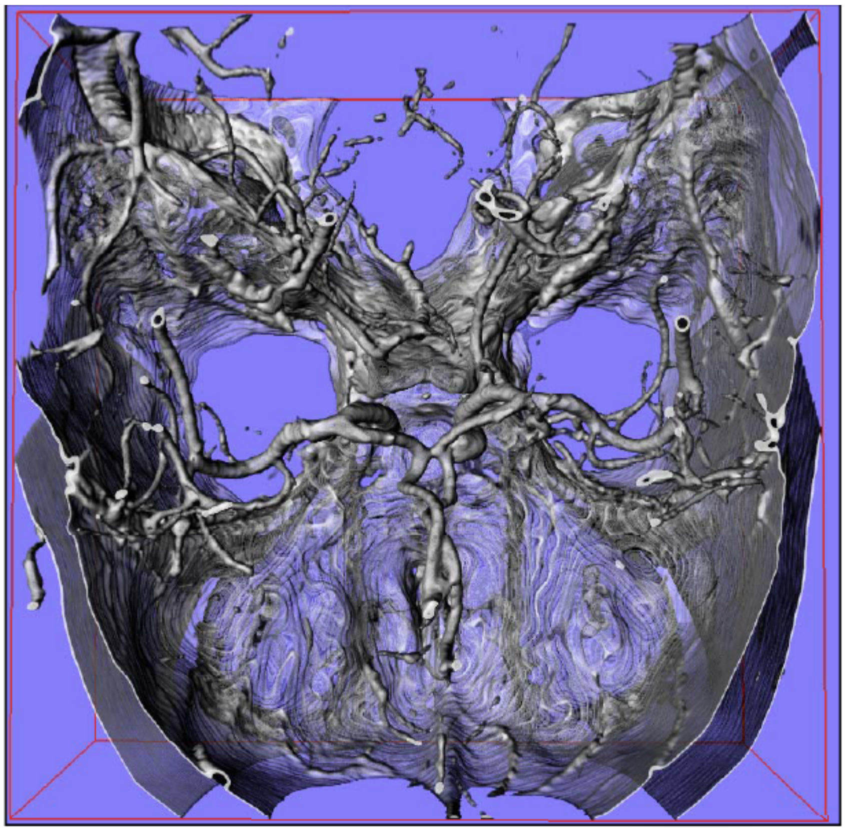
Volume Rendering

Application: Medicine



CT Angiography:
Dept. of Neuroradiology
University of Erlangen, Germany

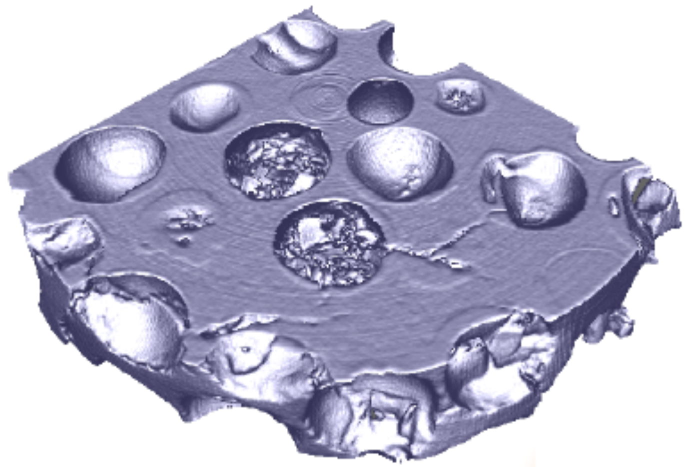
CT Human Head:
Institute for Vision and Graphics
University of Siegen, Germany

Application: Archaeology

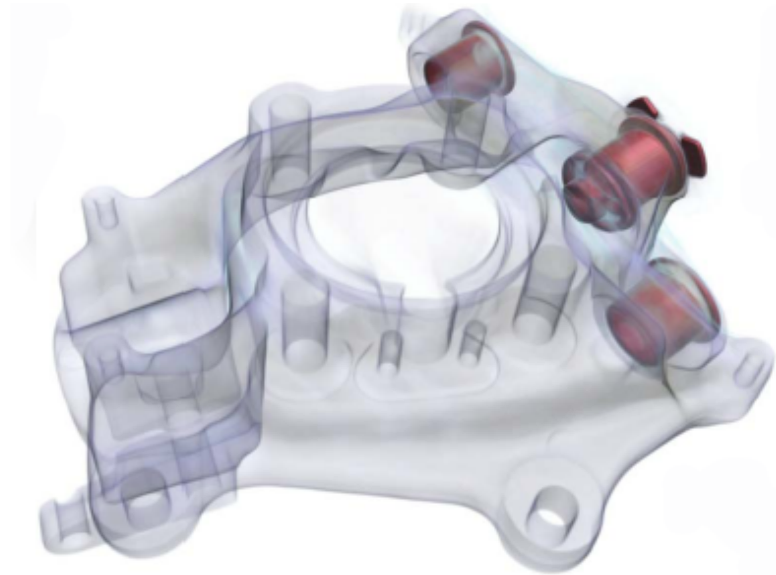
Hellenic Statue of Isis:
ARTIS, University of Erlangen-Nuremberg, Germany

Sotades Pygmaios Statue
ARTIS, University of Erlangen-Nuremberg, Germany

Application: Material Science

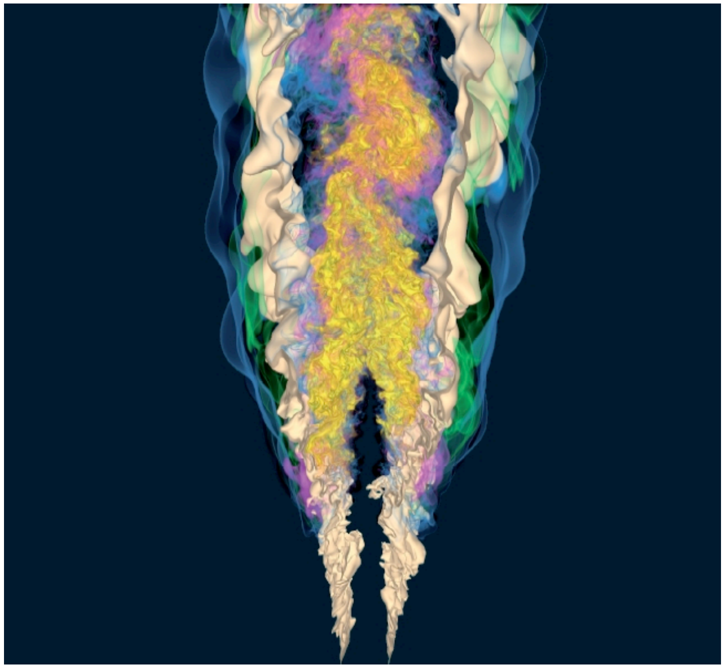


Micro CT, Compound Material,
Material Science Department,
University of Erlangen

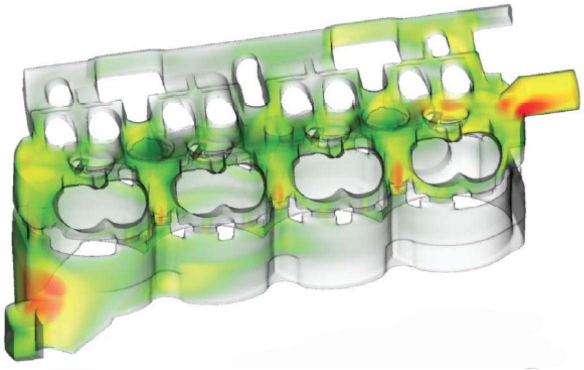


Hinge Bearing,
Austrian Foundry Research Institute

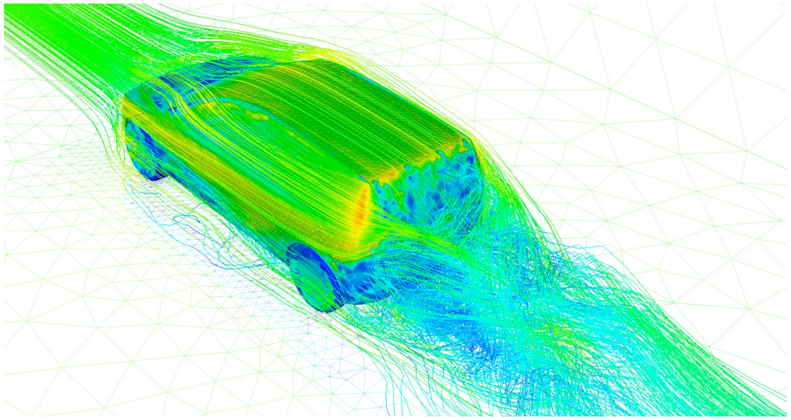
Application: Material Science



Combustion Simulation,
SciDACC

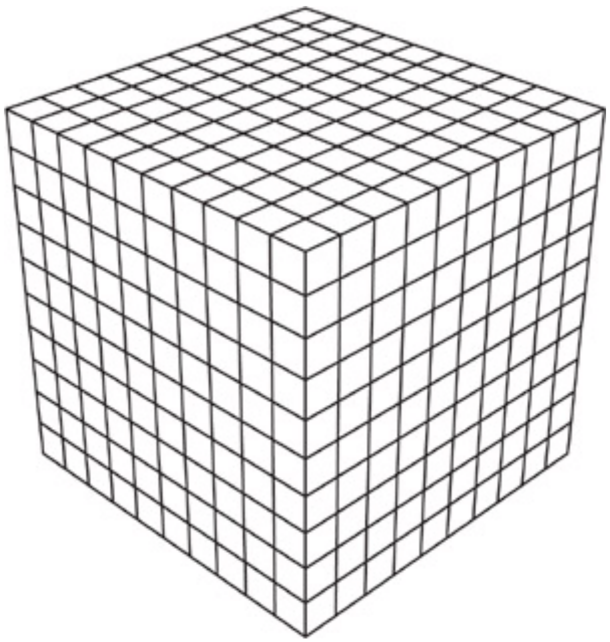


Computational Fluid Dynamic

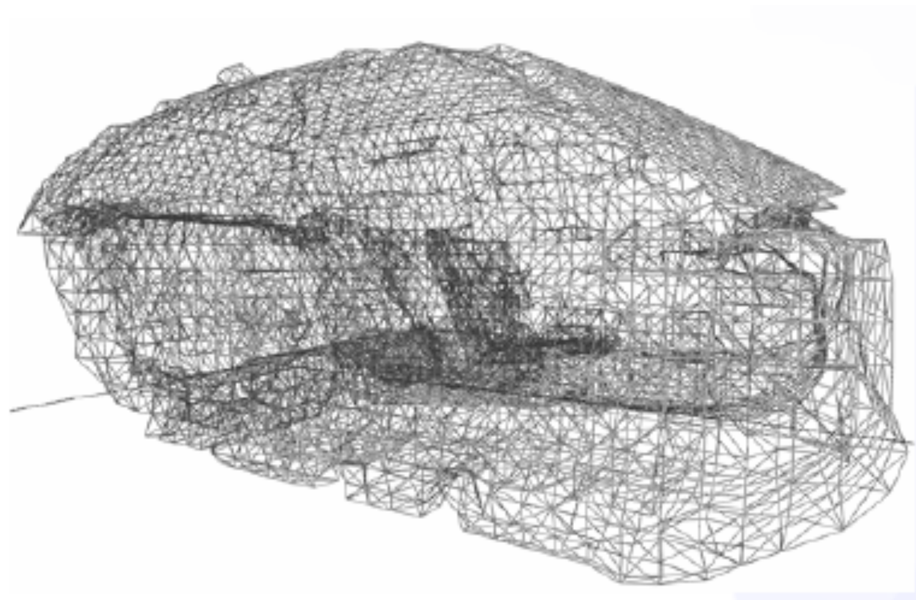


Turbulence Simulation

Volume Data Types:



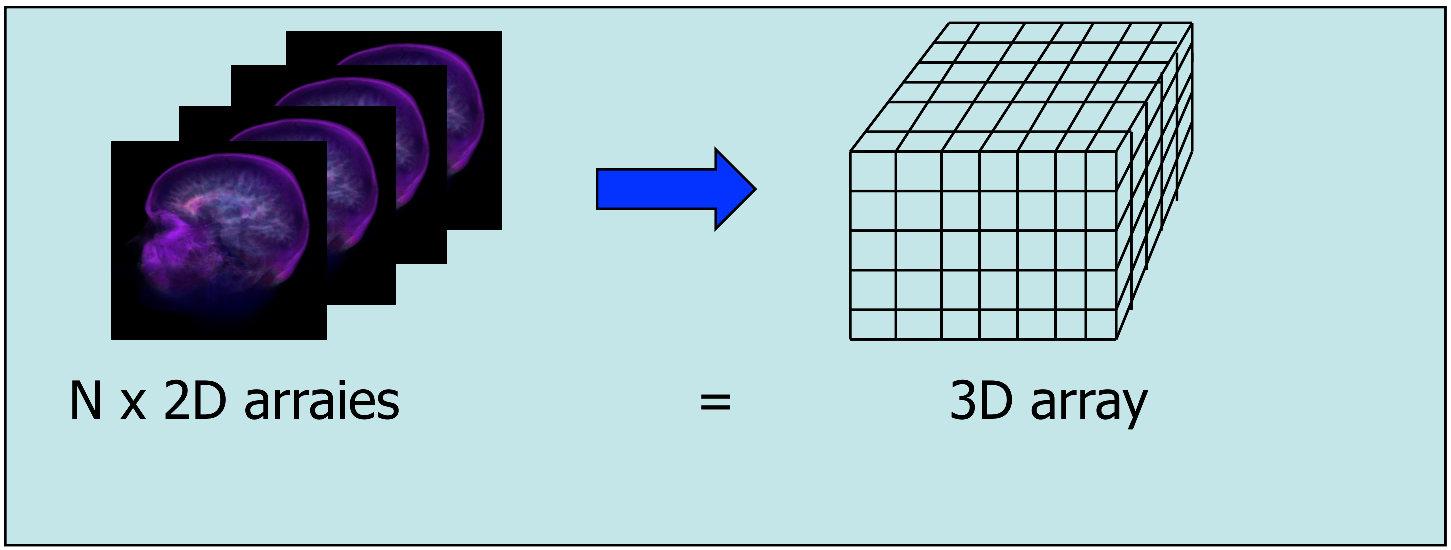
- ***Rigid Grid (Voxel)***
- Reconstruction with trilinear interpolation



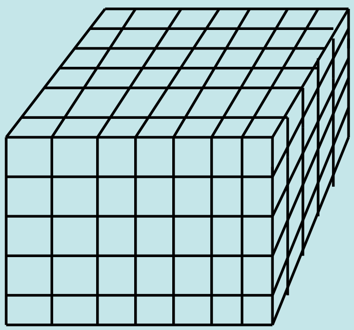
- ***Irregular Structure***
- Decomposed into tetrahedra
- Reconstruction with linear interpolation

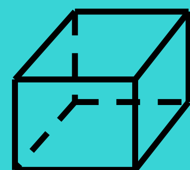
Data Representation (Rigid Grid)

- 3D volume data are represented by a finite number of cross sectional slices (hence a 3D raster)
- On each volume element (voxel), stores a data value (if it uses only a single bit, then it is a binary data set. Normally, we see a gray value of 8 to 16 bits on each voxel.)

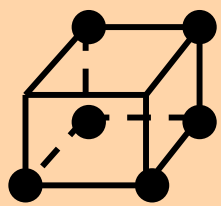


Voxel





A voxel is a cubic cell, which
has a single value cover
the entire cubic region



A voxel is a data point
at a corner of the cubic cell
The value of a point inside the
cell is determined by interpolation

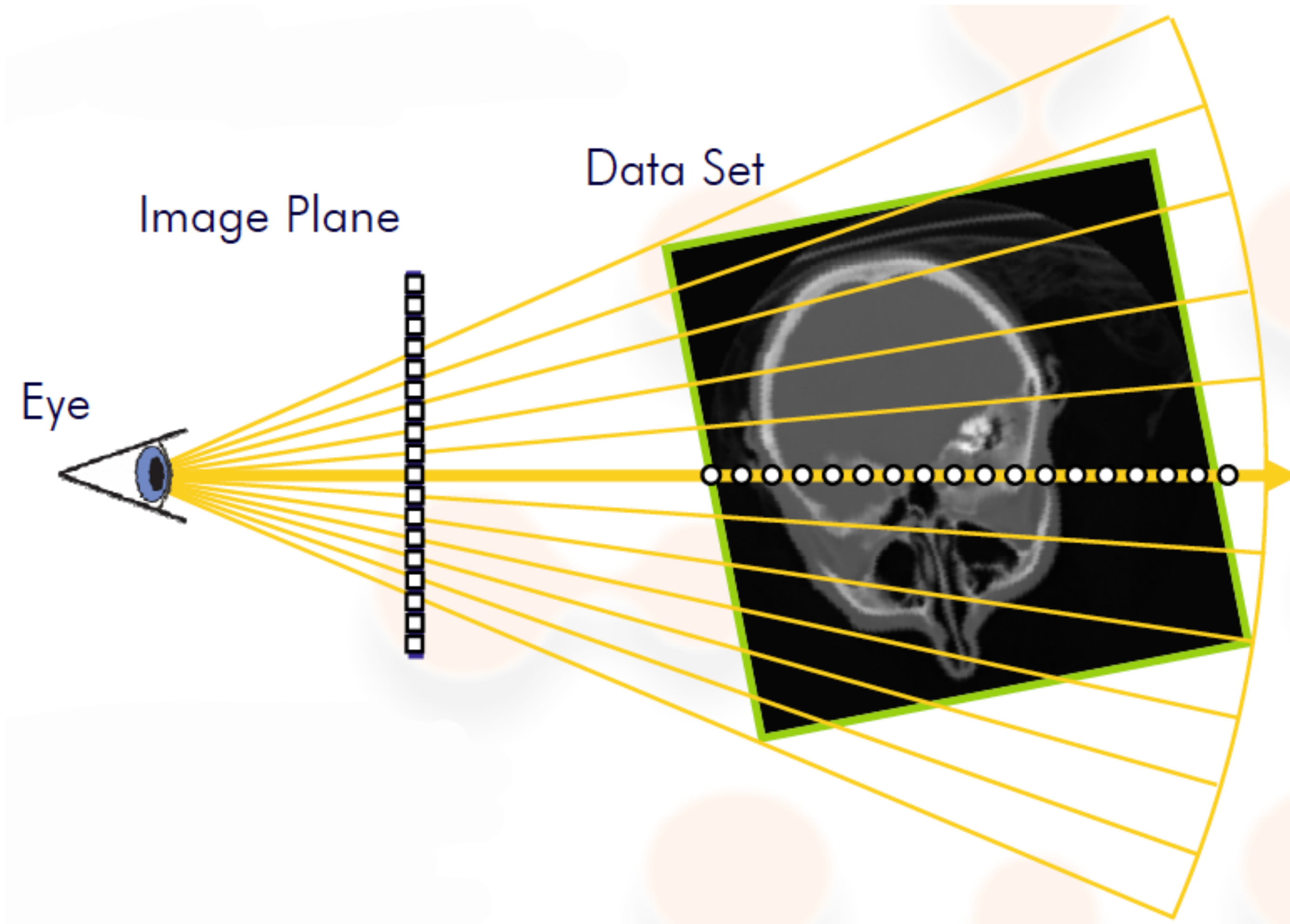
Visualizing Volume Data

- **Mapping values to appearance**
  - A Scalar value mapping to **color** or **opacity**
  - May emphasize certain value ranges (iso-surface) or give all ranges equal emphasis in final image (semi-transparent)

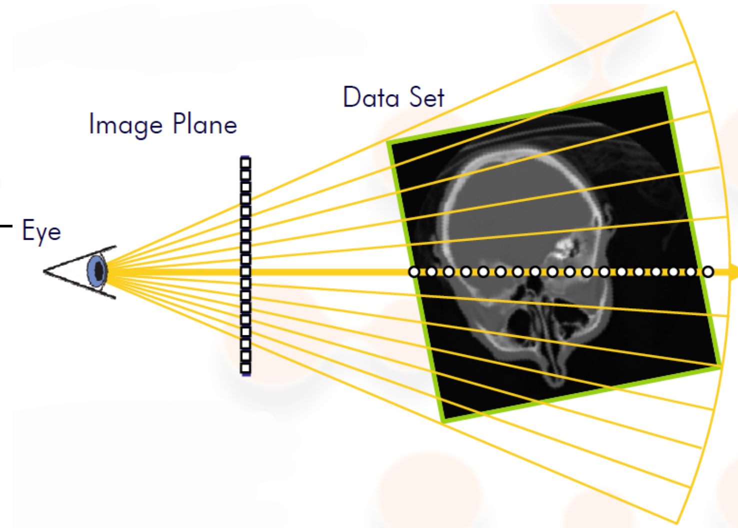
Rendering Methods

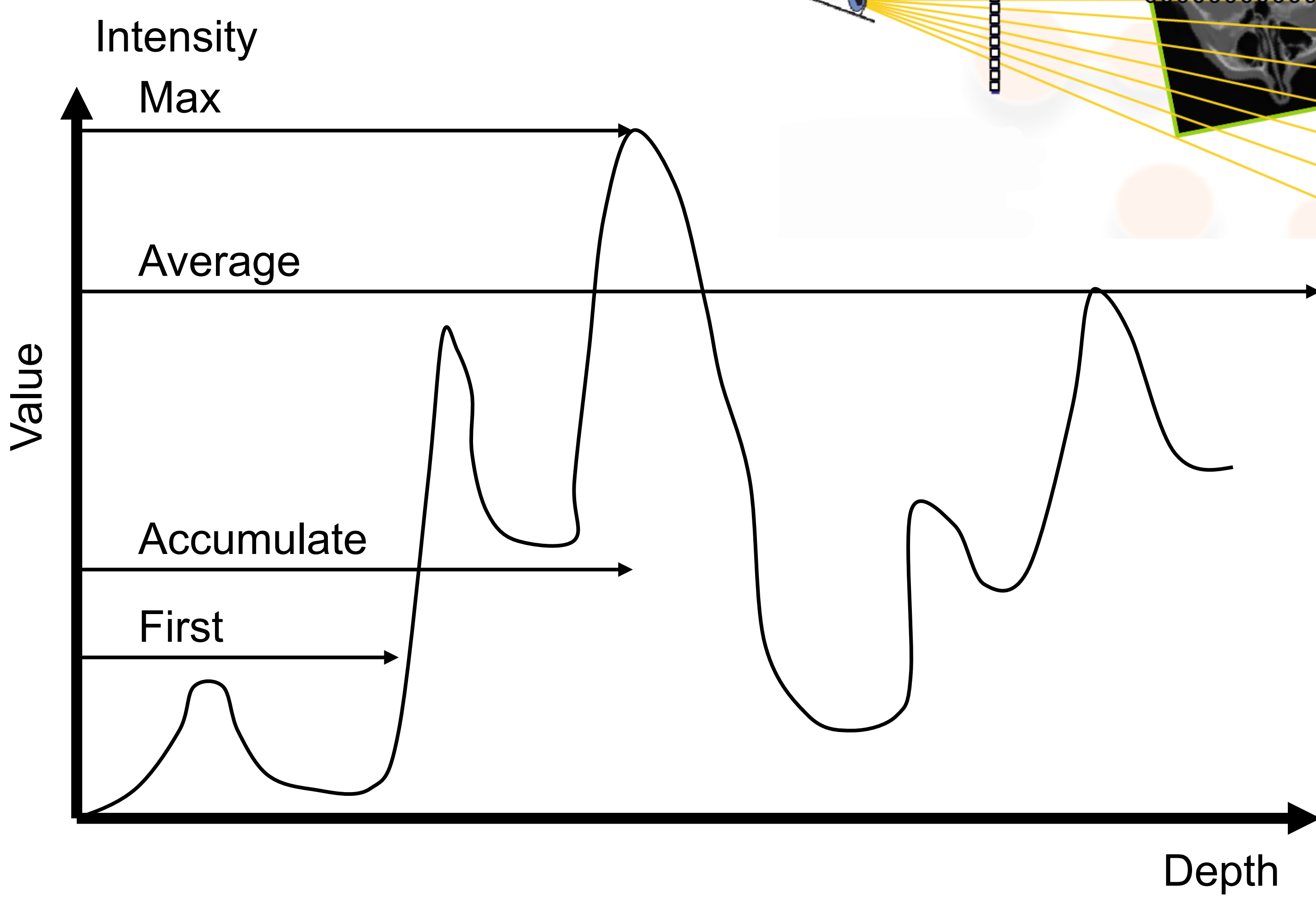
- **Ray Casting**
  - Image-order accumulation
- **Splatting**
  - Object-order accumulation
- **Iso-surface extraction**
  - Marching cube

Ray Casting

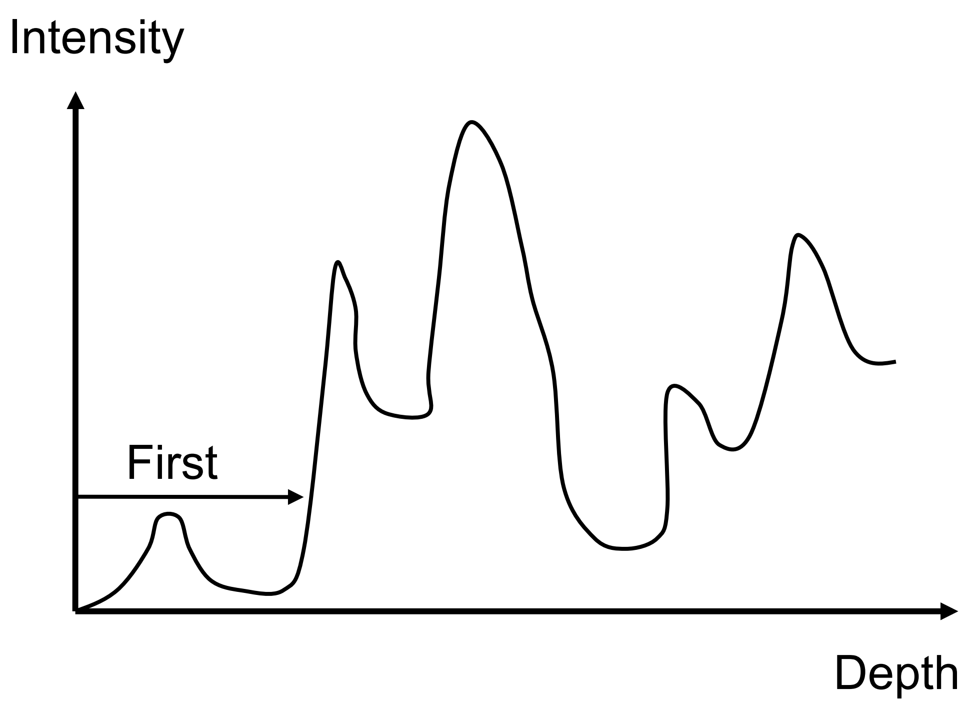


Ray Traversal Schemes



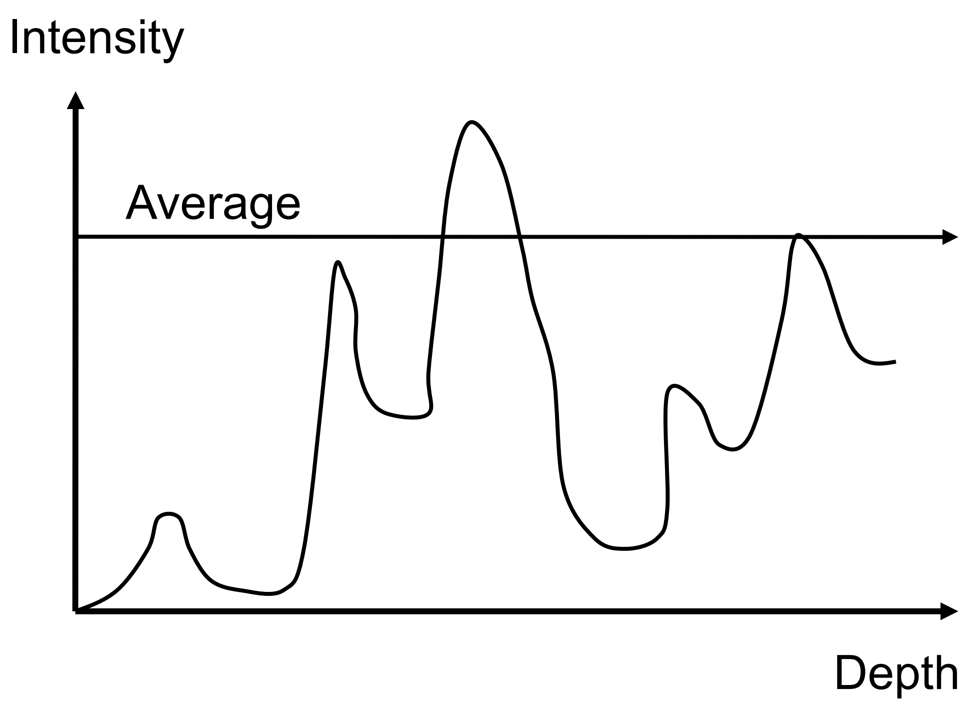


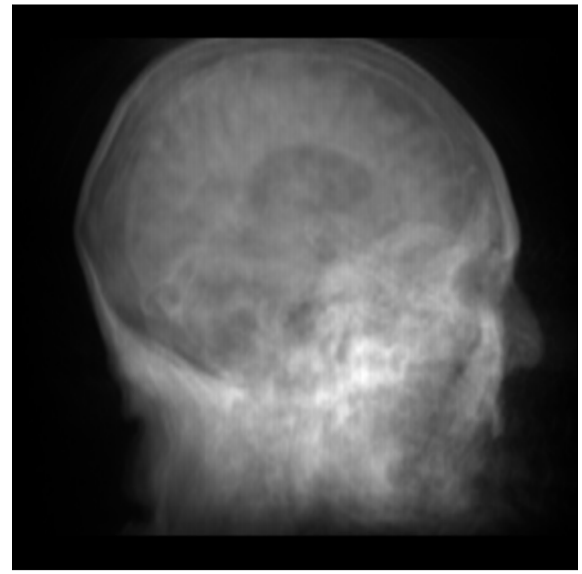
Ray Traversal - First



- **First: direct iso-surface rendering**

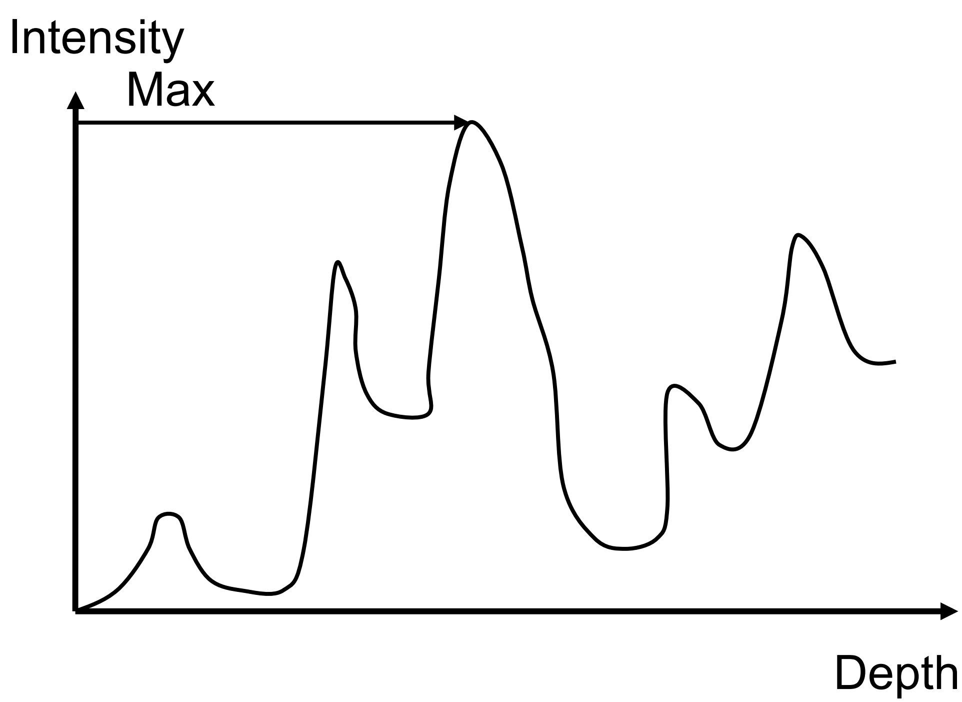
Ray Traversal - Average

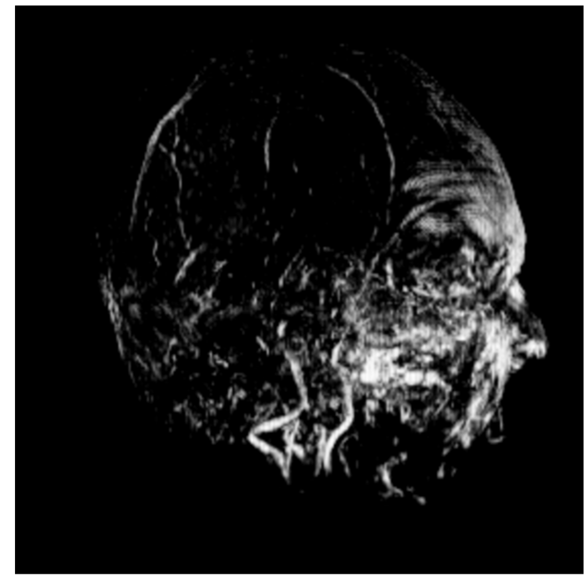




- **Average**: produces basically an X-ray picture

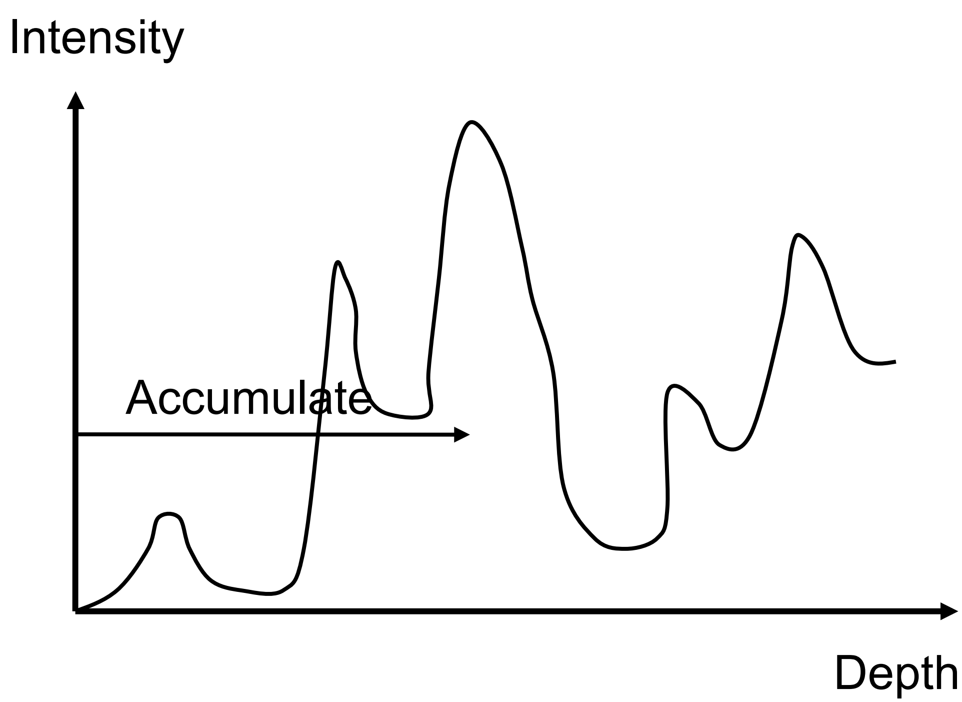
Ray Traversal - MIP

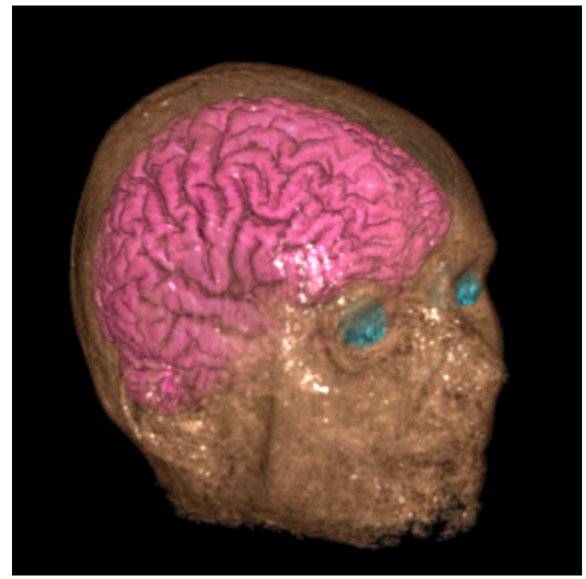




- Max: Maximum Intensity Projection
  used for Magnetic Resonance Angiogram

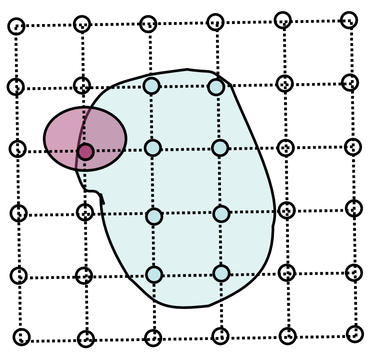
Ray Traversal - Accumulate

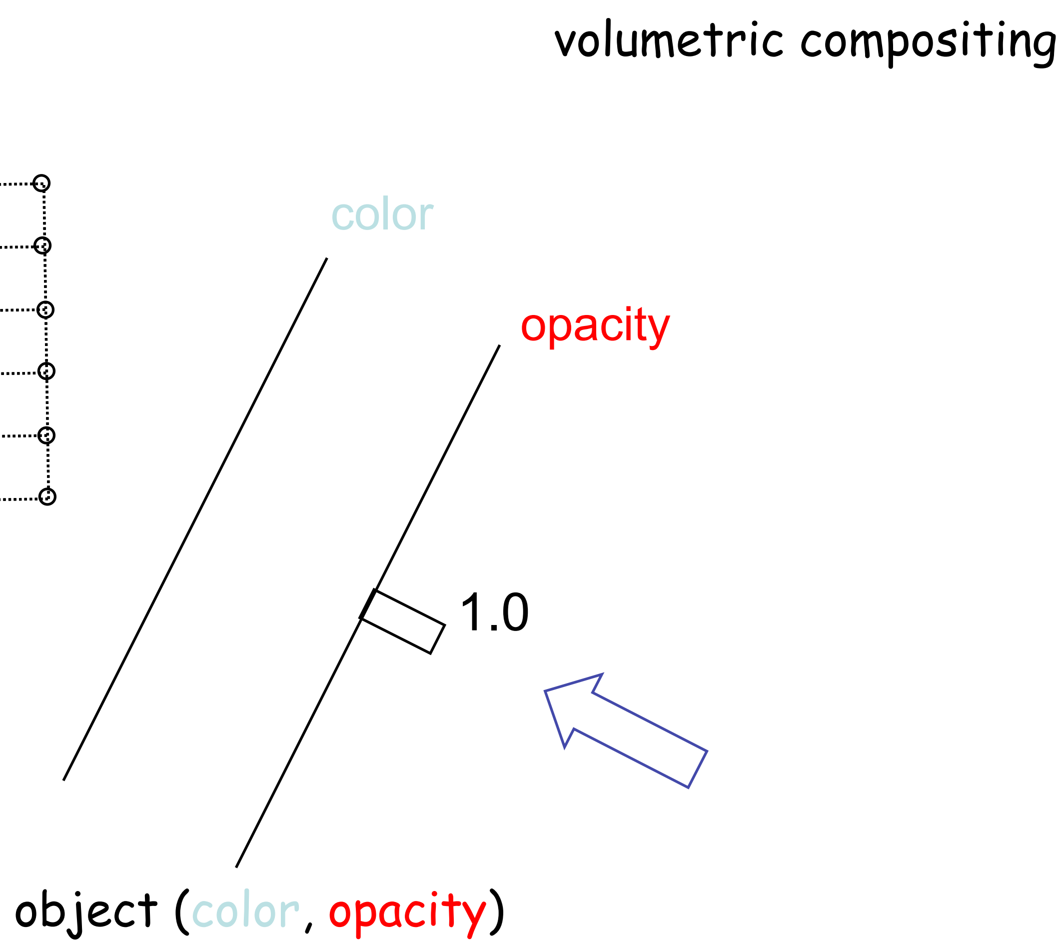




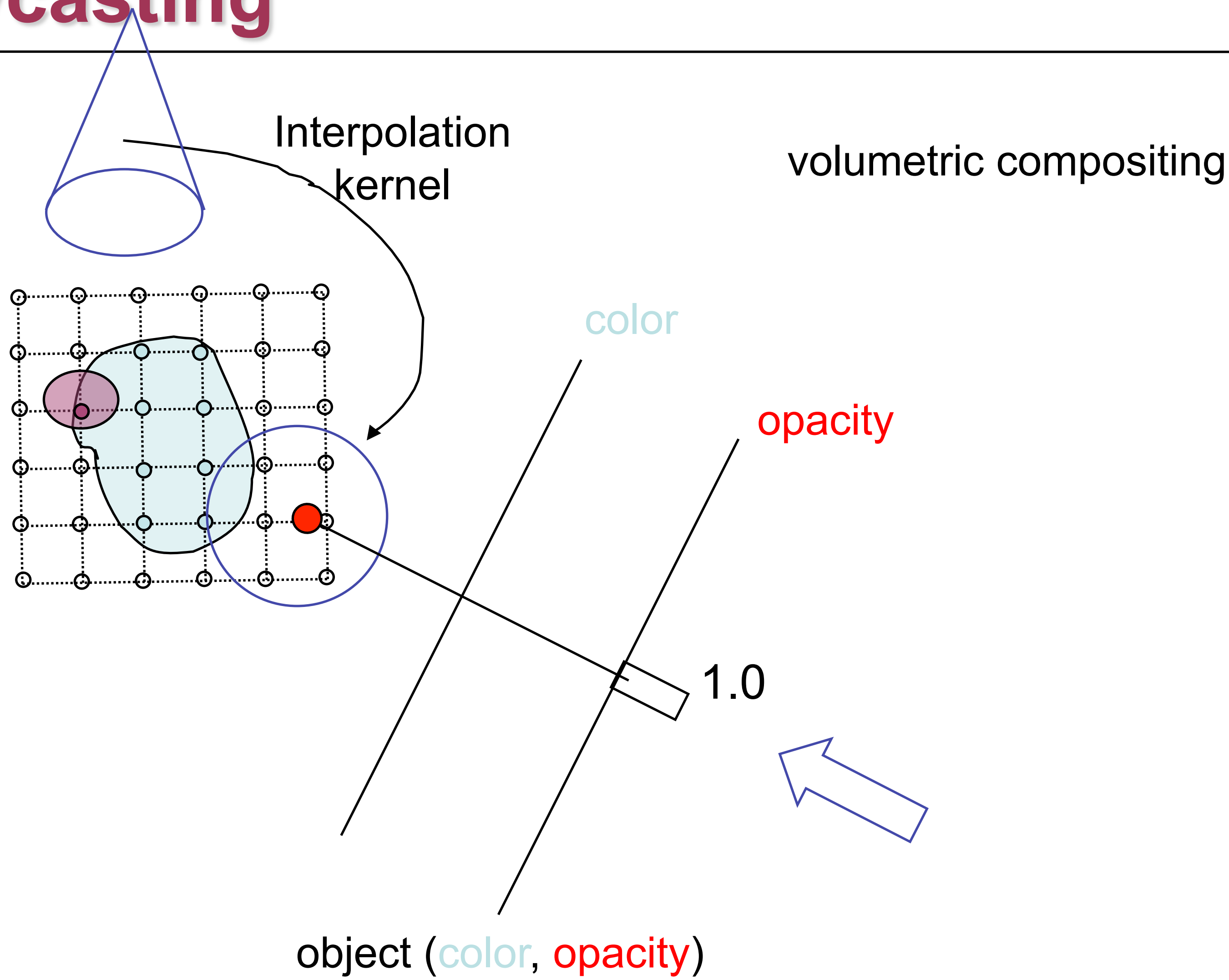
- Accumulate opacity while compositing colors: make transparent layers visible!

Raycasting

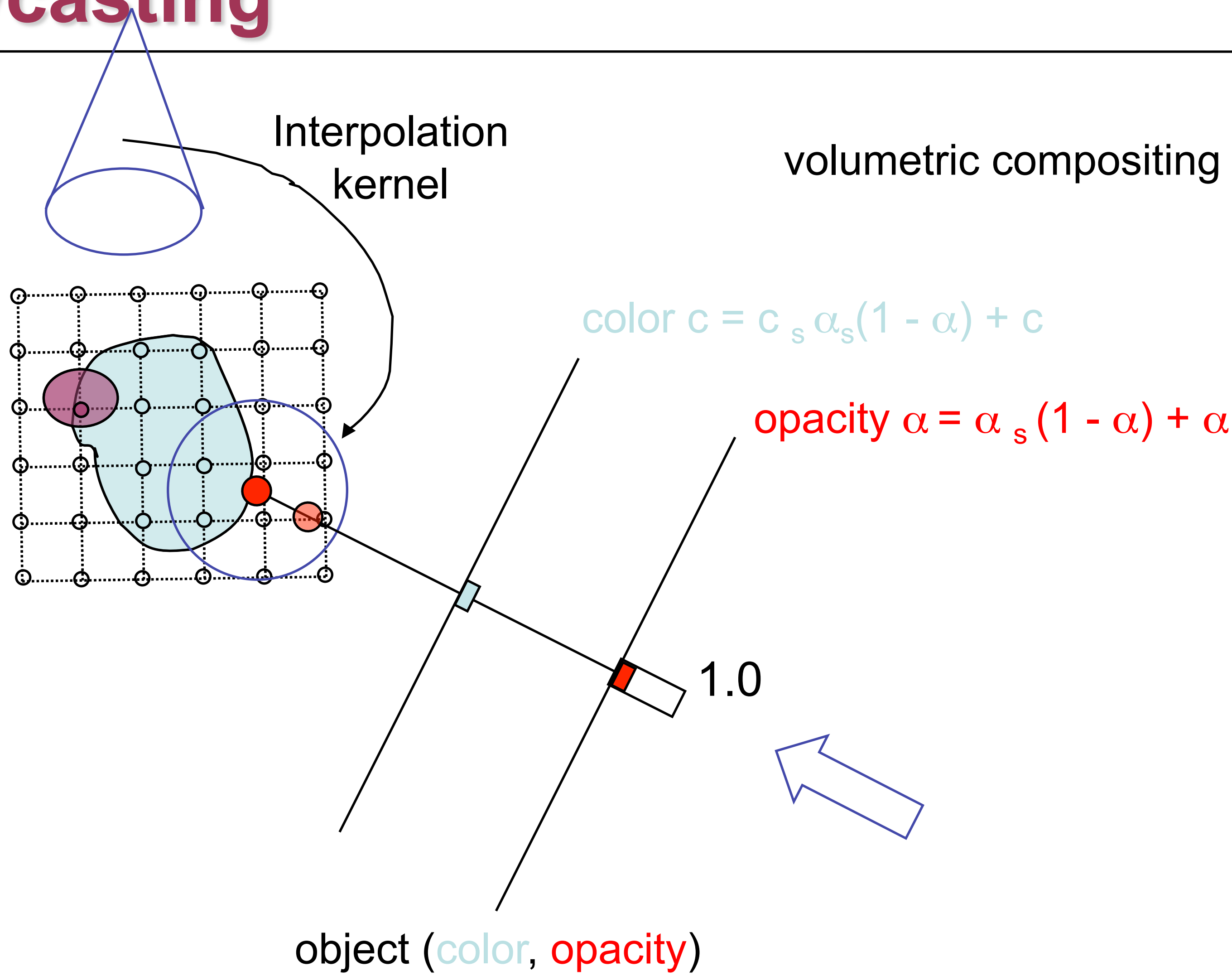



volumetric compositing
color
opacity
1.0
object (color, opacity)

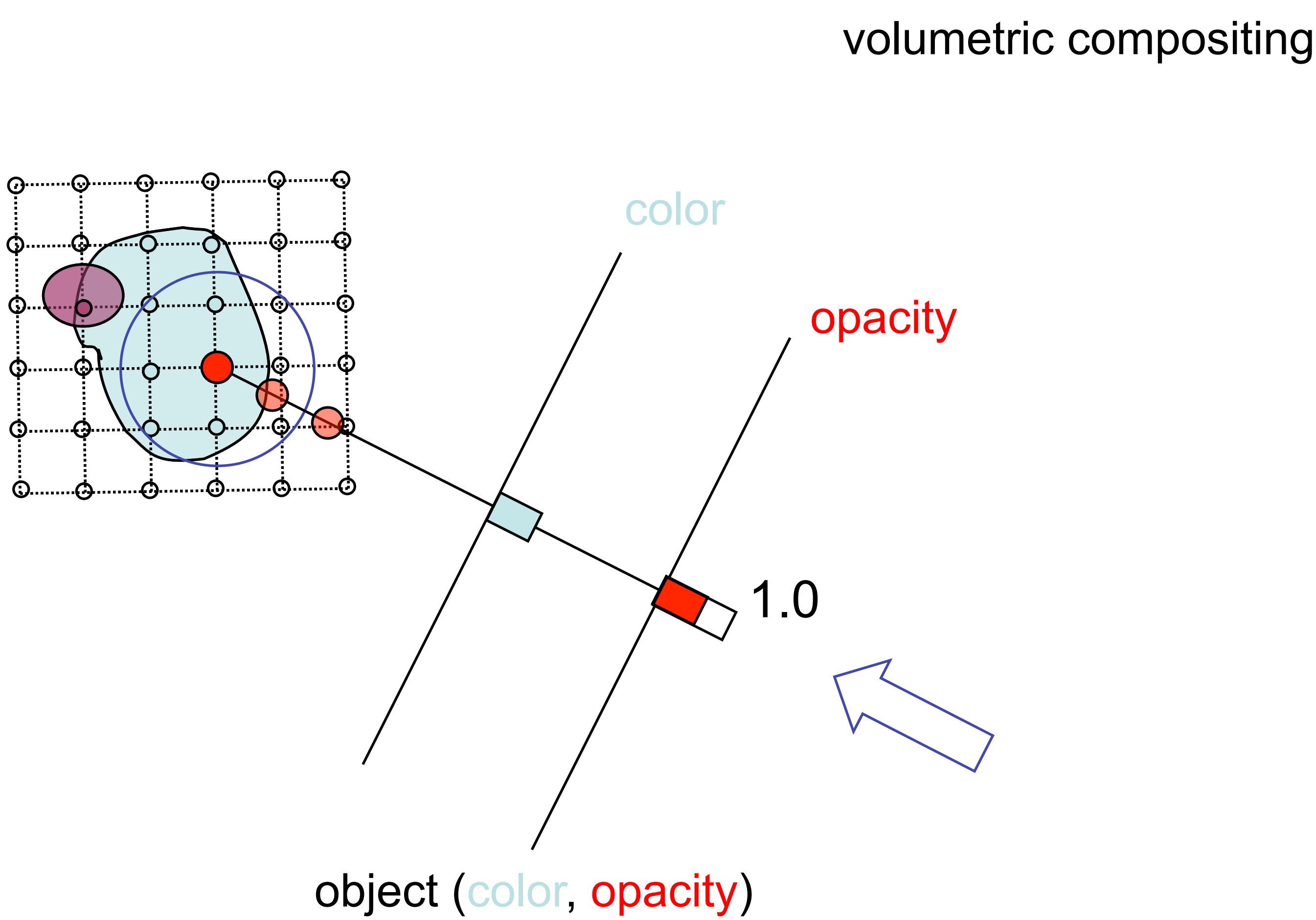
Raycasting


Interpolation kernel
volumetric compositing
color
opacity
1.0
object (color, opacity)

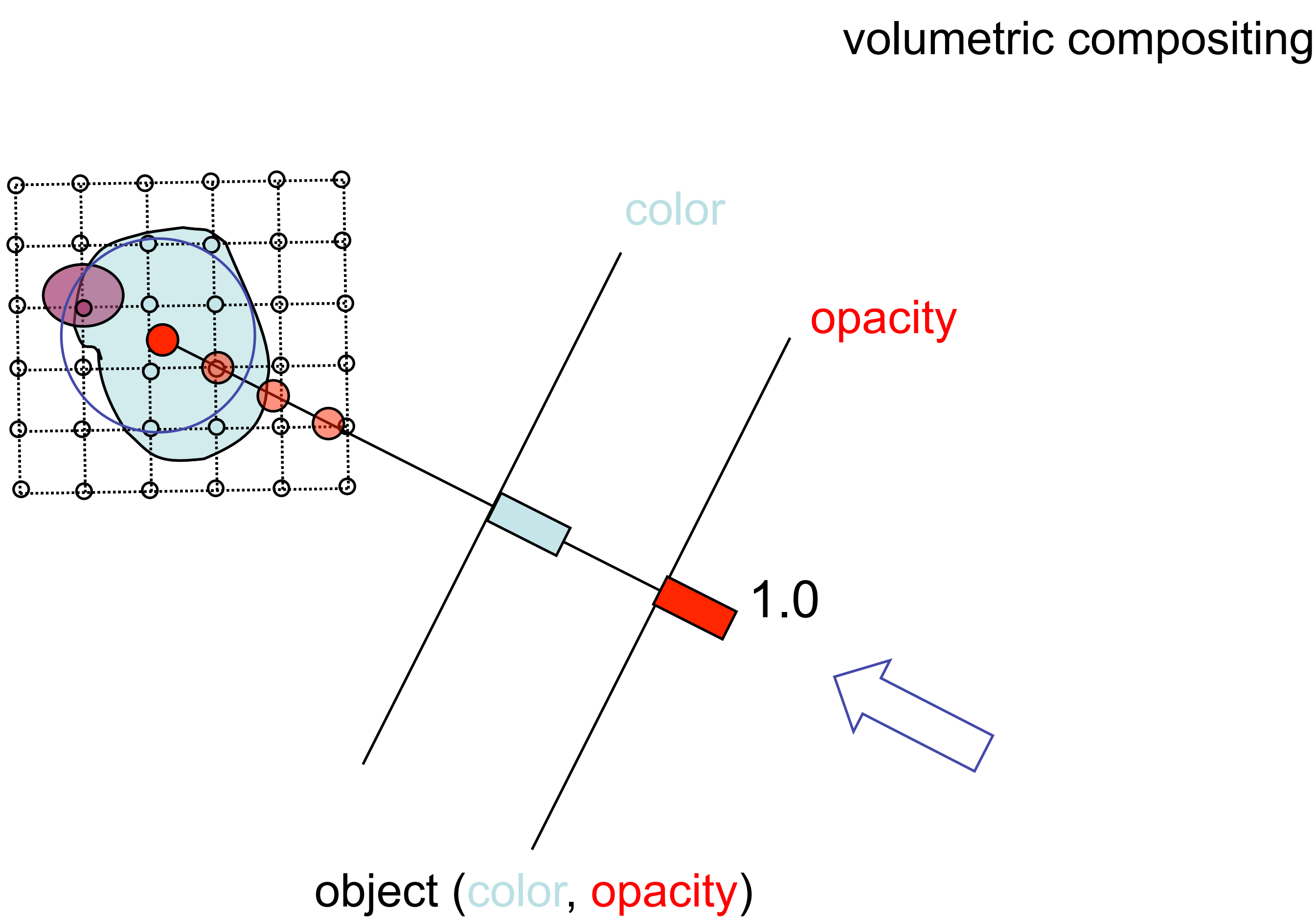
Raycasting


Interpolation kernel
volumetric compositing
color c = c s αs(1 - α) + c
opacity α = α s (1 - α) + α
1.0
object (color, opacity)

Raycasting

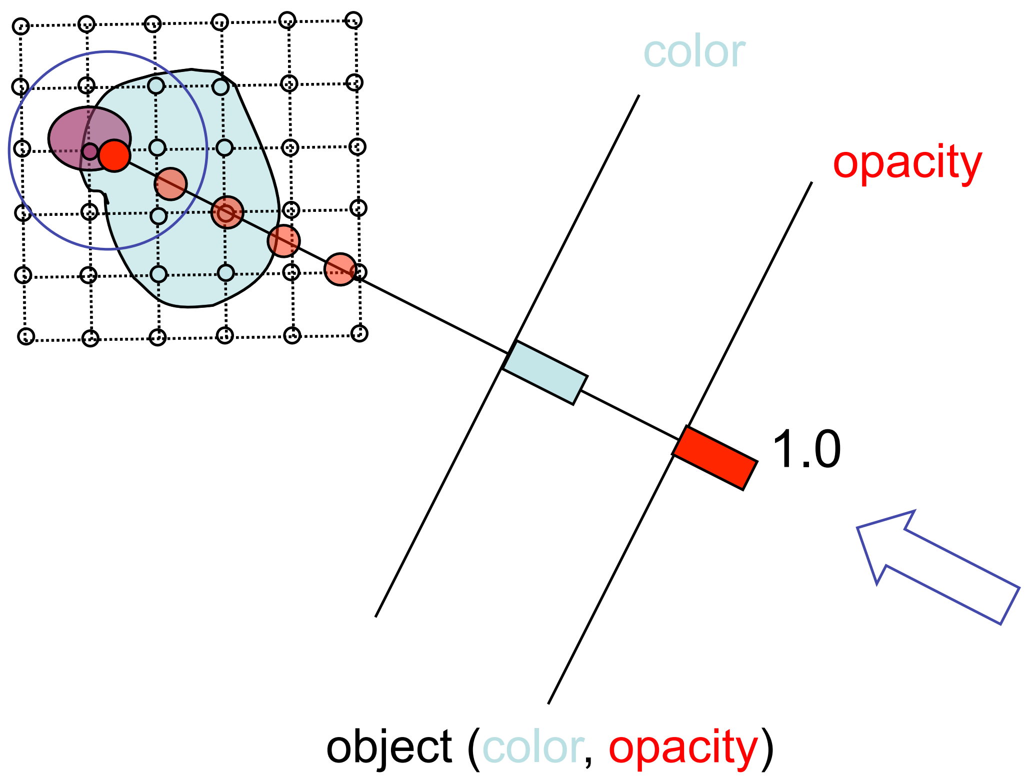

volumetric compositing
color
opacity
1.0
object (color, opacity)

Raycasting


volumetric compositing
color
opacity
1.0
object (color, opacity)

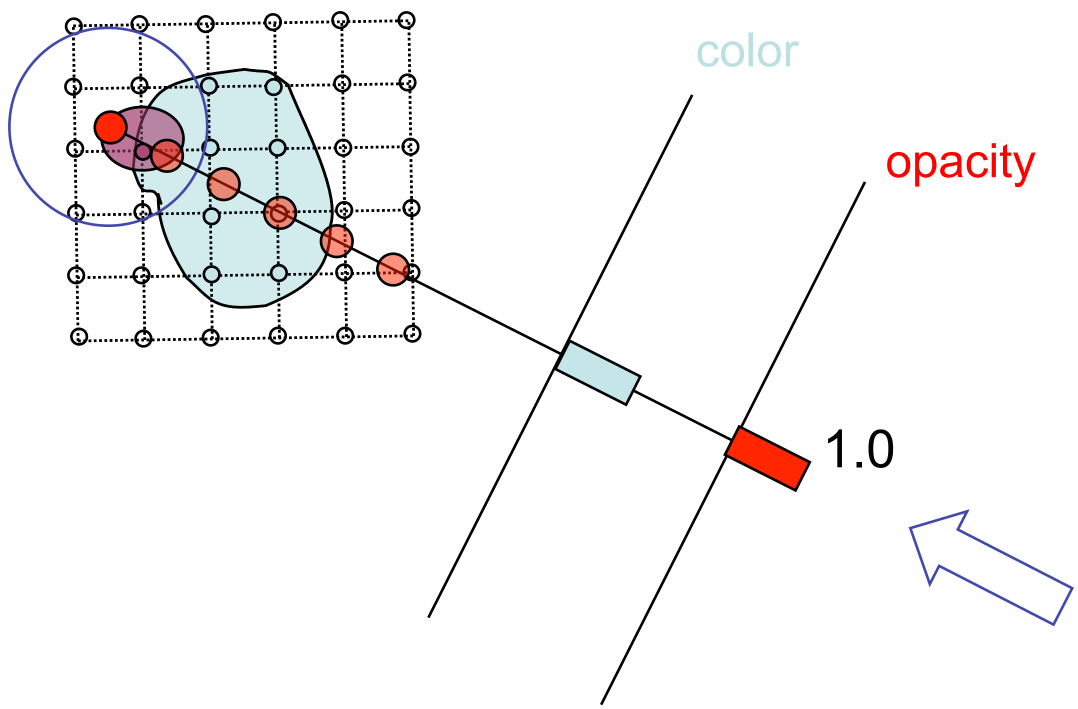
Raycasting

volumetric compositing



Raycasting

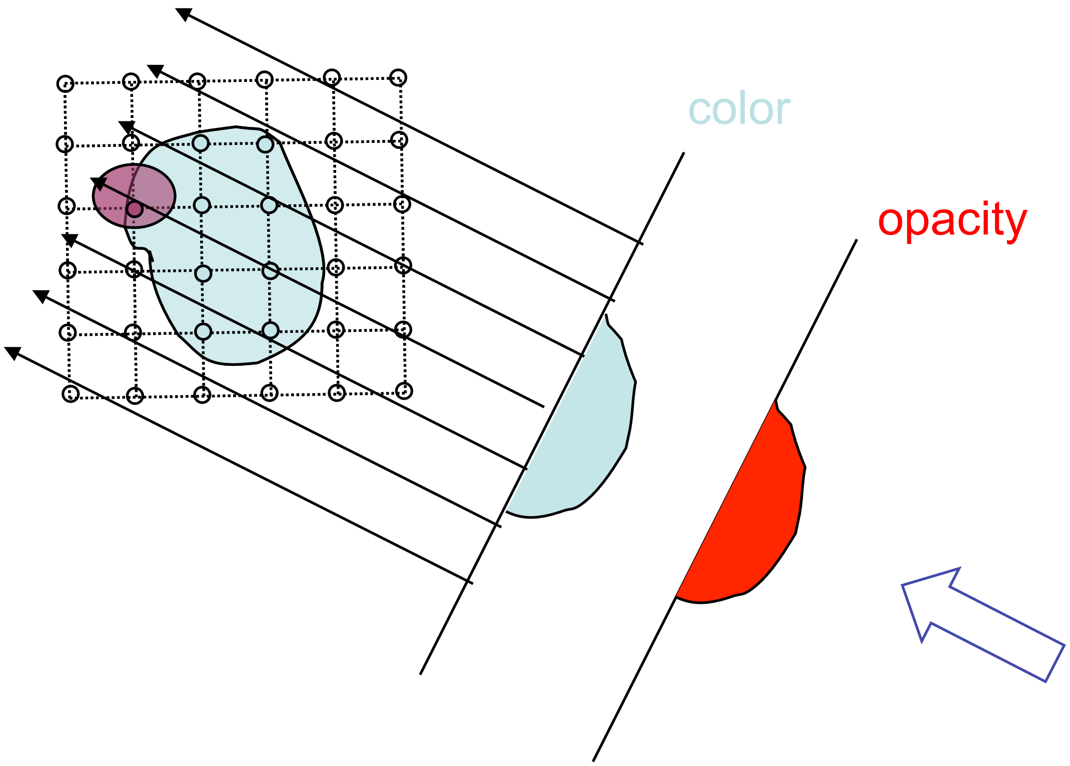
volumetric compositing



object (color, opacity)

Raycasting

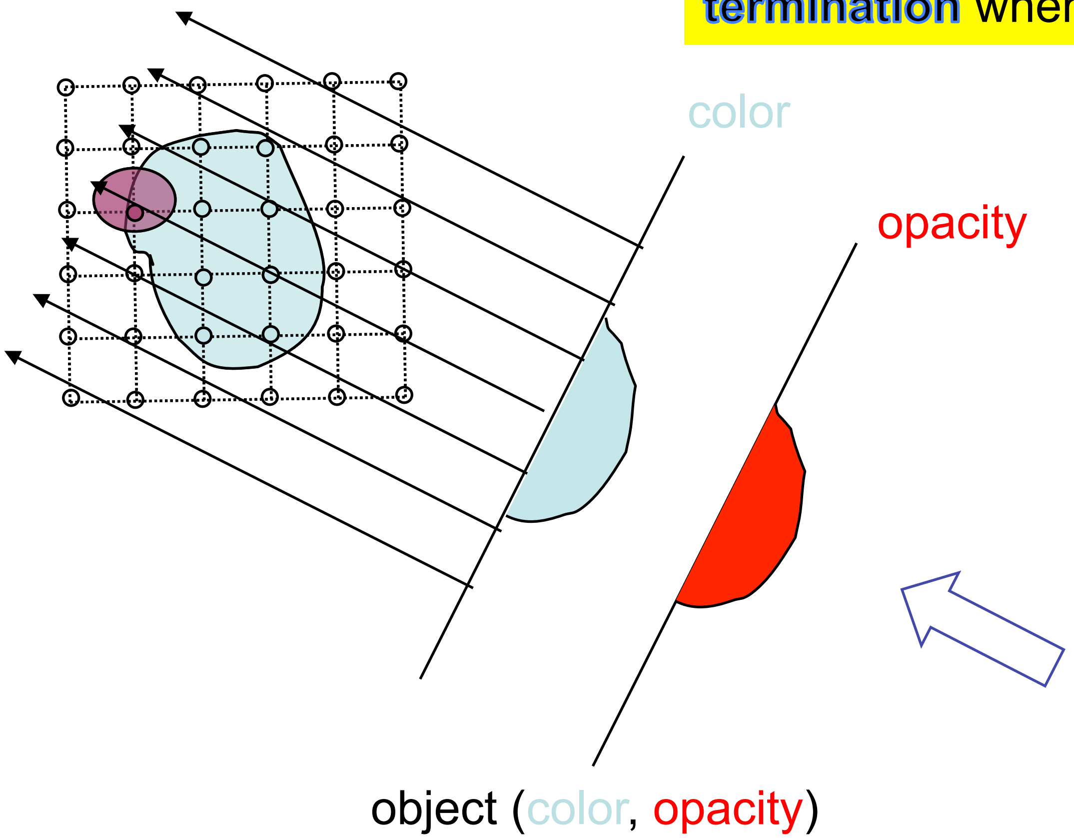
volumetric compositing



object (color, opacity)

Raycasting

If alpha is close enough to 1.0, the color will not change much. Therefore a threshold for alpha (the transparency) may be set, guarantees an **early ray termination** when possible.



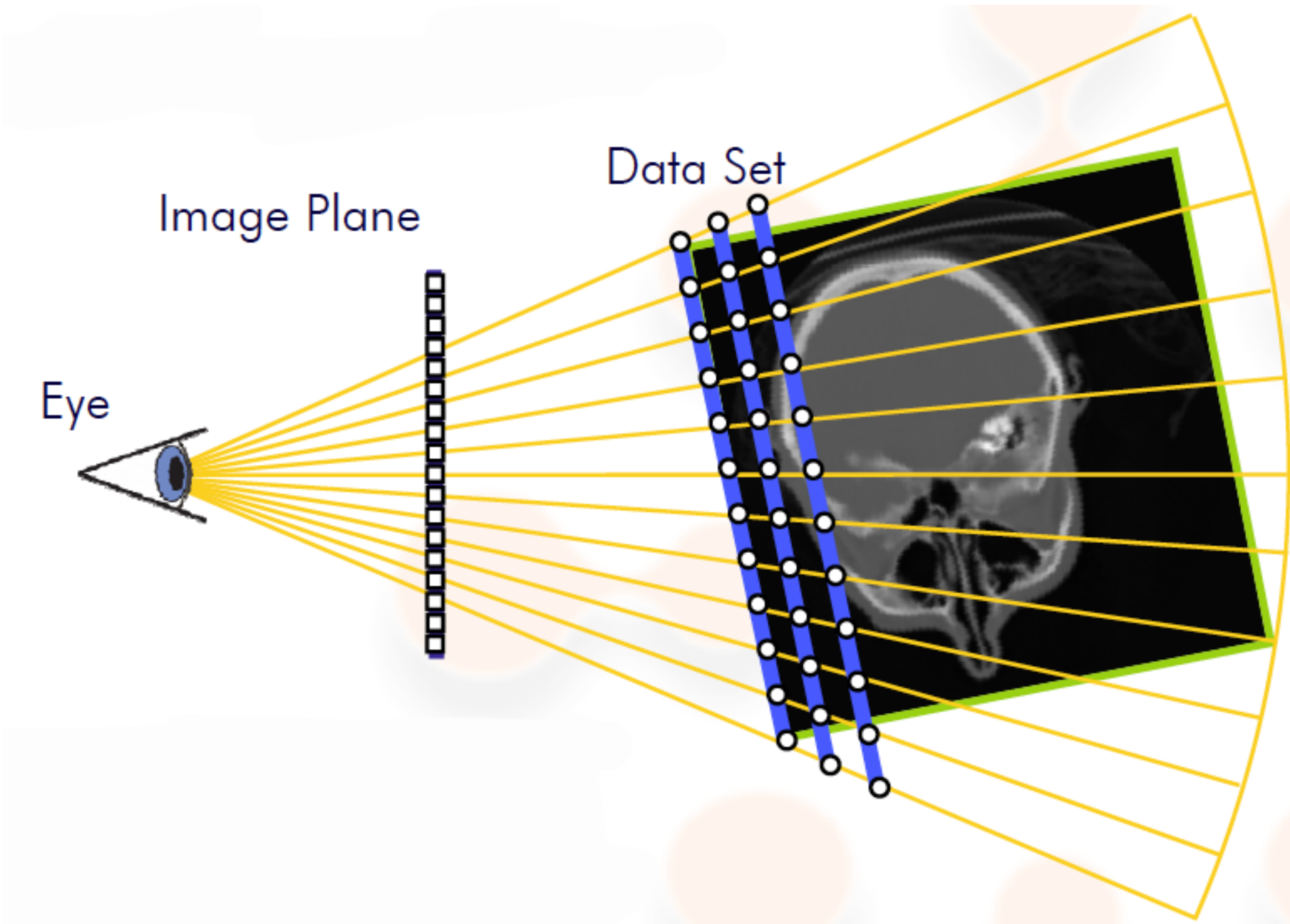
Ray Casting - Performance Improvements

- **Use of Octrees**
  - Minimizes the number transparent voxels during the accumulation, since a group of transparent voxels may be represented as a single node in the octree.
  - More efficient memory usage.
- **Interleaving methods**
  - Sample every two (*n*) voxels as long as voxels are fully transparent.
  - Sample only ¼ of the points in the image and interpolate - Faster for interactive mode, but less quality.
- **Pyramids, k-d trees and other data-structures.**

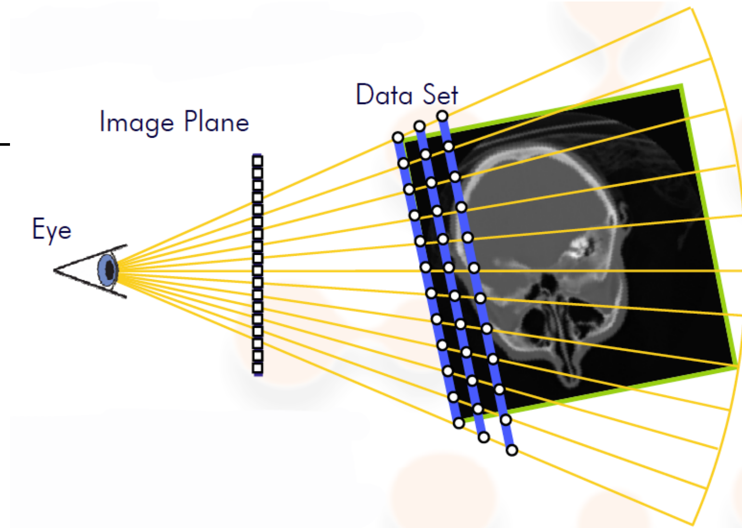
Ray Casting - Improving Image Quality

- **Multi Cast or Super Sampling:**
  - Instead of sampling one ray per pixel, sampling 4 rays per pixel.
  - Better image... but four times longer to render.
- **Ray subdividing:**
  - Used with perspective projection. When the rays draw away from each other, the sampling of the volume is not complete.
  - The solution is to divide the ray when the rays density falls.

Splatting



Splatting



- Traverse voxels in front to back order
  - Traverse each voxel in plane, then move to next plane
- For each voxel, accumulate color and opacity to each pixel it covers
- Voxel projection covers hexagonal footprint
- Smooth interpolation possible by applying kernel with fall-off away from sample point

Ray Casting vs. Splatting

- **Ray casting**
  - Point samples
  - Random data access
  - Easy for parallel or perspective projection
- **Splatting**
  - Area samples
  - Ordered data access
  - More difficult for perspective projection

Shading and Classification

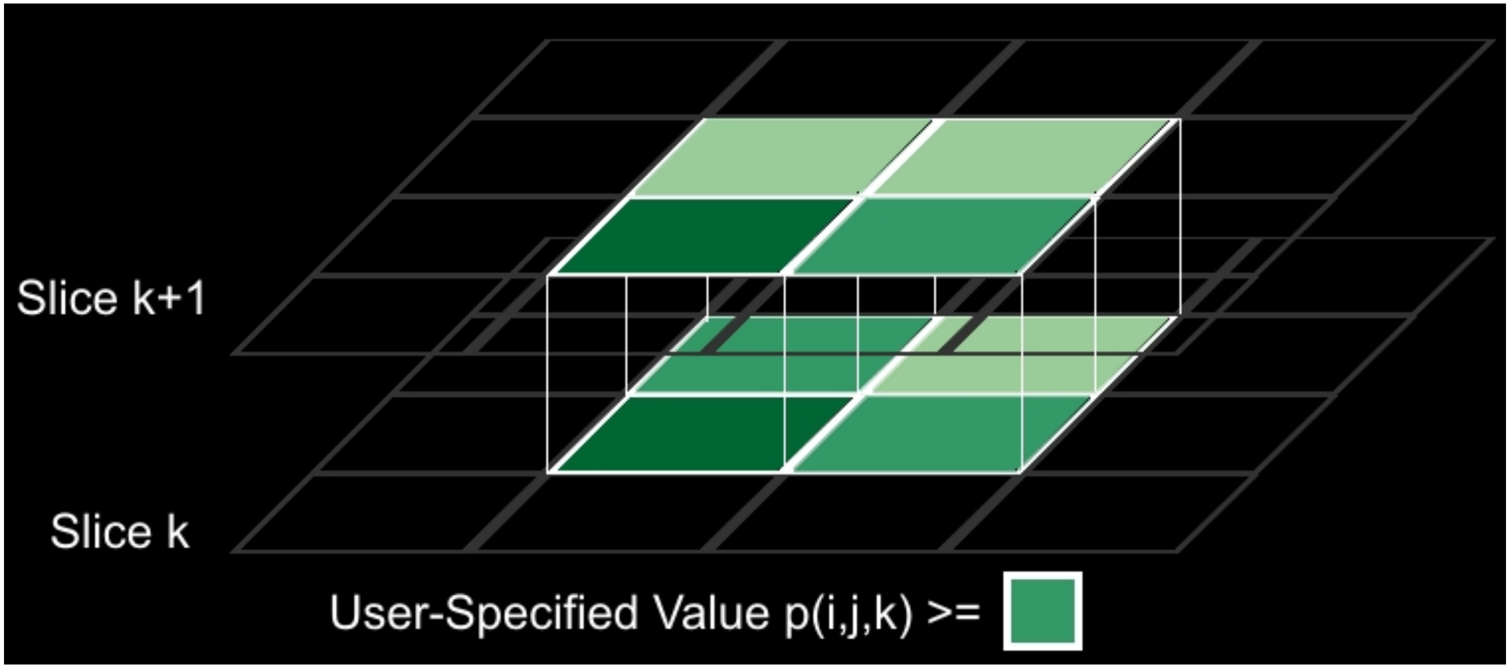
- **Row volume data does not include normal or edges**
  - Edge detection and normal calculation should be done.
  - Using
    - Marching Cube algorithm
    - Gradient estimation
- **Classification**
  - Most volume data is only the density value. Read colors and transparency are set by classifying the voxels using some classification algorithms.

Marching Cubes Algorithm

- **Consists of 3 basic steps:**
  - Locate the surface corresponding to a user -specified value.
  - Create triangles.
  - Calculate normals to the surface at each vertex.

Step 1:Surface Intersection

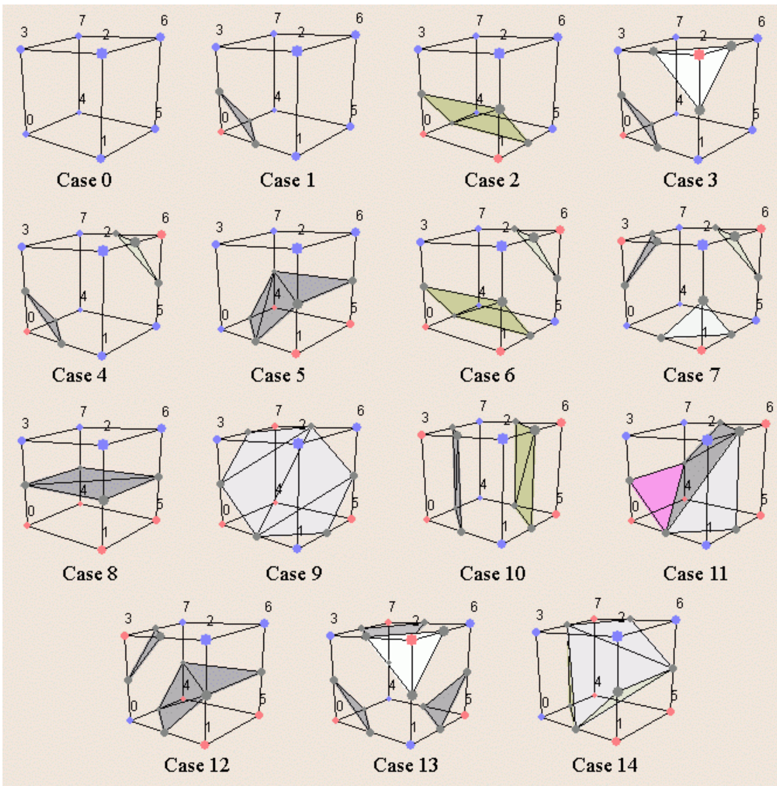
- To locate the surface, it uses a logical cube created from eight pixels (Four each from 2 adjacent layers):



Step 1: Surface Intersection

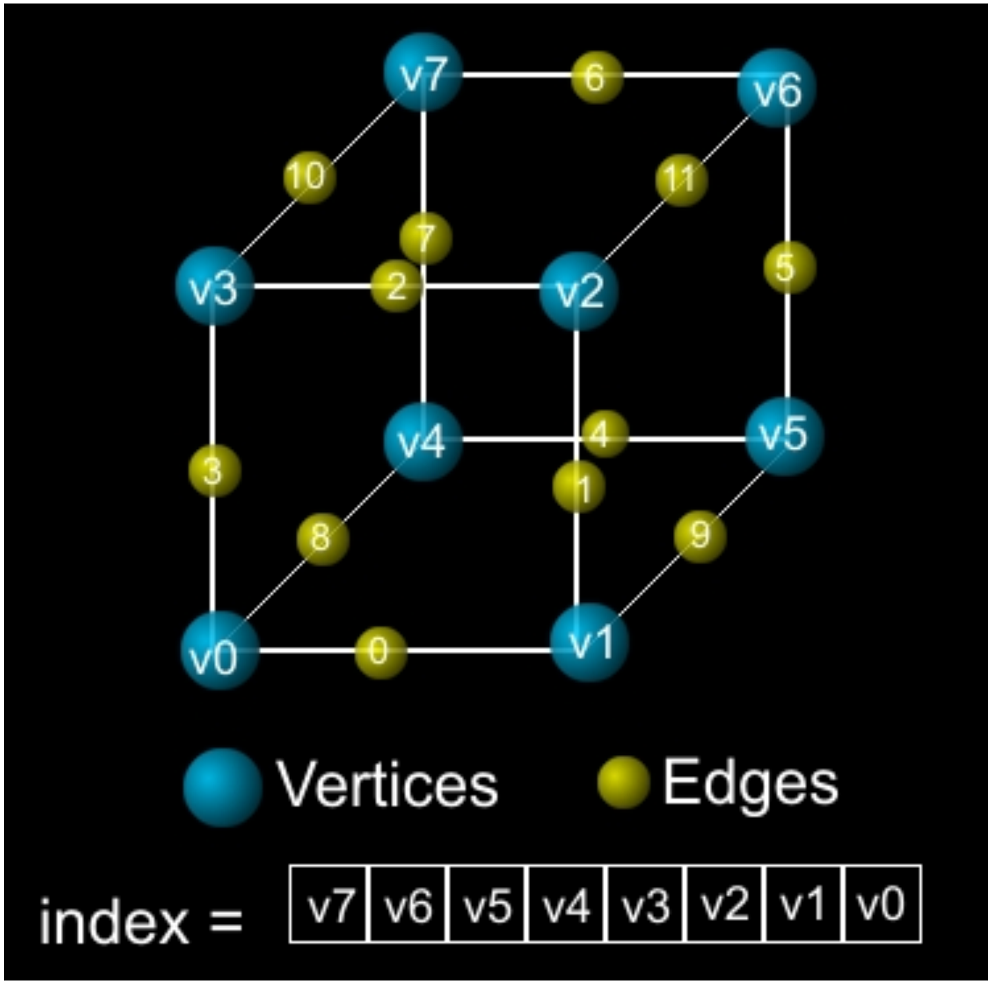
- **Binary vertex assignment: (p (i, j, k) >= TU ? 1: 0)**
  - Set cube vertex to value of 1 if the data value at that vertex exceeds (or equals) the value of the surface we are constructing
  - Otherwise, set cube vertex to 0
- **If a vertex = 1 then it is "inside" the surface**
- **If a vertex = 0 then it is "outside"**
- **Any cube with vertices of both types is "intersected" by the surface.**

Step 2 : Triangulation



- For each cube, we have 8 vertices with 2 possible states each (inside or outside).
- This gives us 28 possible patterns = 256 cases.
- Enumerate cases to create a LUT
- Use symmetries to reduce problem from 256 to 15 cases.

Step 2 : Triangulation



- Use vertex bit mask to create an index for each case based on the state of the vertexes.
- Using the index to tell which edge the surface intersects, we can then can linearly interpolate the surface intersection along the edge.

Step 3 : Surface normals

- To calculate surface normal, we need to determine gradient vector, $\vec{g}$ (derivative of the density function).
- To estimate the gradient vector at the surface of interest, we first estimate the gradient vectors at the vertices and interpolate the gradient at the intersection.
- The gradient at cube vertex (i , j, k), is estimated using central differences along the three coordinate axes by:

D (i, j, k) is the density at pixel (i, j) in slice k.

Δx, Δy, Δz are lengths of the cube edges

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

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

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

Step 3 : Surface normals

- Dividing the gradient by its length produces the unit normal at the vertex required for rendering.
- Then the algorithm linearly interpolates this normal to the point of intersection.

Algorithm Summary

- Scan 2 slices and create cube
- Calculate index for cube based on vertices
- Use index to lookup list of edges intersected
- Use densities to interpolate edge intersections
- Calculate unit normal at each edge vertex using central differences. Interpolate normal to each triangle vertex
- Output the triangle vertices and vertex normals
- March to next position and repeat.