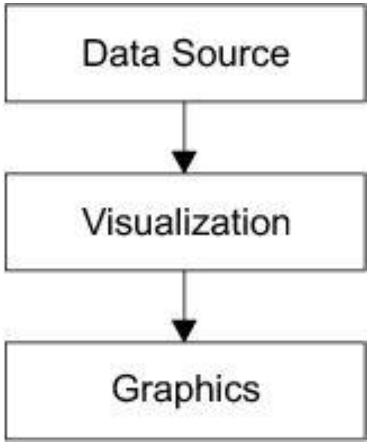
Volume Rendering

Georgios Papaioannou - 2010

Introduction – Data Visualization

- Visualization and computer graphics:
  - Visualization is a procedure for mapping data and calculations to meaningful visual representations that are easy to grasp and interpret
  - Visualization algorithms:
    - Create a visualization object from the raw data
    - Specify its display parameters
  - Graphics algorithms implement these specifications & produce images



Why visualize the data?

- The human visual system can rapidly make meaningful associations of intensity and shape with useful values and their relationship
- Example (raw data):

23	24	25	27	26	25	25	24	24
24	26	28	30	29	27	26	28	31
26	28	29	31	32	29	30	32	36
26	27	30	32	33	34	35	38	41
27	28	28	32	34	35	37	41	42
27	28	31	33	36	38	40	42	43
28	29	32	32	35	37	41	43	44
30	33	33	34	36	38	41	42	44
32	34	27	29	40	42	43	44	45

Why visualize the data? (2)

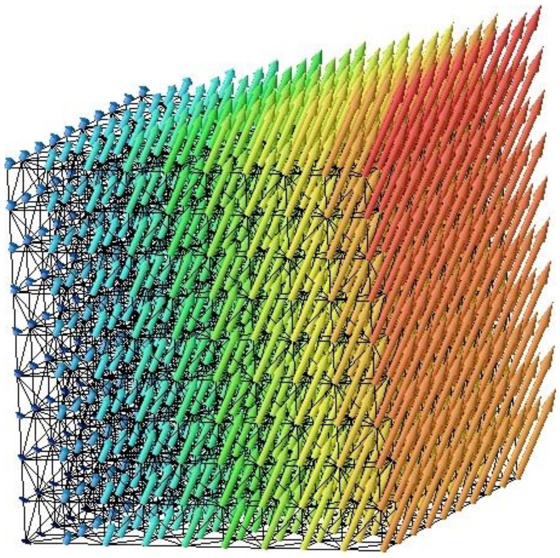
- Example (intensity-coded data):

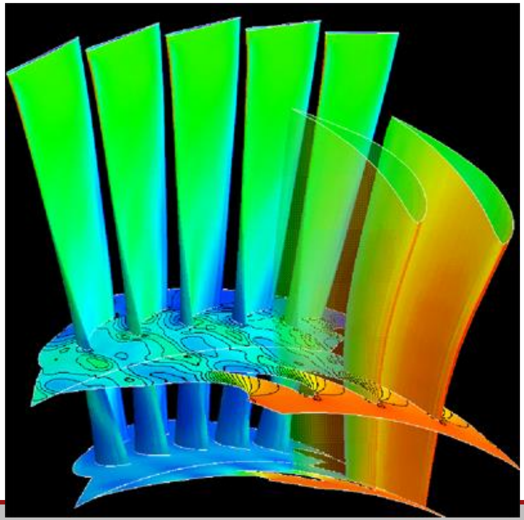
23	24	25	27	26	25	25	24	24
24	26	28	30	29	27	26	28	31
26	28	29	31	32	29	30	32	36
26	27	30	32	33	34	35	38	41
27	28	28	32	34	35	37	41	42
27	28	31	33	36	38	40	42	43
28	29	32	32	35	37	41	43	44
30	33	33	34	36	38	41	42	44
32	34	27	29	40	42	43	44	45

22-25	26-29	30-33
34-37	38-41	42-45

Data Representation

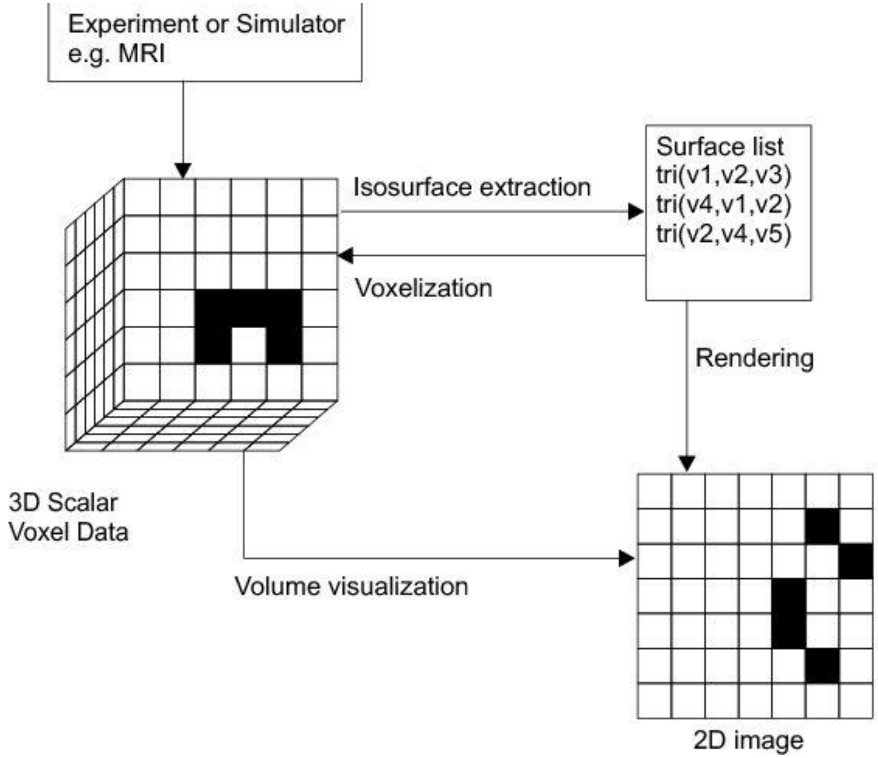
- Data attributes:
  - Dimensionality
  - Scale
  - Regions of Interest (ROI)
  - Structure
  - Critical points
  - Type
  - Sampling type and quantization





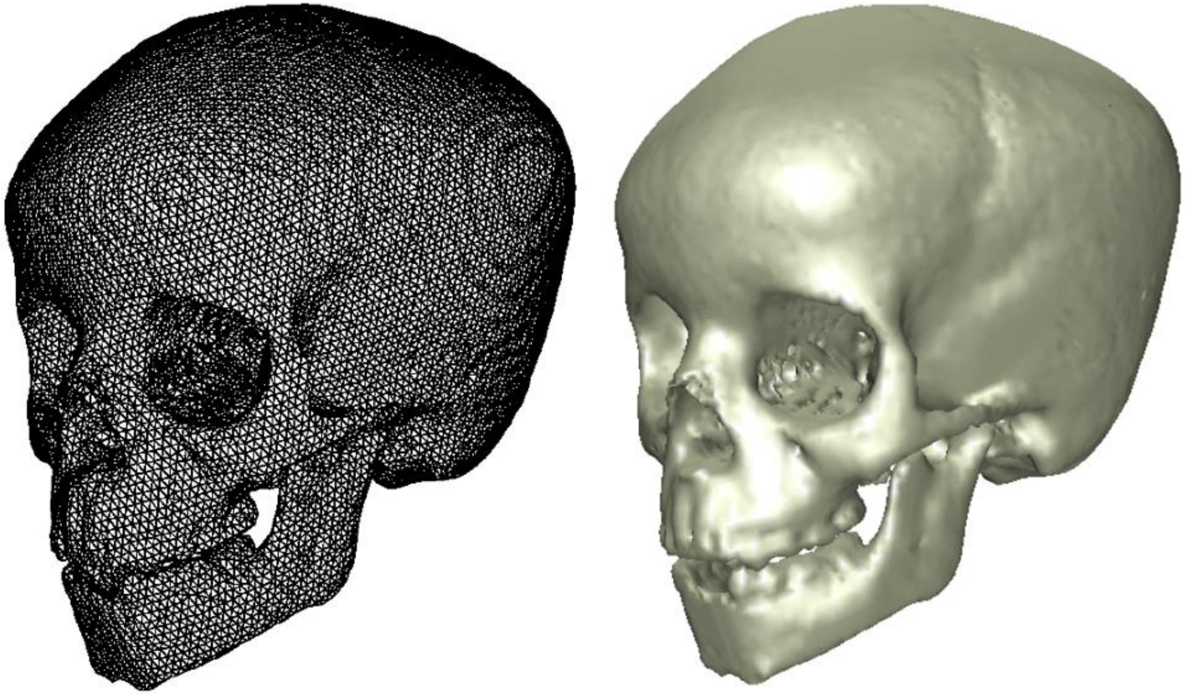
Going 3D: Scalar Data Visualization

- Two major methods:
  - Isosurface visualization
  - Direct volume rendering



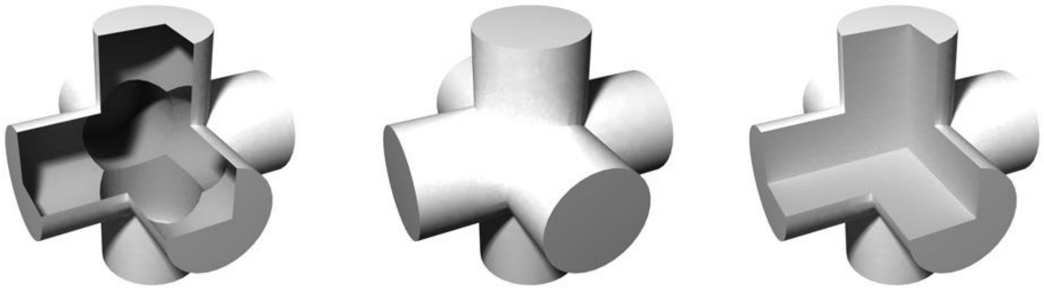
Isosurfaces

- Isosurface is a hyperplane embedded in an N-dimensional space that corresponds to a constant scalar value



Isosurfaces vs Direct Volume Rendering

- Isosurfaces:
  - Create sharp renderings
  - But: only part of the information present in the scalar field is visible on the isosurfaces
- Isosurface rendering requires:
  - Either surface extraction algorithms (and direct rendering)
  - Or direct isosurface rendering (ray tracing, volume splatting/slicing)



Isosurface Extraction Algorithms

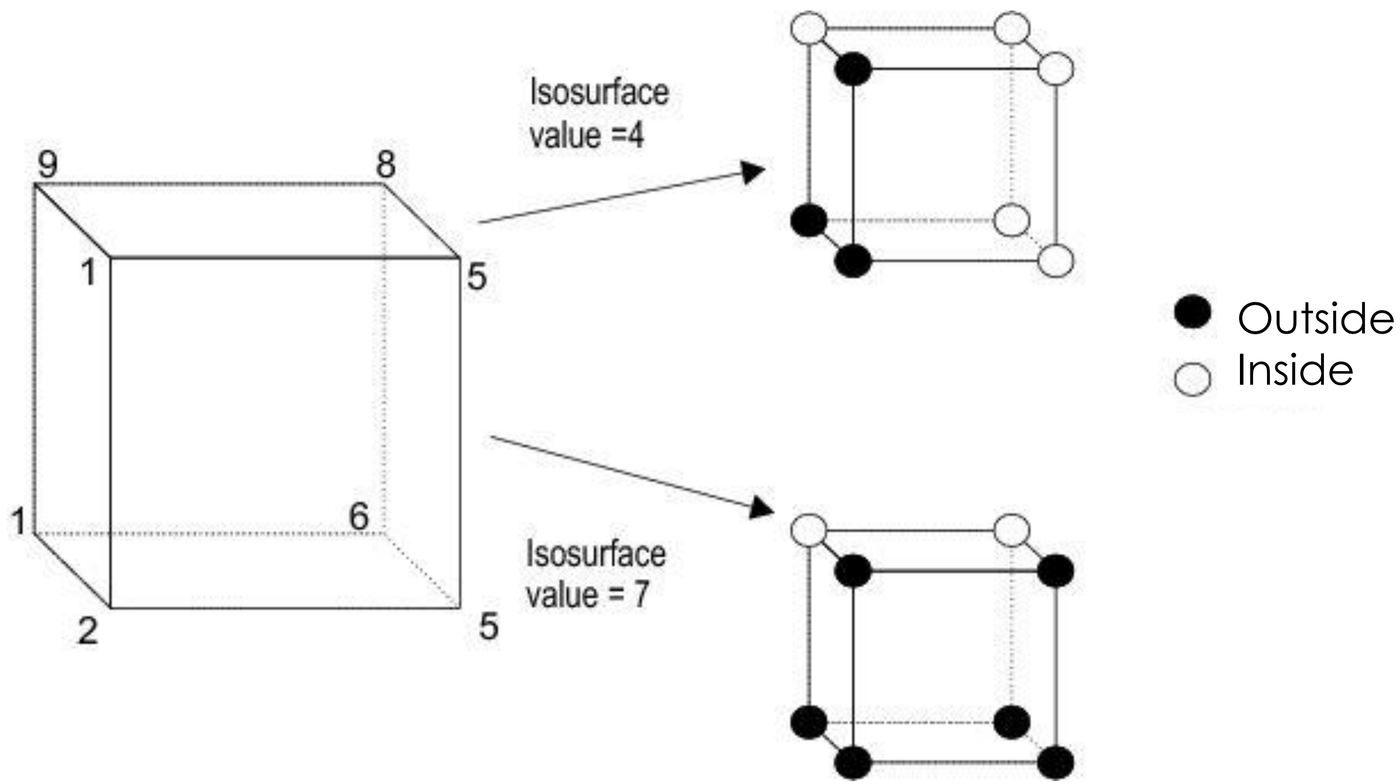
- Often data contain clusters of values, which can be separated by surfaces
- Isosurface algorithms determine these separating surfaces
- Input: surface density thresholds
- Once these isosurfaces are established:
  - Easy to display via standard graphics techniques (polygons)

Marching Cubes (MC) Algorithm

- Input: Scalar volume data set and isosurface threshold
- Output: isosurface polygons
- For every voxel (cube):
- Compare the values at its 8 vertices to the threshold
- Label the vertex as 1 (inside, smaller than isosurface value) or 0 (outside, greater than isosurface value)
- Concatenate all labels and use descriptor to index a table of pre-computed surface-cube intersections

Marching Cubes (MC) Algorithm (2)

- Segmentation example:



Marching Cubes (MC) Algorithm (3)

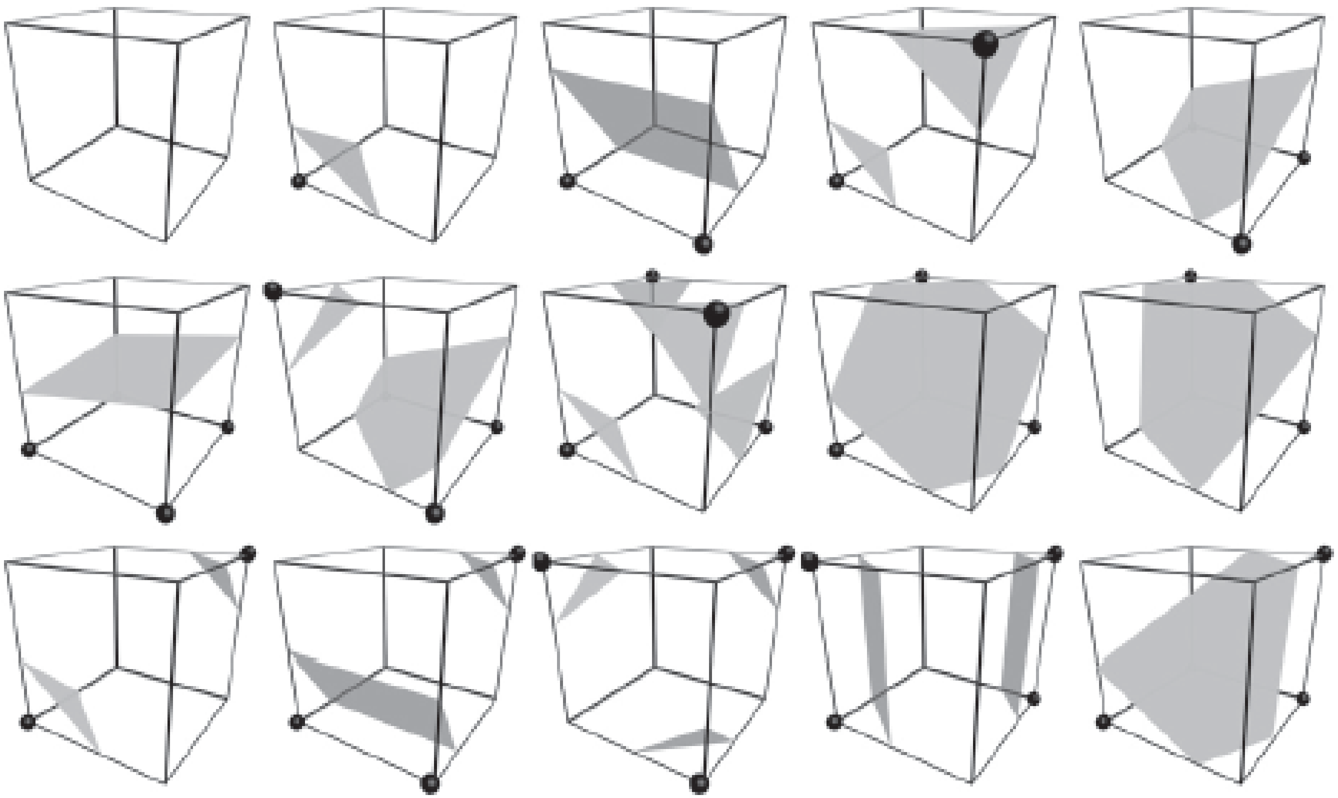
```
for (i,j,k voxels):
    L1=Segment (i,j,k);
    L2=Segment (i+1,j,k);
    ...
    L8=Segment (i+1,j+1,k+1);
    // string/binary concatenation operator
    index=L1++L2++L3++L4++L5++L6++L7++L8;
    // locate corresponding normalized combination (rotated version)
    bindex=MatchSurfaceForm(index);
    // return relative rotation transformation to normalized form
    transform=MatchSurfaceTransform(index);
    // retrieve corresponding (rotated) polygons
    polygons = PrecomputedSurfaces(bindex,transform);
    // ... and adjust edge vertices to fall on isosurface
    for (p=0; p<polygons.size(); p++)
        ComputePreciseEdgePosition(p,voxel(i,j,k));
    for (p=0; p<polygons.size(); p++)
        ComputeNormal(p, voxel(i,j,k));  // also compute vertex normals
```

MC – Pre-computed Voxel Polygons

- 28 ways to label vertices of a cube:
  - Requires 256 pre-computed surface-cube intersection patterns
  - Reduced to just 15 by taking advantage of:
    - Mirror symmetry
    - Rotational symmetry
    - Inside/outside symmetry

MC – Pre-computed Voxel Polygons (2)

- 15 intersection patterns provide the topology of the polygonal intersection surface wrt the cube edges

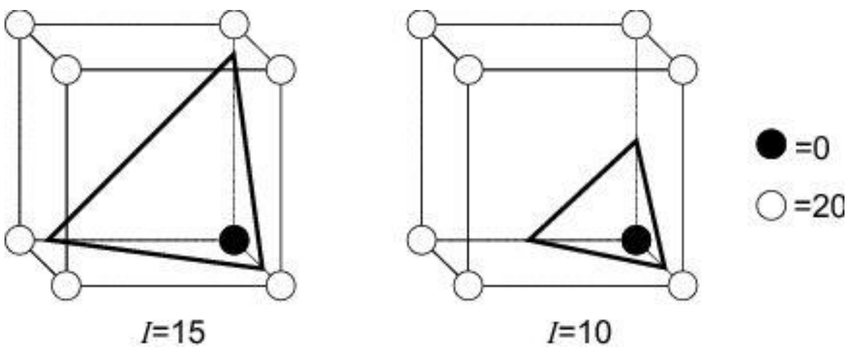


MC – Edge Vertex Adjustment

- The exact points of intersection along each cube edge are determined by interpolation:
  - If the edge vertices $\mathbf{v}_1,\mathbf{v}_2$ have associated field values $val(\mathbf{v}_1)$, $val(\mathbf{v}_2)$ and the isosurface threshold $thres$ → intersection point p can be expressed as:

$$t = \frac{thres - val(\mathbf{v}_1)}{val(\mathbf{v}_2) - val(\mathbf{v}_1)},$$

$$\mathbf{p} = \mathbf{v}_1(1-t) + \mathbf{v}_2 t$$



MC – Normal Vectors

- Normal vectors can be calculated at voxel vertices by the volume density gradient (first order derivatives):

$$g_x(i,j,k) = \frac{v(i+1,j,k) - v(i-1,j,k)}{\Delta x},$$

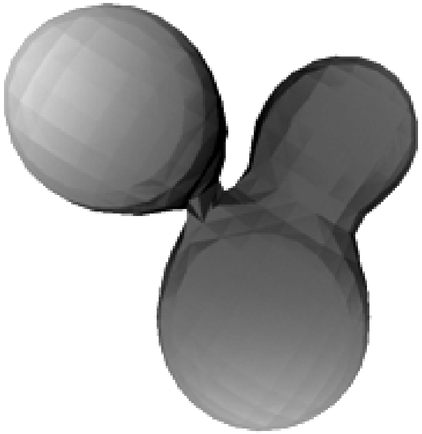
$$g_y(i,j,k) = \frac{v(i,j+1,k) - v(i,j-1,k)}{\Delta y},$$

$$g_z(i,j,k) = \frac{v(i,j,k+1) - v(i,j,k-1)}{\Delta z}$$

- They are interpolated to obtain the isosurface polygon vertex normals

MC - Comments

- Major disadvantages of MC algorithm:
  - Large number of polygons created for the isosurface
  - This number is not proportional to the isosurface complexity:
    - Depends primarily on the density of the grid
- MC can be fully accelerated by the GPU (see example)



Direct Volume Visualization

- Can be used to render isosurface data but also
- Display transparency-weighted density clouds
- Can use complex shading (shadows, absorption, forward scattering etc)
- Central Techniques to this genre are:
  - Ray marching
  - Volume slicing

Direct Volume Rendering Operations

- Sampling
  - Establishes the sampling pattern and evaluates volume values at sample locations
- Classification
  - Classifies and maps volume data to density and color
- Shading
  - Illuminates the samples. For isosurfaces, the normal vectors are also extracted and used.
- Combination
  - Combines the samples with other samples in the line of sight

Volume Sampling

- Samples are projected on the view plane
- Commonly samples are drawn on the line of sight through each pixel (ray marching)
- The location of the samples is determined by the **rendering algorithm**
- Data samples are interpolated at sample locations from the initial volume data structure.
- Usually, tri-linear interpolation is used, although cubic interpolation is also common

Classification

- Converts scalar values to density and color
  - Density is used to define the transparency of a point.
- More complex classifiers do not just use the local scalar value, but also other features
- Classification can be performed before or after sample interpolation (pre-/post-classification)

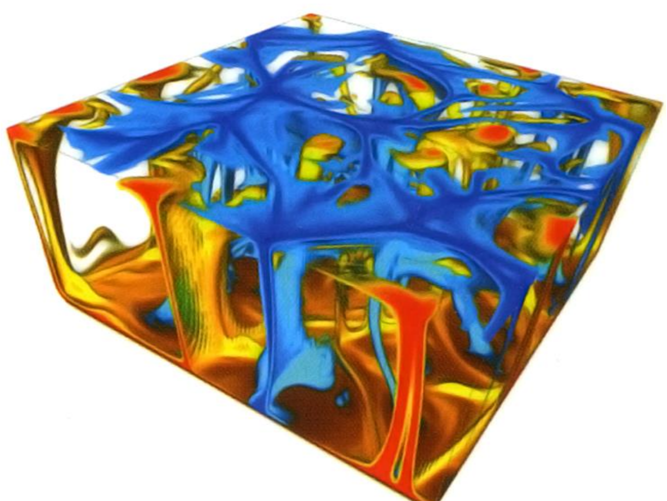
Transfer Functions

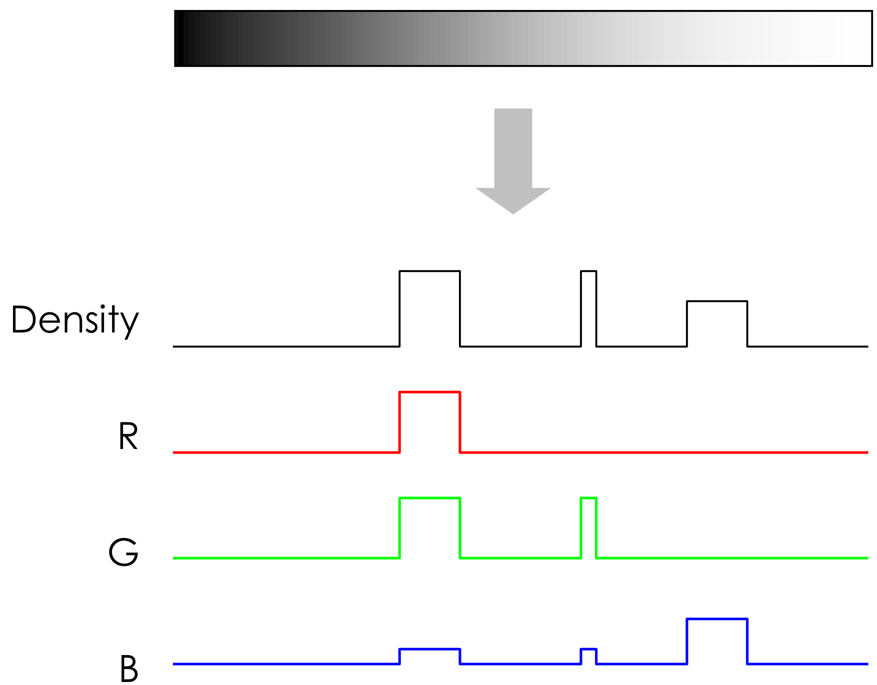
- Volume data express measured quantities or normalized intensity
- Not always adjusted to the visible color range
- We need to highlight and visualize only certain intensity ranges (as in isosurface rendering)
- We need to enhance contrast for clarity

- Transfer functions map the scalar data values to volume density and color, in order to enhance the useful information

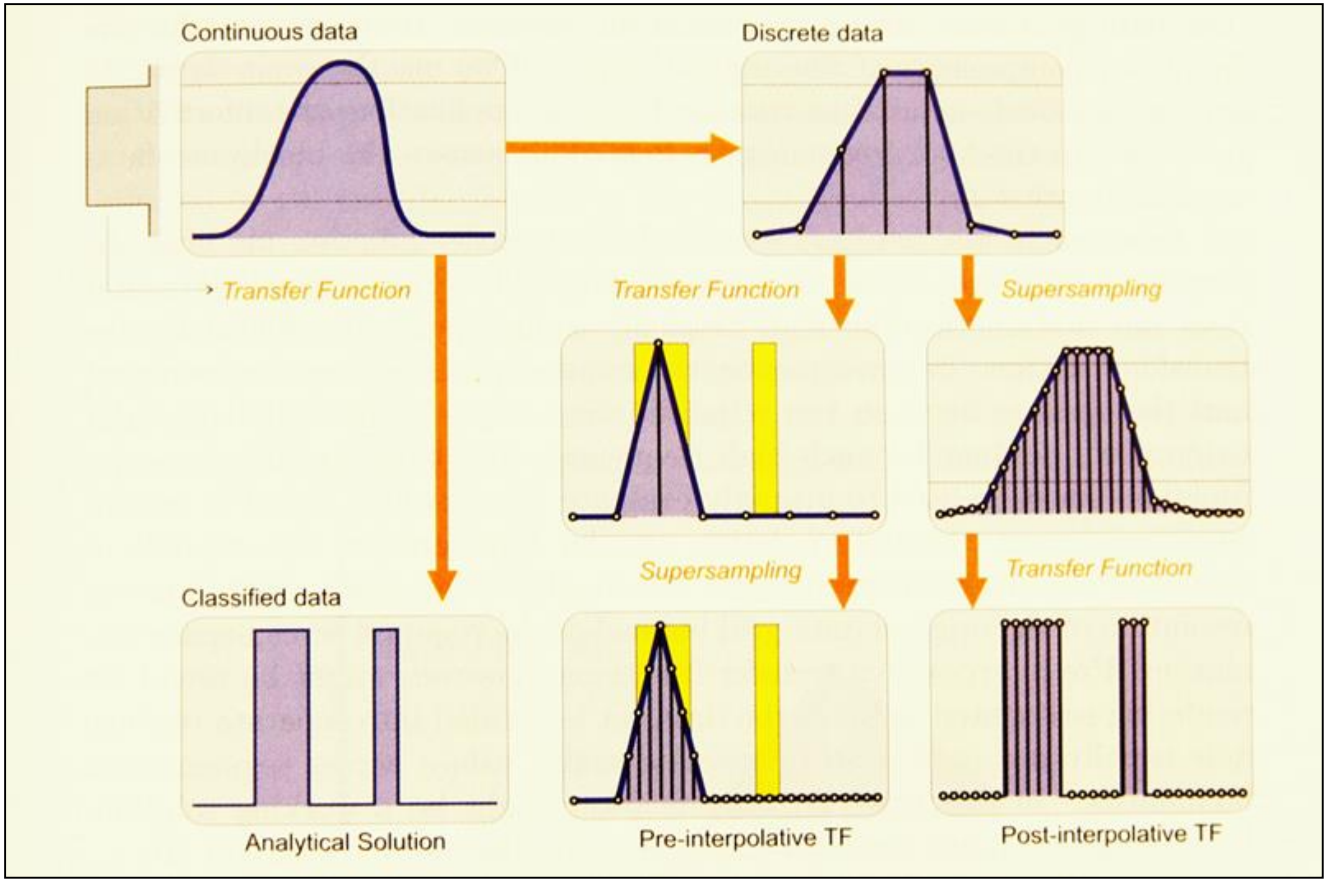
Transfer Functions (2)

- Density and color are usually separately mapped and encoded as RGBA values
- Any function or user-defined curve can be used
- Common functions:
  - Step functions
  - Sigmoid functions





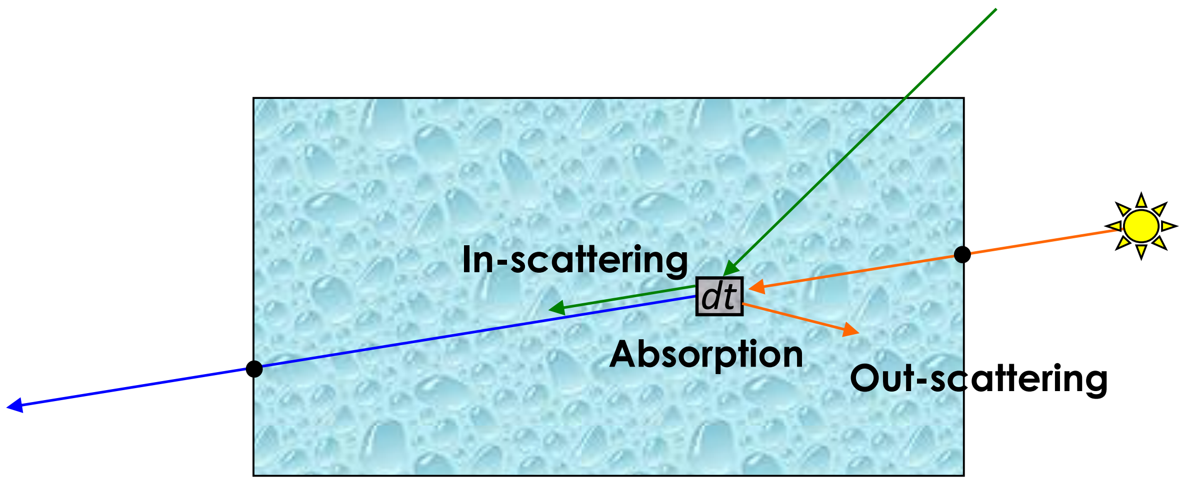
Pre-classification vs Post-classification



Light Propagation Equations

- 4 phenomena affect light traveling through a medium:
  - Absorption
  - Out-scattering
  - Emission
  - In-scattering

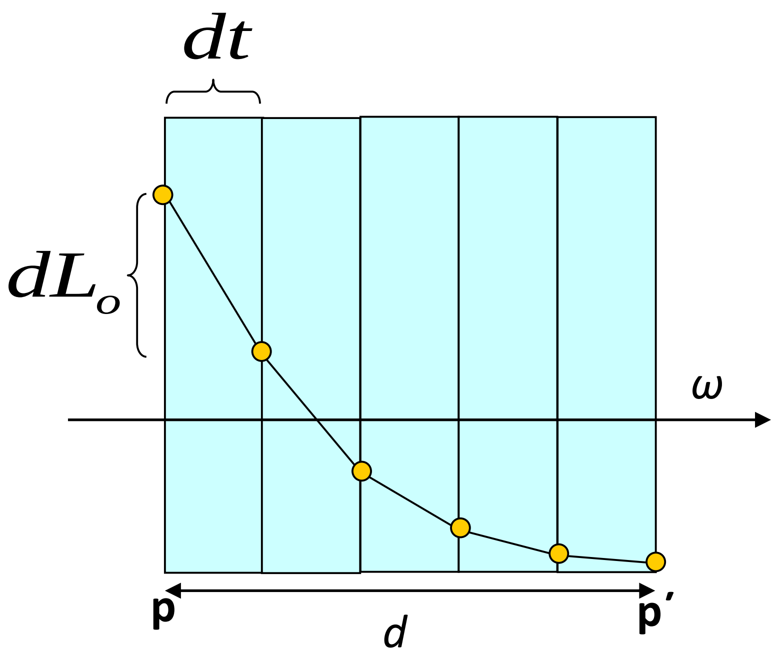
Attenuation



Attenuation

$$L_o(\mathbf{p},\omega) - L_i(\mathbf{p},\omega) = dL_o(\mathbf{p},\omega) \Rightarrow \frac{dL_o(\mathbf{p},\omega)}{dt} = -\sigma(\mathbf{p},\omega)L_i(\mathbf{p},-\omega)$$

$$\sigma(\mathbf{p},\omega) = \sigma_a(\mathbf{p},\omega) + \sigma_s(\mathbf{p},\omega)$$



Transmittance:
Fraction of light transmitted from **p** to **p′**

$$T_r(\mathbf{p} \to \mathbf{p}') = e^{-\int_0^d \sigma(\mathbf{p}+t\omega,\omega)dt}$$

Beer's Law

- For constant σ (homogeneous medium), transmittance becomes:

$$T_r(\mathbf{p} \rightarrow \mathbf{p}') = e^{-\sigma d}$$

- If absorption is constant along small ray segments: ,from Beer's law and the definition of transmittance we get:

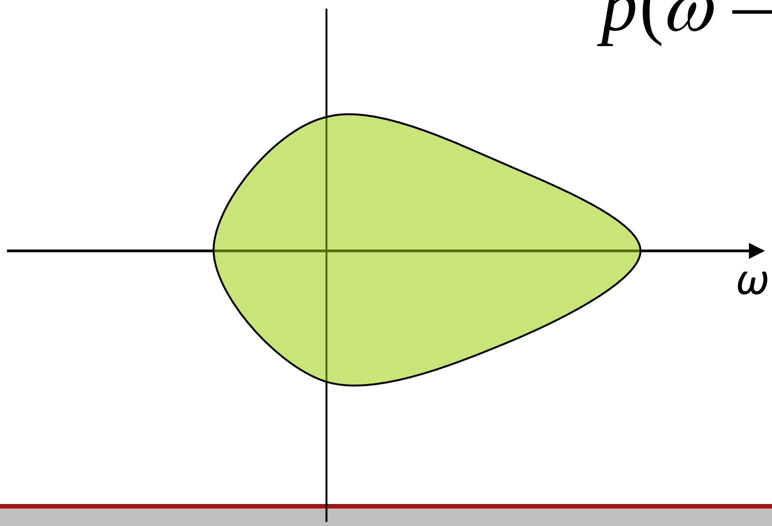
$$T_r(\mathbf{p}_1 \rightarrow \mathbf{p}_N) = e^{-(\sigma_1 d_1 + \sigma_2 d_2 + \ldots + \sigma_{N-1} d_{N-1})} \Leftrightarrow$$

$$T_r(\mathbf{p}_1 \rightarrow \mathbf{p}_N) = \prod_{i=1}^{N-1} T(\mathbf{p}_i \rightarrow \mathbf{p}_{i+1})$$

In-scattering – Phase Functions

- The directional distribution of scattered light at a point is called a **phase function**.
- It is similar to the BSDF but expresses the probability that light from ω is deflected towards ω' :

$$p(\omega \rightarrow \omega'): \qquad \int_S p(\omega \rightarrow \omega')\, d\omega' = 1$$



In-scattering – Phase Functions (2)

- Popular phase functions:
  - Isotropic

$$p_{isotropic}(\omega \to \omega') = \frac{1}{4\pi}$$

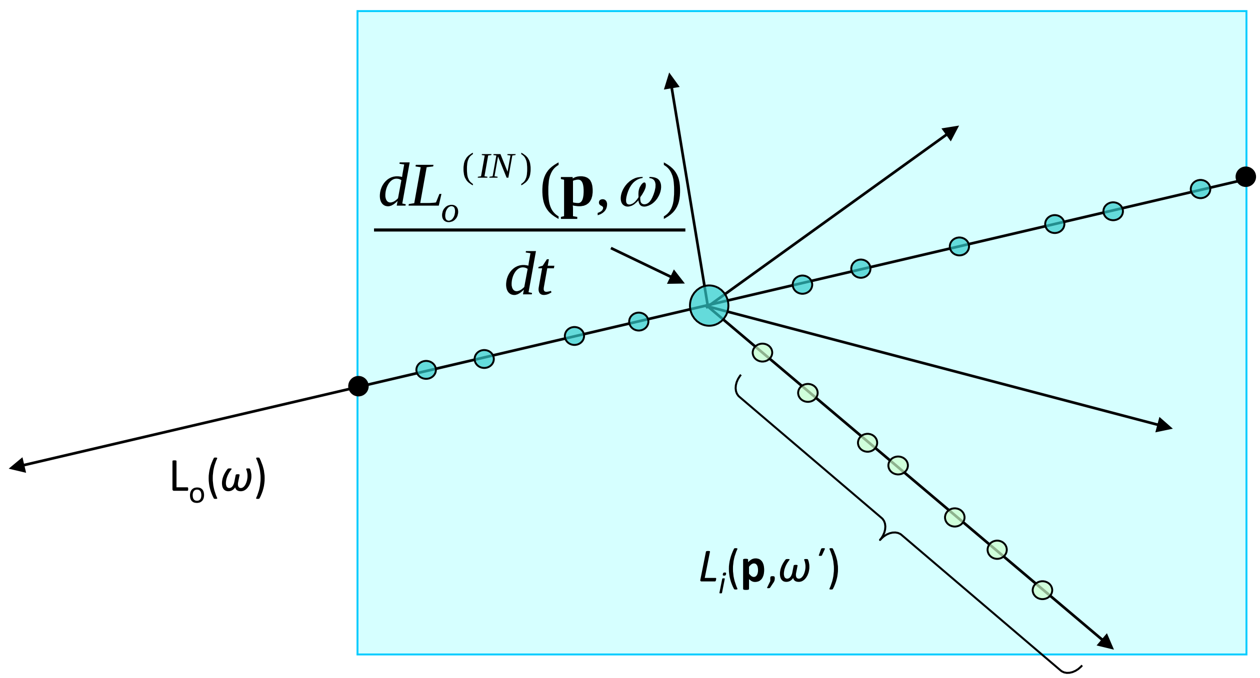
  - Henyey-Greenstein

$$p_{Henyey-Greenstein}(\omega \to \omega') = \frac{1}{4\pi} \frac{1-g^2}{\left(1+g^2-2g\cos\theta\right)^{3/2}}$$

  - Mie (atmosphere)
  - Rayleigh (droplets, steam etc)

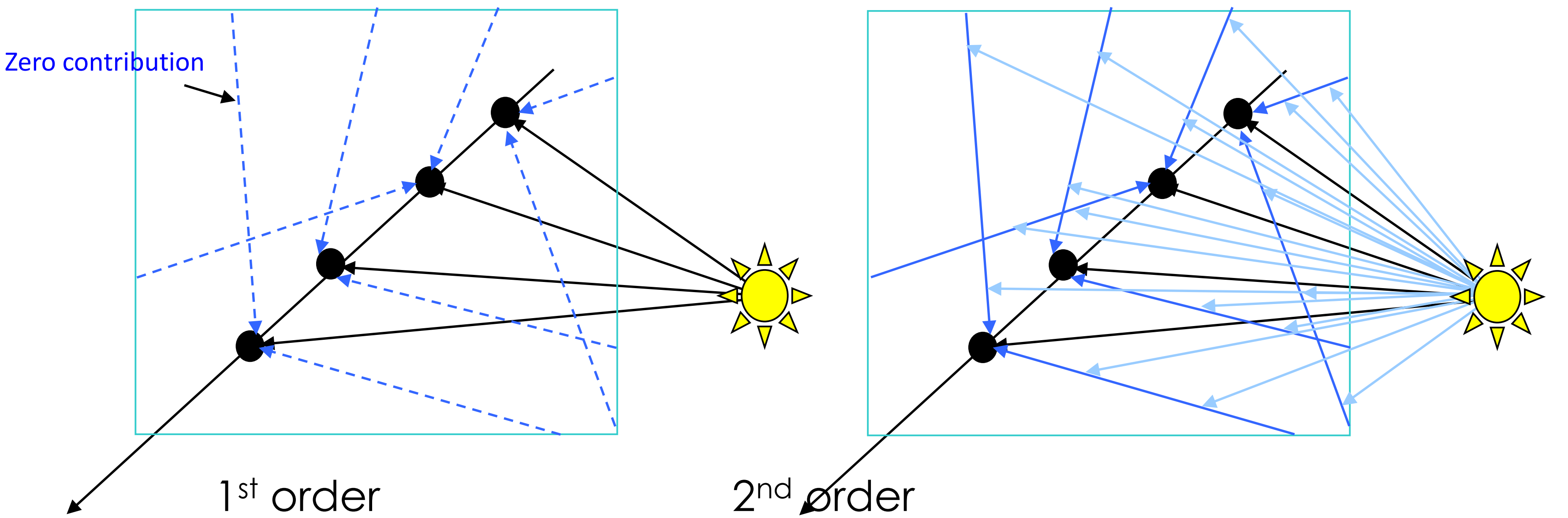
In-scattering Equation

$$\frac{dL_o^{(IN)}(\mathbf{p},\omega)}{dt} = L_e(\mathbf{p},\omega) + \int_S p(\mathbf{p},-\omega' \to \omega) L_i(\mathbf{p},\omega') d\omega'$$



In-scattering Equation (2)

- In-scattering equation is actually computed recursively, although usually 1-2 levels are used:



Combining Out/In-scattering

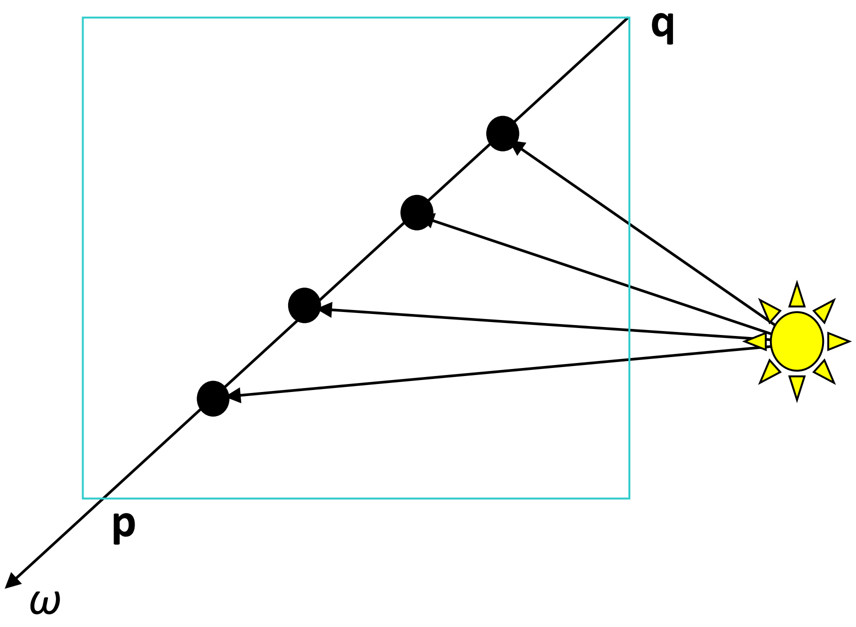
$$L(\mathbf{p},\omega)=L_o^{(IN)}(\mathbf{p},\omega)+L_o^{(OUT)}(\mathbf{p},\omega)$$

$$L_o^{(OUT)}(\mathbf{p},\omega)=T_r(\mathbf{p}\rightarrow\mathbf{q})L_i(\mathbf{q},\omega)$$

$$L_o^{(IN)}(\mathbf{p},\omega)=\int_0^t\left(L_e(\mathbf{p}+\omega t,\omega)+\int_S p(\mathbf{p}+\omega t,-\omega'\rightarrow\omega)L(\mathbf{p}+\omega t,\omega')d\omega'\right)T_r(\mathbf{p}\rightarrow\mathbf{p}+\omega t)dt$$

Direct Volume Illumination

- We can simplify the equation by omitting all indirect (in-scattering) paths:



Direct Volume Illumination (2)

- This is the first order light bounce
- Includes shadow calculation
- It is very common to use uniform phase function

$$L(\mathbf{p},\omega) = T_r(\mathbf{p} \to \mathbf{q} + \omega t) L_{background} +$$

$$\int_0^t \left(L_e(\mathbf{p} + \omega t, \omega) + \cancel{\int_S} p(\mathbf{p} + \omega t, -\omega' \to \omega) L(\mathbf{p} + \omega t, \omega') d\omega' \right) T_r(\mathbf{p} \to \mathbf{p} + \omega t) dt =$$

$$T_r(\mathbf{p} \to \mathbf{q}) L_{background} + \int_0^t \left(L_e(\mathbf{p} + \omega t, \omega) + \frac{color(\mathbf{p} + \omega t)}{4\pi} \cdot L(\mathbf{p} + \omega t, \omega_L) \right) T_r(\mathbf{p} \to \mathbf{p} + \omega t) dt$$

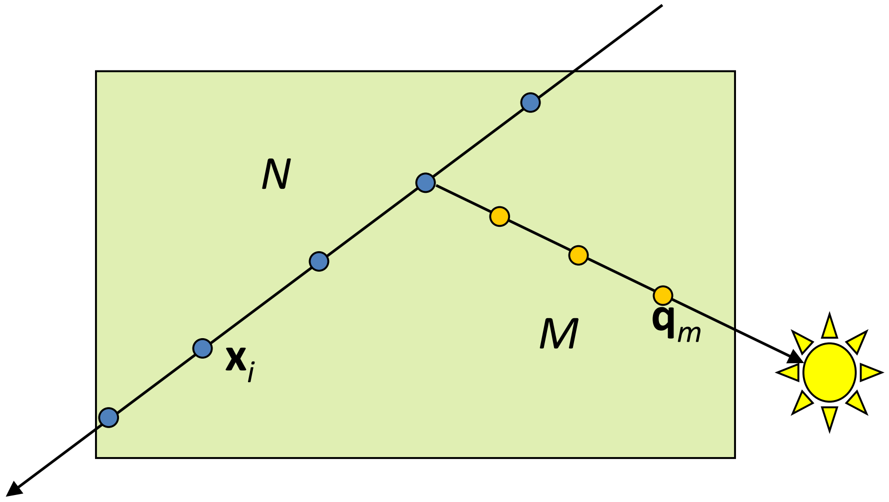
Direct Volume Illumination (3)

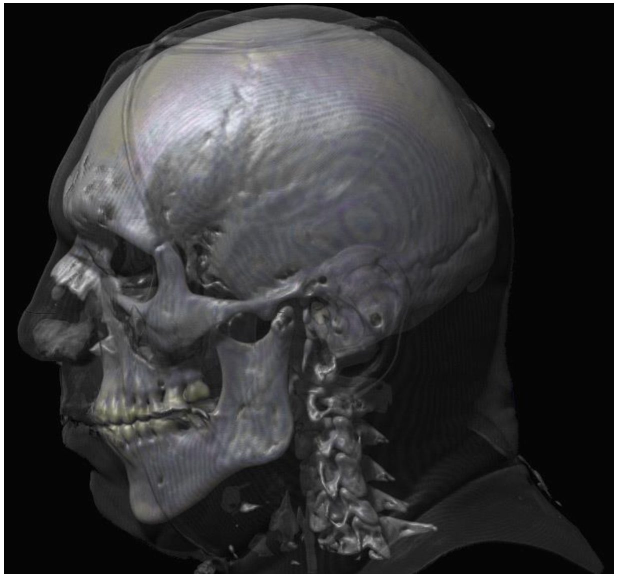
- In discrete sampling form:

$$L(\mathbf{p},\omega) = T_r(\mathbf{x}_1 \to \mathbf{x}_\mathbf{N}) L_{background} +$$

$$L(\mathbf{x}_1) + \sum_{i=2}^{N-1} L(\mathbf{x}_i) \prod_{k=1}^{i-1} e^{-\sigma(\mathbf{x}_k)\|\mathbf{x}_{k+1}-\mathbf{x}_k\|}$$

$$L(\mathbf{x}_i) = L_e(\mathbf{x}_i) + \frac{color(\mathbf{x}_i)}{4\pi} \cdot L_L \prod_{m=1}^{M-1} e^{-\sigma(\mathbf{q}_m)\|\mathbf{q}_{m+1}-\mathbf{q}_m\|}$$

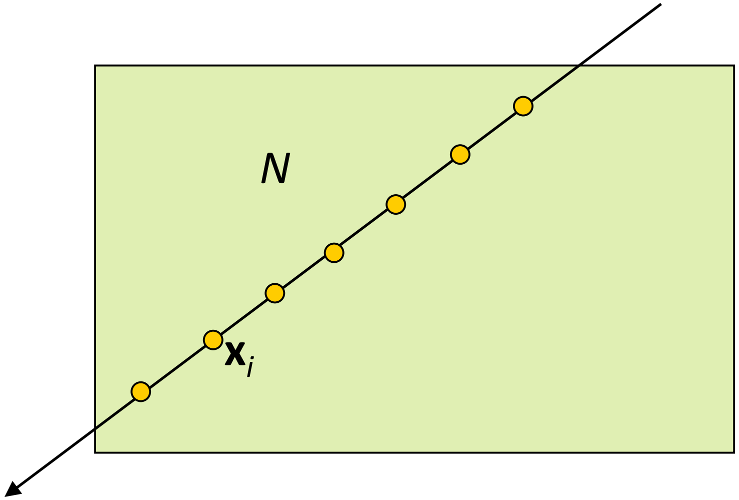


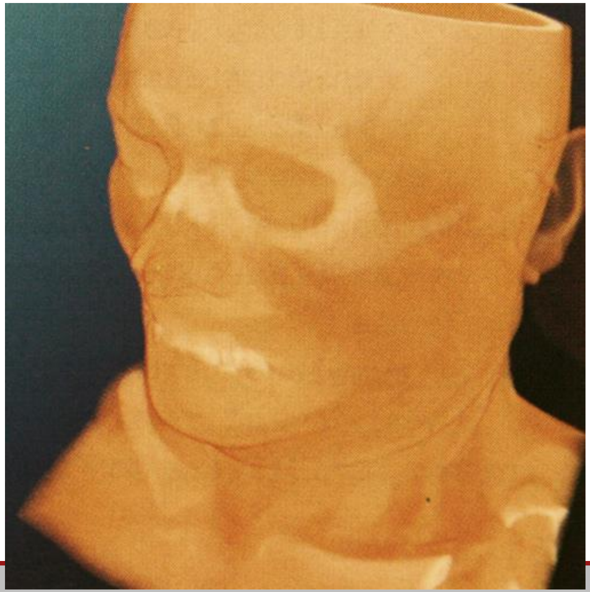


Direct Volume Illumination (4)

- No Shadow, just the illuminated samples (L_e):

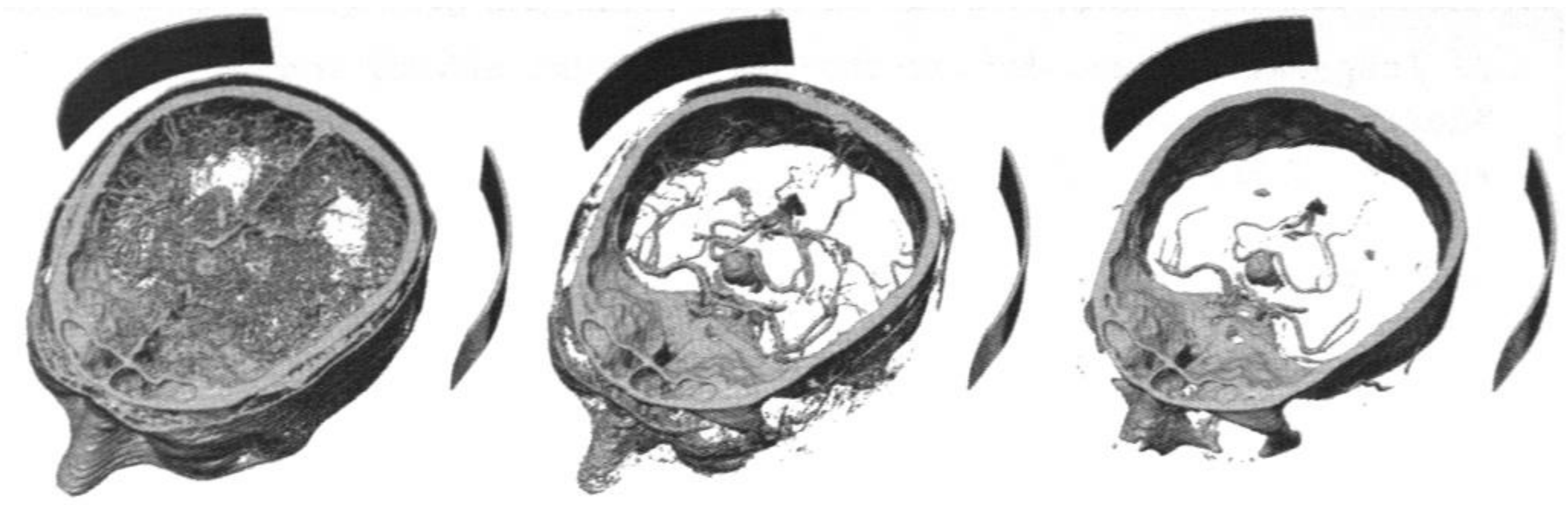
$$L(\mathbf{p},\omega) = T_r(\mathbf{x}_1 \to \mathbf{x}_\mathrm{N}) L_{background} + L_e(\mathbf{x}_1) + \sum_{i=2}^{N-1} L_e(\mathbf{x}_i) \prod_{k=1}^{i-1} e^{-\sigma(\mathbf{x}_k)\|\mathbf{x}_{k+1}-\mathbf{x}_k\|}$$





Isosurface Rendering

- After classification, local sample density determines the "presence" of the sample in the integral
- If the transfer function has sharp transitions, then isosurfaces at various density values are formed



Isosurface Shading

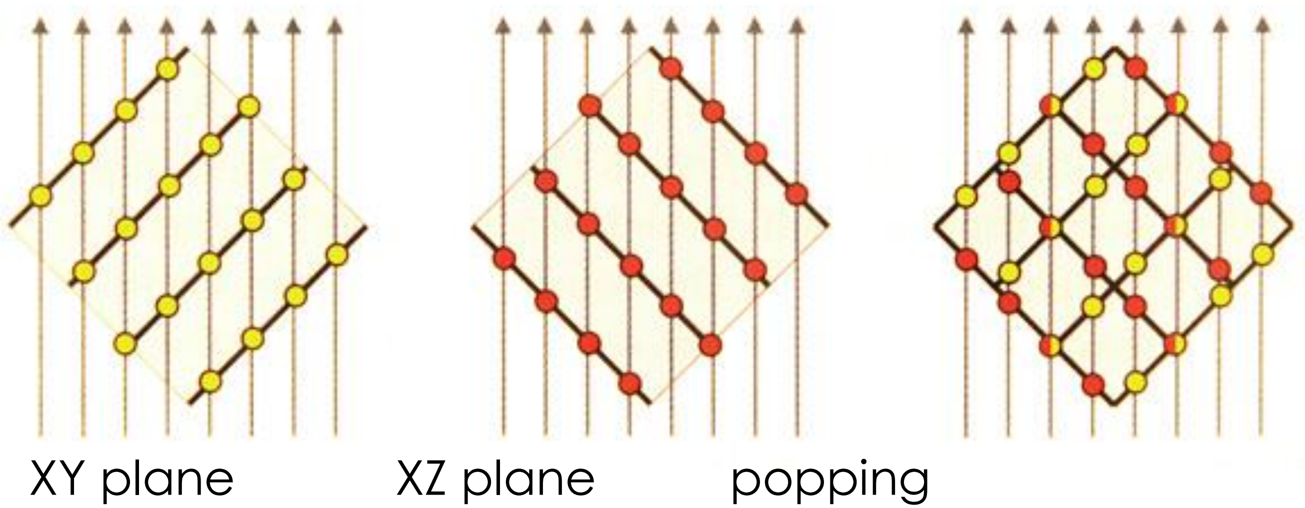
- To shade the isosurface samples, a local illumination mode can be used →
- Requires normal vectors
- Normals are directly computed from the density gradient
  - Pro-processed: After precomputing the transfer function and its effect. Can be restrictive
  - On the fly: Best quality and versatility, but requires six texture fetches for gradient computation

Rendering

- Texture-based:
  - Volume slicing using 2D axis-aligned slice texures
  - Volume slicing using 3D textures
- Direct volume ray marching
  - CPU
  - GPU

Volume Rendering – Axes-Aligned Slicing

- A number of slice textures is generated from the volume along the 3 axes
- For GPU rendering: Post-classification is possible
- The axis that is most perpendicular to the view plane is chosen for slice projection

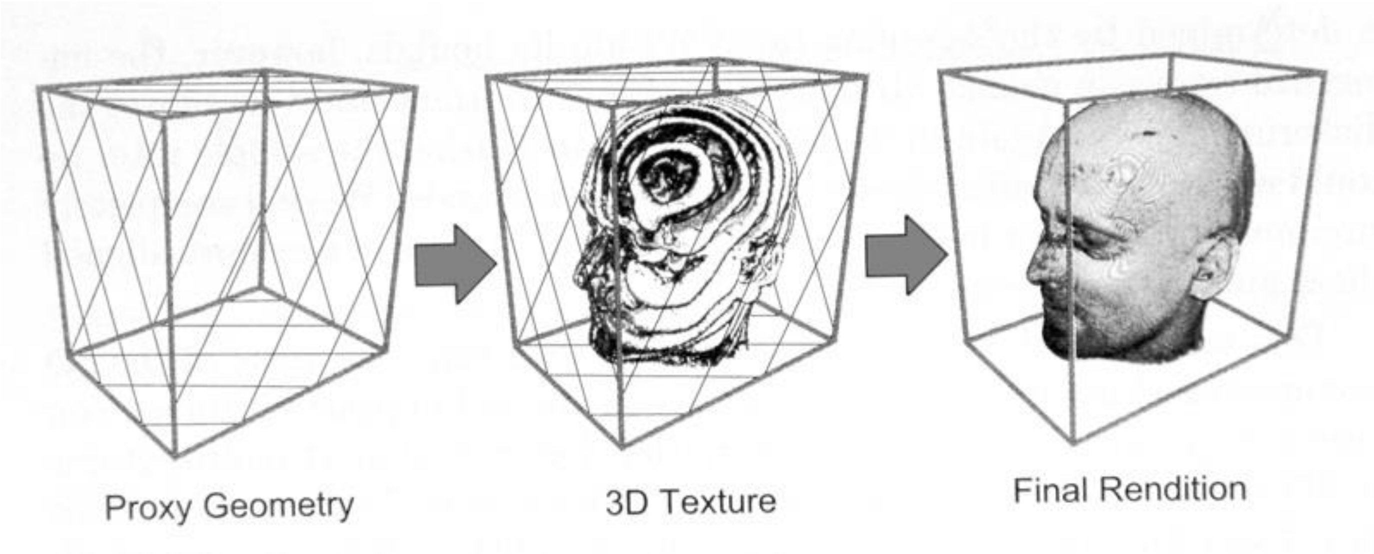


AA Slicing - Comments

- Popping effects when changing axis
- Can perform full illumination only if the volume is also accessible to a fragment shader as a 3D texture
- Blending is performed via standard direct rendering modes →
- Slice **alpha** values are mapped to transmittance →
- Transmittance is precalculated for specific slice thickness

Volume Rendering – View-Aligned Slicing

- View-aligned cross sections of the volume space are created and textured using the volume data as 3D texture
- Slice distance determines transmittance → alpha
- Slices must be conventionally blended

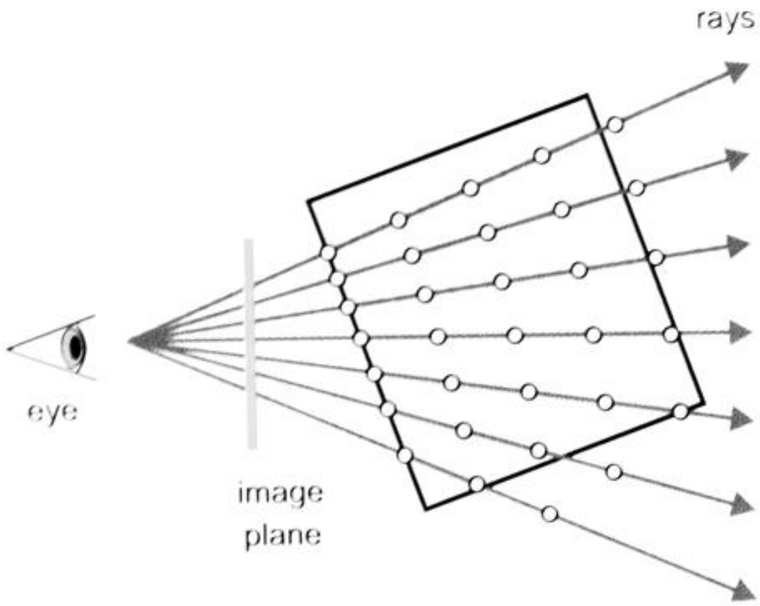


View-Aligned Slicing - Comments

- Very efficient GPU implementation
- Full shading possible via fragment shaders and volume (3D texture) access

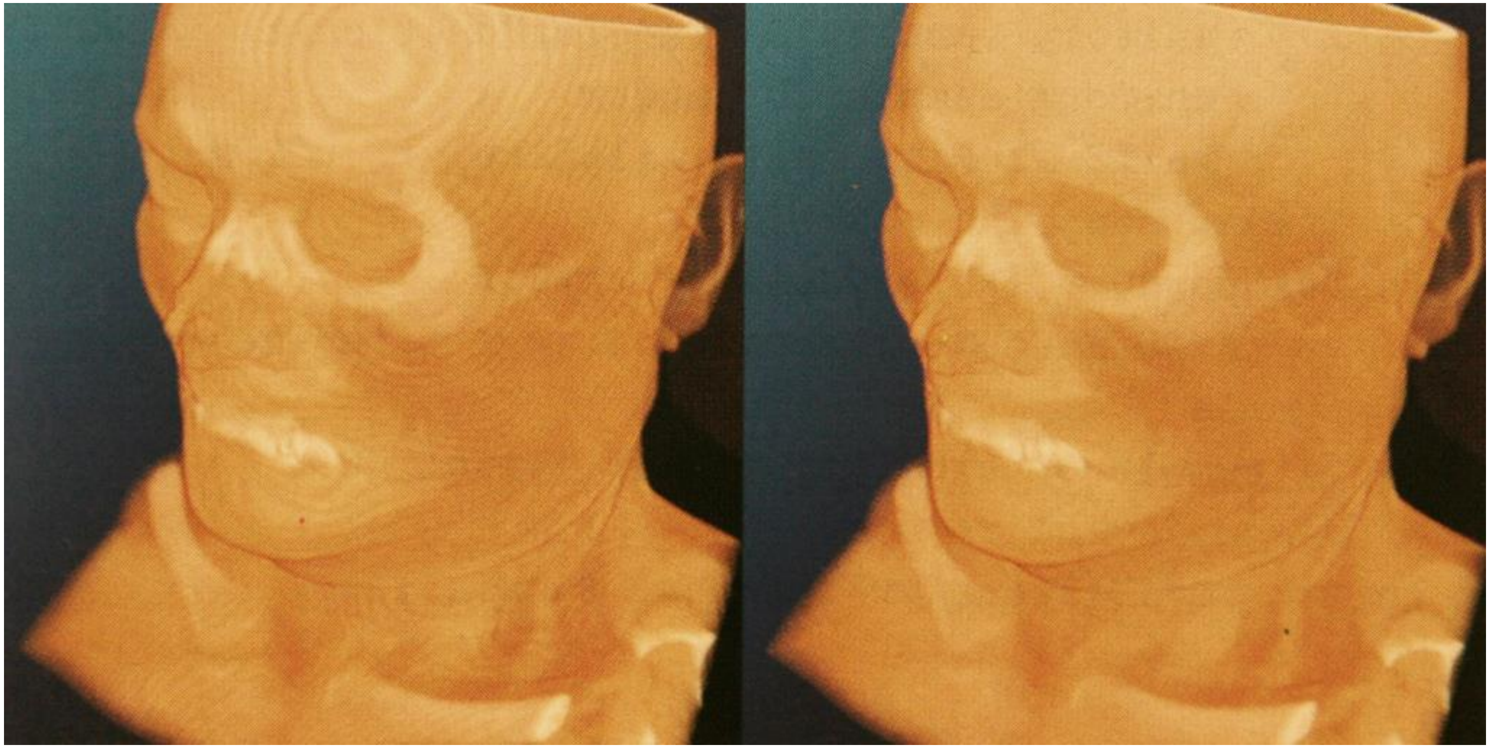
Volume Rendering – Volume Ray Marching

- Most generic technique
- CPU and GPU implementation
- Cast rays from view plane towards the volume



Ray Marching – Sampling Intervals

- Constant sampling intervals create visible banding
- Stratified stochastic jittering produces smoother results and avoids banding



GPU Volume Visualization - Ray Marching

- Requires a surface to draw fragments as initial ray points:
- View-aligned plane (e.g. screen-filling quad)
- Volume-aligned closed surface such as box, sphere
- Easy to combine with cutting planes
- Iterate for a number of samples along each ray
- Add jittering
- Perform arbitrarily complex lighting computations and post-classification

Contributors

- Georgios Papaioannou