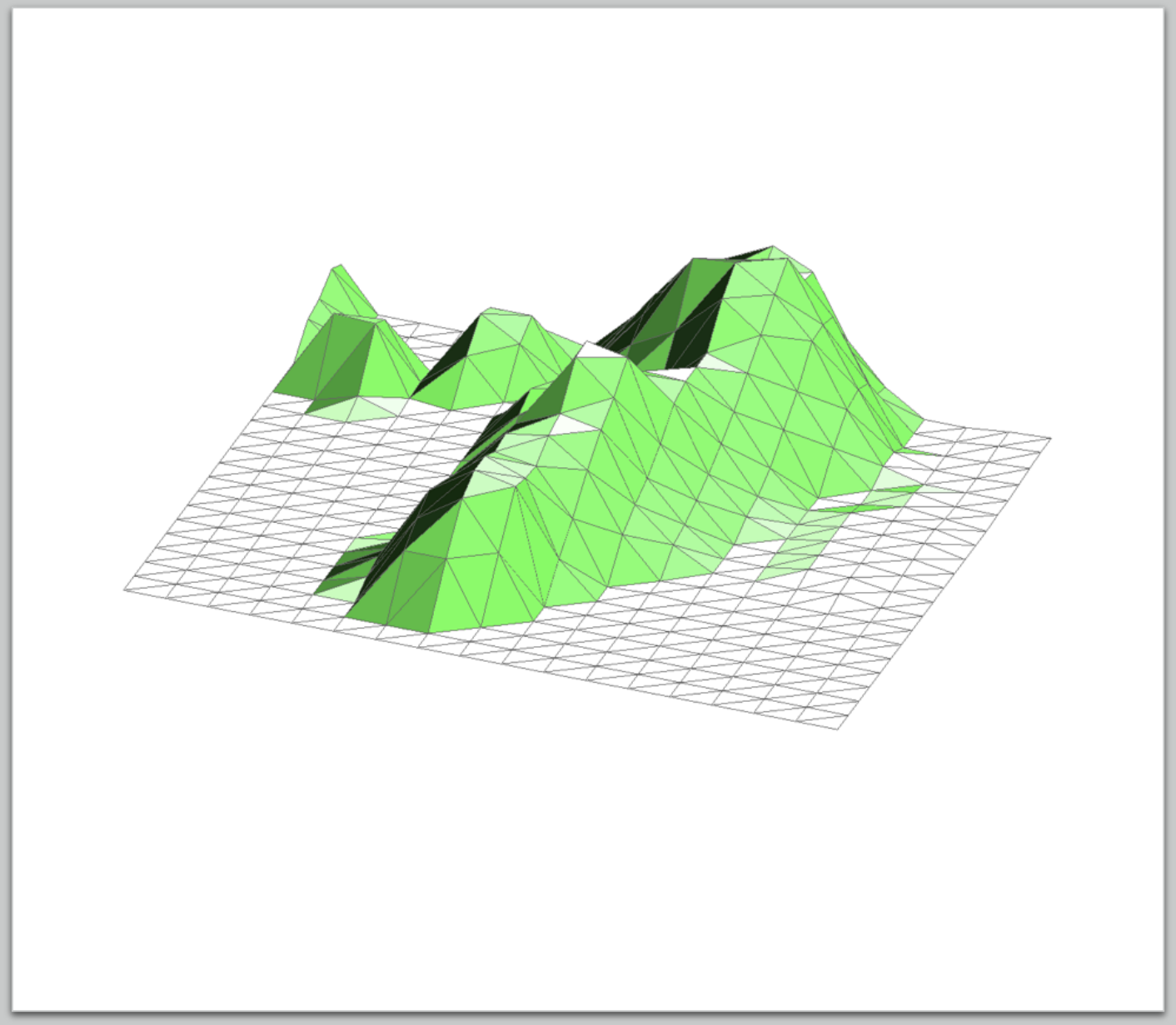
19- Volume Rendering

Dr. Hamish Carr & Dr. Rafael Kuffner dos Anjos

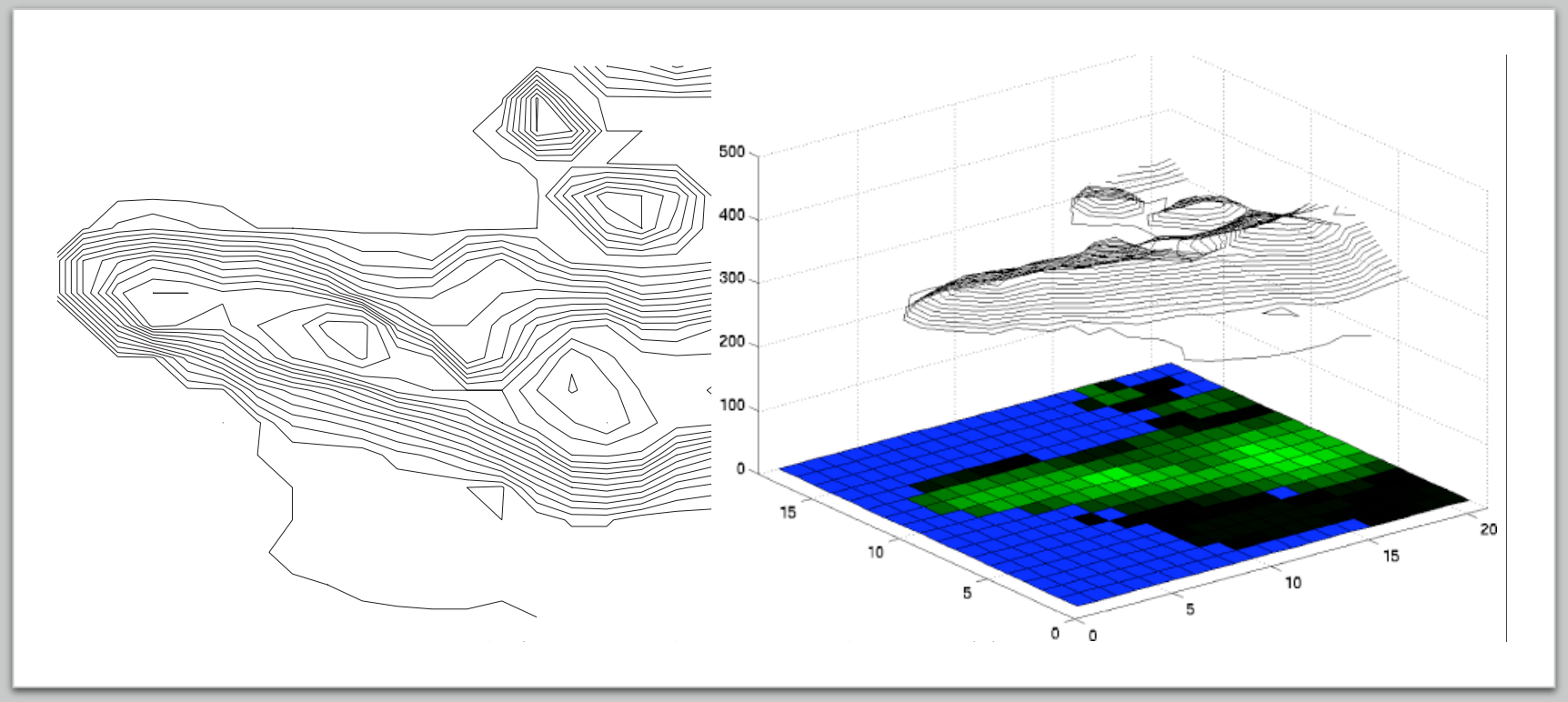
$f: \mathbb{R}^2 \rightarrow \mathbb{R}$: Height Field

- For every point in the plane, define a *height*
- The graph is then a *terrain*
  - i.e. a surface in a 3D embedding space

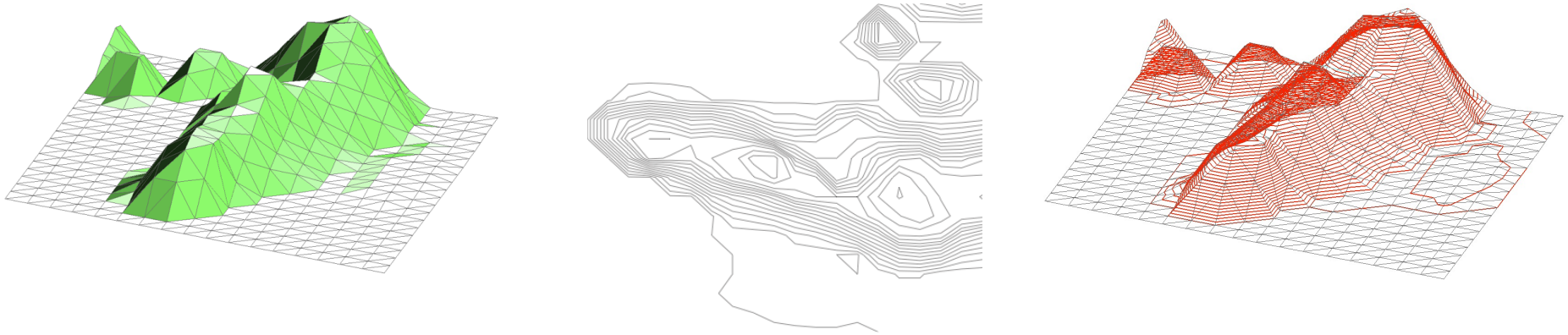


Contours

- We can define curves using contours
- Let an artist design the landscape
- Pick a threshold, generate a curve



Manifold Dimension



- f is a 2-manifold embedded in 3-D
- It's contours are 1-manifolds
  - embedded in 2-D
  - or in 3-D
- What happens if we add a dimension?

Implicit (Blobby) Surfaces

- Define a function $f: \mathbb{R}^3 \rightarrow \mathbb{R}$
  - Often a sum of Gaussian distributions
  - It's a 3-manifold embedded in 4-D
  - It's contours are 2-manifolds in 3-/4-D
- Choose a threshold h
- Extract the level set / contours
  - They are *guaranteed* to be manifold

Implicit Surfaces

- We can use many types of functions:
  - CT/MRI scans
  - Numerical simulations
  - Mathematical models – eg. sum of Gaussians
  - Distance fields
  - *any* property we can compute in 3-D
- In practice, we need to start with a 3-D mesh

Meshes in 3D

- In 2-D, we used triangles and squares
- In 3-D, we use tetrahedra and cubes
  - also pyramids, octahedra & triangular prisms
- Interpolation follows the same rules as in 2-D
  - geometric interpolation (meshes)
  - kernel filters (sampling theory)

Isosurfaces

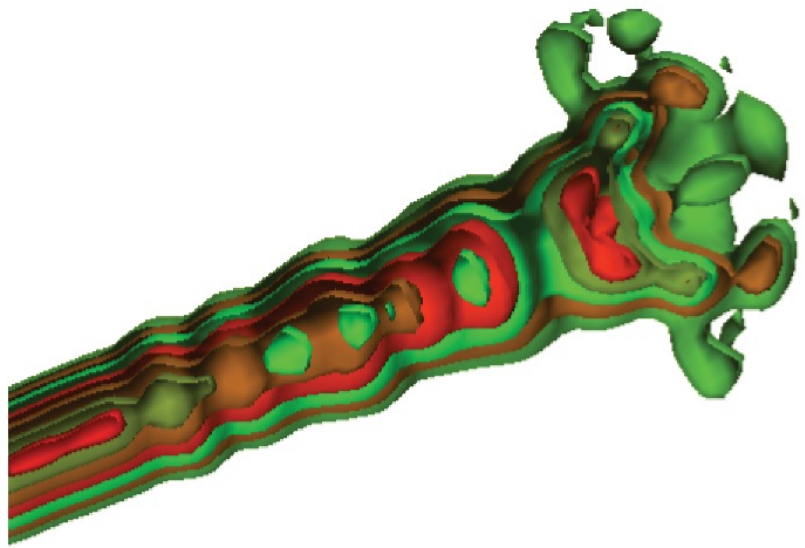
- Contours in 3D
- Basically the same as we saw in 2D

$$f^{-1}(h) = \{(x, y, z) : f(x, y, z) = h\}$$

- 2-manifolds embedded in 3 or 4 space
- But we can't usually show multiples
  - because we've run out of dimensions

Result

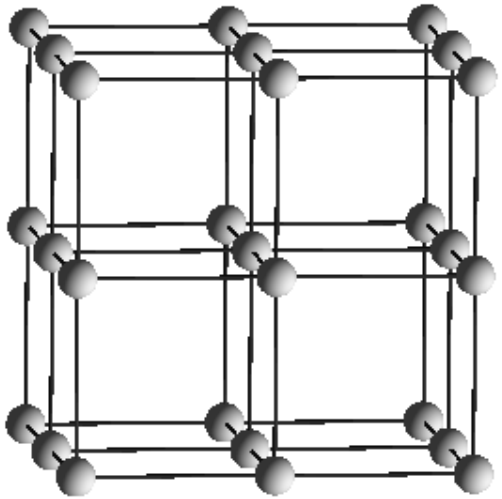
Method:	BCC	Data/lobster.dat.raw.bc	
Cells:	160372	Tris/Cell:	1.291
Triangles:	207034	Isovalue:	112.500



An Example

Cubic Meshes

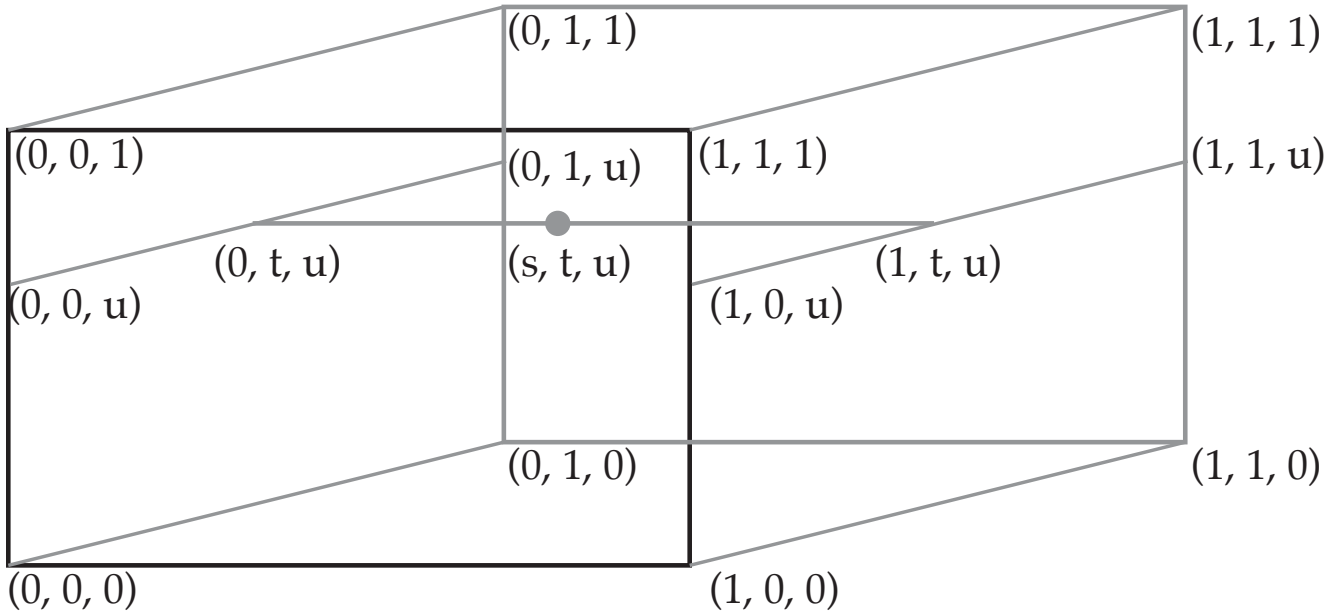
Samples



Mesh

- By far the most common case
  - Generalisation of squares in 2-D

Trilinear Interpolation



- Repeat linear interpolation in 3 dimensions

Trilinear Interpolation

$$f(x,y,z) = a\ xyz + b\ yz + c\ xz + d\ xy + ex + fy + gz + h$$

$$a = b_{111} - b_{110} - b_{101} - b_{011} + b_{100} + b_{010} + b_{001} + b_{000}$$

$$b = b_{011} - b_{010} - b_{001} + b_{000}$$

$$c = b_{101} - b_{100} - b_{001} + b_{000}$$

$$d = b_{110} - b_{100} - b_{010} + b_{000}$$

$$e = b_{100} - b_{000}$$

$$f = b_{010} - b_{000}$$

$$g = b_{001} - b_{000}$$

$$h = b_{000}$$

Computing Normal

- Normal vector is based on gradient
  - always perpendicular to contours
  - describes steepest ascent
  - but outside is typically low-valued
  - use *negative* gradient as normal

$$\vec{n} = -\nabla f(x)$$
$$= (-\frac{\delta f}{\delta x}, -\frac{\delta f}{\delta y}, -\frac{\delta f}{\delta z})$$

Central Differencing:

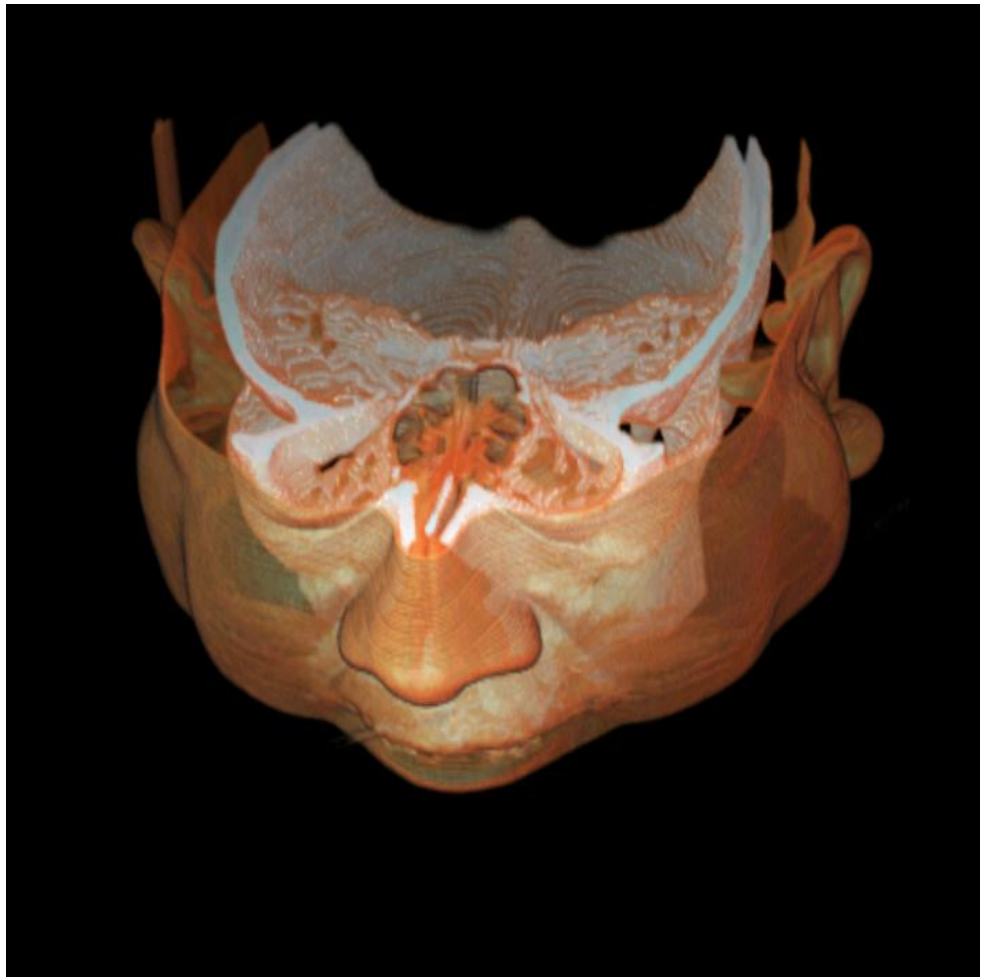
- Approximation of gradient:

$$\vec{n}_{i,j,k} \approx (-\frac{f(x_{i+1},y_j,z_k)-f(x_{i-1},y_j,z_k)}{2}, -\frac{f(x_i,y_{j+1},z_k)-f(x_i,y_{j-1},z_k)}{2}, -\frac{f(x_i,y_j,z_{k+1})-f(x_i,y_j,z_{k-1})}{\delta z},$$

- Compute at vertices of cell
- Interpolate along edges

Direct Volume Rendering

courtesy T. Möller, SFU



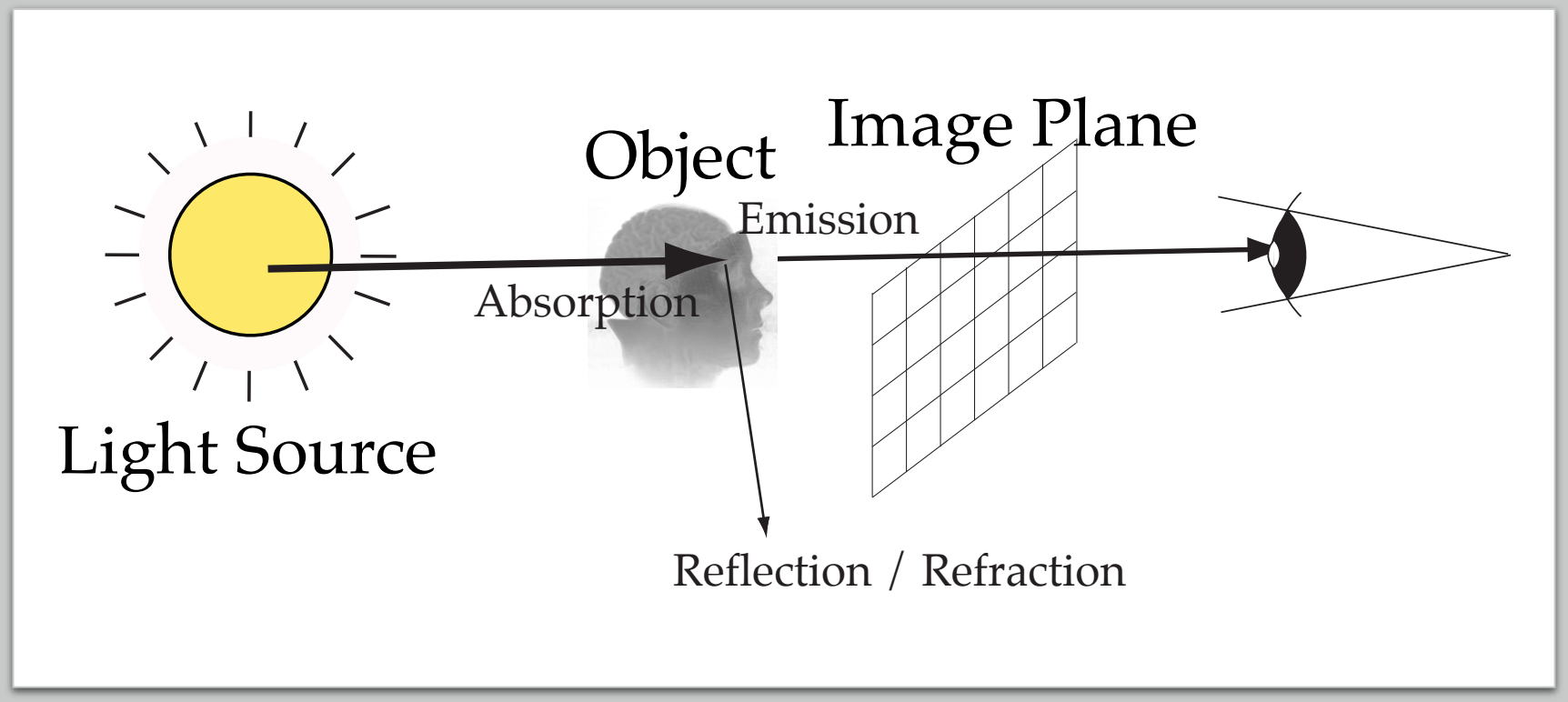
Created with 3D doctor

Direct Volume Rendering

- 3D equivalent of colour mapping
- Define a transfer function:
  - maps from data to colour opacity
- Then compute light transport through data
  - i.e. a computational X-ray
- Or think of it as luminous jelly

Modelling Transmission

- No longer modelling *surfaces*
- Modelling *diffuse* internal reflection
- We get light added *and* subtracted



Density Clouds

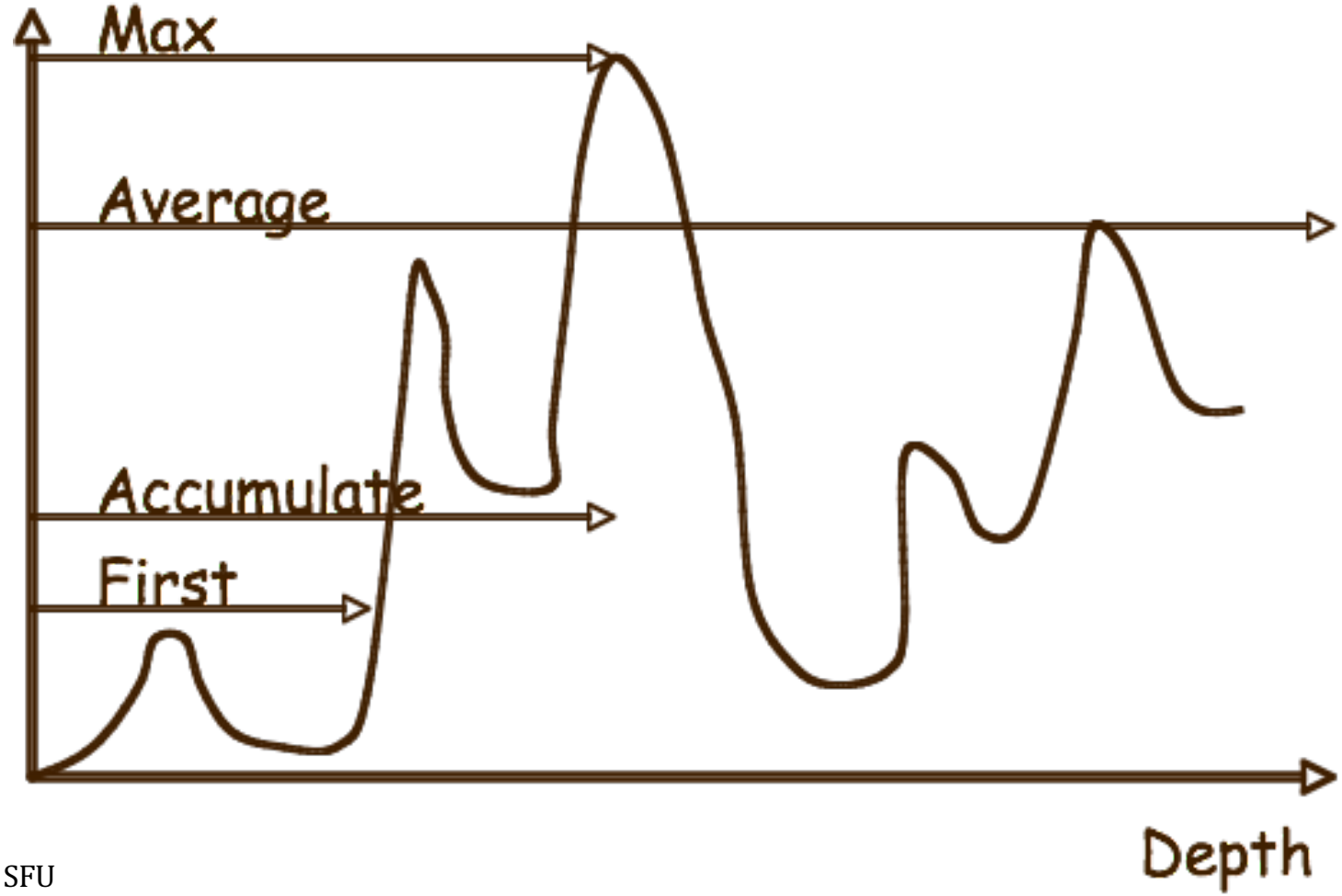
- Photons are blocked by absorption
- Probability depends on density of material
  - i.e. how many particles are in the way
- Measured as optical density
  - # of particles / mm
  - E.g. 5mm of lead might block 90% of photons
  - How much would 10mm block?

Volume Rendering Integral

$$I_\lambda(x,r) = \int_0^L C_\lambda(s)m(s)e^{-\int_0^2 m(t)dt}\,ds$$

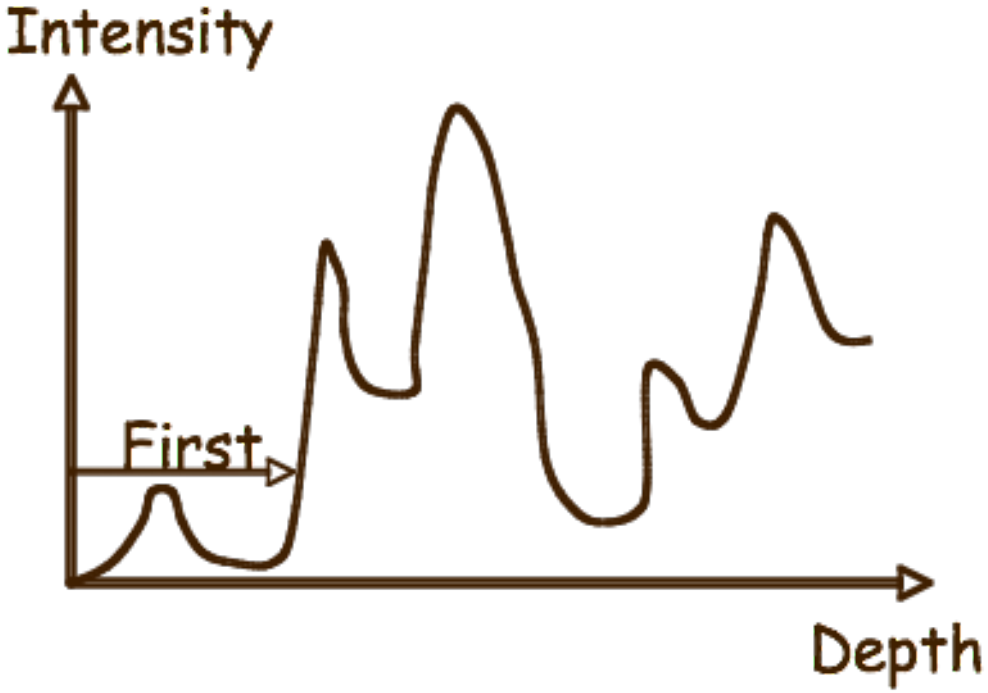
- Due to Kajiya, Sabella, Max (1986) (3 papers!)
  - λ is the wavelength of light
  - $x + \vec{r}t$ is a ray of length L through pixel x
  - $C_\lambda(s)$ is the colour contribution at point s
  - $m(s)$ is the mass density at s

TF Design



courtesy T. Möller, SFU

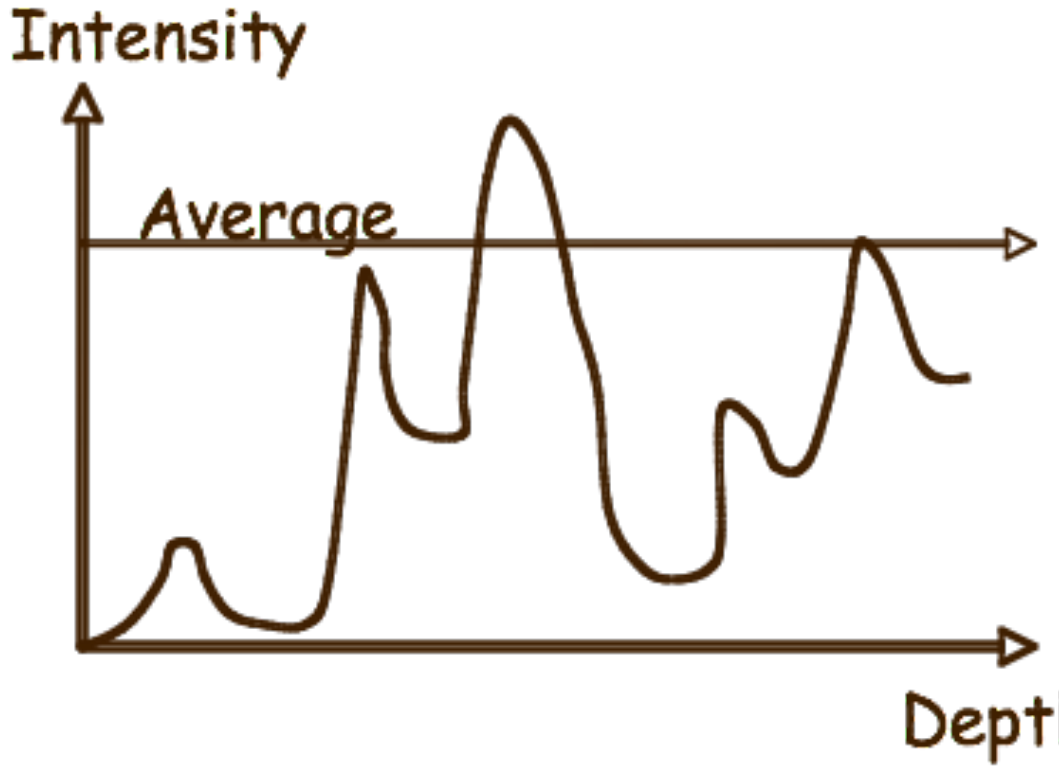
First Intersection

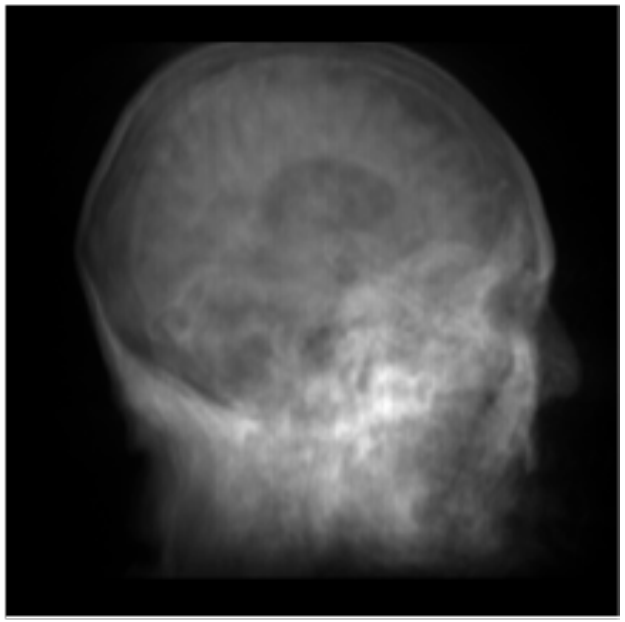


courtesy T. Möller, SFU

- Equivalent to an Isosurface (Tuy&Tuy, 1984)
- 100% opacity at chosen isovalue

Average Intensity

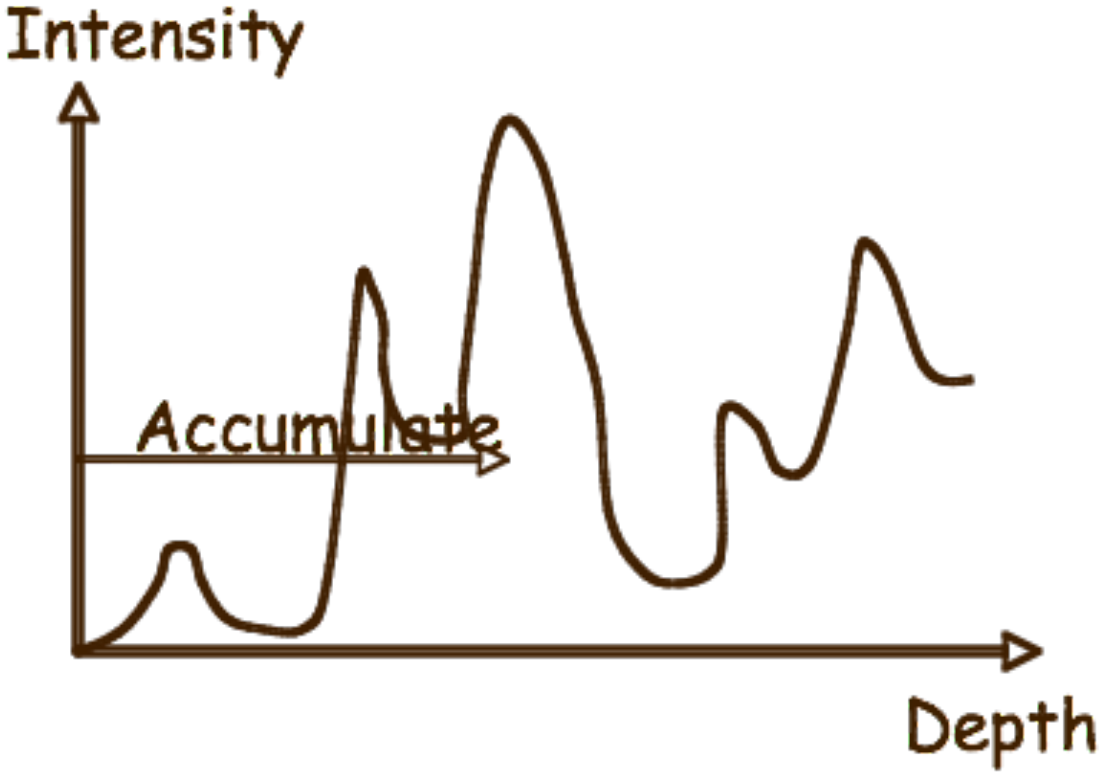


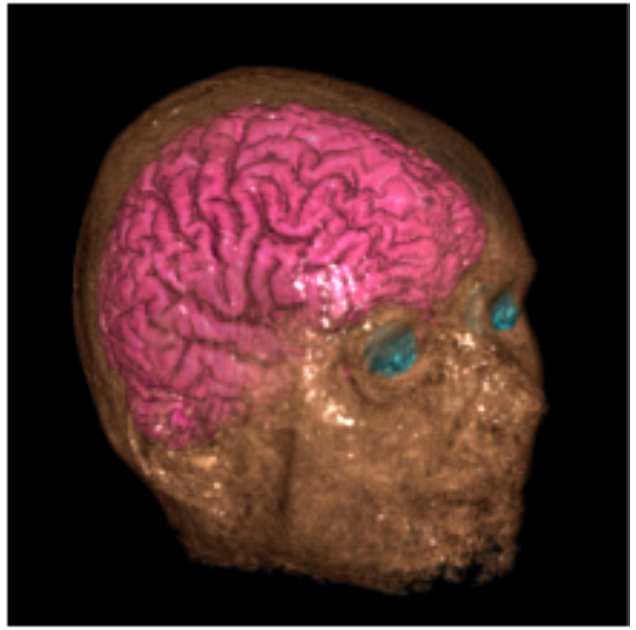


courtesy T. Möller, SFU

- Constant Opacity throughout volume
- Equivalent to an x-ray

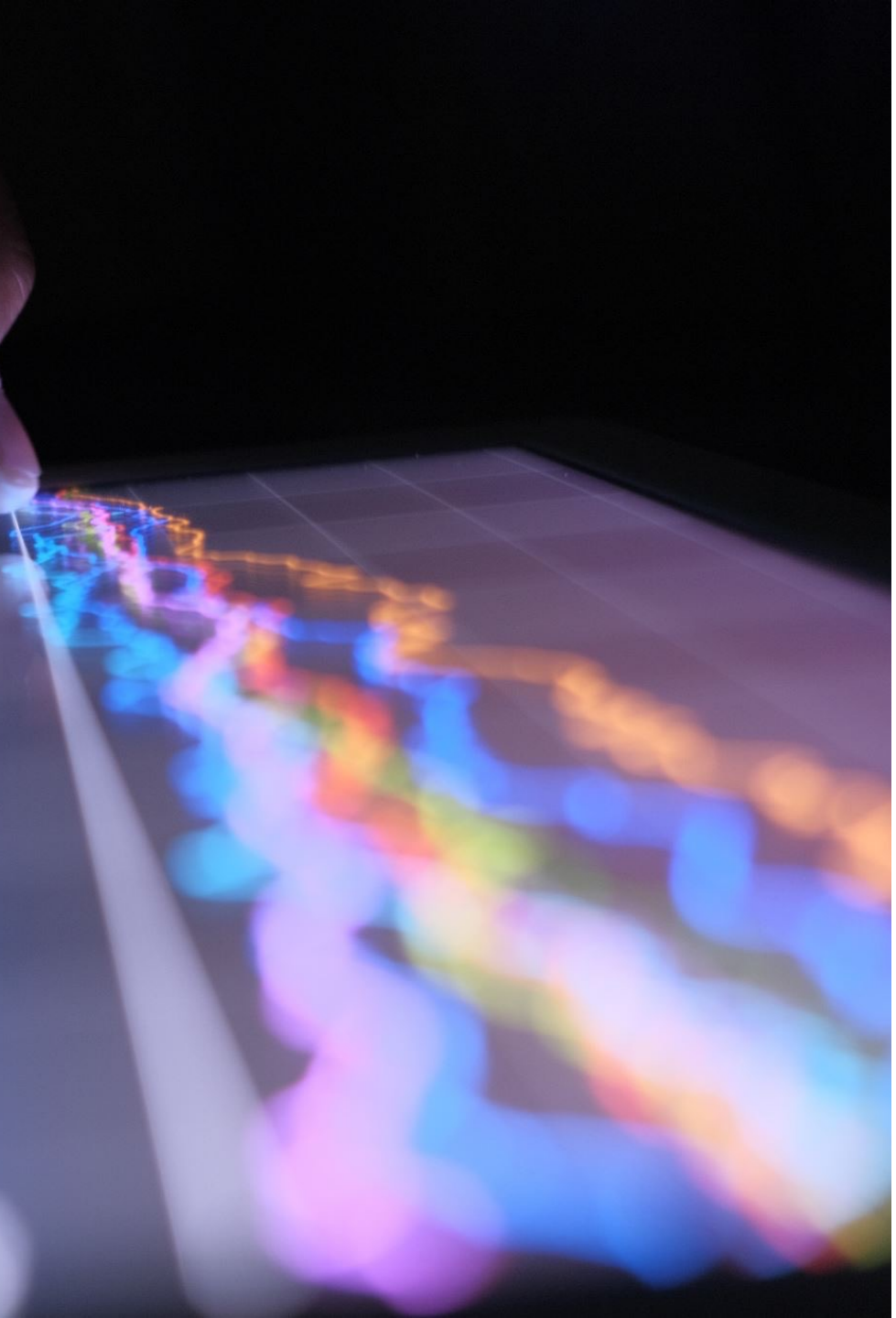
Accumulate





courtesy T. Möller, SFU

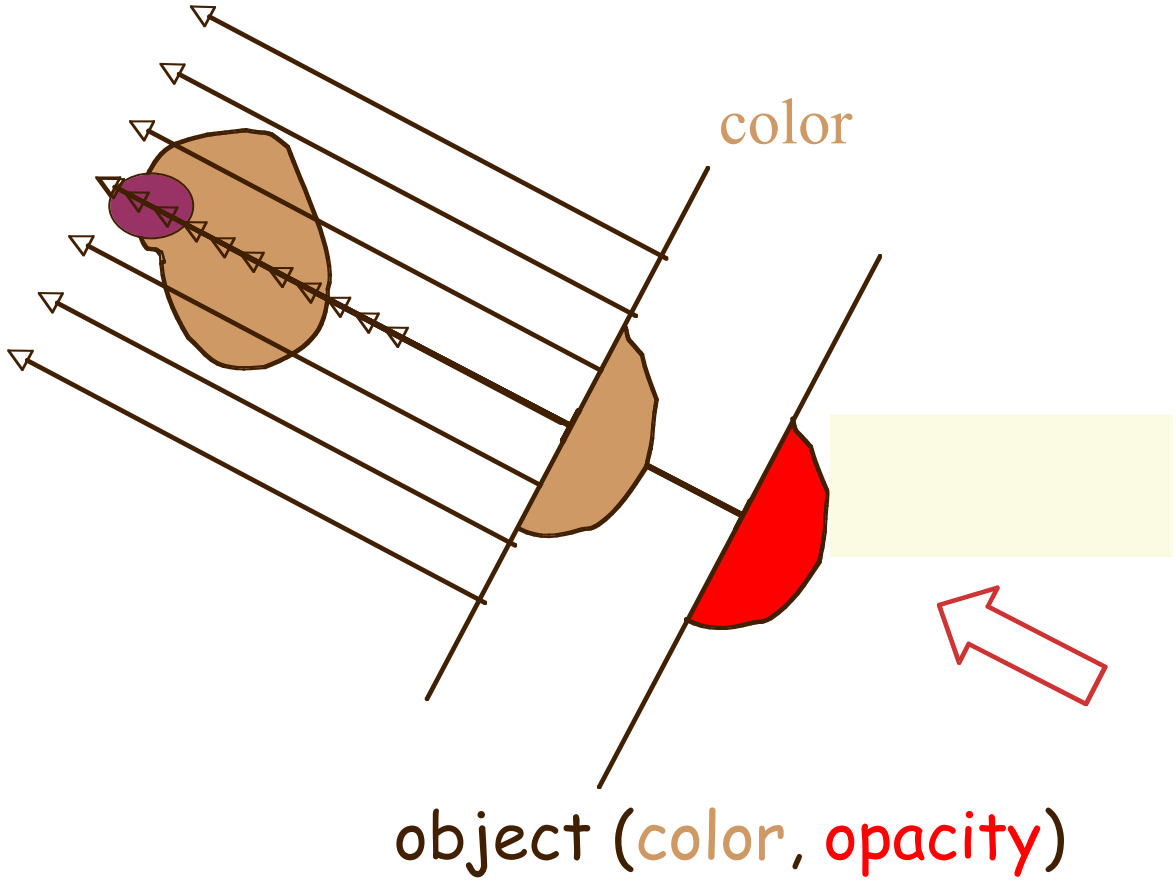
- Use TF to highlight structures
- Levoy, 1988



DVR Ray-tracing

- For each pixel
- Compute corresponding ray in volume
- Walk along ray in small increments
- Computing contribution to integral
- Then display as image
- This makes regular raytracing look cheap!
  - So sci. vis. has *many* hacks/speedups

Example



Pseudocode

```
// loop through all pixels
for (x = 0; x < width; x++)
  for (y = 0; y < height; y++)
    { // for each pixel
    // start with 0 intensity
    float totalIntensity = 0.00;
    // choose two points along the ray
    Point p = unProject(x, y, -1);
    Point q = unProject(x, y, -2);
    // compute the vector for the line
    Vector v = q - p;
    // take very small steps
    tStep = 0.01;
    // iterate along the ray until we get to the other side
    for (t = 0.0; t < maxThicknessOfData; t+=tStep)
      { // for t
      // compute where this point is
      Point r = p + v*t;
      // look up the value, then use that to find opacity
      alpha = TF_alpha(r,tStep);
      // also use it to look up intensity
      intensity = TF_intensity(r);
      // compute the opacity & update
      totalIntensity = totalIntensity * (1-alpha) + intensity * alpha;
      } // for t
    // when we're done, save the pixel intensity
    setPixel(x,y,totalIntensity);
    } // for each pixel
```

Volume rendering algorithm

- This is tracking the integration from the back forwards
- If you want to have a lightsource at the side,
- Modify to account for the reflection of the particles.
- And for that you will need everything we talked about before, and a normal vector.
- Scattering, montecarlo etc…

Applications to Rendering

- Mist
- Fog
- Smoke
- Explosions
- Particle Systems
- X-rays
- Visualisation in general