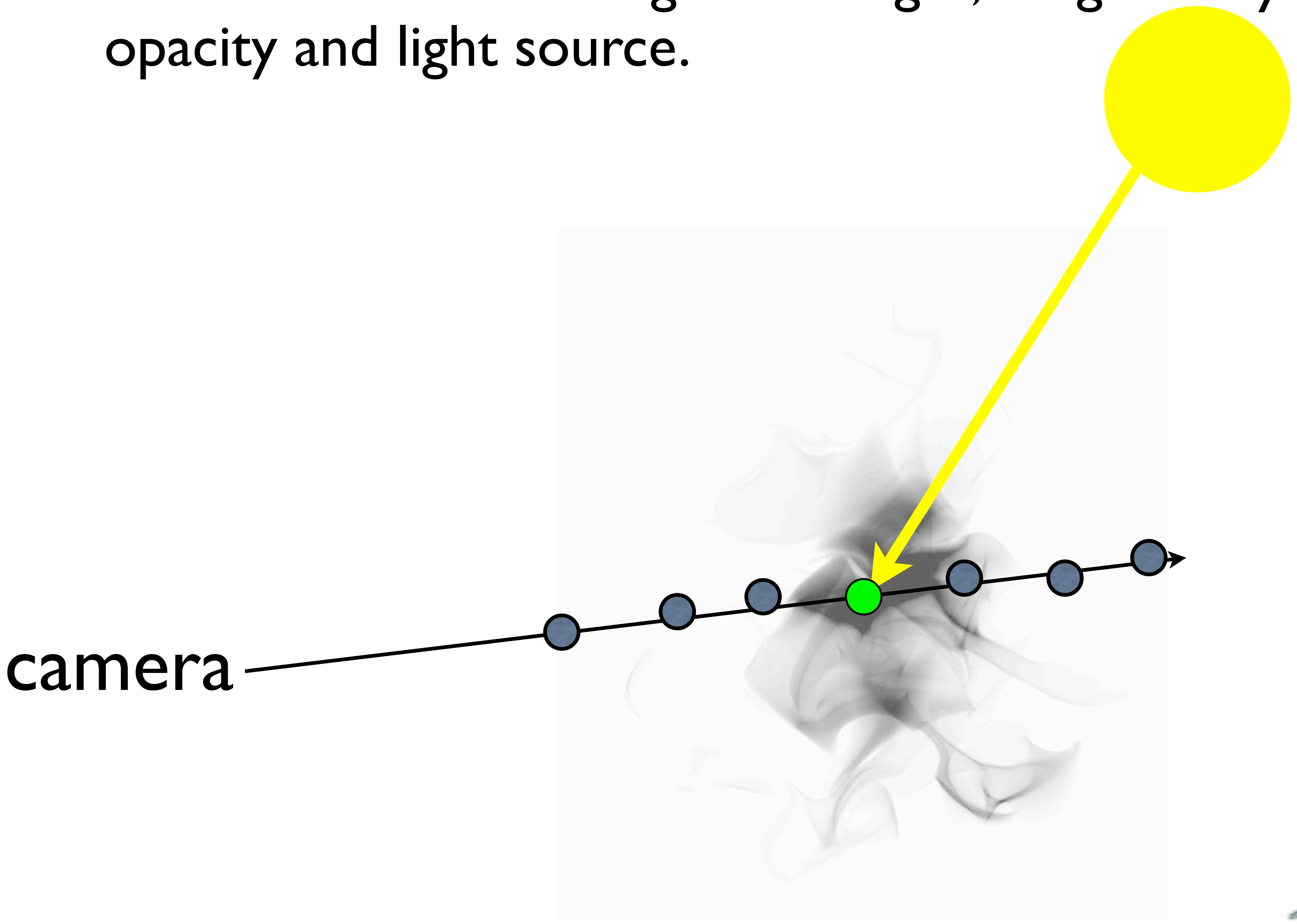
- Accumulate opacity along light of sight.
- Accumulate color along line of sight, weighted by accumulated opacity and light source.



Rhythm&Hues STUDIOS

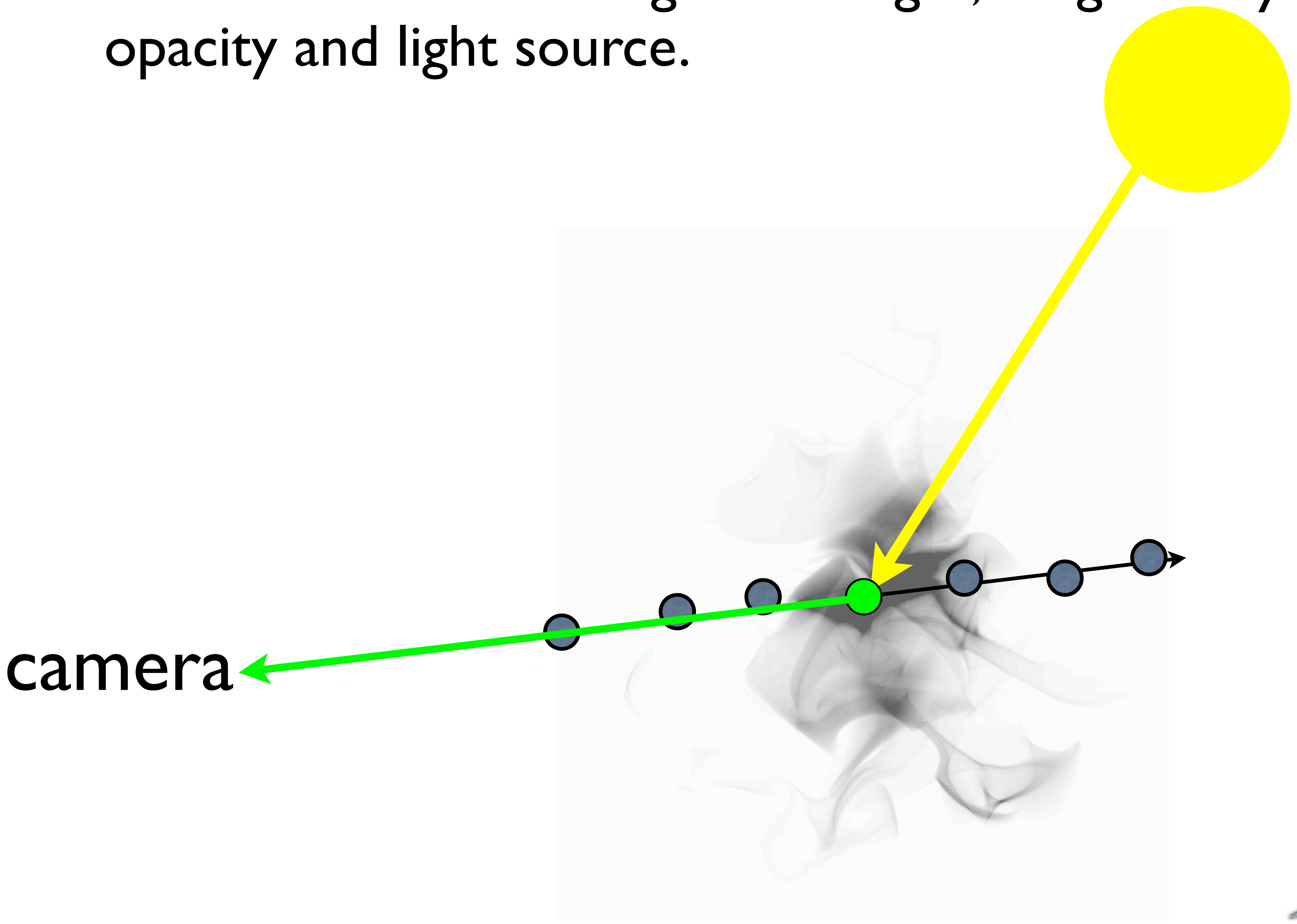
Volume Rendering

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



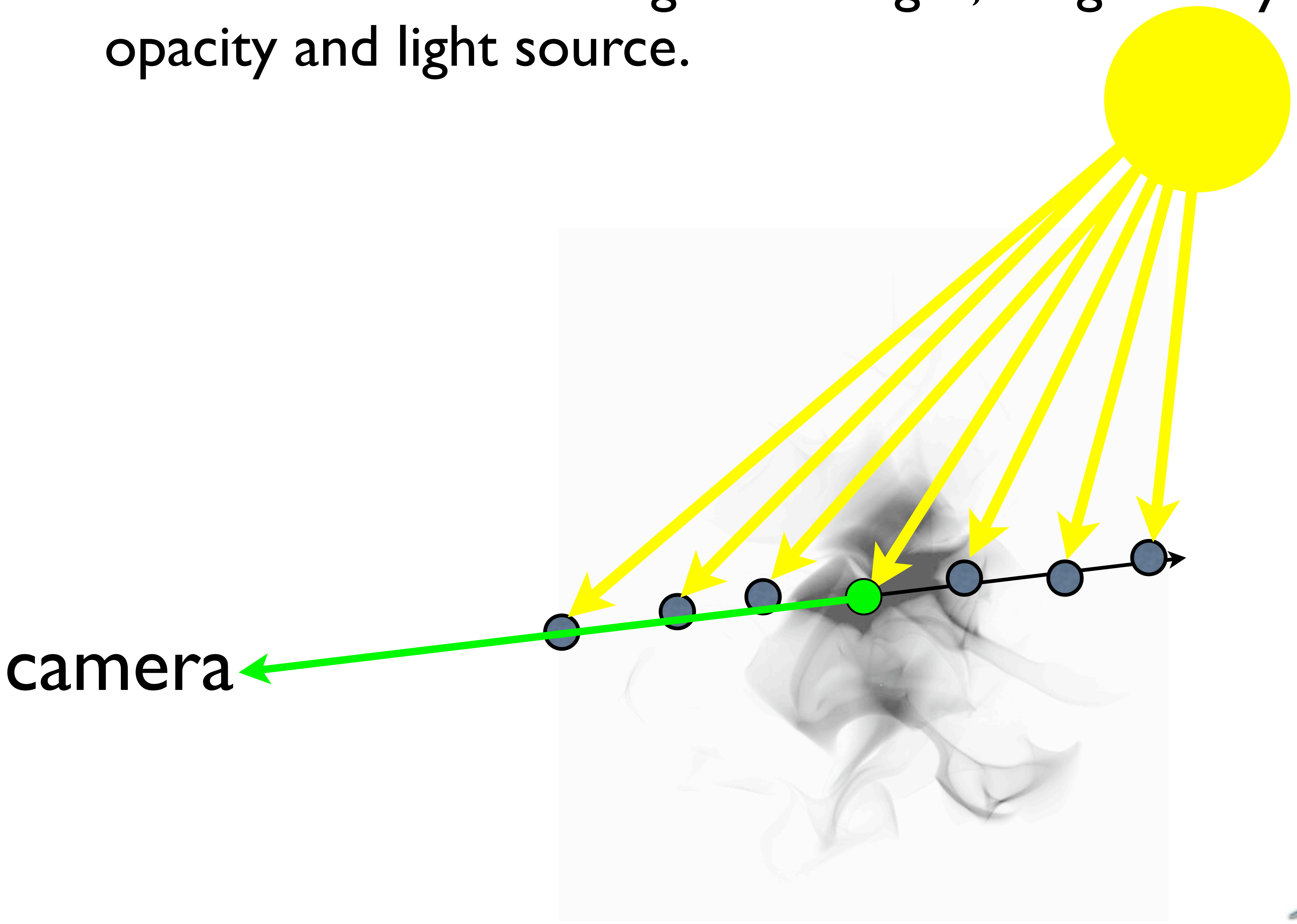
Volume Rendering

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



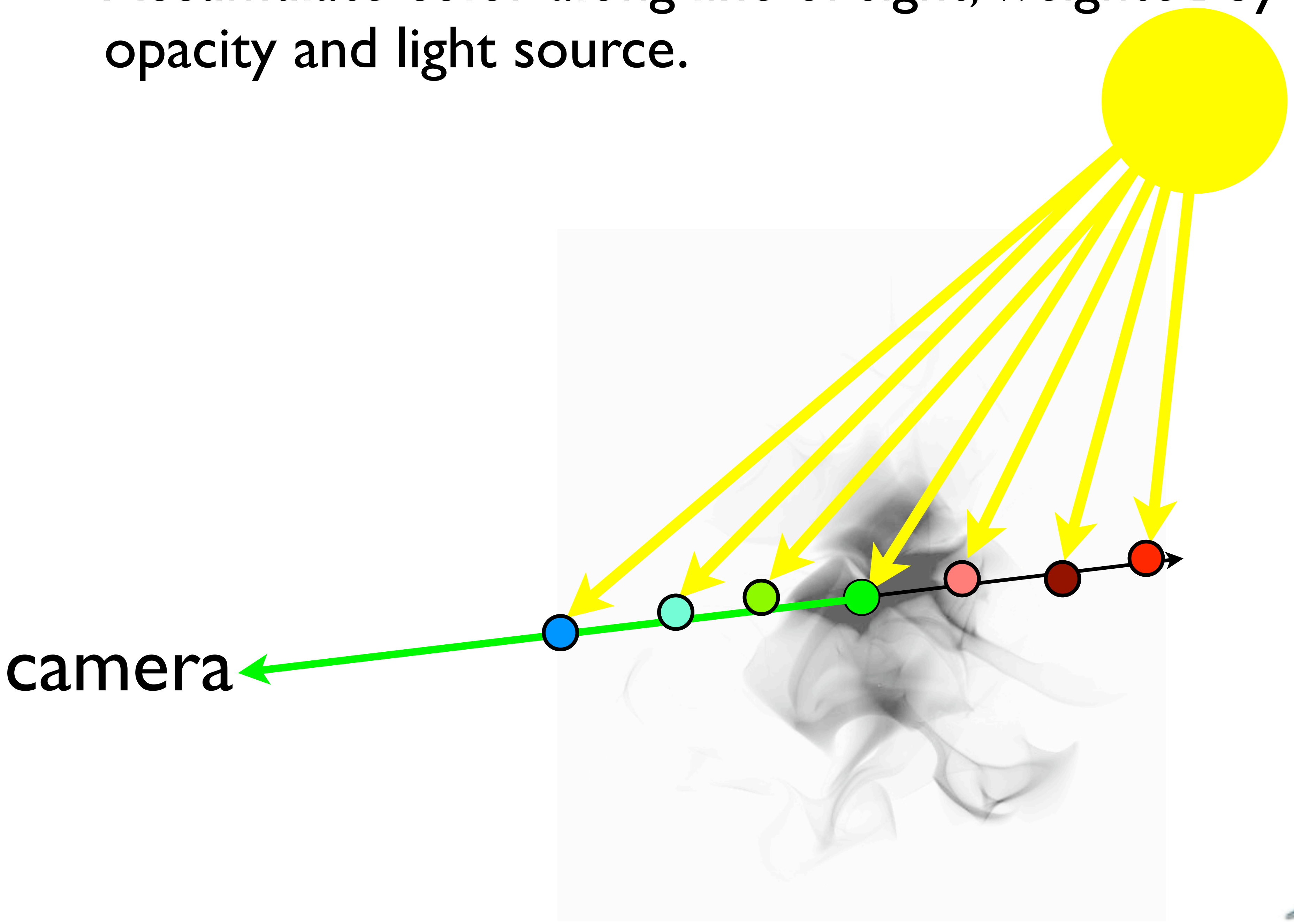
Volume Rendering

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



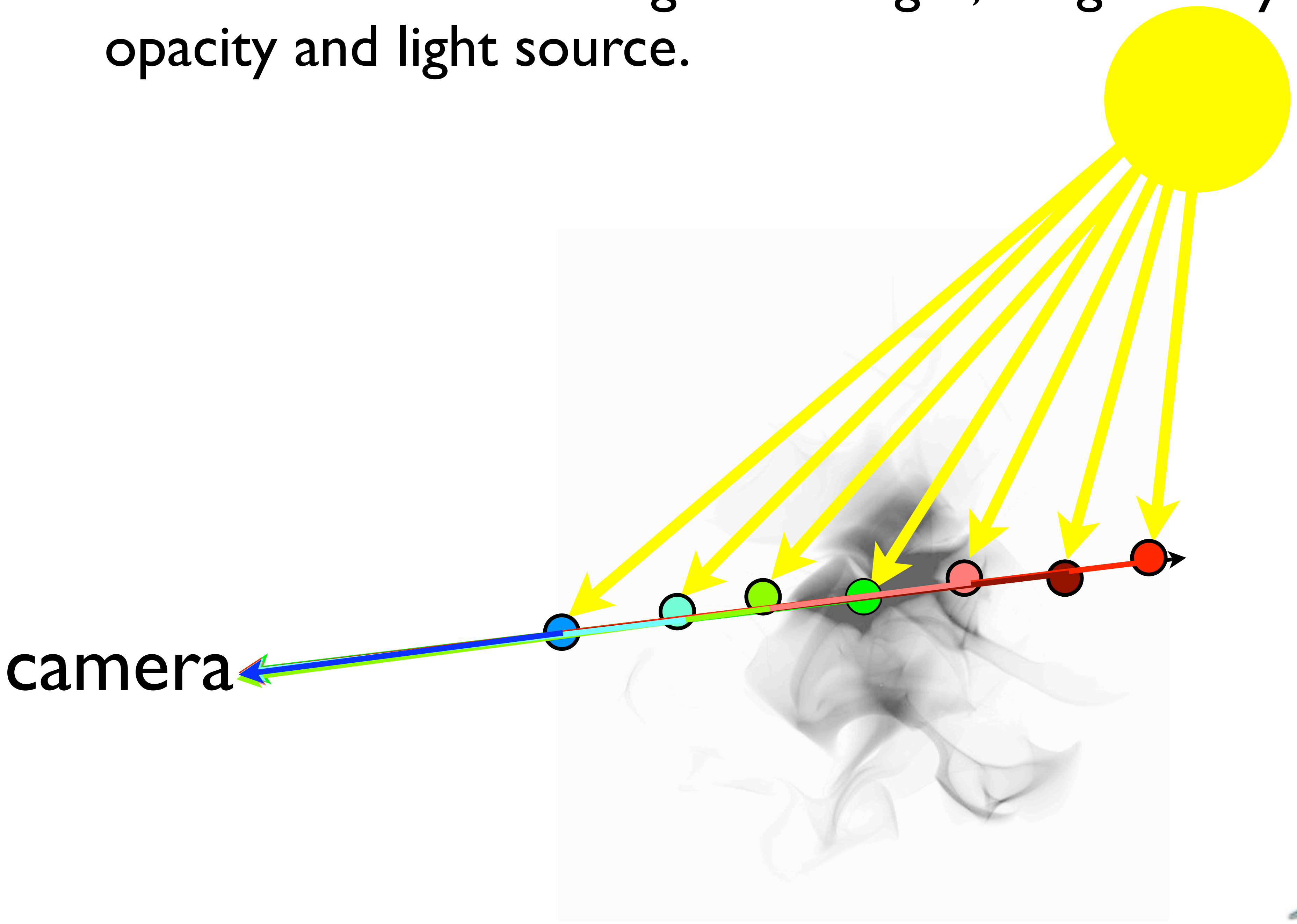
Volume Rendering

- 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

$$\rho(\mathbf{x}) = \begin{cases} > 0 & \text{inside material} \\ 0 & \text{everywhere else} \end{cases}$$

- Material color is a triplet as a function of position

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

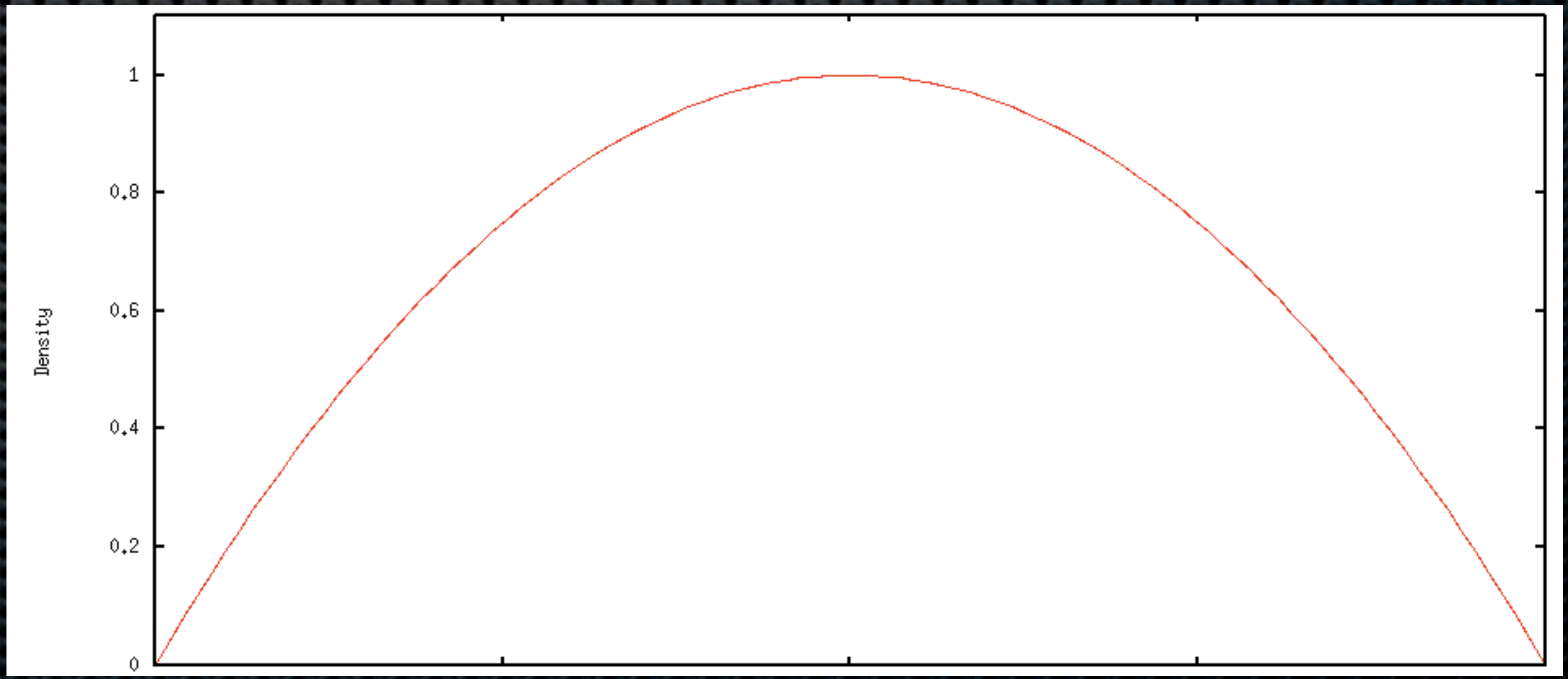
- Density and color are the fundamental inputs

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 $\mathbf{x}_c$
- Pixel looks in direction $\hat{\mathbf{n}}$
- Pixel sees color accumulated along the line

$$\mathbf{x}_c + \hat{\mathbf{n}}\, s$$

- Accumulated color $\vec{C}(\mathbf{x}_c, \hat{\mathbf{n}})$ has mathematical form

$$\vec{C}(\mathbf{x}_c, \hat{\mathbf{n}}) = \int_0^\infty ds\, \vec{G}(\mathbf{x}_c, \hat{\mathbf{n}}, s)$$

Proportional to material color and density

$$\vec{G}(\mathbf{x}_c, \hat{\mathbf{n}}, s) = \vec{c}(\mathbf{x}_c + \hat{\mathbf{n}}s)\ \rho(\mathbf{x}_c + \hat{\mathbf{n}}s)\ T(\mathbf{x}_c, \hat{\mathbf{n}}, s)$$

Transmissivity & Opacity

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

$$T(\mathbf{x}_c, \hat{\mathbf{n}}, s) = \exp\left\{-\kappa \int_0^s ds' \; \rho(\mathbf{x}_c + \hat{\mathbf{n}} s')\right\}$$

- Opacity is the complement of transmissivity

$$O(\mathbf{x}_c, \hat{\mathbf{n}}, s) = 1 - T(\mathbf{x}_c, \hat{\mathbf{n}}, s)$$

Rendering Equation: No Lights

$$\vec{C}(\mathbf{x}_c, \hat{\mathbf{n}}) = \int_0^\infty ds\ \rho(\mathbf{x}_c + \hat{\mathbf{n}}s)\ \vec{c}(\mathbf{x} + \hat{\mathbf{n}}s)\ \exp\left\{-\kappa \int_0^s ds'\ \rho(\mathbf{x}_c + \hat{\mathbf{n}}s')\right\}$$

Discretize Integration

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

$$\mathbf{x}_i = \mathbf{x}_c \;+\; \hat{\mathbf{n}}\,\Delta s\; i, \quad i = 0, \ldots, \infty$$

- Full sum looks like

$$\vec{C}(\mathbf{x}_c, \hat{\mathbf{n}}) = \sum_{i=0}^{\infty} \Delta s\; \rho(\mathbf{x}_i)\; \vec{c}(\mathbf{x}_i)\; T(\mathbf{x}_c, \hat{\mathbf{n}}, i\Delta s)$$

Ray Marching

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

$$\mathbf{x}_i \mathrel{+}= \hat{\mathbf{n}}\, \Delta s$$

$$\vec{C} \mathrel{+}= \Delta s\ \rho(\mathbf{x}_i)\ \vec{c}(\mathbf{x}_i)\ T(\mathbf{x}_c, \hat{\mathbf{n}}, i\Delta s)$$

Updating Transmissivity

- At start of march, initialize $T = 1$
- As march proceeds, update transmissivity as

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

Full Iterative Ray March

$$\mathbf{x}_i \mathrel{+}= \ \hat{\mathbf{n}}\,\Delta s$$

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

$$\vec{C} \mathrel{+}= \ \Delta s\ \rho(\mathbf{x}_i)\ \vec{c}(\mathbf{x}_i)\ T$$

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

$$\vec{C} \mathrel{+}= \Delta s\ \rho(\mathbf{x}_i)\ \vec{c}(\mathbf{x}_i)\ T$$

- A better representation of the integral is

$$\vec{C} \mathrel{+}= \frac{(1 - e^{-\kappa \Delta s \rho(\mathbf{x}_i)})}{\kappa}\ \vec{c}(\mathbf{x}_i)\ T$$

Complete Ray March (No Lights)

$$\mathbf{x}_i \mathrel{+}= \hat{\mathbf{n}}\,\Delta s$$

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

$$T \mathrel{*}= \Delta T$$

$$\vec{C} \mathrel{+}= \frac{1-\Delta T}{\kappa}\ \vec{c}(\mathbf{x}_i)\ T$$

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

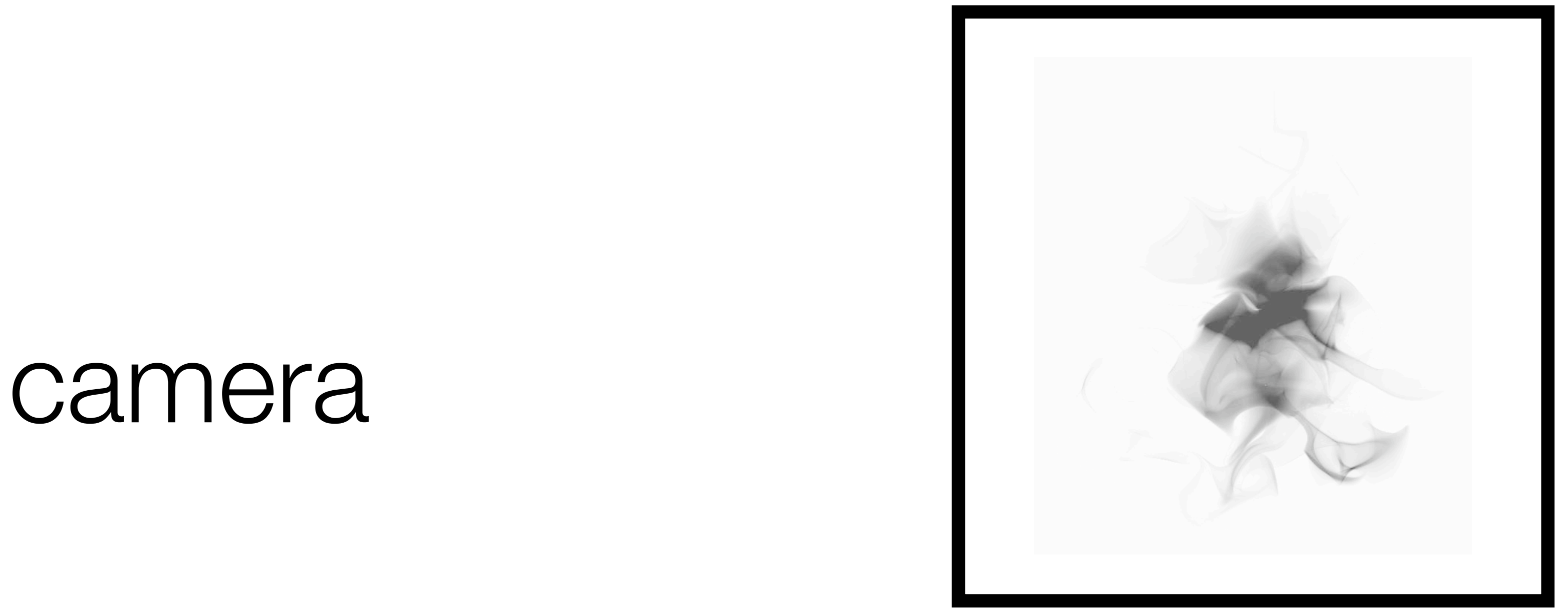
Containers

- 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

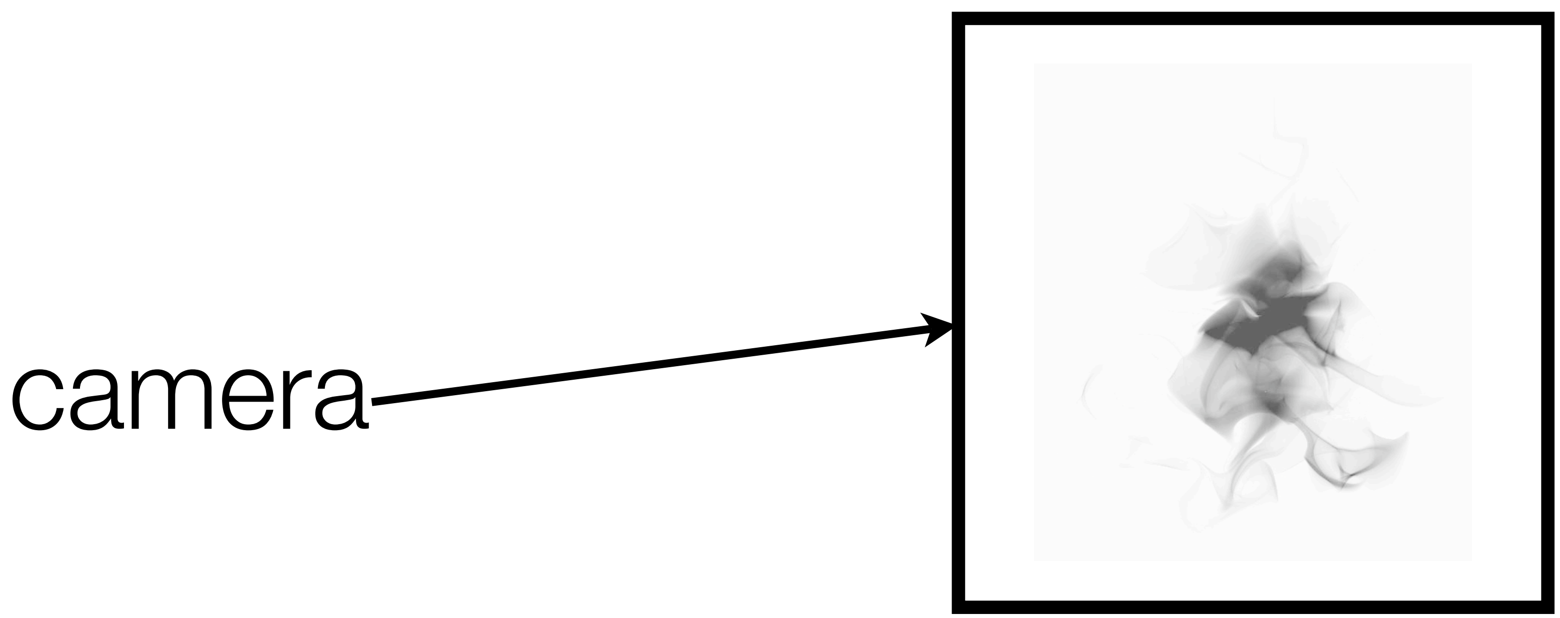
Containers

- 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



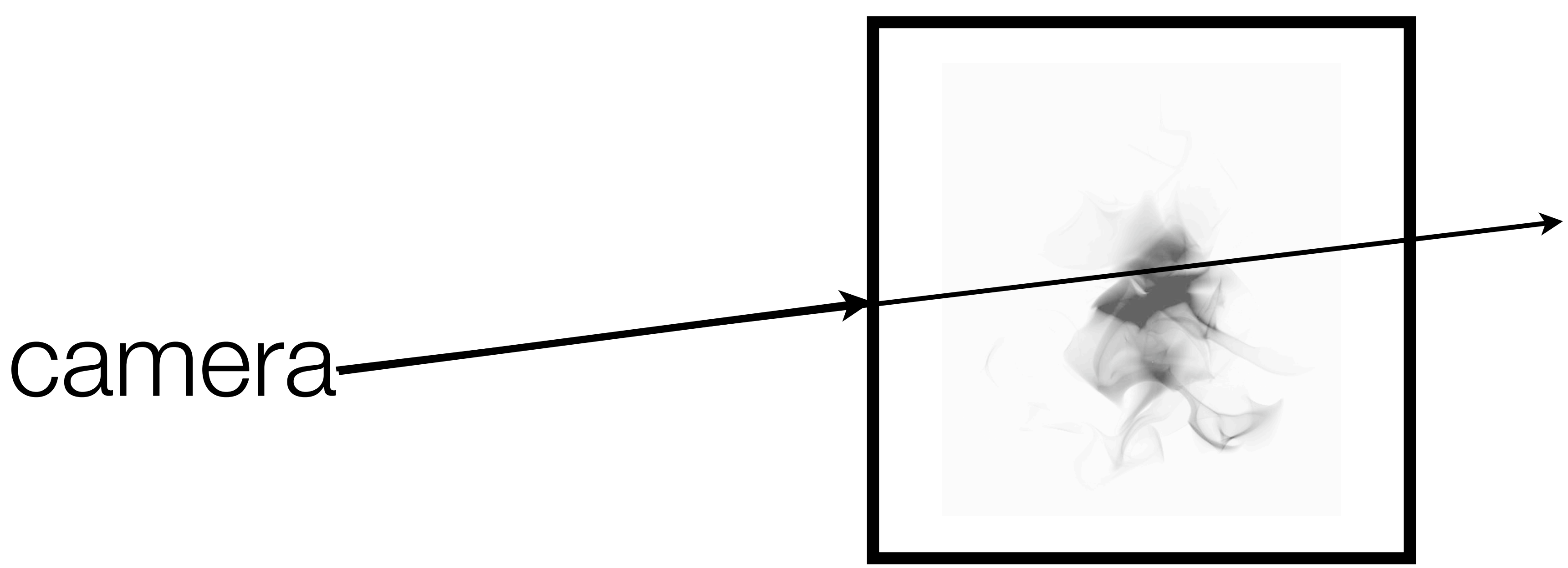
Containers

- 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



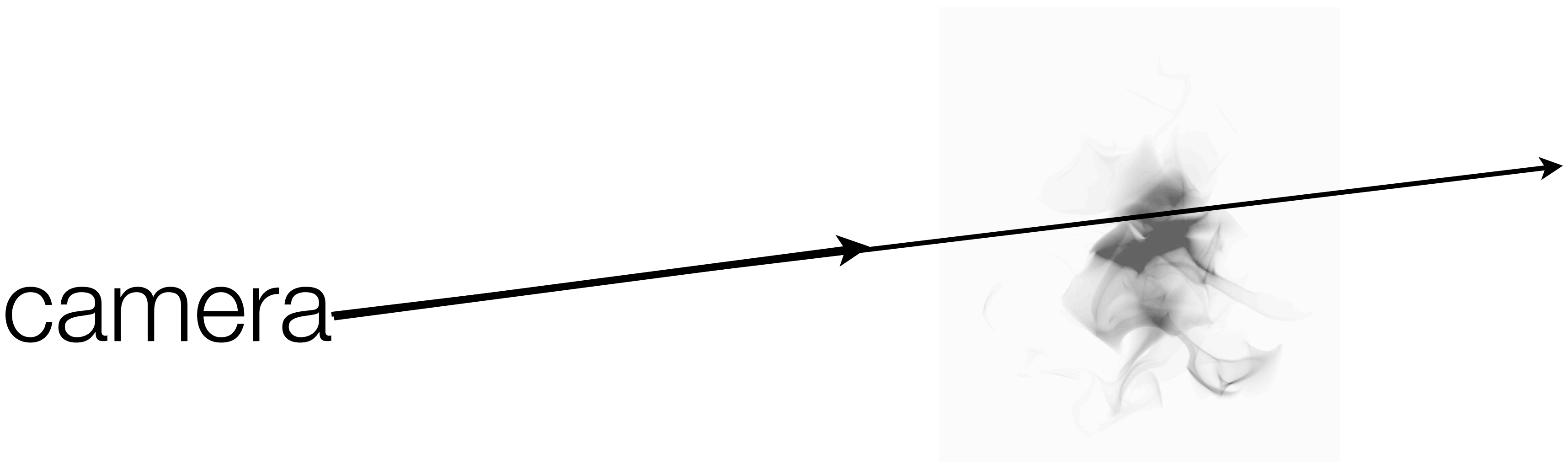
Containers

- 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

camera

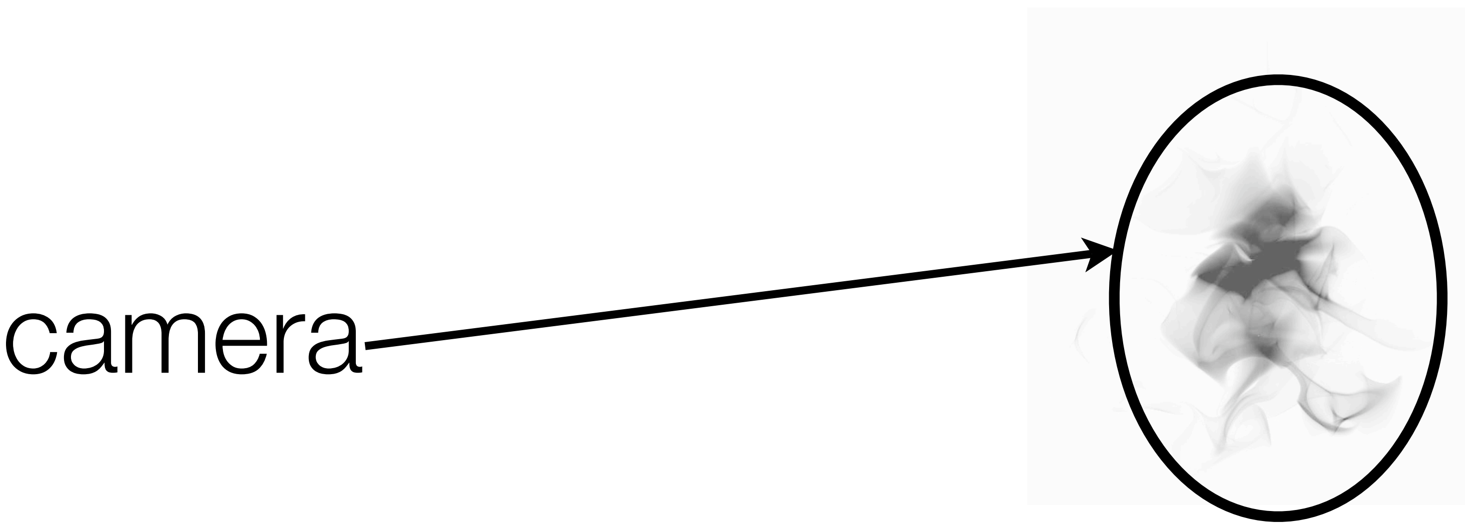
Containers

- 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

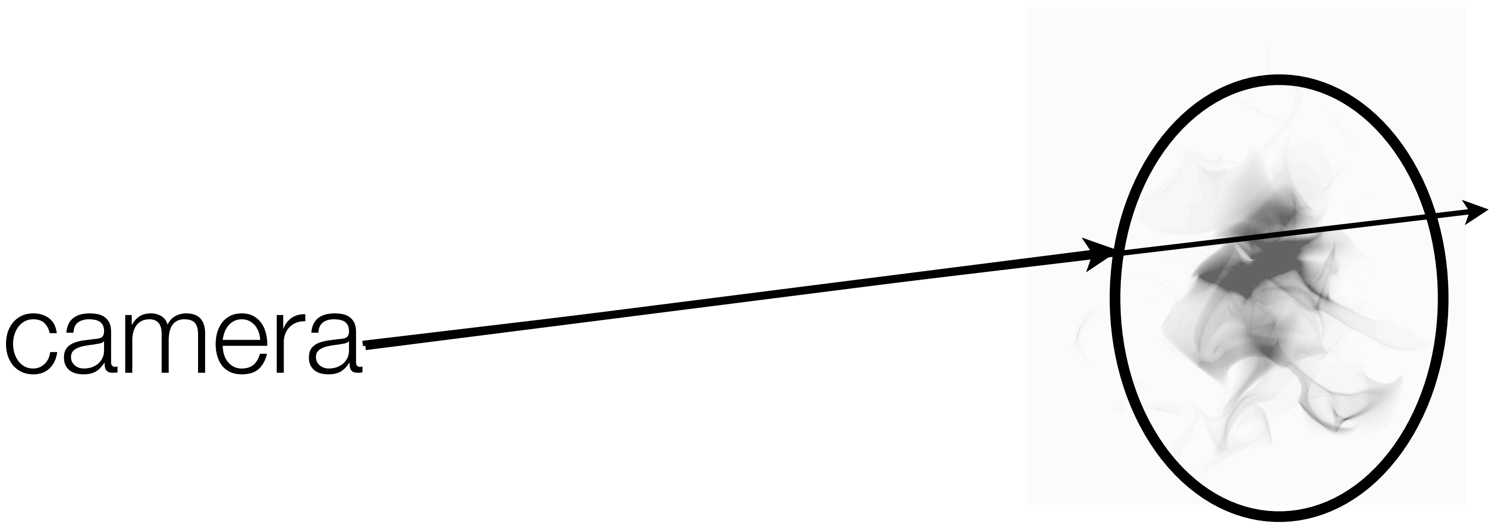
Containers

- 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



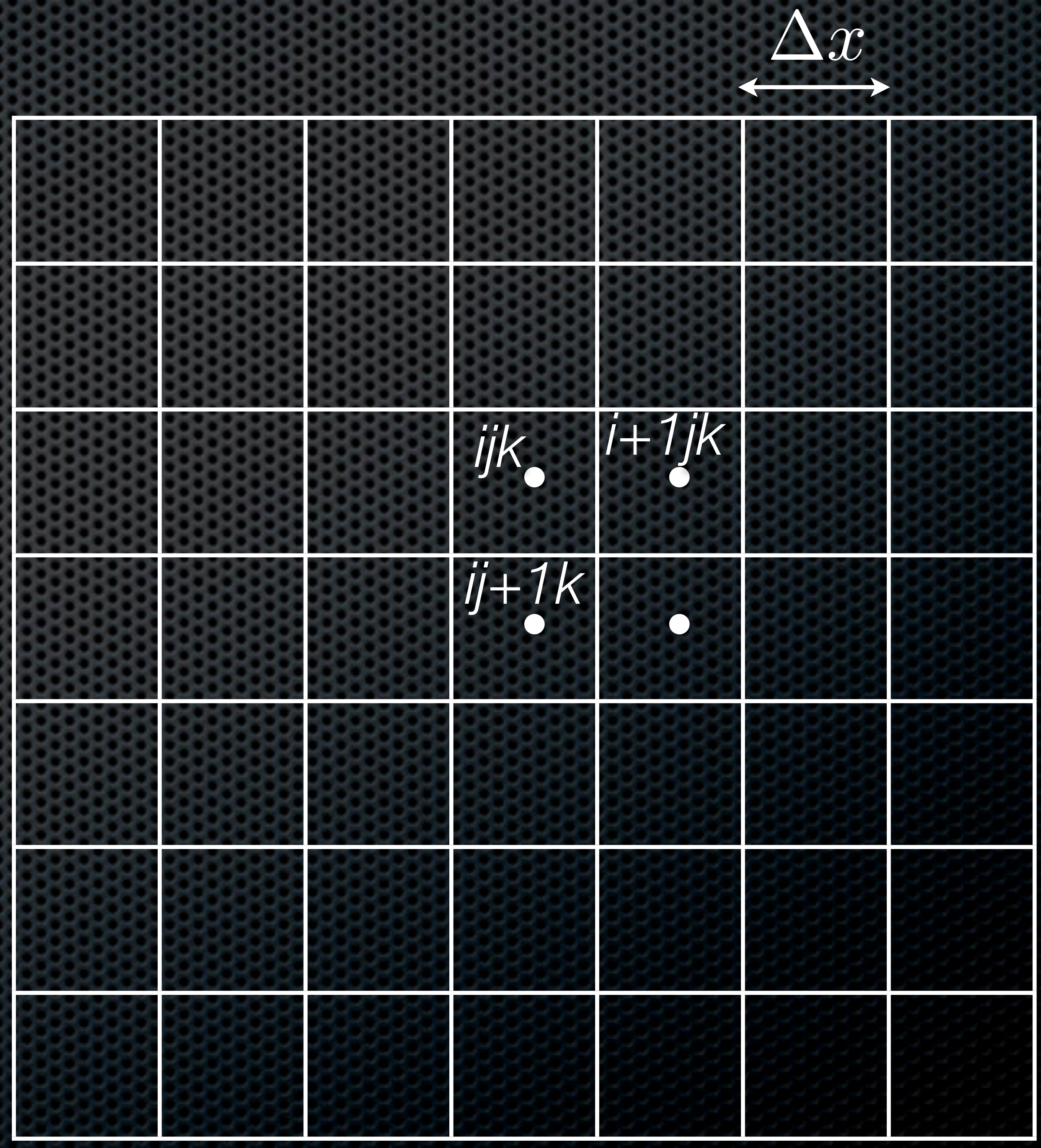
Why Use Gridded Volumes

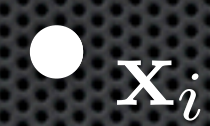
- Some algorithms for density & color work better with grids
- Even if density & color can be written as equations, they may be so slow to evaluate that a gridded sample is better
- Can store gridded values on disk for later use.

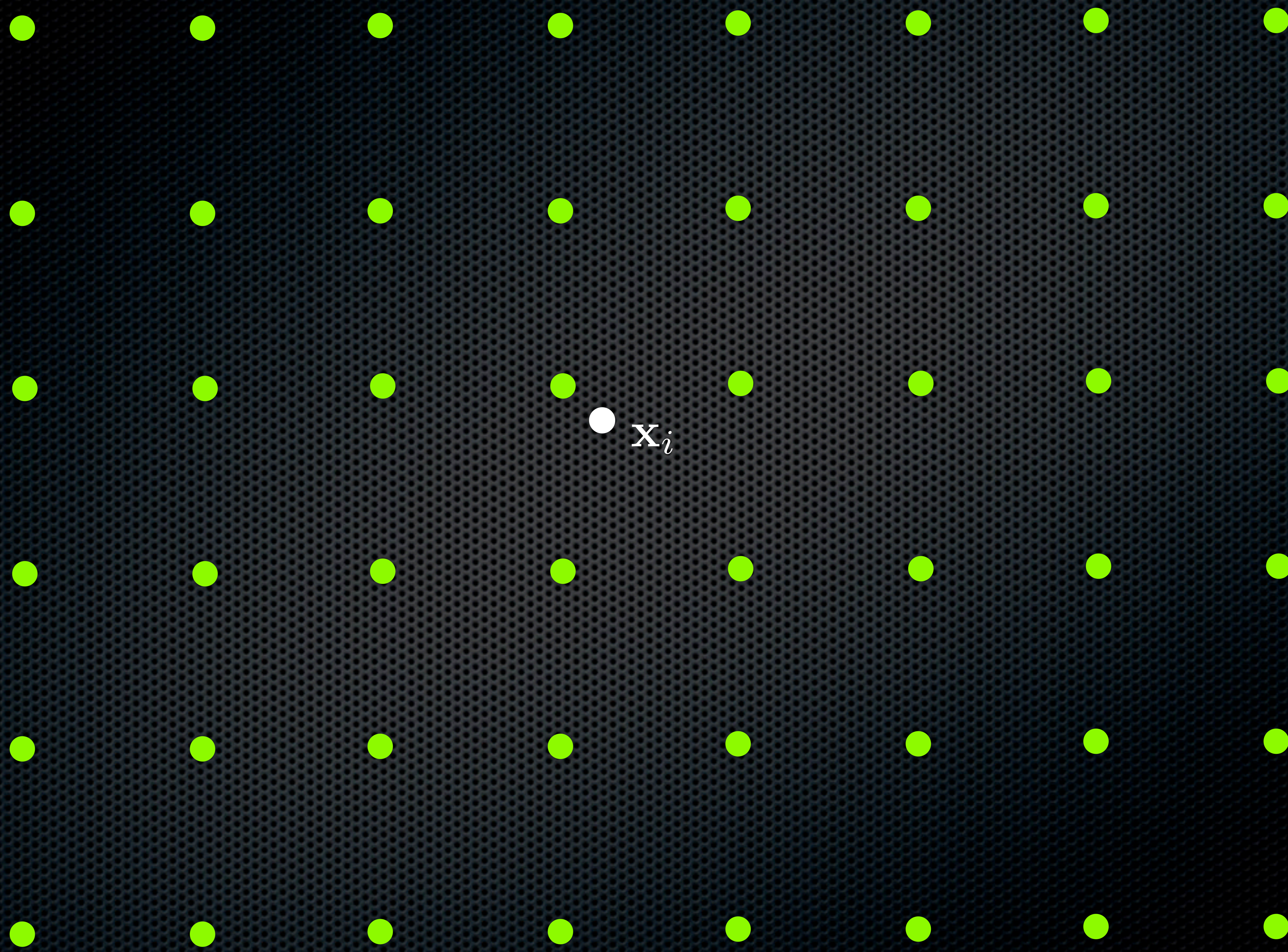
Gridded Volumes: Voxels

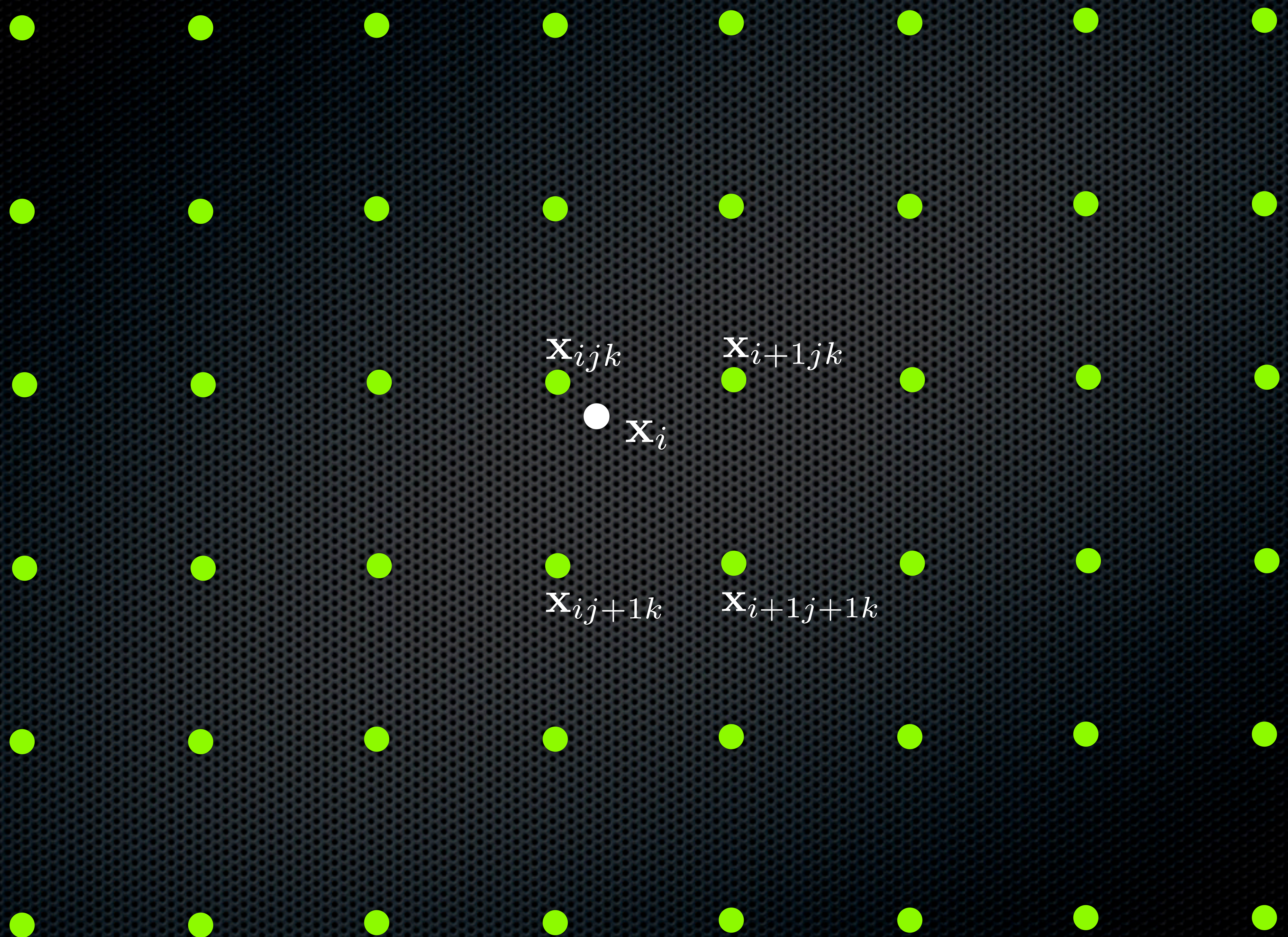
- Rectangular mesh of points $\mathbf{x}_{ijk}$ at the center of *voxels* labelled *ijk*
- *i,j,k = 1,.....,M*
- At each voxel center, density and color have values ρ_{ijk} $\vec{c}_{ijk}$

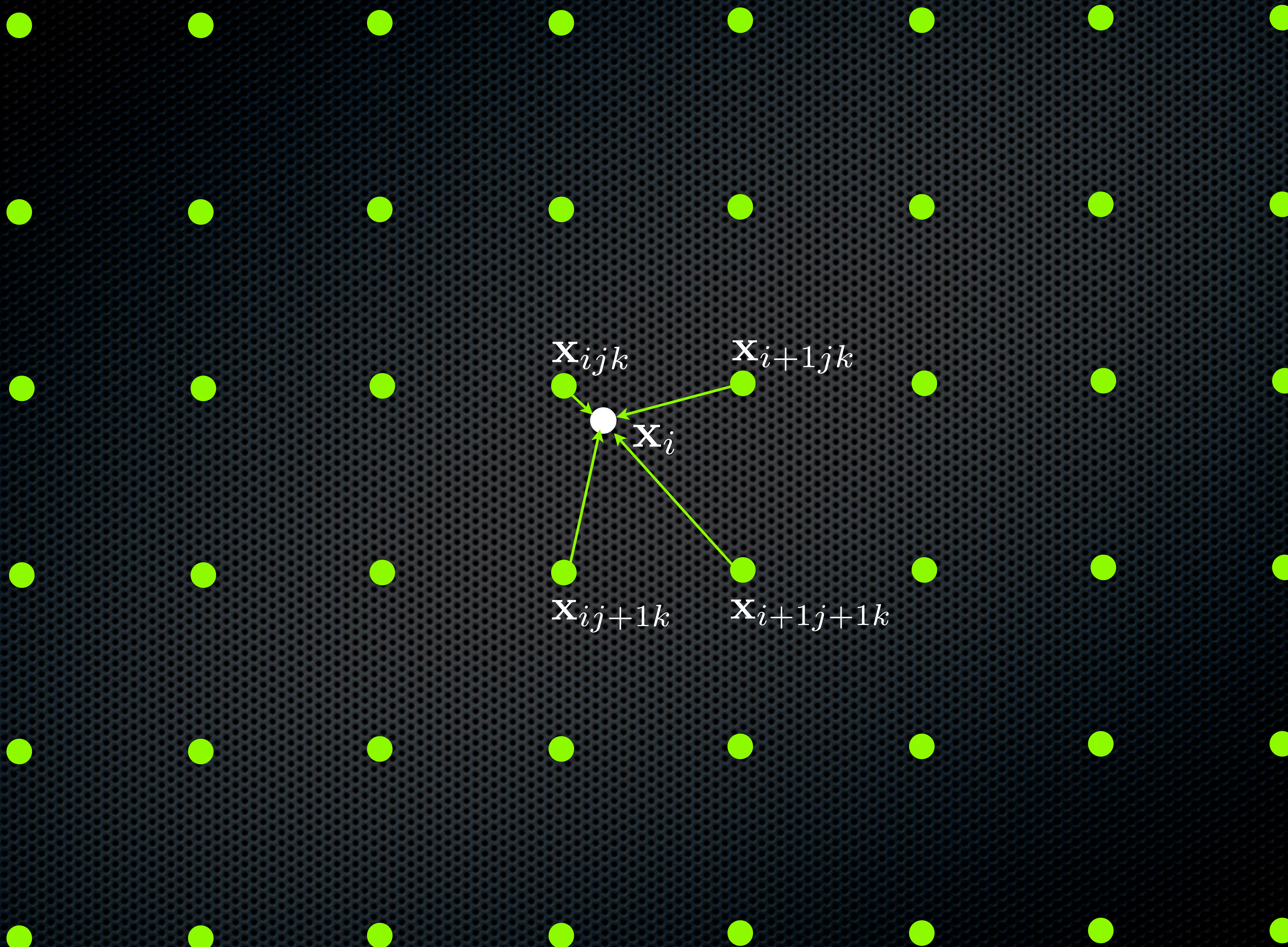
$$\rho(\mathbf{x}_{ijk}) = \rho_{ijk}$$
$$\vec{c}(\mathbf{x}_{ijk}) = \vec{c}_{ijk}$$




$\mathbf{x}_i$


$\mathbf{x}_i$


$\mathbf{x}_{ijk}$
$\mathbf{x}_{i+1jk}$
$\mathbf{x}_i$
$\mathbf{x}_{ij+1k}$
$\mathbf{x}_{i+1j+1k}$



$\mathbf{x}_{ijk}$
$\mathbf{x}_{i+1jk}$
$\mathbf{x}_i$
$\mathbf{x}_{ij+1k}$
$\mathbf{x}_{i+1j+1k}$

Trilinear Interpolation

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

$$\mathbf{x}_i = \sum_{a=i}^{i+1} \sum_{b=j}^{j+1} \sum_{c=k}^{k+1} \mathbf{x}_{abc}\, \omega_{abc}$$

- 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

$$\rho(\mathbf{x}_i) = \sum_{a=i}^{i+1} \sum_{b=j}^{j+1} \sum_{c=k}^{k+1} \rho_{abc}\, \omega_{abc} \qquad \vec{c}(\mathbf{x}_i) = \sum_{a=i}^{i+1} \sum_{b=j}^{j+1} \sum_{c=k}^{k+1} \vec{c}_{abc}\, \omega_{abc}$$

Interpolation Weights

$$\mathbf{x}_i = (x_i, y_i, z_i) \qquad \mathbf{x}_{abc} = (x_{abc}, y_{abc}, z_{abc})$$

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

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

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

$$\omega_{abc} = \omega^x_{abc}\, \omega^y_{abc}\, \omega^z_{abc}$$

Motion Tests with Particles

Motion Tests with Gridded Volumes

Light Color Transmission

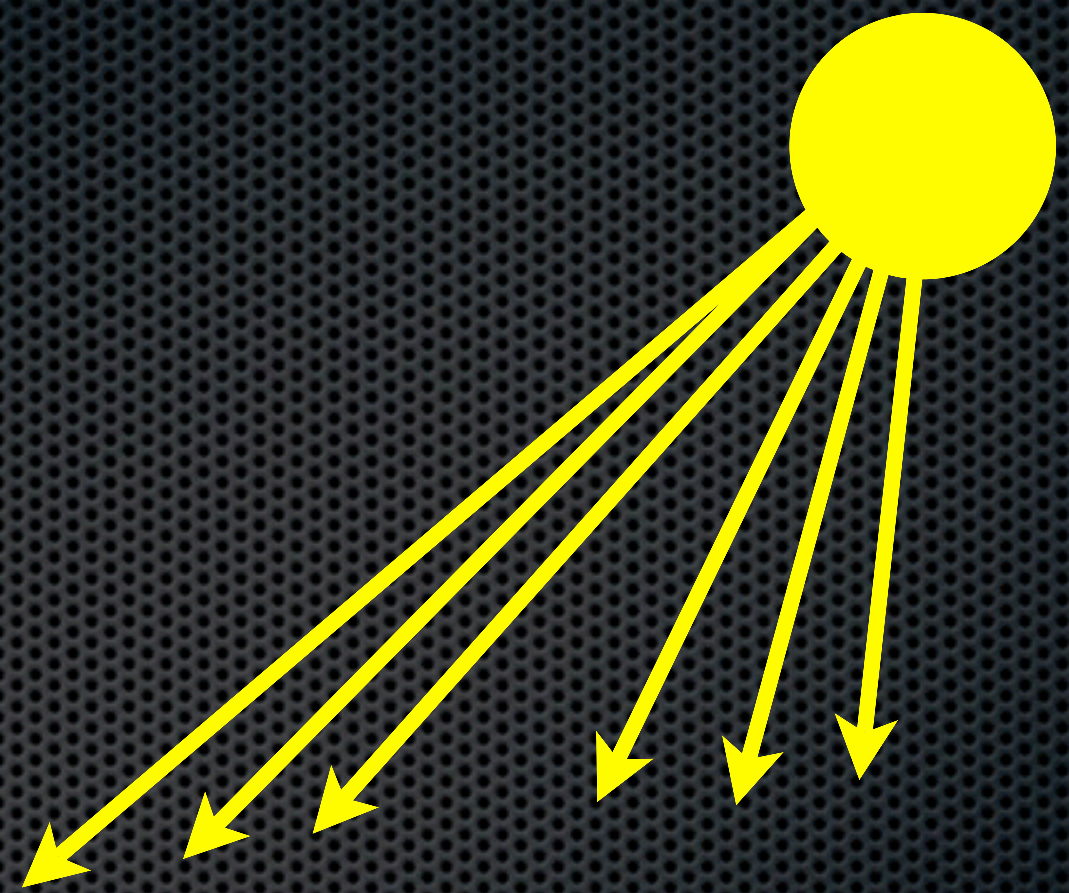
- Lights modify this ray march procedure
- The color of the material is “multiplied” by the color of the light.
- The color of the light is attenuated by the volume material between the light and the ray march points.

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

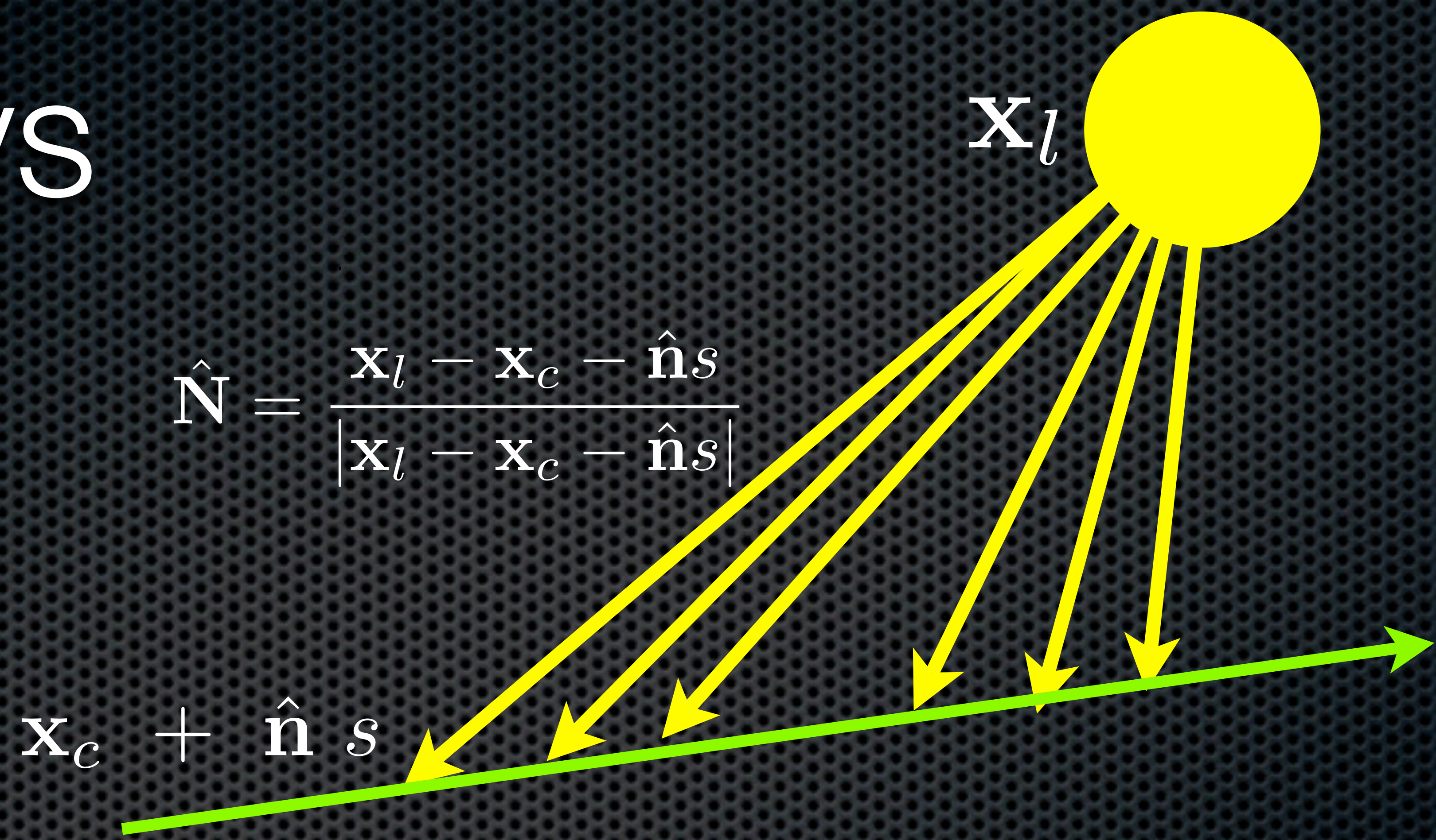
$$\vec{c} \odot \vec{F} \;=\; (r\;F_r,\;\; g\;F_g,\;\; b\;F_b)$$

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.

$$Q(\mathbf{x}_c, \hat{\mathbf{n}}, s, \mathbf{x}_l) = \exp\left\{ -\kappa \int_0^D ds' \rho(\mathbf{x}_c + \hat{\mathbf{n}}s + \hat{\mathbf{N}}s') \right\}$$

$$D = |\mathbf{x}_l - \mathbf{x}_c - \hat{\mathbf{n}}s|$$

Rendering Equation: Lights

$$\vec{C}(\mathbf{x}_c, \hat{\mathbf{n}}) = \int_0^\infty ds\, \rho(\mathbf{x}_c{+}\hat{\mathbf{n}}s)\, (\vec{c}(\mathbf{x}{+}\hat{\mathbf{n}}s){\odot}\vec{F})\, \exp\left\{-\kappa \int_0^s ds'\, \rho(\mathbf{x}_c + \hat{\mathbf{n}}s')\right\} Q(\mathbf{x}_c, \hat{\mathbf{n}}, s, \mathbf{x}_l)$$

Ray March with Lights

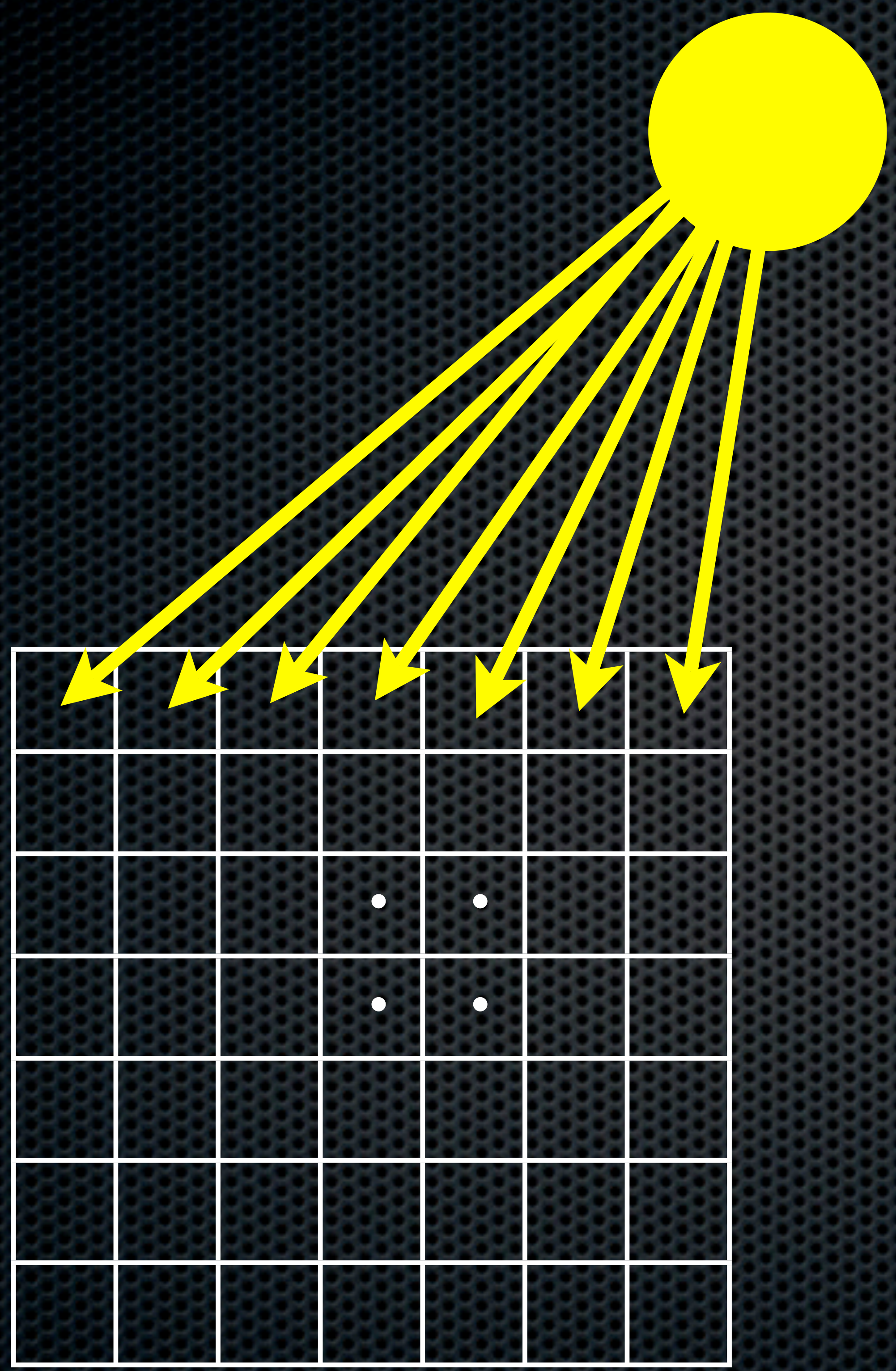
$$\mathbf{x}_i \mathrel{+}= \hat{\mathbf{n}}\,\Delta s$$

$$\Delta T = \exp\left\{-\kappa \Delta s\ \rho(\mathbf{x}_i)\right\}$$

$$T \mathrel{*}= \Delta T$$

$$\vec{C} \mathrel{+}= \frac{1-\Delta T}{\kappa}\left(\vec{c}(\mathbf{x}_i)\odot \vec{F}\right)\, T\ Q(\mathbf{x}_c, \hat{\mathbf{n}}, \Delta s i, \mathbf{x}_l)$$

Precomputed Light Transmissivity



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

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

$$\hat{\mathbf{N}}_{ijk} = \frac{\mathbf{x}_l - \mathbf{x}_{ijk}}{|\mathbf{x}_l - \mathbf{x}_{ijk}|}$$

- Use trilinear interpolation for points off of voxels centers.

Full Ray March with Lights

$$\mathbf{x}_i \mathrel{+}= \hat{\mathbf{n}}\,\Delta s$$

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

$$T \mathrel{*}= \Delta T$$

$$\vec{C} \mathrel{+}= \frac{1-\Delta T}{\kappa}\,(\vec{c}(\mathbf{x}_i) \odot \vec{F})\; T\; Q(\mathbf{x}_i)$$

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

$$f(\mathbf{x}) \;=\; 0$$

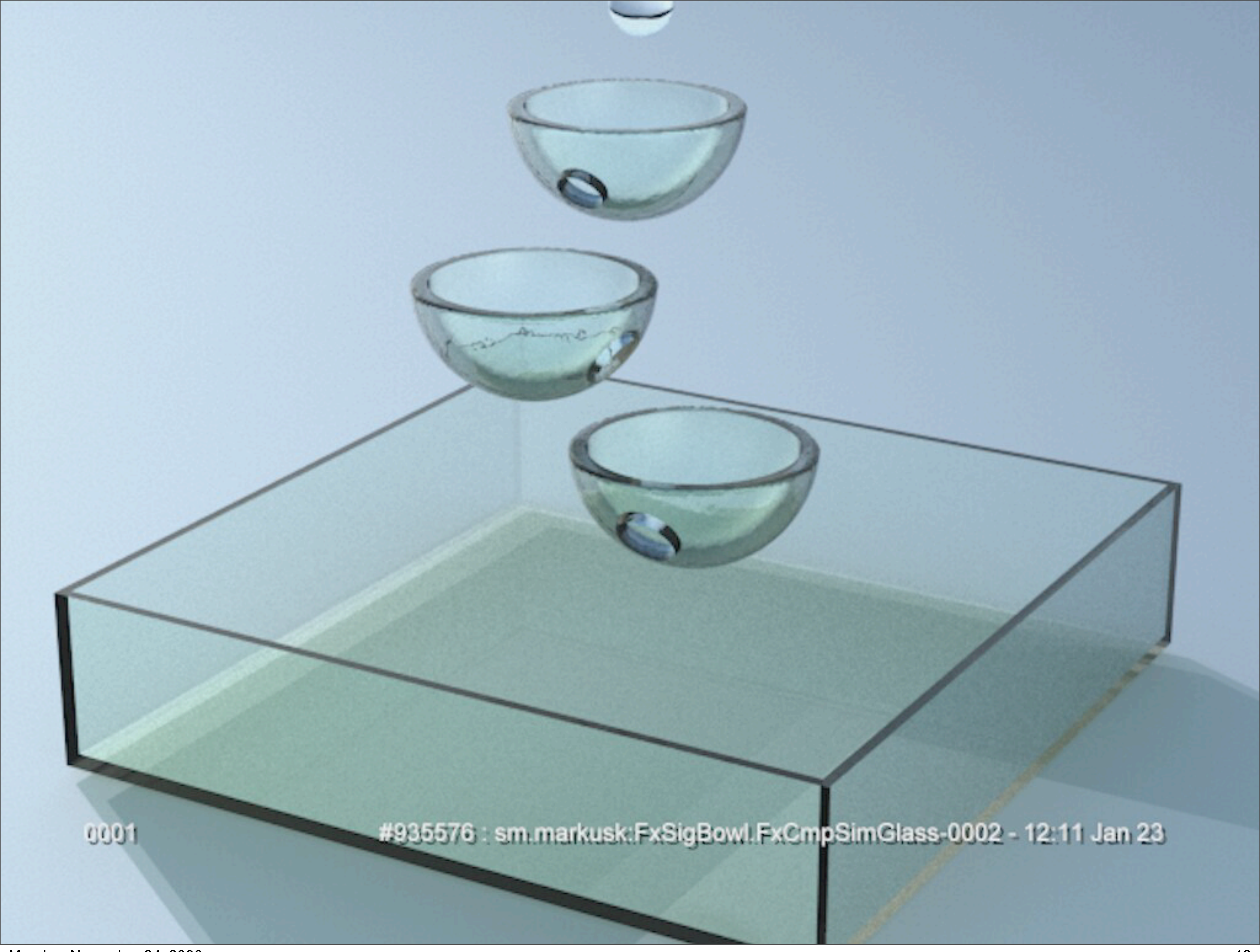
- Examples:
  - sphere $$1 - \frac{|\mathbf{x}|^2}{r^2} \;=\; 0$$
  - torus $$4R^2(x^2 + z^2) - (|\mathbf{x}|^2 + R^2 - r^2)^2 \;=\; 0$$
  - cone $$\cos^{-1}\left(\frac{\mathbf{x} \cdot \hat{\mathbf{a}}}{|\mathbf{x}|}\right) - \theta \;=\; 0$$

Levelset Density

- 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
http://www.unchainedgeometry.com/jbloom/book.html


0001
#935576 : sm.markusk:FxSigBowl.FxCmpSimGlass-0002 - 12:11 Jan 23

Density from Implicit Function

$$\rho(\mathbf{x}) = \begin{cases} f(\mathbf{x})/f_{max} & f(\mathbf{x}) > 0 \text{ (inside)} \\ 0 & f(\mathbf{x}) \leq 0 \text{ (outside)} \end{cases}$$

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


perspl

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
- Algorithm has noticeable patterns in the sequence
- Sequence 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

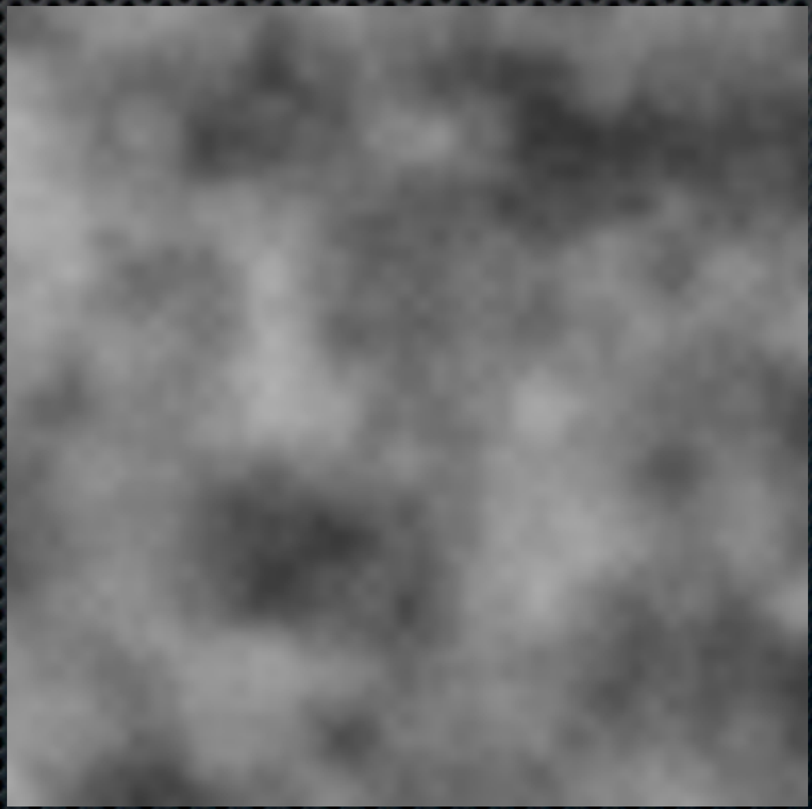
code: http://cobweb.ecn.purdue.edu/~ebertd/texture/

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*.

$$\mathrm{fBm}(\mathbf{x}) = \sum_{\ell=1}^{L} A^{\ell-1}\, \mathrm{Perlin}(\mathbf{x}\ f^{\ell})$$



Pyroclastic Puff

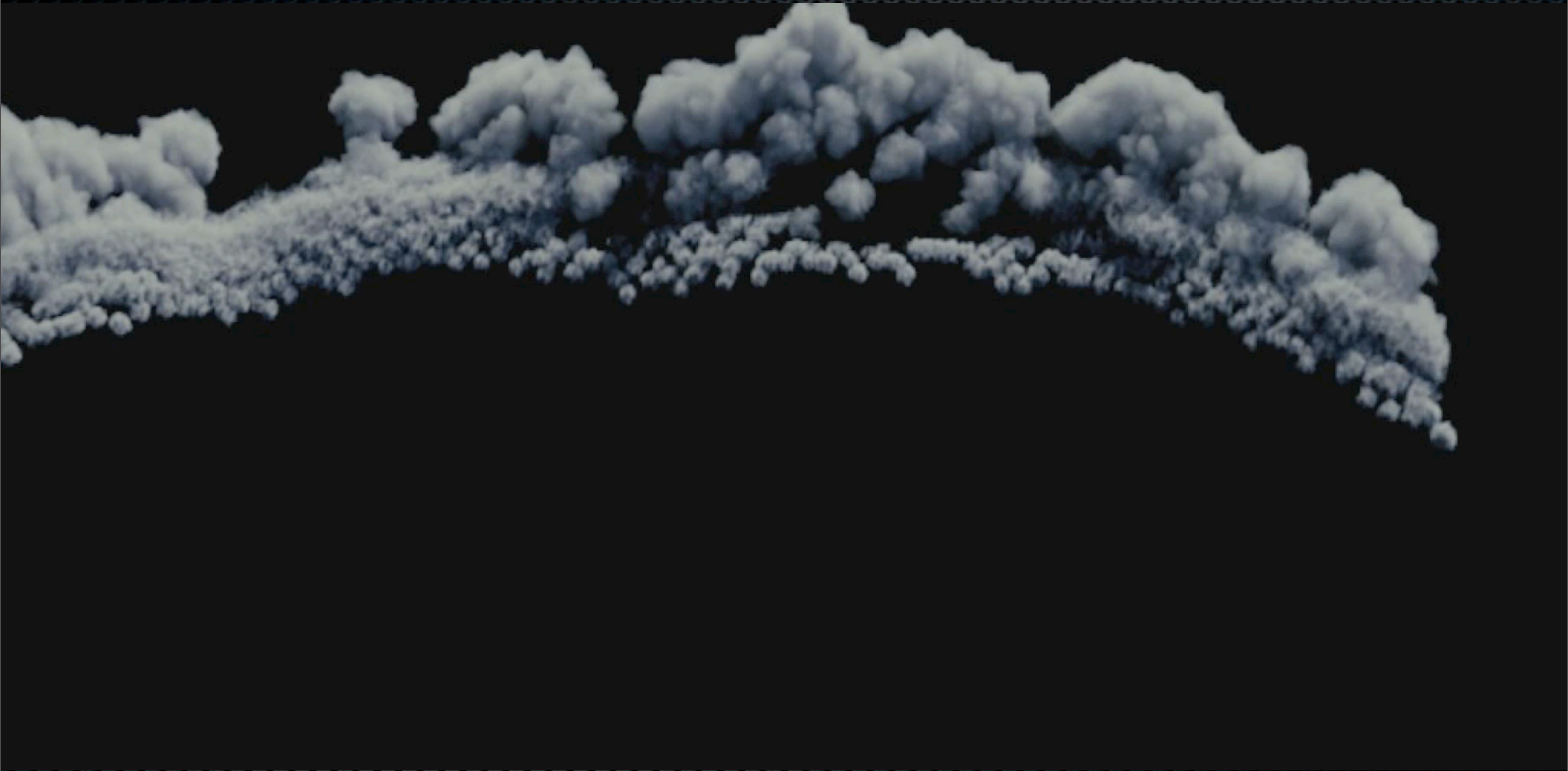
- Implicit function for a sphere:

$$1 - \frac{|\mathbf{x} - \mathbf{x}_s|^2}{r^2}$$

- Use fBm of perlin noise to displace boundary

$$\left|\mathrm{fBm}\left(\frac{\mathbf{x} - \mathbf{x}_s}{|\mathbf{x} - \mathbf{x}_s|}\right)\right| + a - \frac{|\mathbf{x} - \mathbf{x}_s|^2}{r^2}$$

- Update density of each voxel inside this implicit function



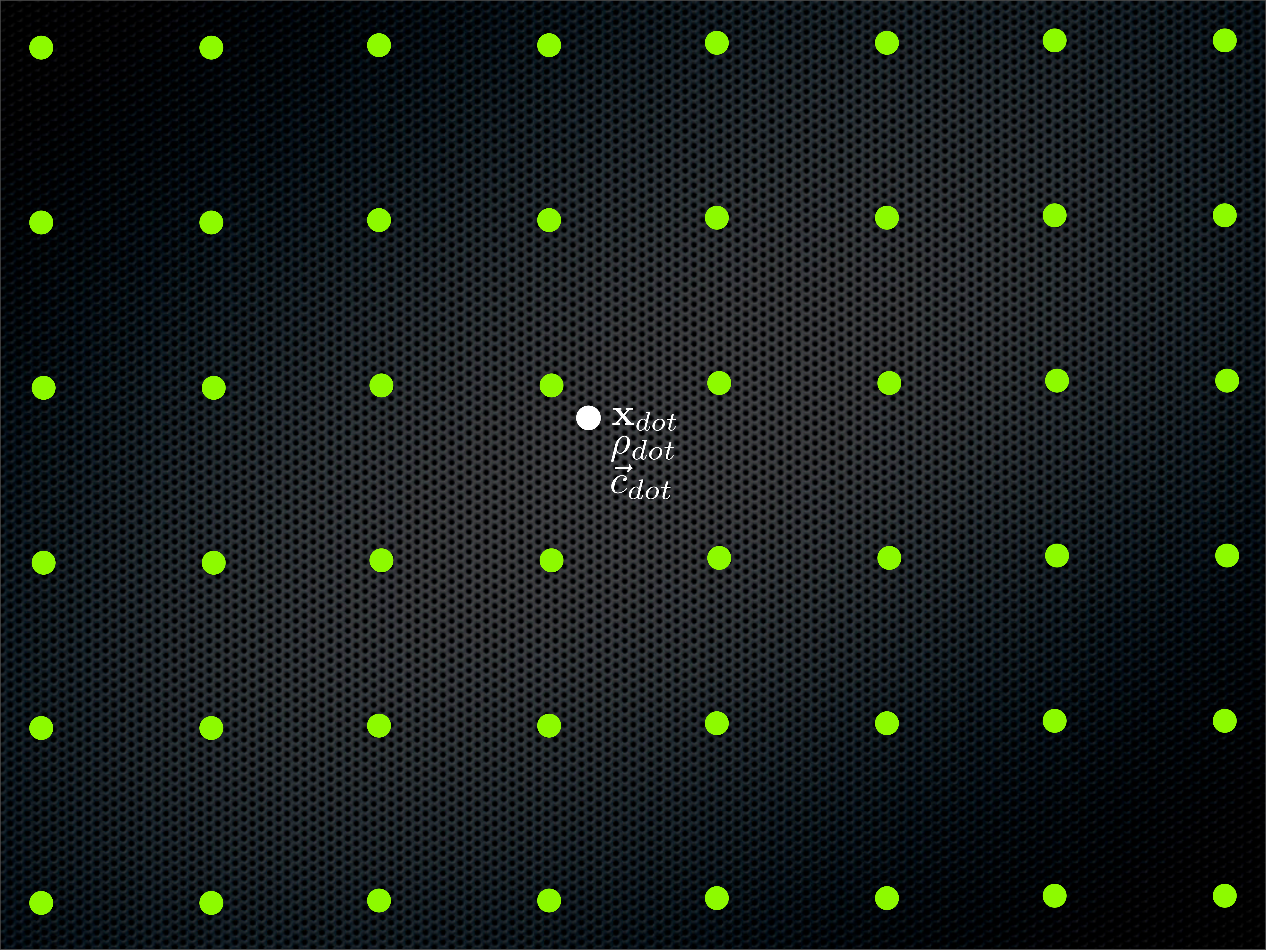
Baking anti-aliased dots

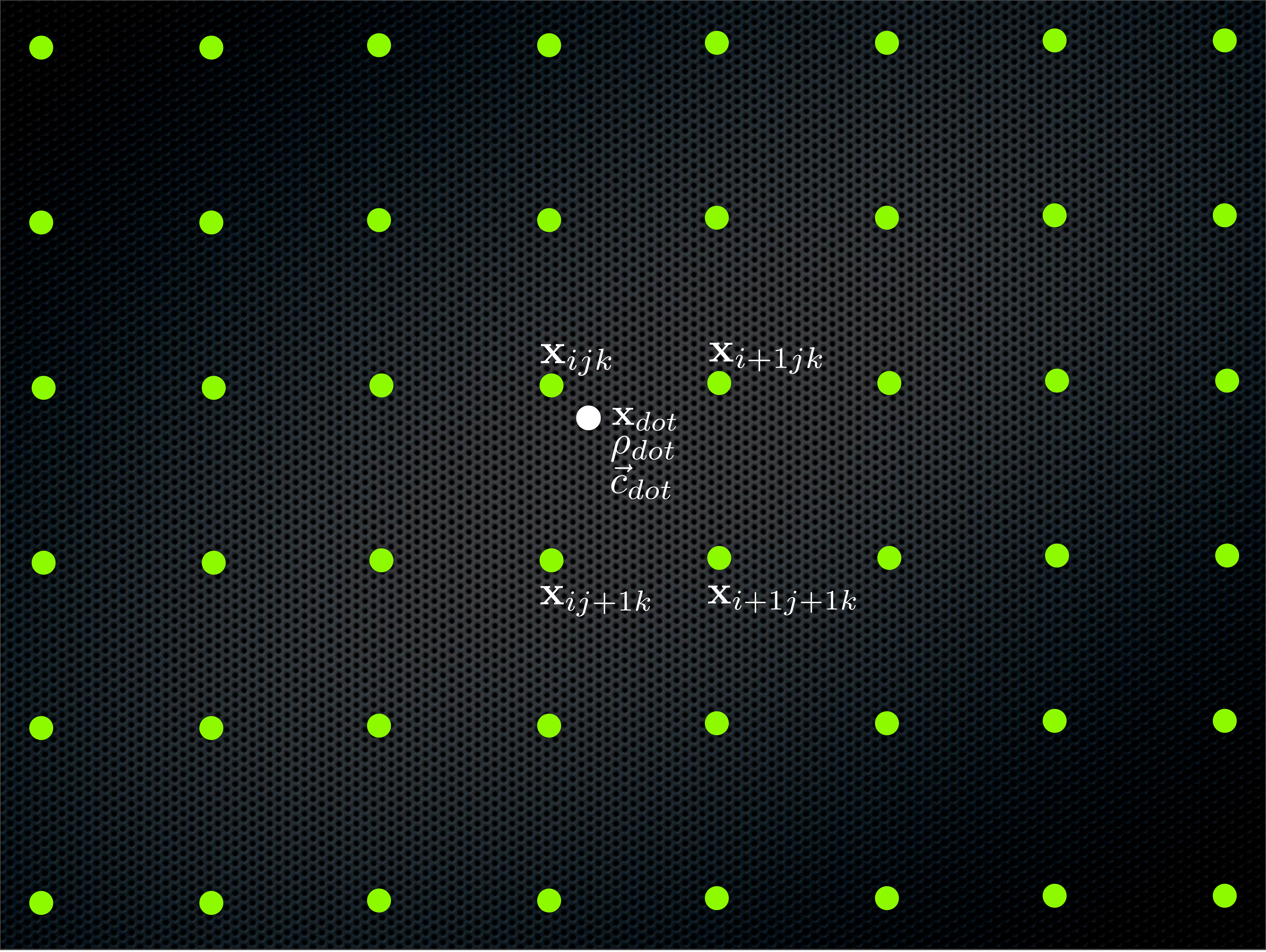
- Some algorithms generate tiny dots of density & color
- Bake many of them into a grid one by one.
- Since they are tiny, baking has to be done with anti-aliasing, smearing dot across eight neighboring voxels.
- This is a very powerful & flexible technique

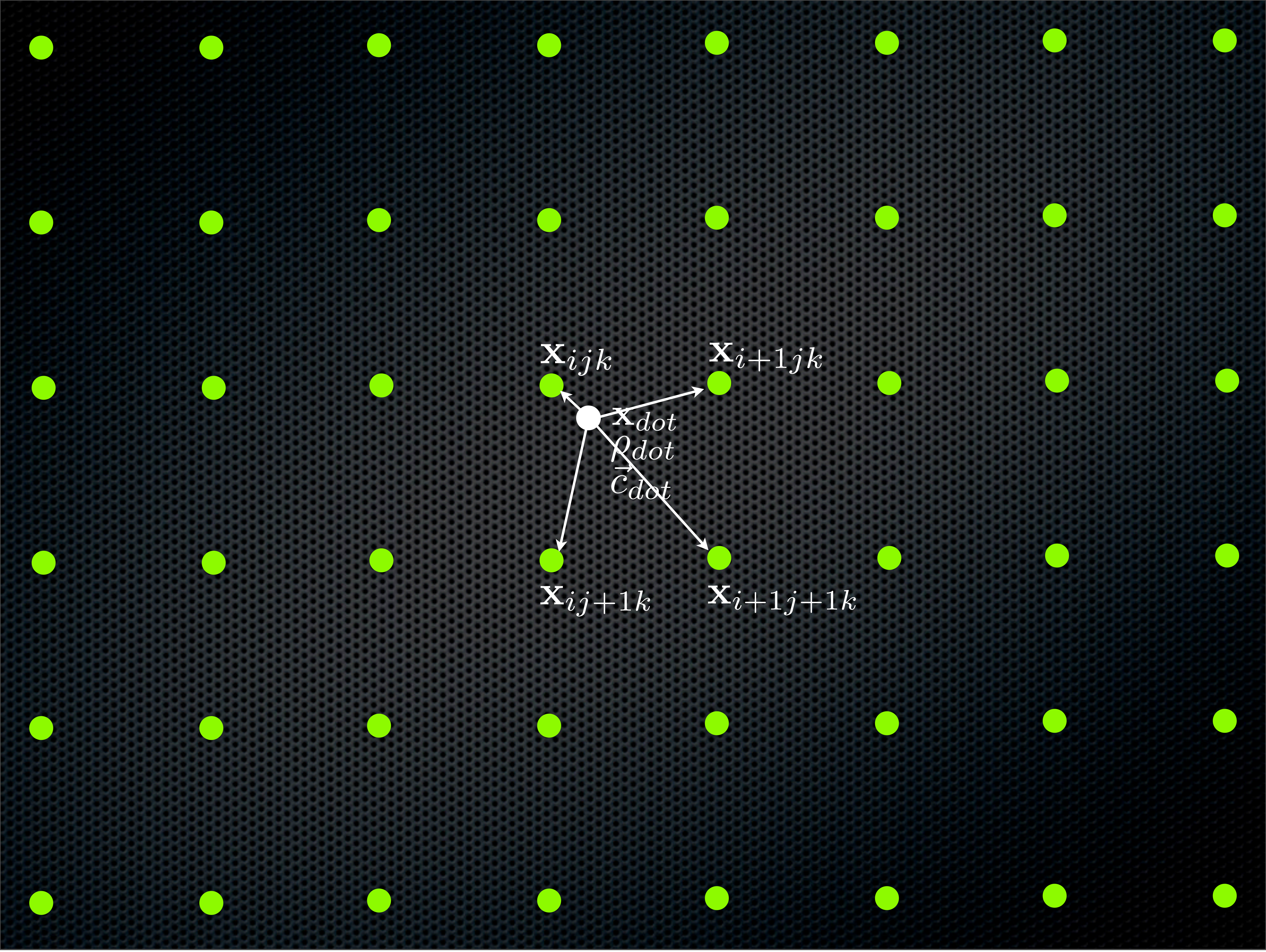
$\mathbf{x}_{dot}$

ρ_{dot}

$\vec{c}_{dot}$


$\mathbf{x}_{dot}$
ρ_{dot}
$\vec{c}_{dot}$


$\mathbf{x}_{ijk}$
$\mathbf{x}_{i+1jk}$
$\mathbf{x}_{dot}$
ρ_{dot}
$\vec{c}_{dot}$
$\mathbf{x}_{ij+1k}$
$\mathbf{x}_{i+1j+1k}$


$\mathbf{x}_{ijk}$
$\mathbf{x}_{i+1jk}$
$\mathbf{x}_{dot}$
ρ_{dot}
$\vec{c}_{dot}$
$\mathbf{x}_{ij+1k}$
$\mathbf{x}_{i+1j+1k}$

Bake dot by updating voxels

- Dot located at $\mathbf{x}_{dot}$ with density & color ρ_{dot} $\vec{c}_{dot}$

- 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 ω_{abc}

- Update density & color at the 8 nearest voxels

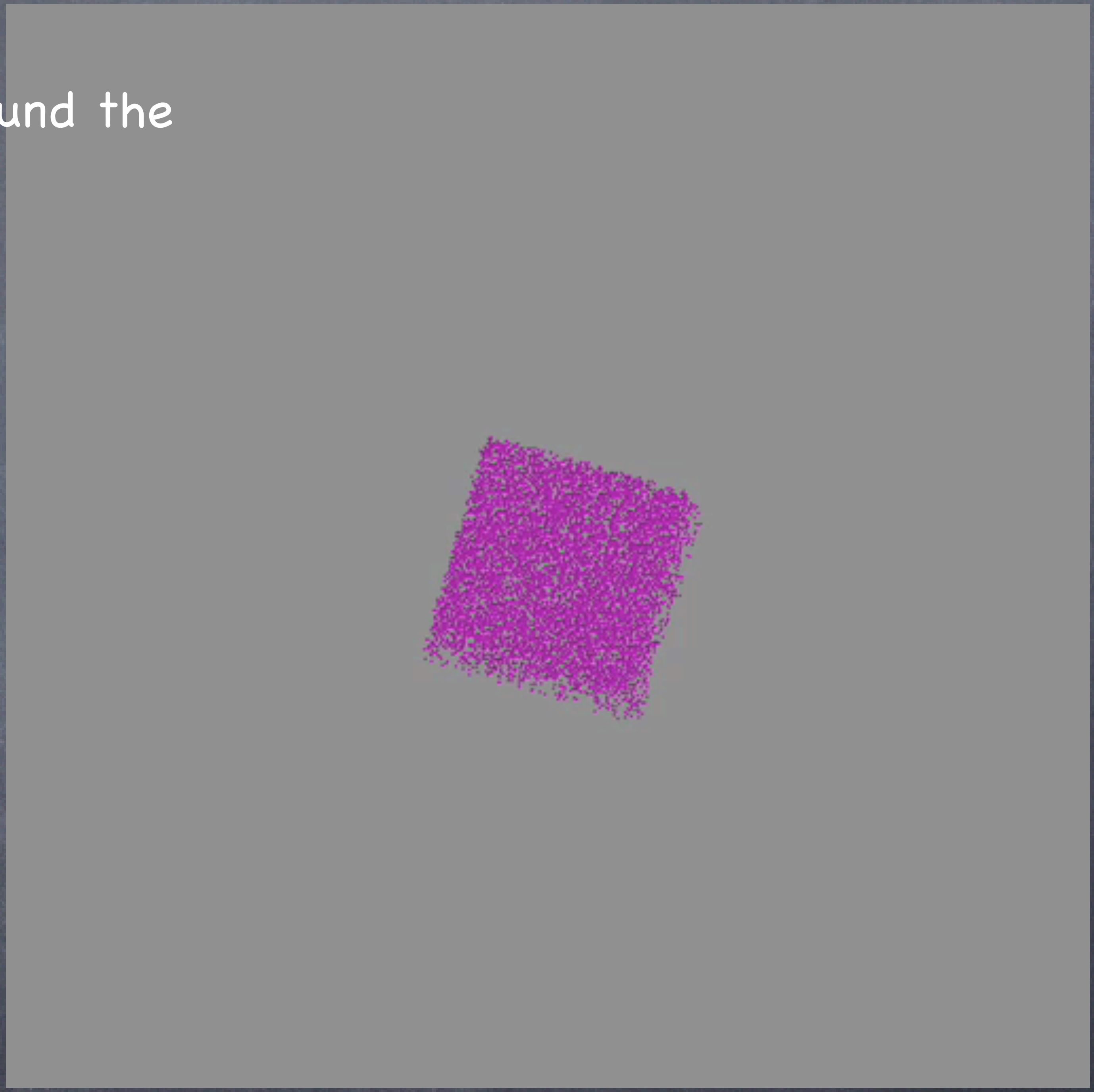
$$\rho_{abc} += \rho_{dot}\ \omega_{abc}$$

$$\vec{c}_{abc} += \vec{c}_{dot}\ \omega_{abc}$$

Steps to Grow Point Wisps

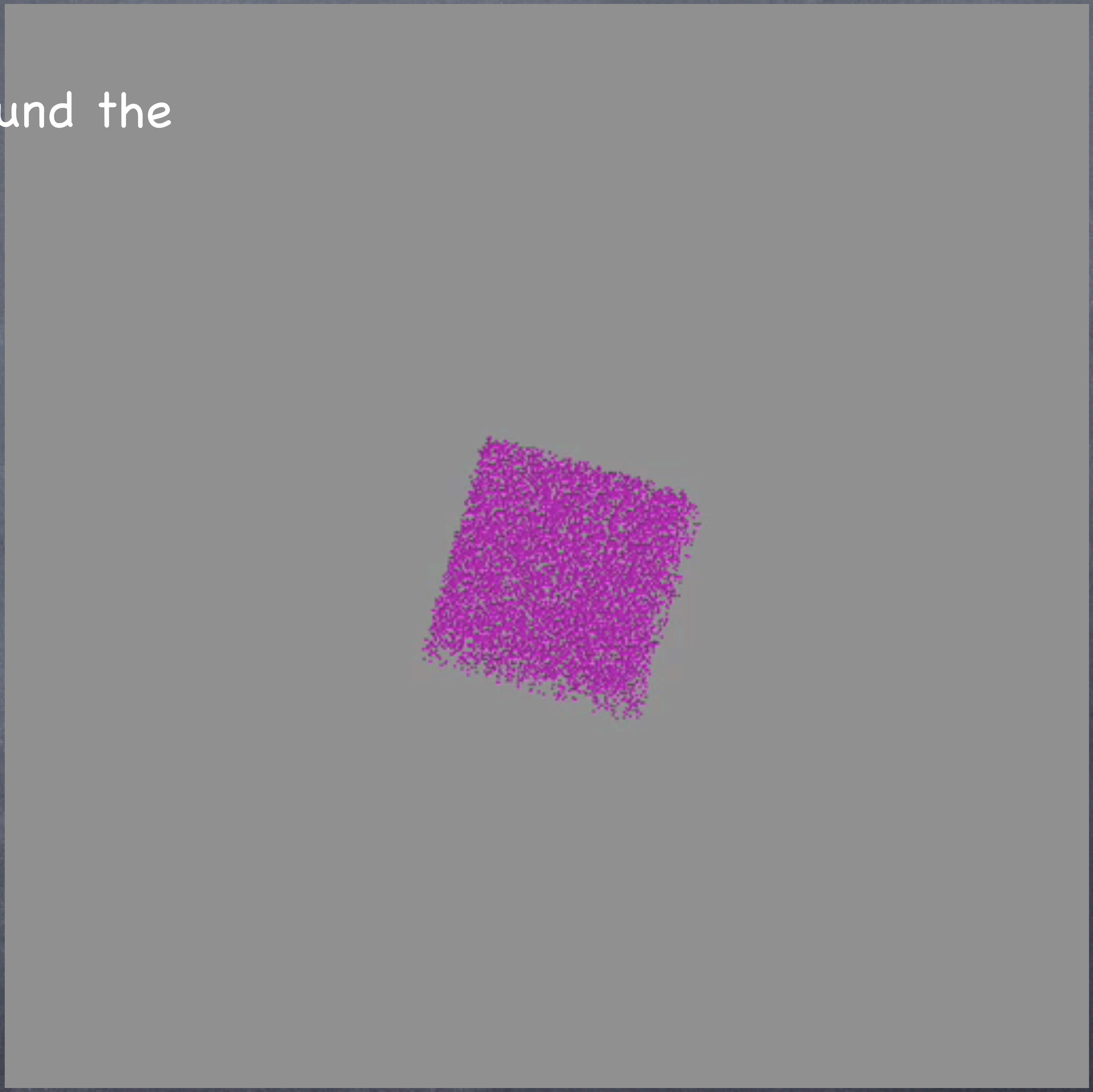
Steps to Grow Point Wisps

- Distribute points randomly in space around the guide point
  - correlated random walk
  - hundreds or thousands of points



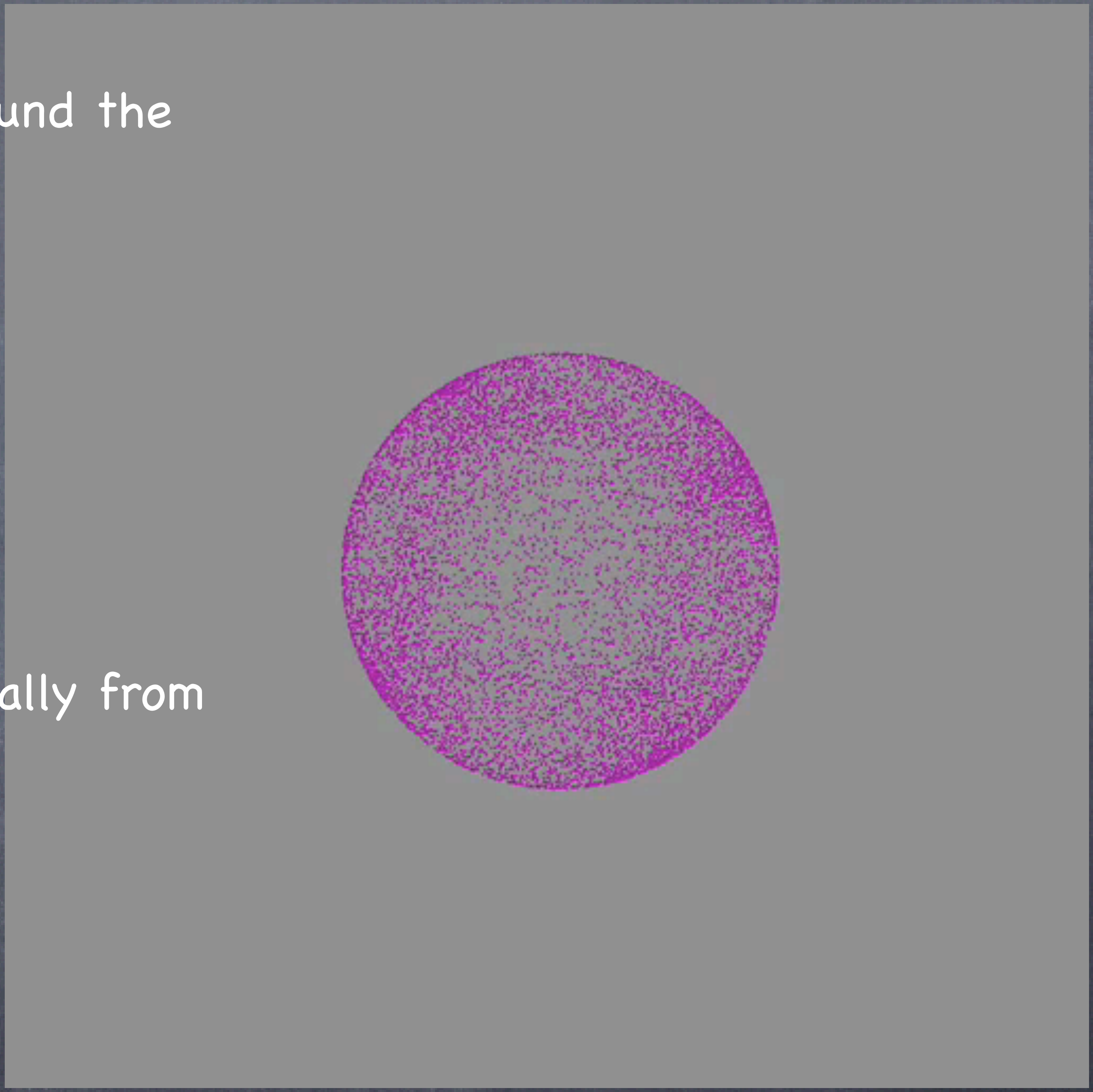
Steps to Grow Point Wisps

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



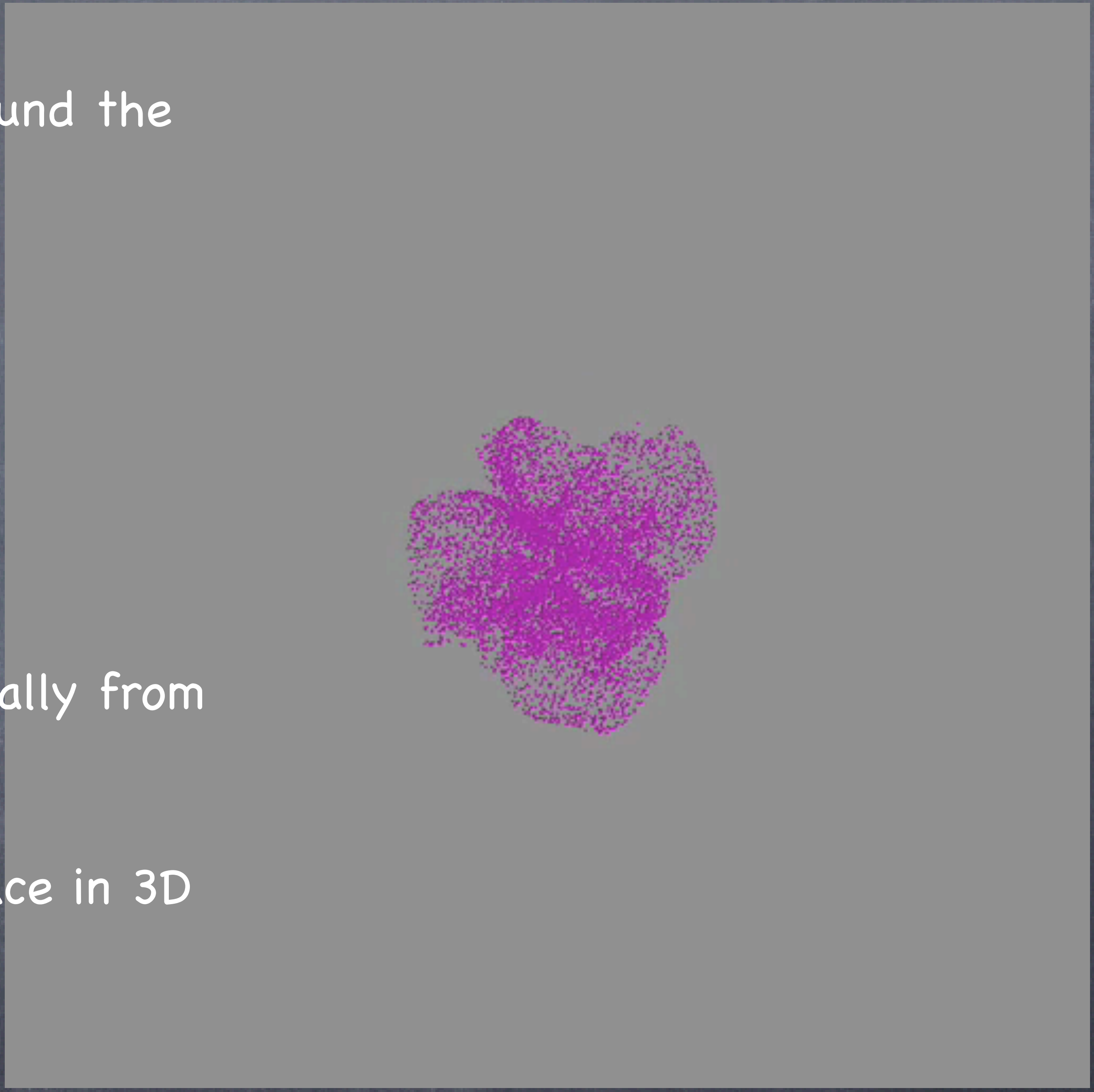
Steps to Grow Point Wisps

- 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



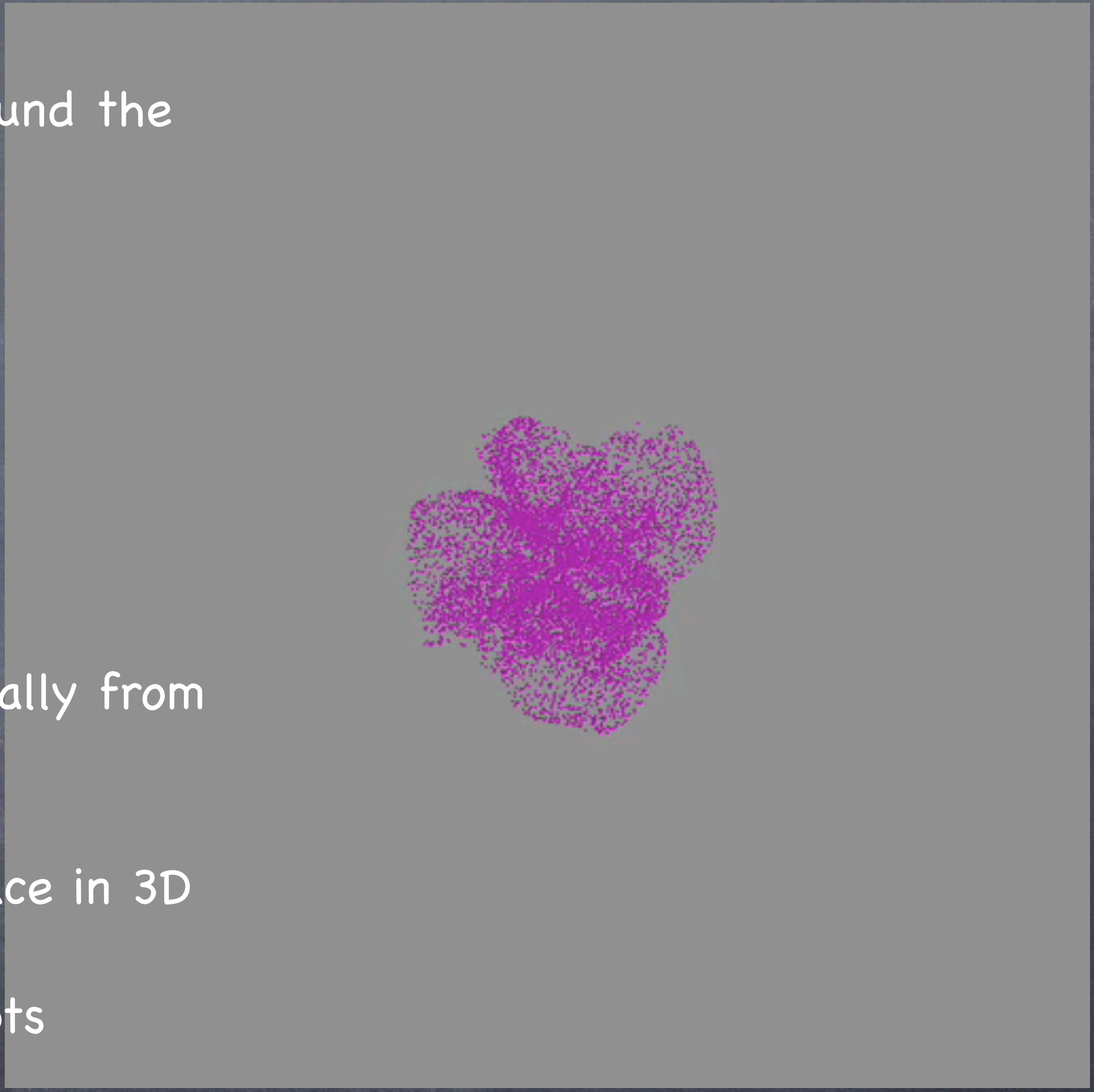
Steps to Grow Point Wisps

- 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

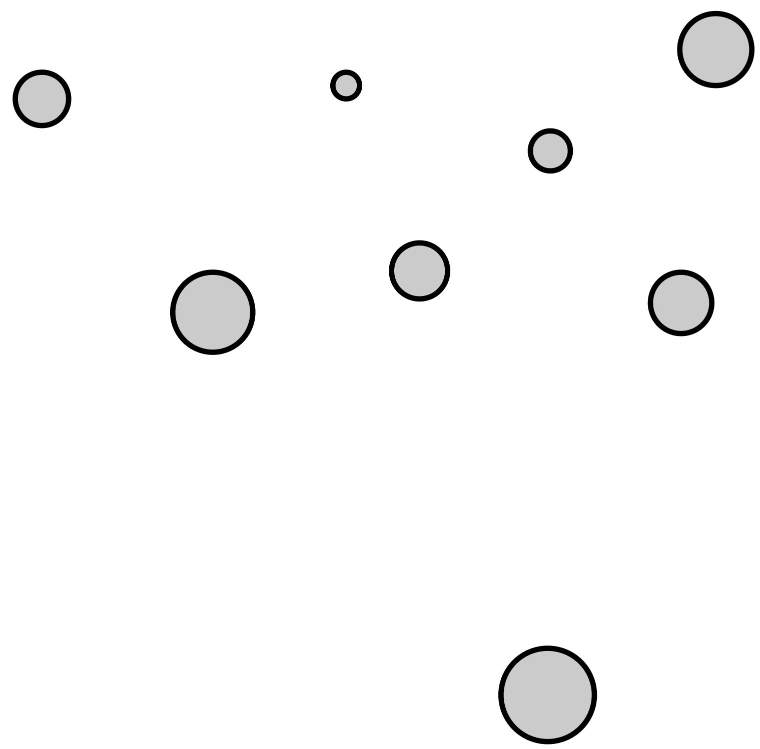


Steps to Grow Point Wisps

- 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

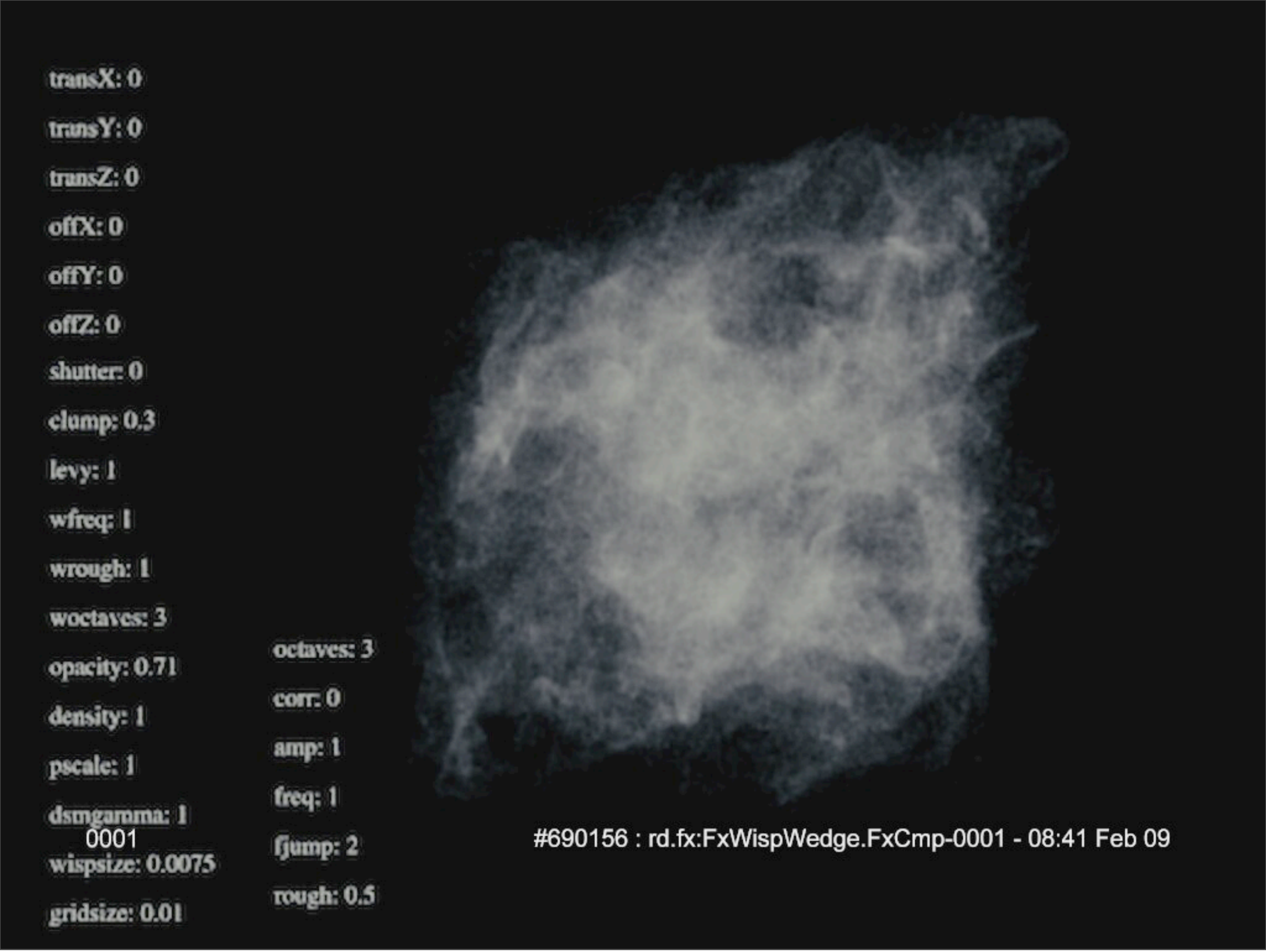
```
// displace again with 3D fractal sum noise
xfsn = fBm( xsphere, ysphere, zsphere );
yfsn = fBm( xsphere + 0.1, ysphere + 0.1, zsphere + 0.1 );
zfsn = fBm( xsphere - 0.1, ysphere - 0.1, zsphere - 0.1 );

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.

Pseudo-code

```
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 );
   }
}
```

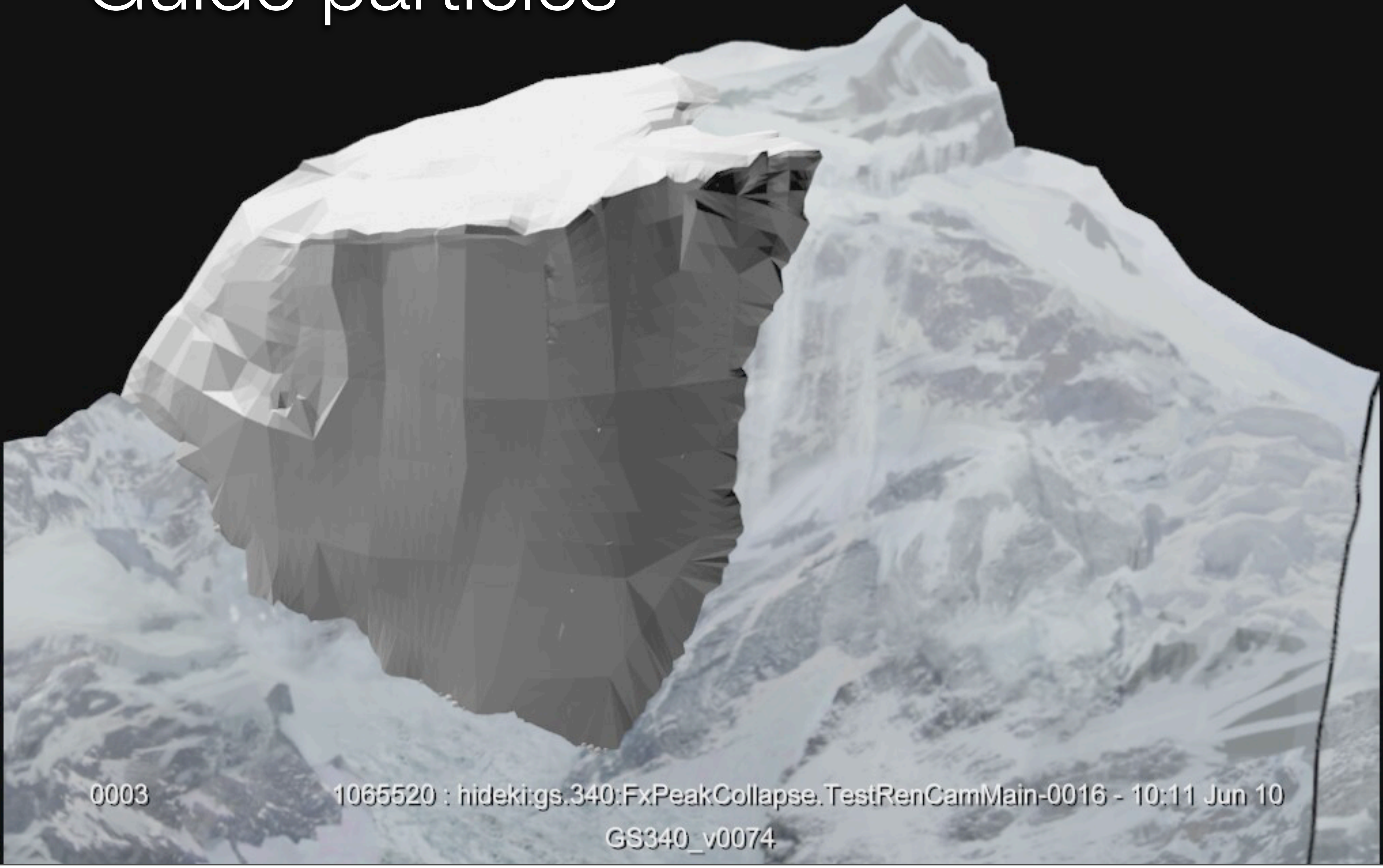

transX: 0
transY: 0
transZ: 0
offX: 0
offY: 0
offZ: 0
shutter: 0
clump: 0.3
levy: 1
wfreq: 1
wrough: 1
woctaves: 3
opacity: 0.71
density: 1
pscale: 1
dsmgamma: 1
0001
wispsize: 0.0075
gridsize: 0.01
octaves: 3
corr: 0
amp: 1
freq: 1
fjump: 2
rough: 0.5
#690156 : rd.fx:FxWispWedge.FxCmp-0001 - 08:41 Feb 09


0001
#457678 : rd.jamesa:MasterFeltTest.Boat-0008 - 01:49 Oct 03

0188

#560449 : SR_266_039:Cmp-0054 - 09:44 Apr 06

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