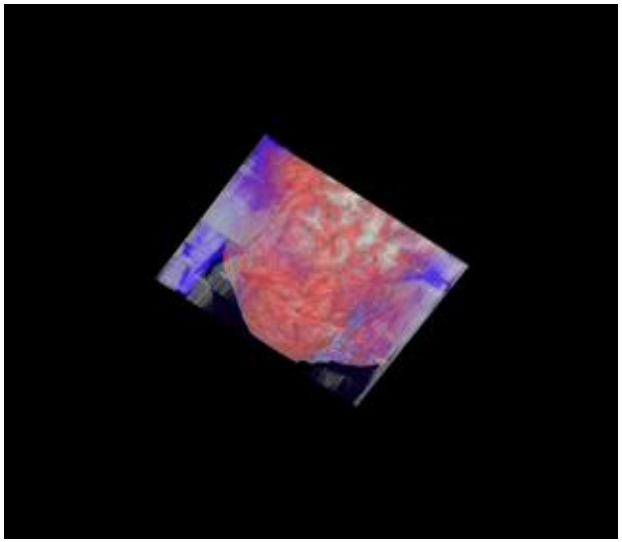
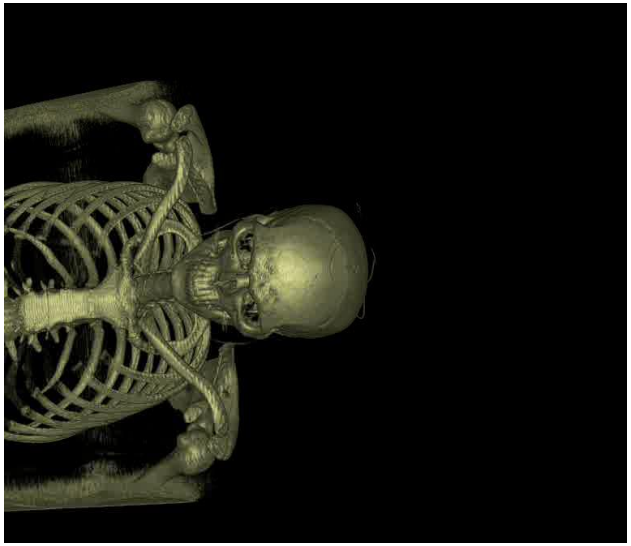
Visualization

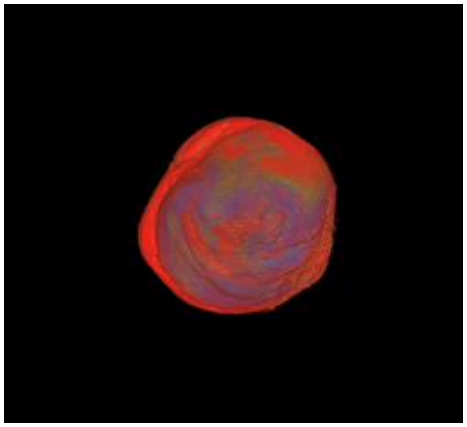
Prof. Vaibhav Vasani

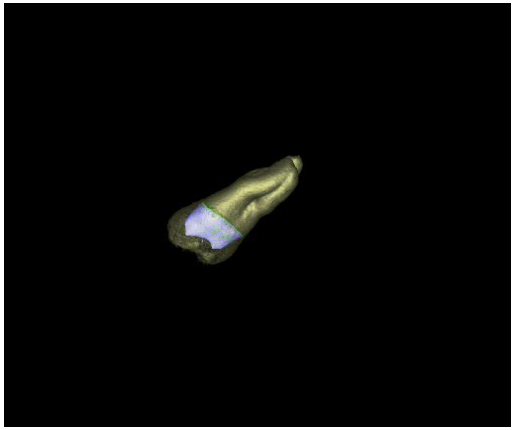
Volume Rendering

Example Volume Renderings



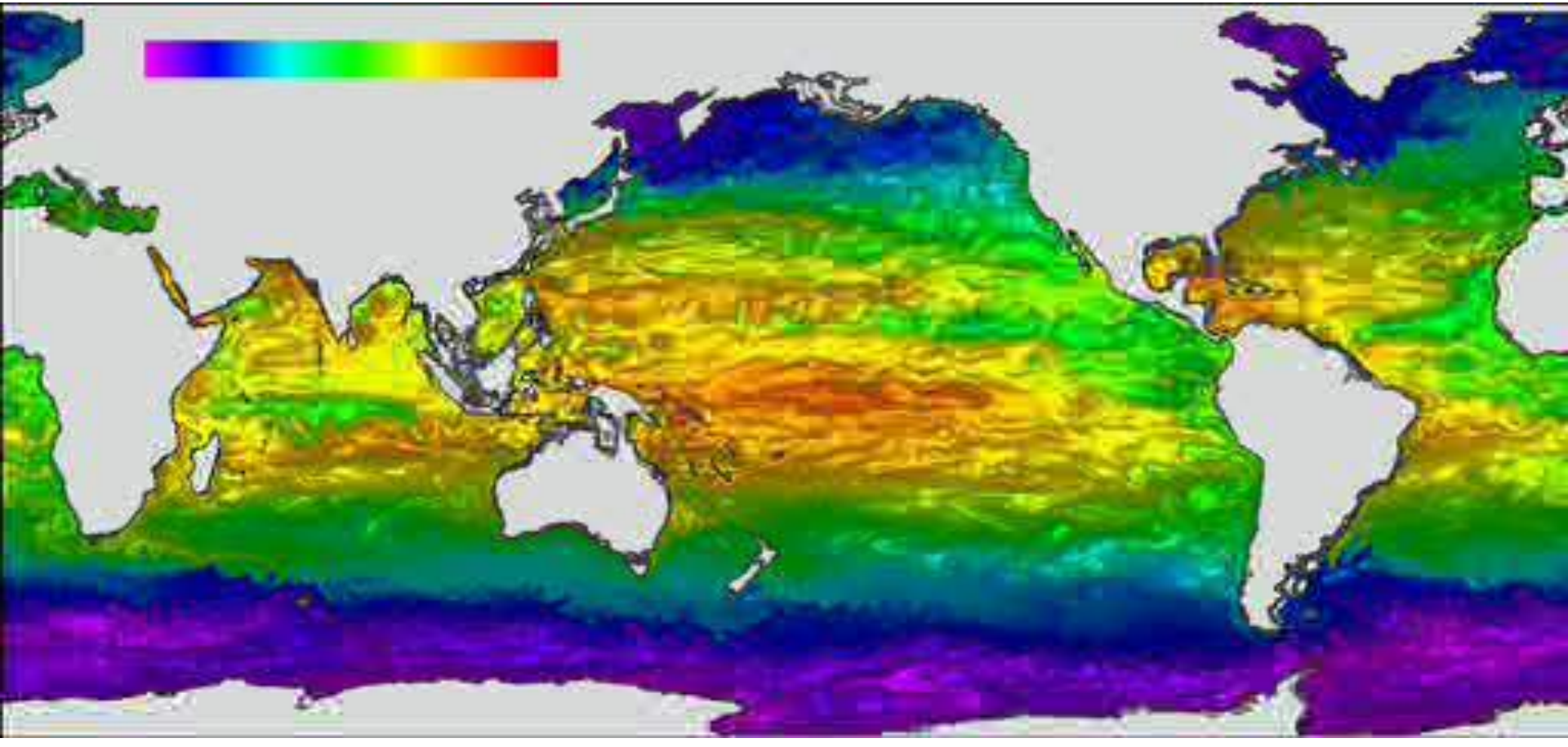






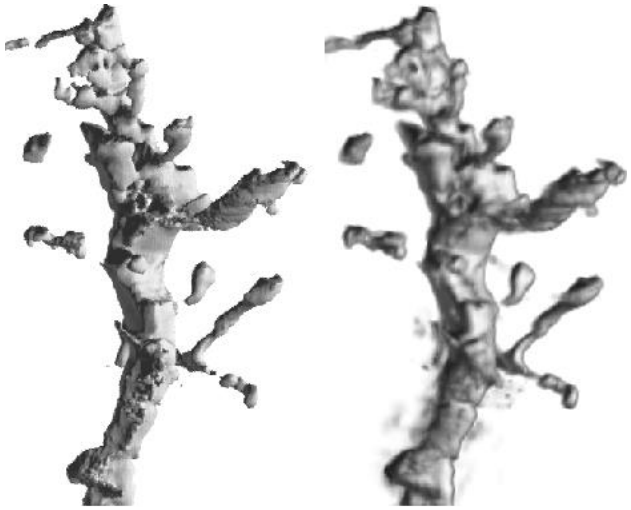
Oceanographic Simulations

- 2160×960×30×4(bytes) = 237 MB
- 237(MB)×115(timesteps) = 27 GB



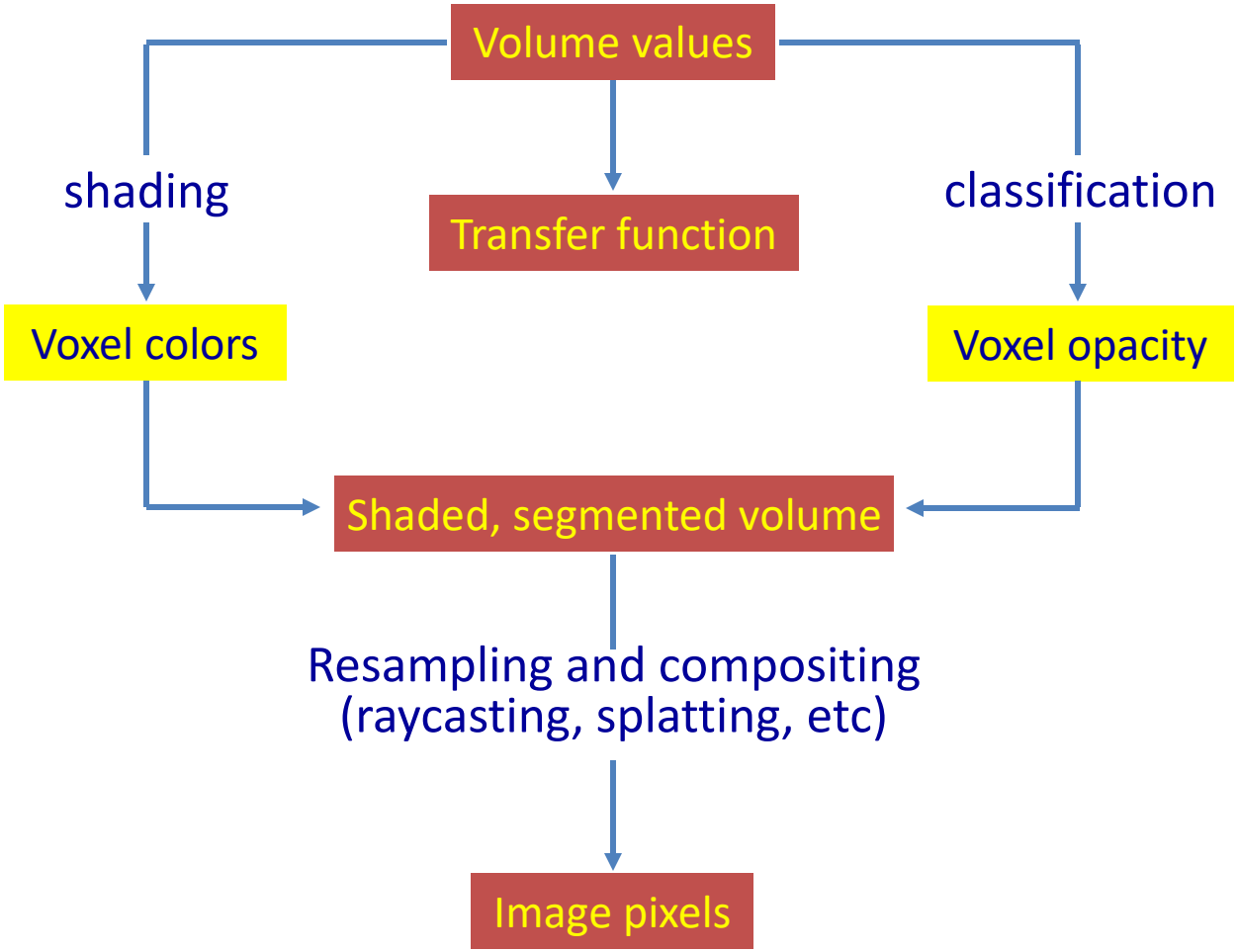
Volume Rendering Algorithm

- Direct volume rendering
  - Ray-casting
  - Splatting
- Indirect volume rendering
  - Fourier
- Texture based volume rendering
  - 3D Texture mapping hardware



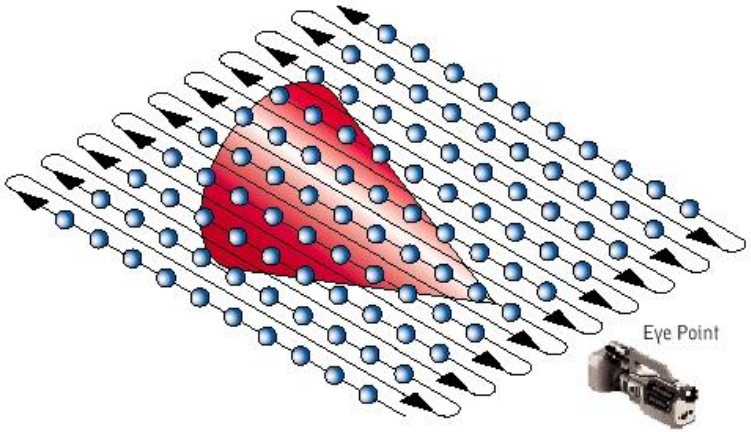
(a) Isosurface Rendering (b) Direct Volume Rendering

Traditional Volume Rendering Pipeline

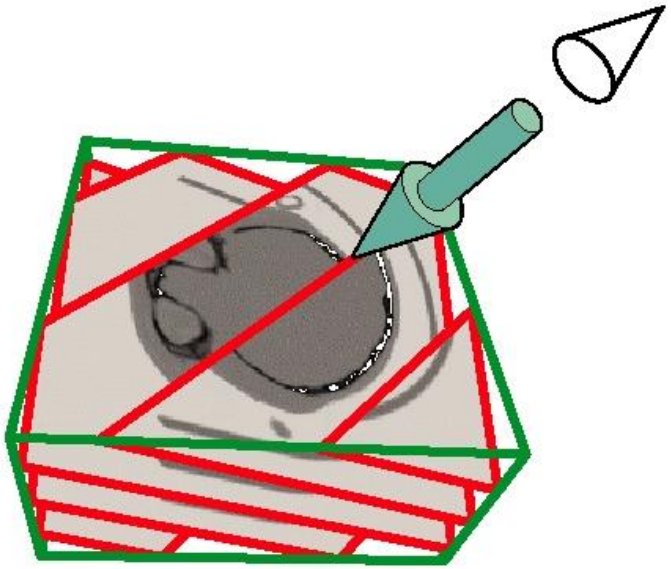


Texture Based Volume Rendering

- 3D Texture mapping hardware

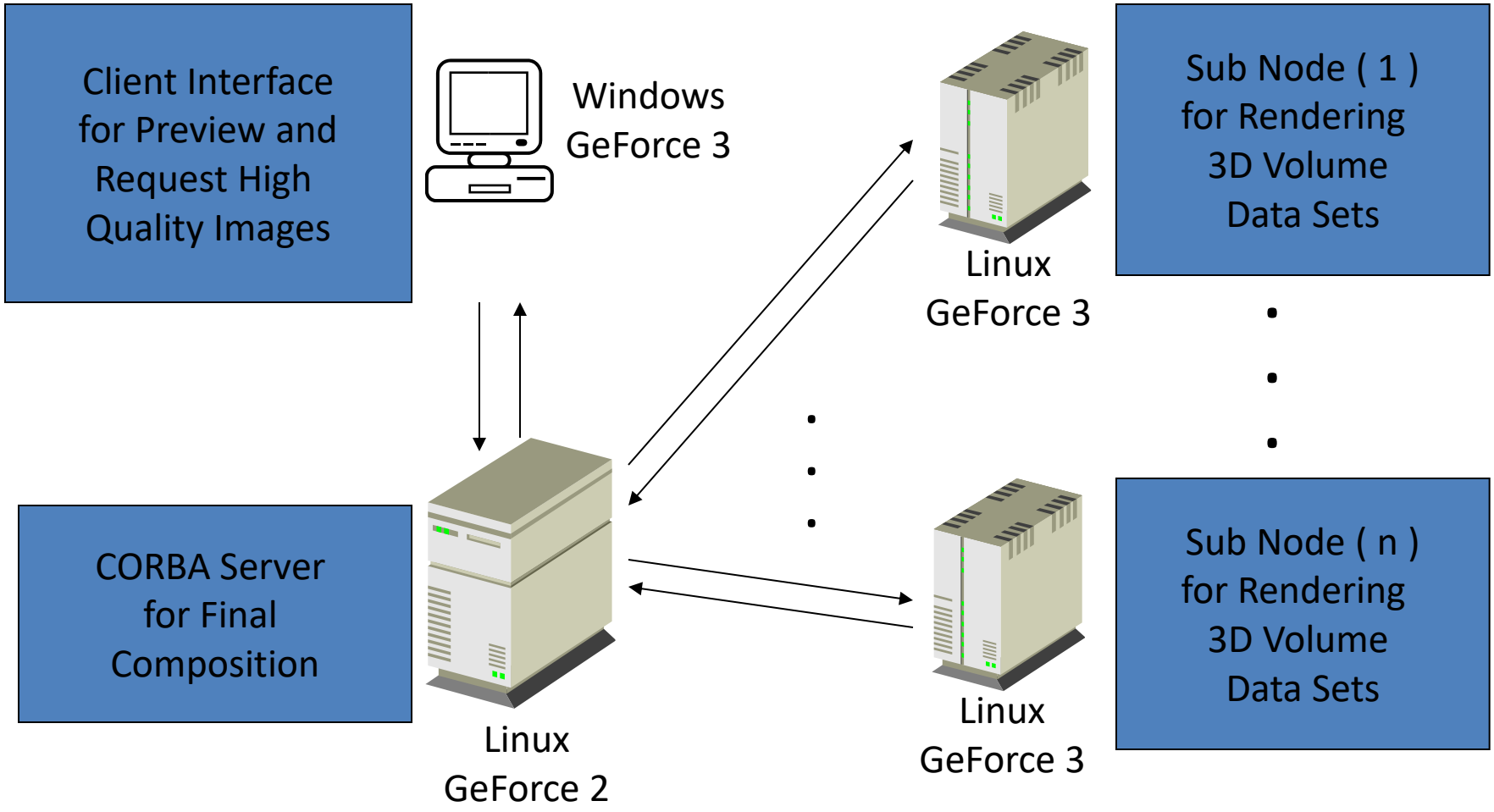


Ray Casting



3D Texture Mapping

System Diagram



Client-server Algorithm

1. Adjust color table & transfer function using Windows interface.
2. Send a request to CORBA server.
3. The CORBA server distributes work to each node using MPI.
4. Each node renders each part of data using back-to-front composition.
5. The CORBA server takes the image pieces from each node and composites them into an image.
6. The Windows interface takes the final image.

Hardware Accelerated Rendering Algorithm

1. Load a 3D indexed volume data and normal vectors as RGB to texture memory on GF3
2. Set up a color look-up table
3. Set up combiners of GF3 for shading for color of texture , diffuse and specular
4. Calculate intersection between texture cube and texture mapped planes parallel with view planes
5. Composite the texture mapped planes using back-to-front composition

Hardware Accelerated Rendering

3D Indexed Volume Data

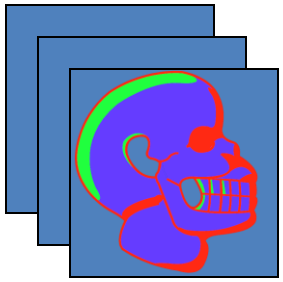
Color Look-up Table

Index	R	G	B	A
0	0	0	0	0
1	10	10	0	100
…				
255	100	0	255	255

Texture Mapped Color Images

Pixels are Combined by GF3 Combiners

Normal Vectors as RGB Texture



Normal Mapped Images

Dot Products with a Light Vector for Diffuse

Dot Products with a H Vector for Specular

Shaded Final Image

Direct Volume Rendering (DVR): Ray-casting

Light: bouncing photons

- Lights send off photons in all directions
  - Model these as particles that bounce off objects in the scene
  - Each photon has a wavelength and energy (color and intensity)
  - When photon bounce, some energy is absorbed, some reflected, some transmitted
- If we can model photon bounces we can generate image
- Techniques: follow each photon from the light source until
  - All of its energy is absorbed (after too many bounces)
  - It departs the known universe
  - It strikes the image and its contribution is added to appropriate pixel

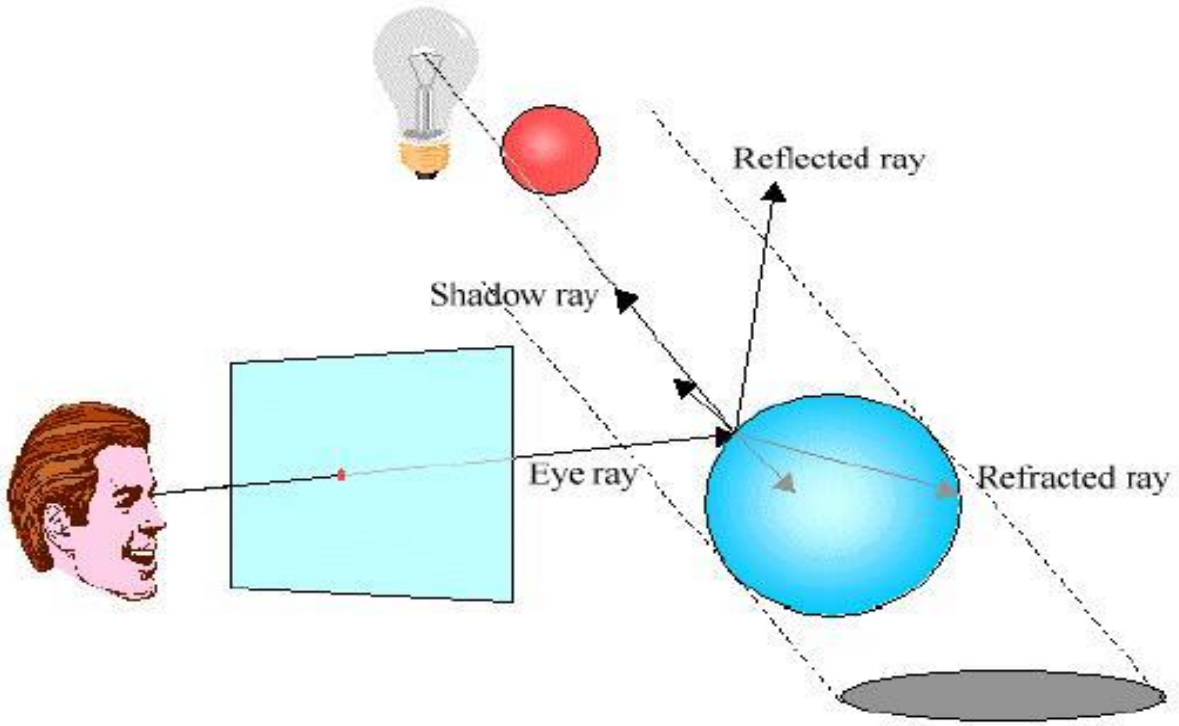
Forward Ray Tracing

- Rays are the path of these photons
- This method of rendering by following photon path is called ray tracing
- Forward ray tracing follows the photon in direction that light travels (from the source)
- Big problem with this approach:
  - Only a tiny fraction of rays will reach the image
  - Extremely slow
- Ideal Scenario:
  - We'd like to magically know which rays will eventually contribute to the image, and trace only those

Direct: No conversion to surface geometry

Backward Ray Tracing

- The solution is to start from the image and trace backwards
  - Start from the image and follow the ray until the ray finds or fail to find a light source

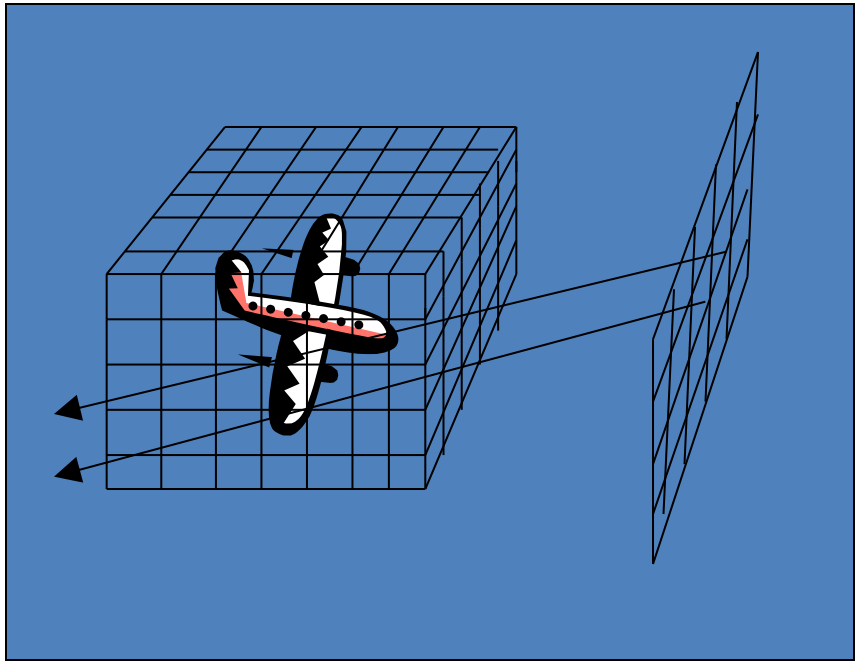


Backward Ray Tracing

- Basic ideas:
  - Each pixel gets light from just one direction- the line through the image point and focal point
  - Any photon contribute to that pixel's color has to come from this direction
  - So head in that direction and find what is sending light this way
  - If we hit a light source-we're done
  - If we find nothing-we're done
- At the end we've done forward ray tracing, but only for rays that contribute to the image

Basic Idea: Ray Casting

Based on the idea of ray tracing



- Only consider Primary Ray
- Trace from eat each pixel

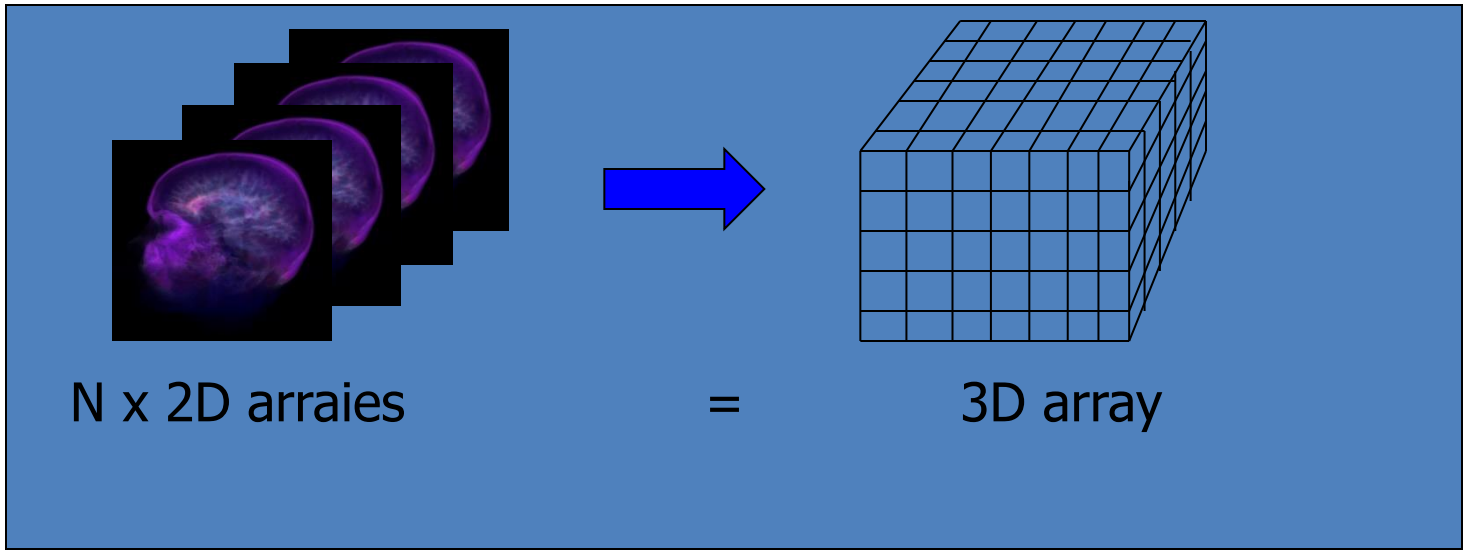
as a ray into object space

- Compute color value along the ray, composite
- Assign the value to the pixel

Data Representation

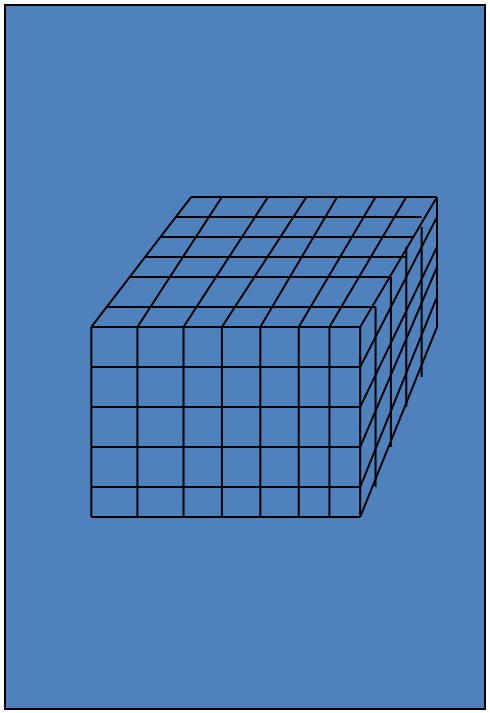
•3D volume data are represented by a finite number of cross sectional slices (hence a 3D raster)

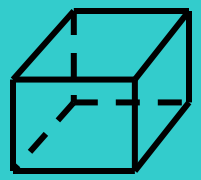
•On each volume element (voxel), stores a data value (if it uses only a single bit, then it is a binary data set. Normally, we see a gray value of 8 to 16 bits on each voxel.)



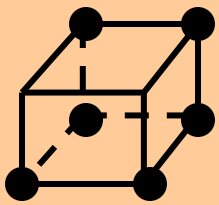
Data Representation (2)

What is a Voxel? – Two definitions





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

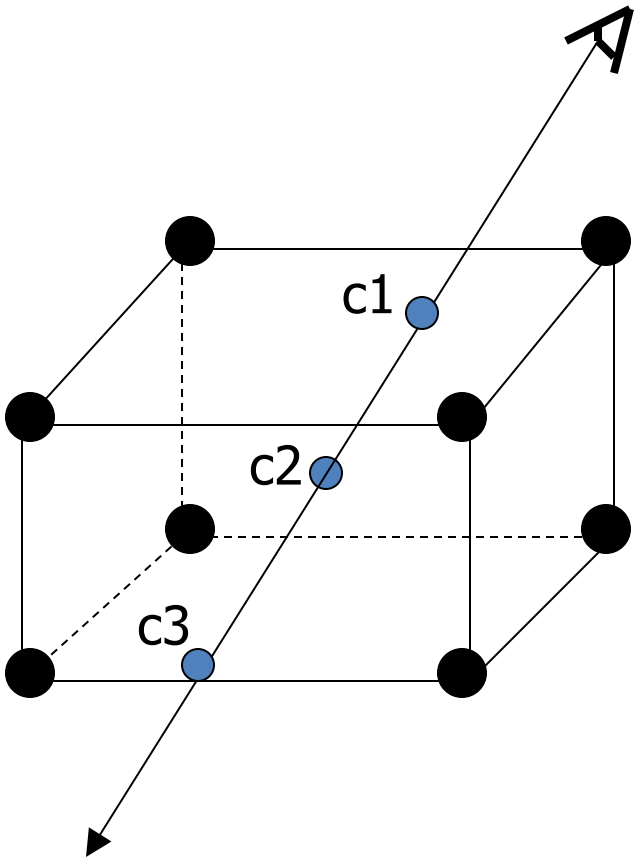


A voxel is a data point at a corner of the cubic cell
The value of a point inside the cell is determined by interpolation

Ray Casting

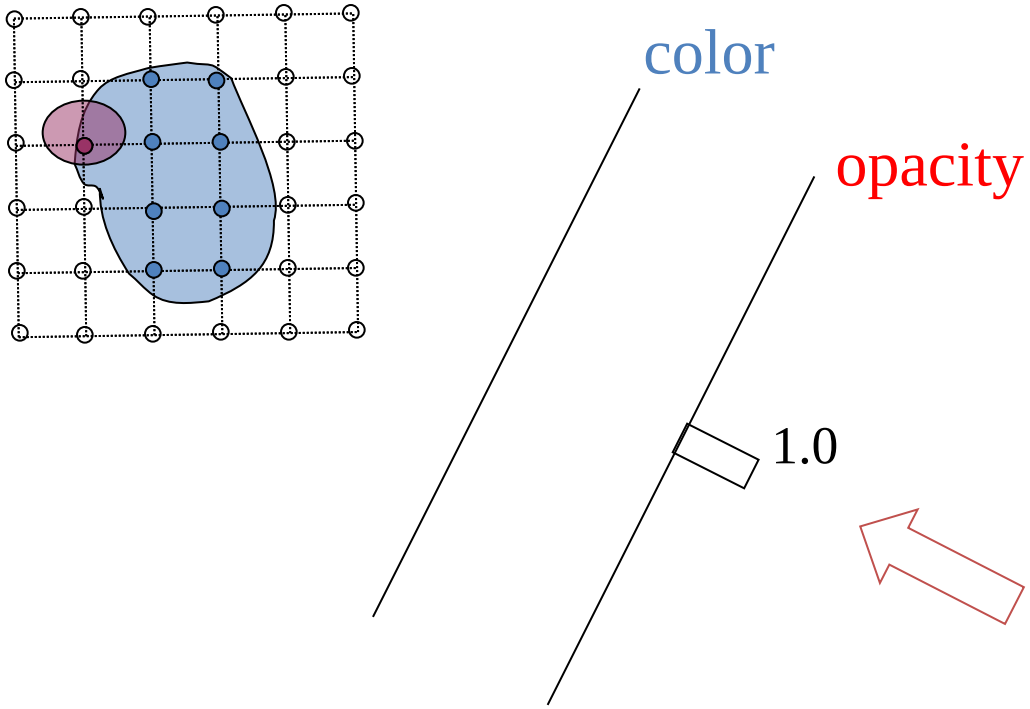
- Stepping through the volume: a ray is cast into the volume, sampling the volume at certain intervals
- The sampling intervals are usually equi-distant, but don’t have to be
- At each sampling location, a sample is interpolated / reconstructed from the grid voxels
- popular filters are: nearest neighbor (box), trilinear (tent), Gaussian, cubic spline
- Classification: map from density to color, opacity
- Composite colors and opacities along the ray path

Basic Idea of Ray-casting Pipeline

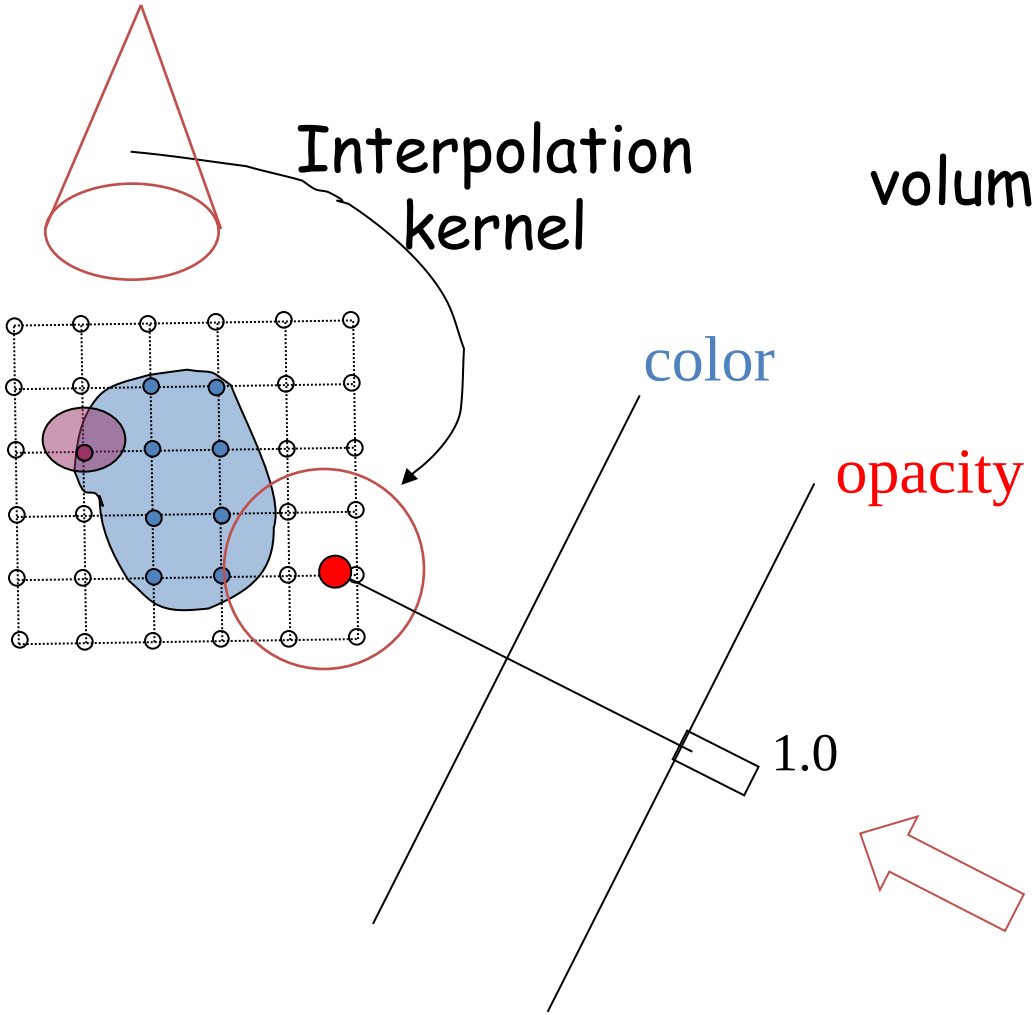


Raycasting

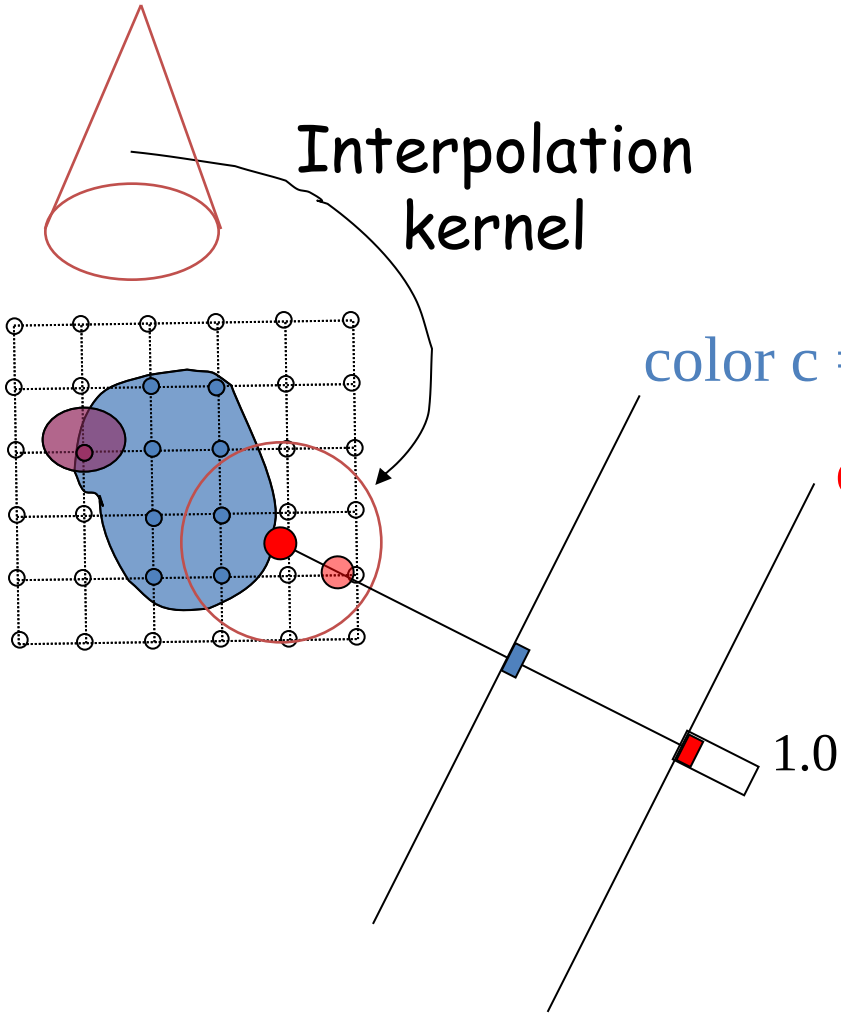
volumetric compositing



Raycasting


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

Raycasting



volumetric compositing

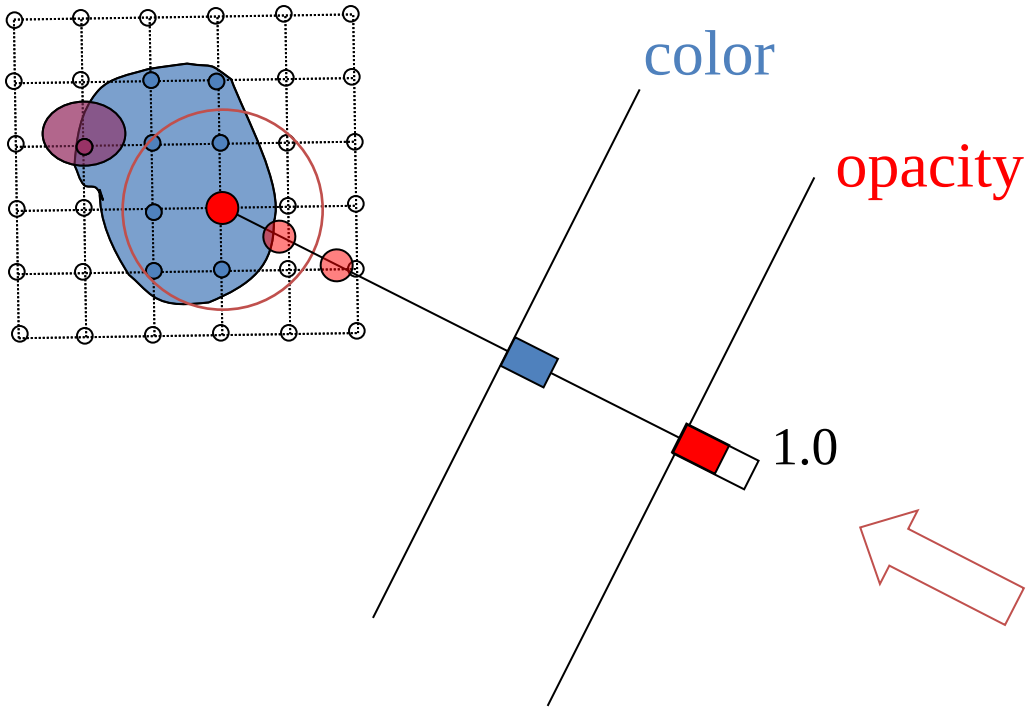
color $c = c_s \alpha_s(1 - \alpha) + c$

opacity $\alpha = \alpha_s (1 - \alpha) + \alpha$

object (color, opacity)

Raycasting

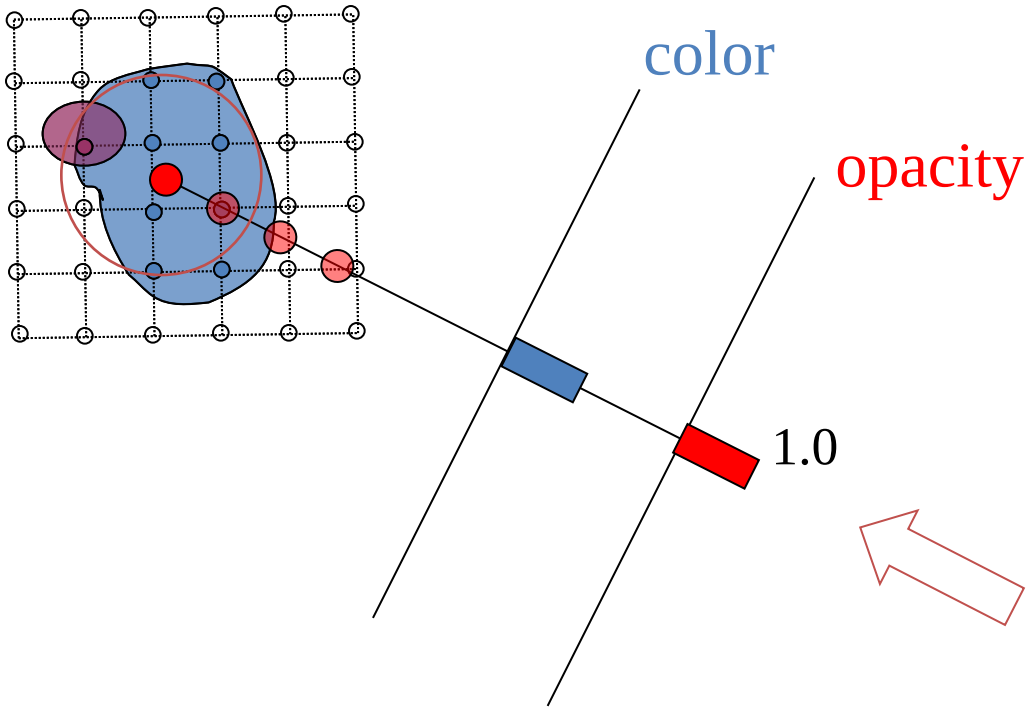
volumetric compositing



object (color, opacity)

Raycasting

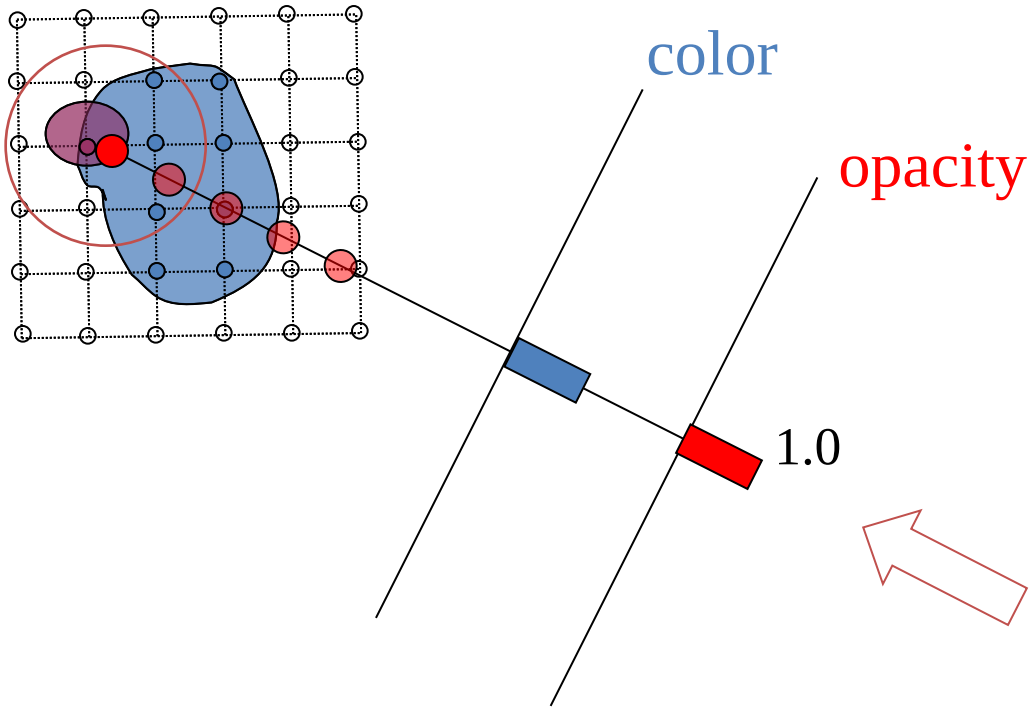
volumetric compositing



object (color, opacity)

Raycasting

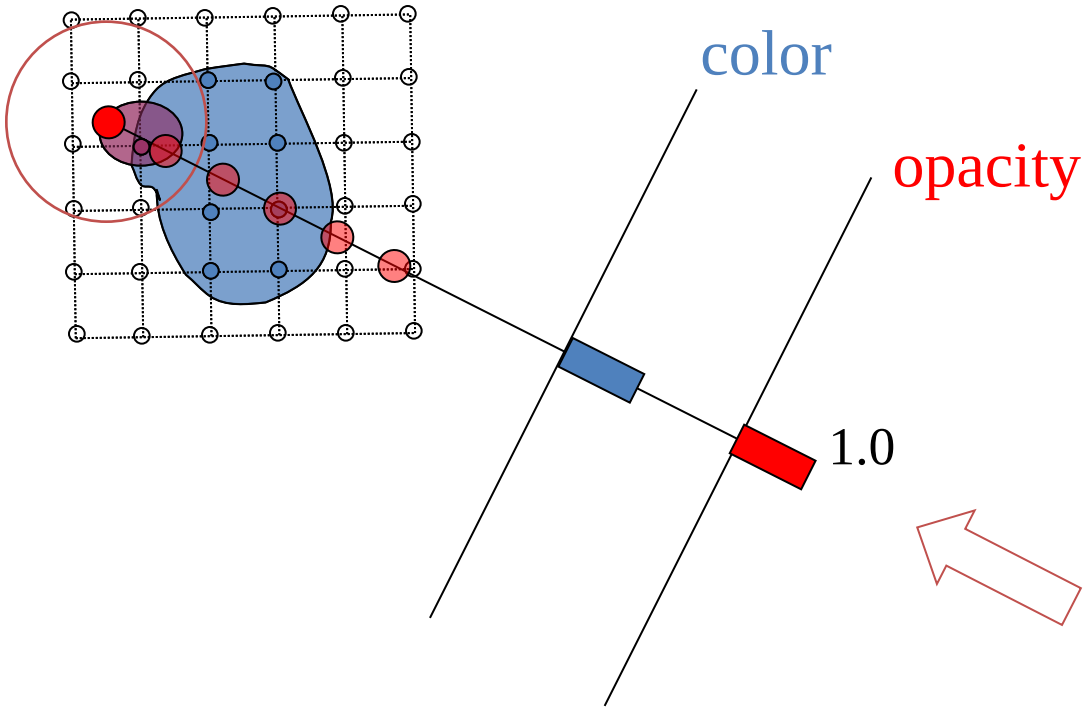
volumetric compositing



object (color, opacity)

Raycasting

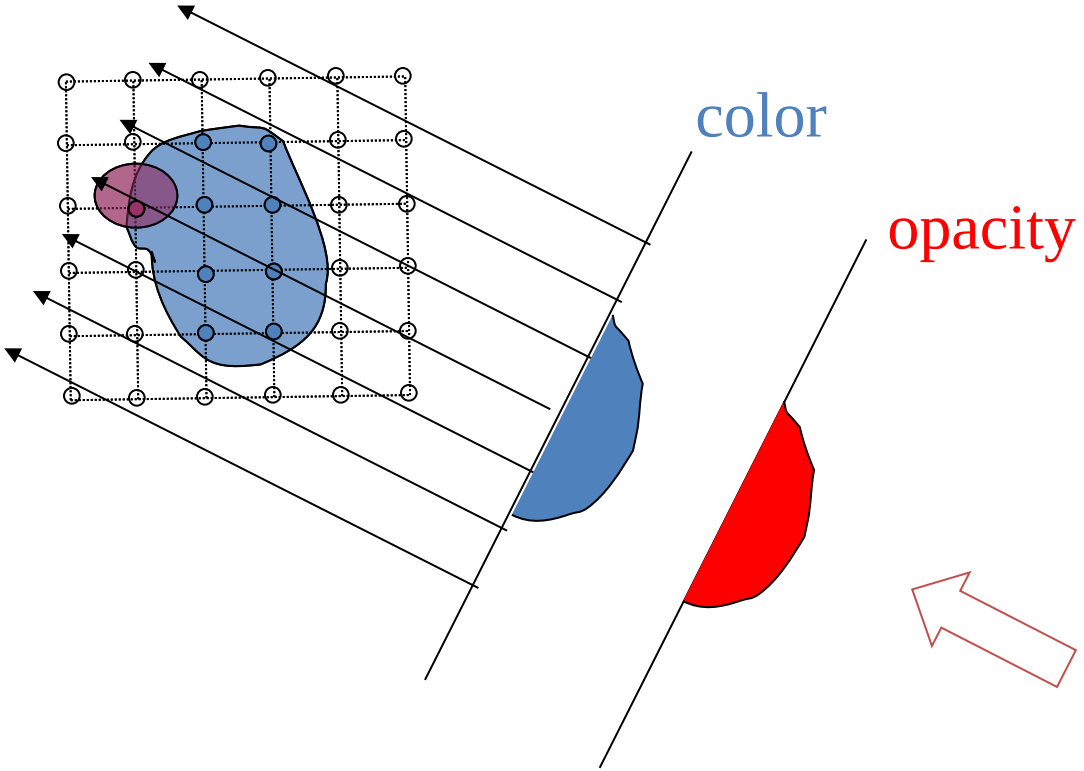
volumetric compositing



object (color, opacity)

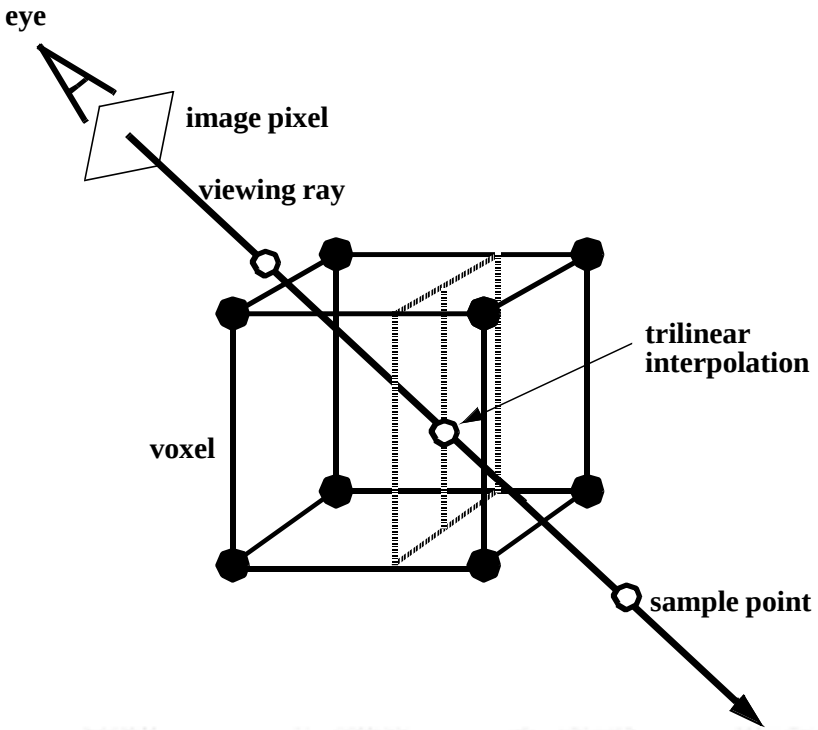
Raycasting

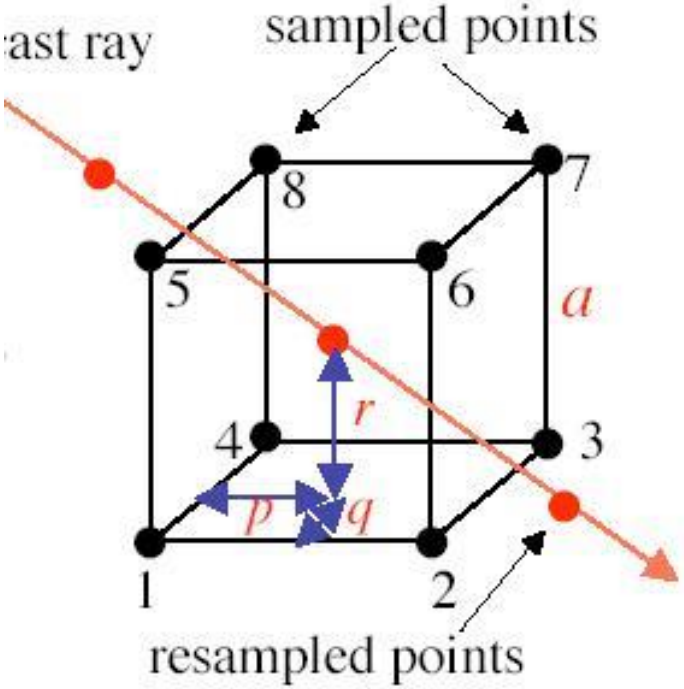
volumetric compositing



object (color, opacity)

Trilinear - Interpolation





$$f_v = f_1(1-p/a)(1-q/a)(1-r/a) + f_2(p/a)(1-q/a)(1-r/a)$$
$$+ f_3(p/a)(q/a)(1-r/a) + f_4(1-p/a)(q/a)(1-r/a)$$
$$+ f_5(1-p/a)(1-q/a)(r/a) + f_6(p/a)(1-q/a)(r/a)$$
$$+ f_7(p/a)(q/a)(r/a) + f_8(1-p/a)(q/a)(r/a)$$

Classification/Transfer Function

- Maps raw voxel value into presentable entities: color, intensity, opacity, etc.

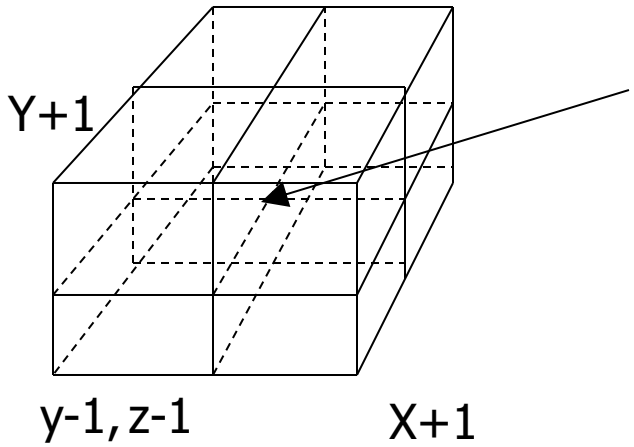
  Raw-data $\rightarrow$ material (R, G, B, α, K_a, K_d, K_s, .)

- Region of interest: high opacity (more opaque)
- no interest: translucent or transparent

- Often use look-up tables (LUT) to store the transfer function

Levoy – Gradient/Normals

- Central difference
- per voxel



$$G_x = \frac{v_{i+1,j,k} - v_{i-1,j,k}}{2}$$

$$G_y = \frac{v_{i,j+1,k} - v_{i,j-1,k}}{2}$$

$$G_z = \frac{v_{i,j,k+1} - v_{i,j,k-1}}{2}$$

Levoy - Shading

- Phong Shading + Depth Cueing

$$C(x) = C_p k_a + \frac{C_p}{k_1 + k_2 d(x)} (k_d (N(x) \cdot L) + k_s (N(x) \cdot H)^n)$$

- C_p = color of parallel light source
- k_a / k_d / k_s = ambient / diffuse / specular light coefficient
- k_1, k_2 = fall-off constants
- d(x) = distance to picture plane
- L = normalized vector to light
- H = normalized vector for maximum highlight
- $N(x_i)$ = surface normal at voxel x_i

Compositing

It is the accumulation of colors weighted by opacities

- Front-to-back

$$C = c_s \cdot \alpha_s (1 - \alpha) + C$$

$$\alpha = \alpha_s \cdot (1 - \alpha) + \alpha$$

Advantage: early ray termination

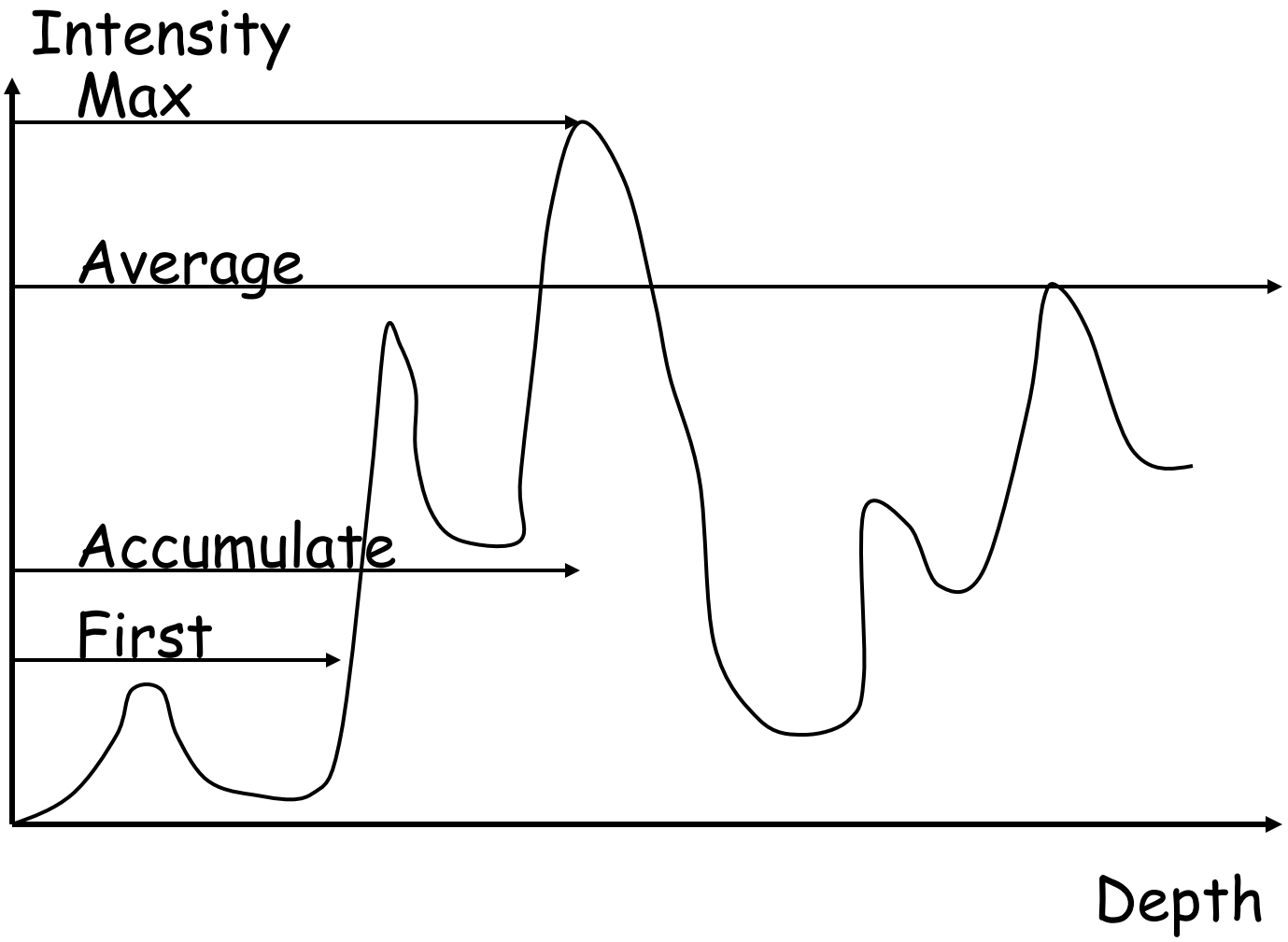
- back-to-front

$$C = C \cdot (1 - \alpha_s) + c_s$$

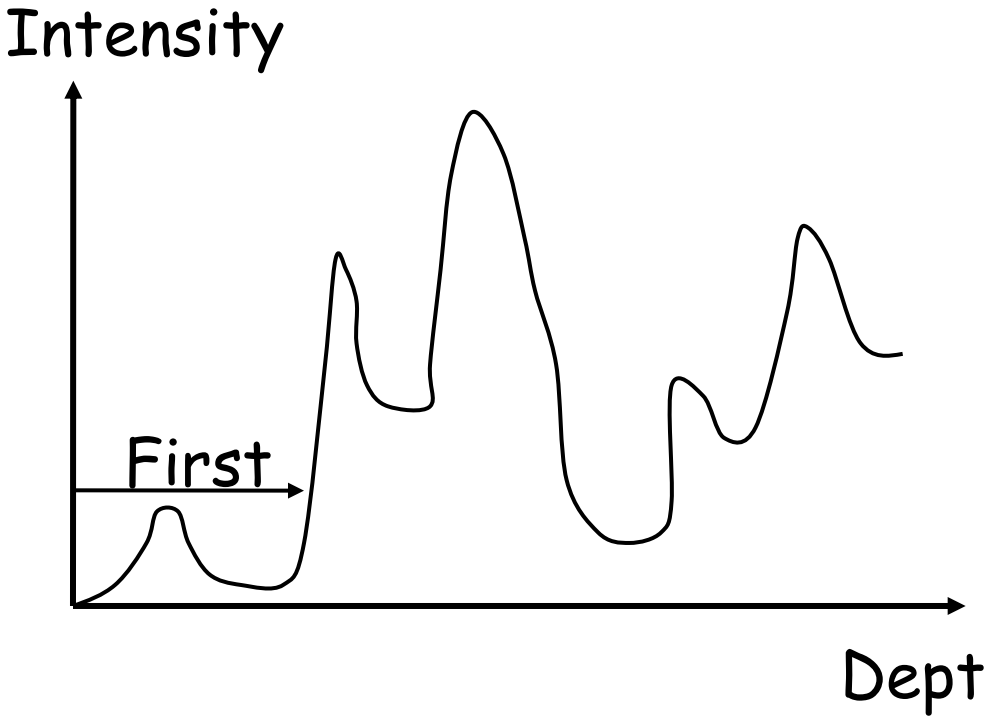
$$\alpha = \alpha \cdot (1 - \alpha_s) + \alpha_s$$

Advantage: object approach suitable for hardware implementation

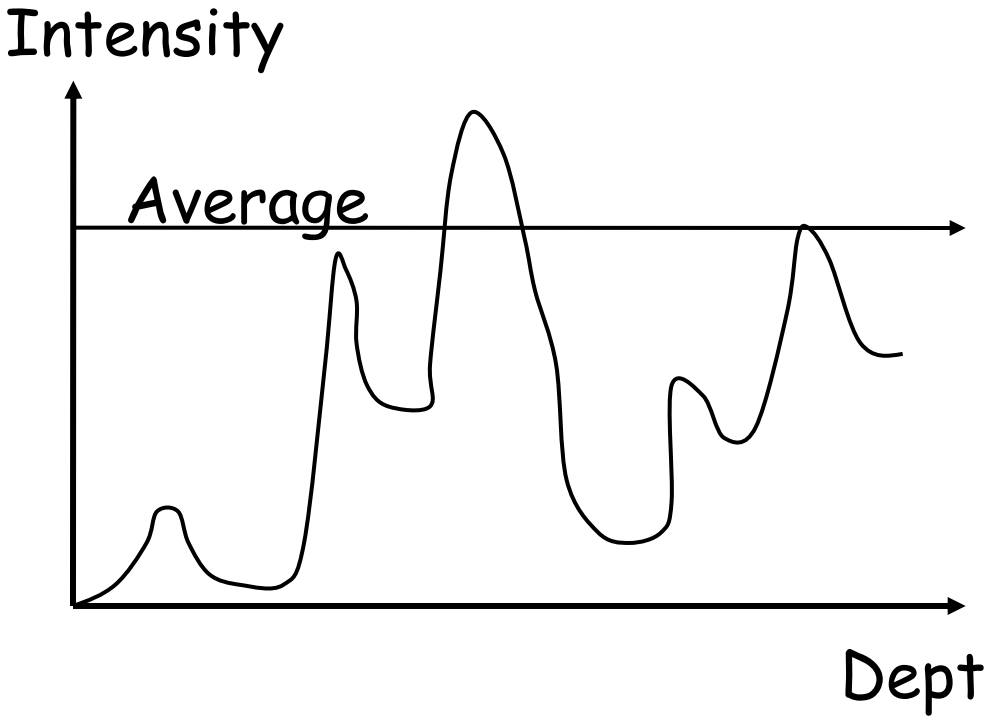
Ray Traversal Schemes

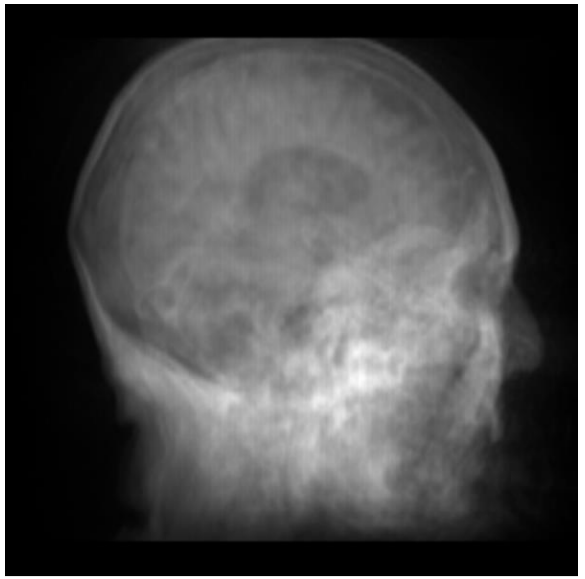


Ray Traversal - First



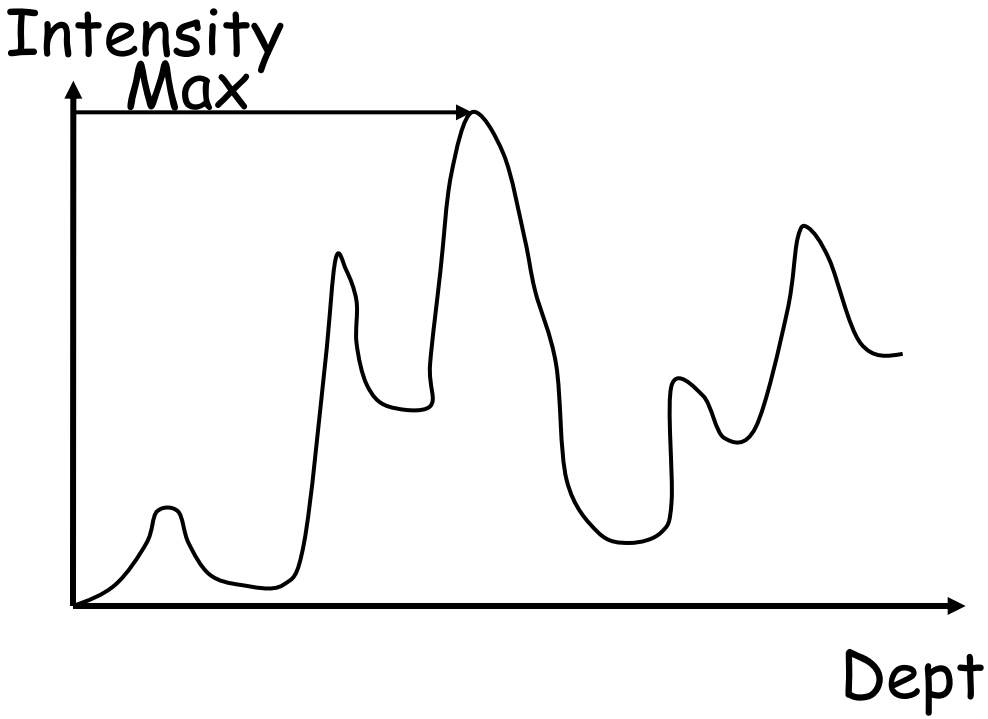
Ray Traversal - Average

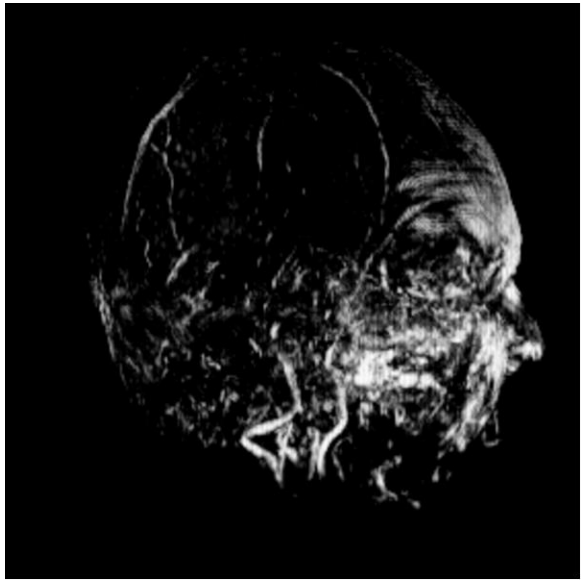




- **Average**: produces basically an X-ray picture

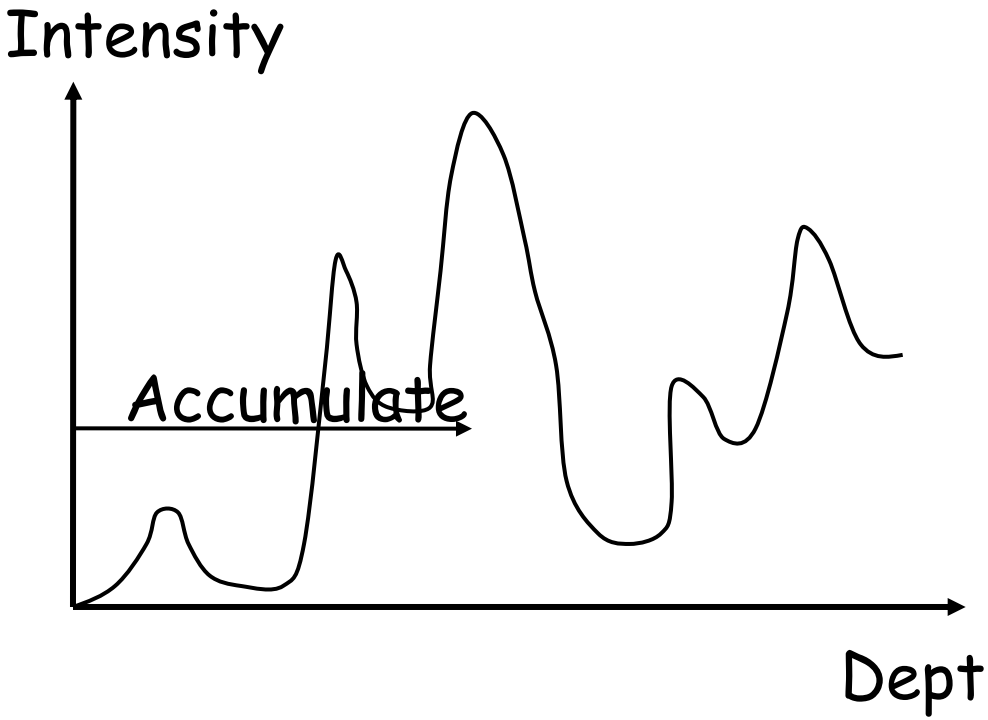
Ray Traversal - MIP

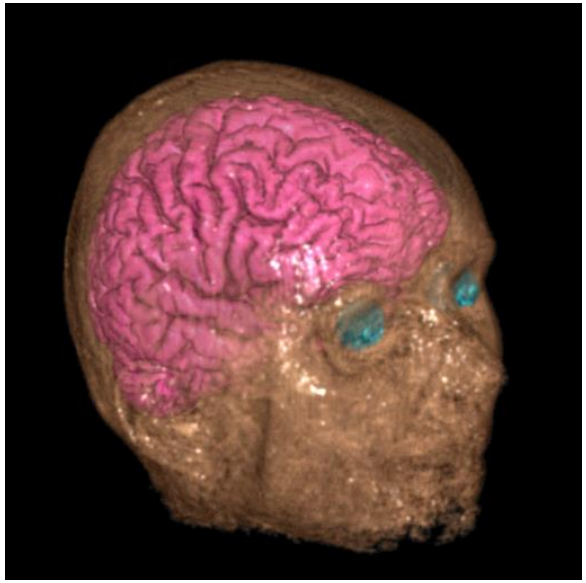




- **Max**: Maximum Intensity Projection used for Magnetic Resonance Angiogram

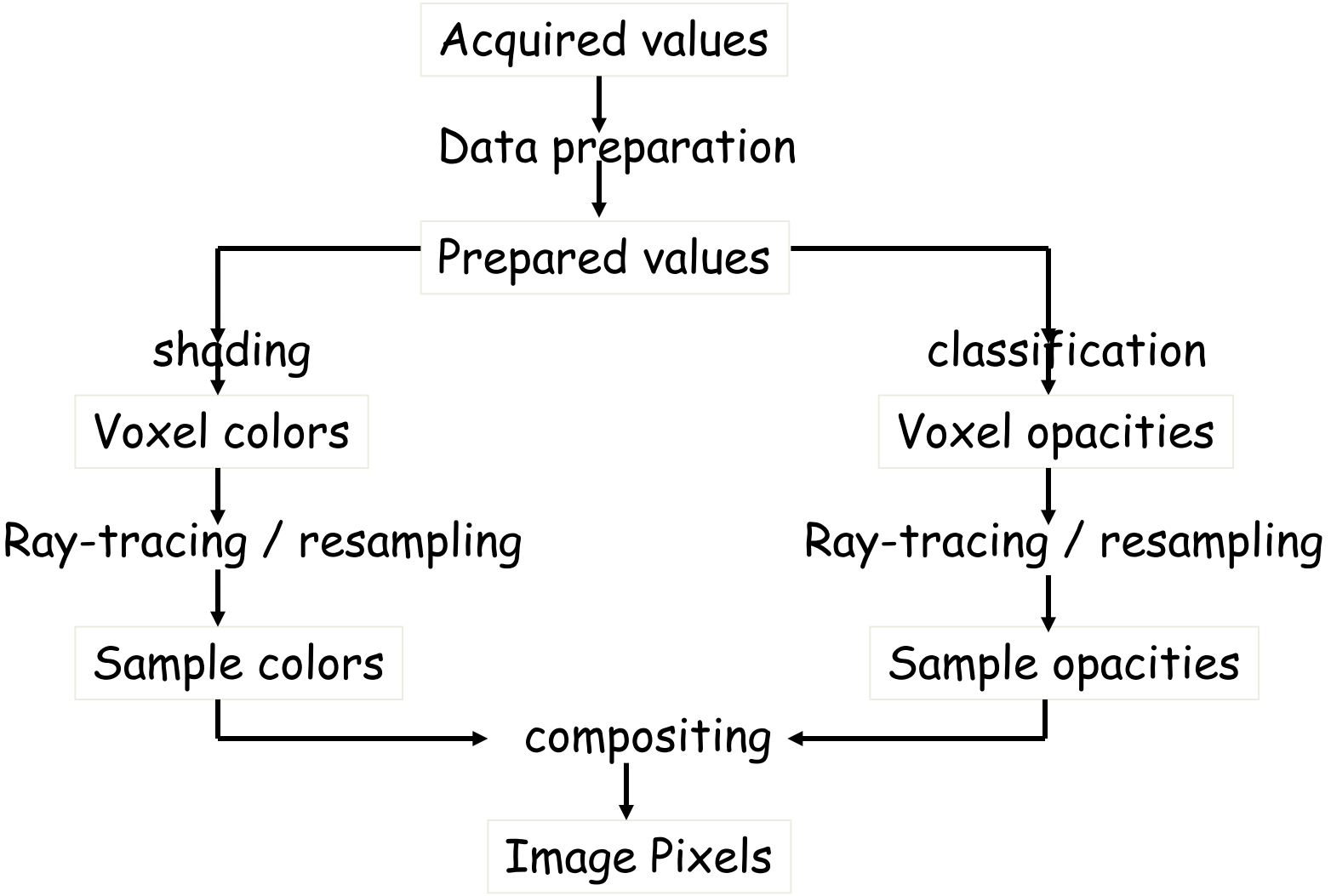
Ray Traversal - Accumulate



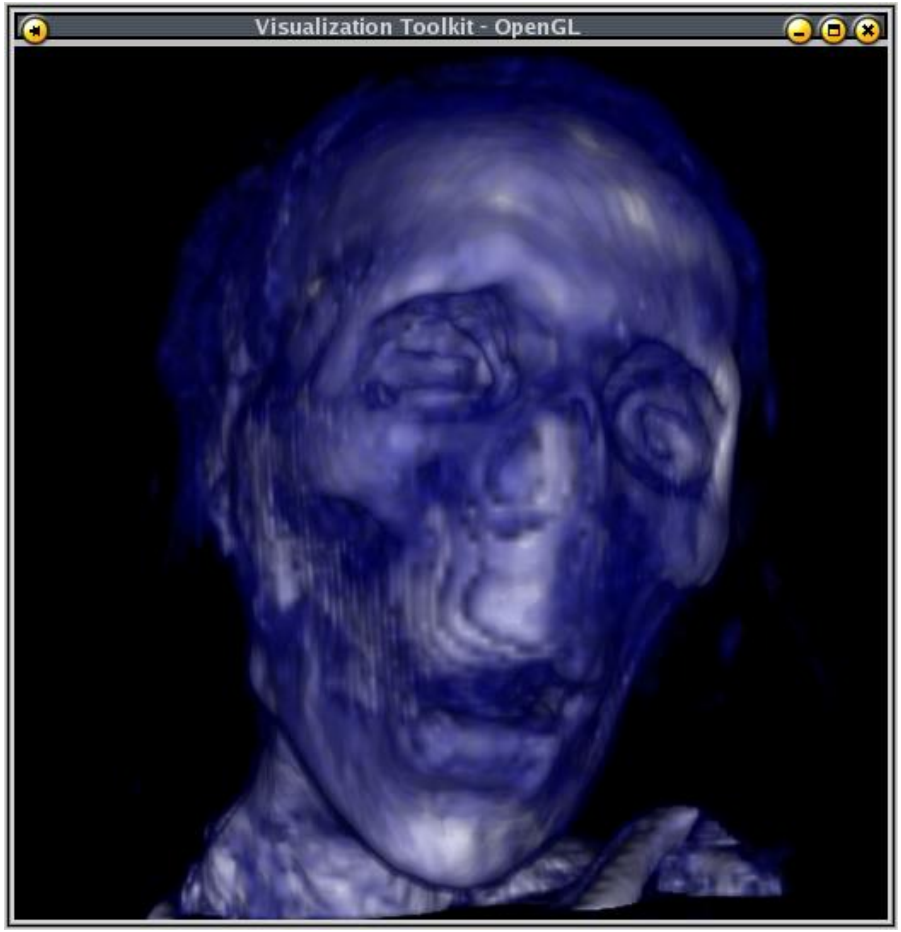


- **Accumulate opacity while compositing colors**: make transparent layers visible!
  Levoy '88

Volume Rendering Pipeline



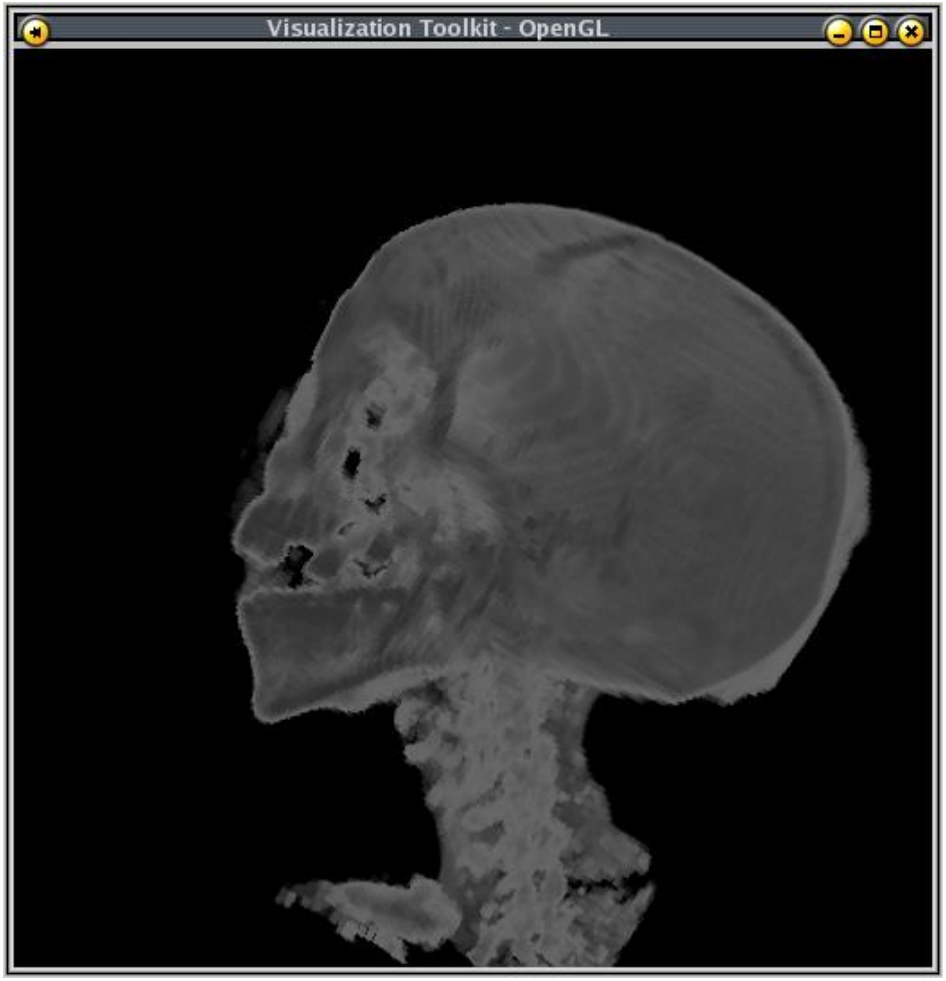
VTK Ray Casting Example



- The given data represents a mummy preserved for a long time. So the skin is dried and not very crisp when compared to the data set given in Project 3. The dried skins iso value was found to be around 70 to 90. The skull iso value was around 100 to 120. In order to visualize this data set a opacity transfer function and a color transfer function are constructed. The opacity for values ranging from 0 - 50 is chosen to be 0.0 and 55 - 80 is chosen to be 0.1 (semi translucent) and finally the bone values ranging from 90 - 120 is given a complete opaque value of 1.0. The colors are chosen in such a way that the skin range has a light blue and the bone has a complete white and all other values have a color value of 0.0.

- **www.cs.utah.edu/.../sci_vis/prj4/REPORT.html**

VTK Ray Casting Example



- create the maximum intensity projection of the image. This looks more like an x-ray of the mummy. I used the inbuilt method in VTK called *vtkVolumeRayCastMIPFunction* . The opacity transfer function plays a major role in this technique and the color transfer function is used to adjust the contrast and get good looking images.
- **www.cs.utah.edu/.../sci_vis/prj4/REPORT.html**

Surface Modeling

Visualization using BrainVISA

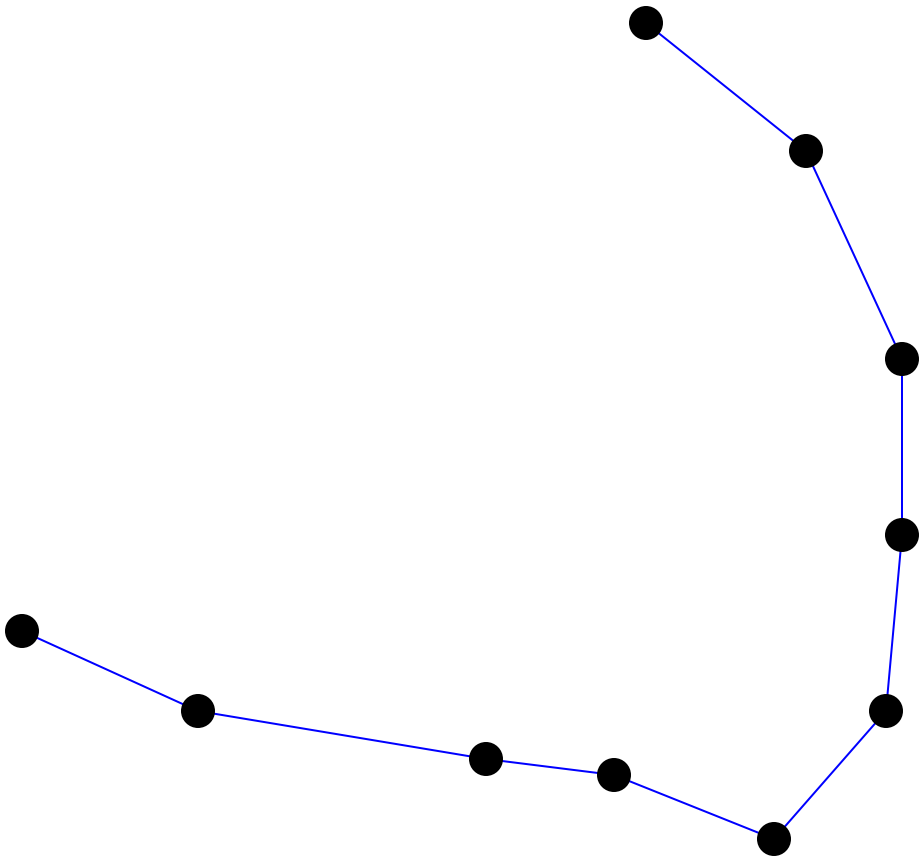
Why Use Surface Modeling

- Visualization of Structure
- Analysis of Structure
- Dynamic control of view

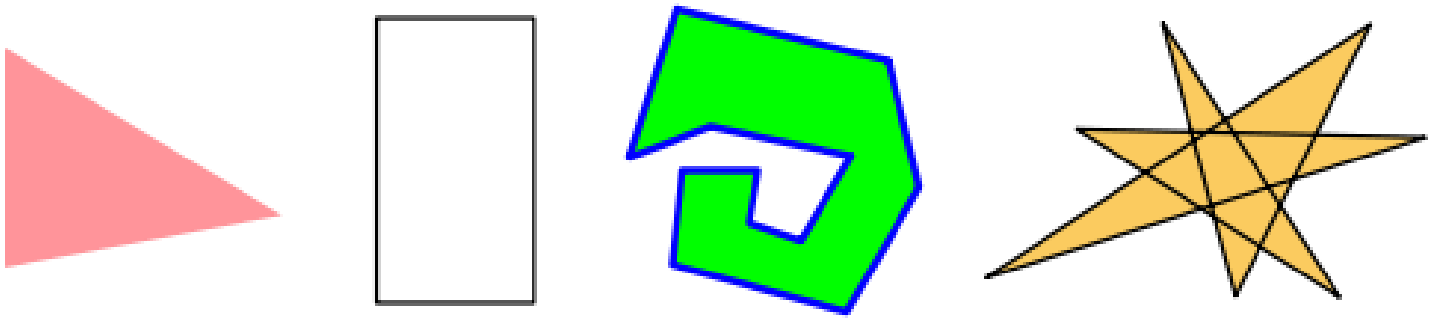
What Makes a Surface

- A surface is usually defined as a mesh
- The mesh is composed of vertices, edges, normals and polygons
- The vertices define the surface boundary
- Vertices are connected by edges
- Edges are combined to make polygons
- Normals determine side of surface as well as viewing properties

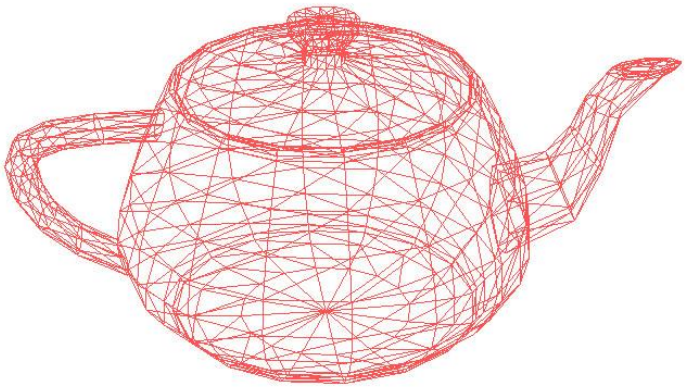
Vertices and Edges

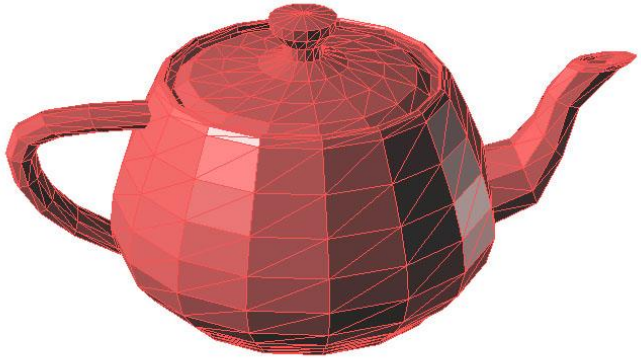


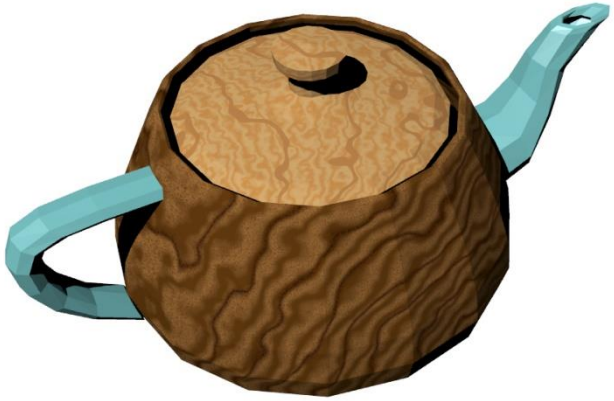
Polygons



Polygon Mesh Surface



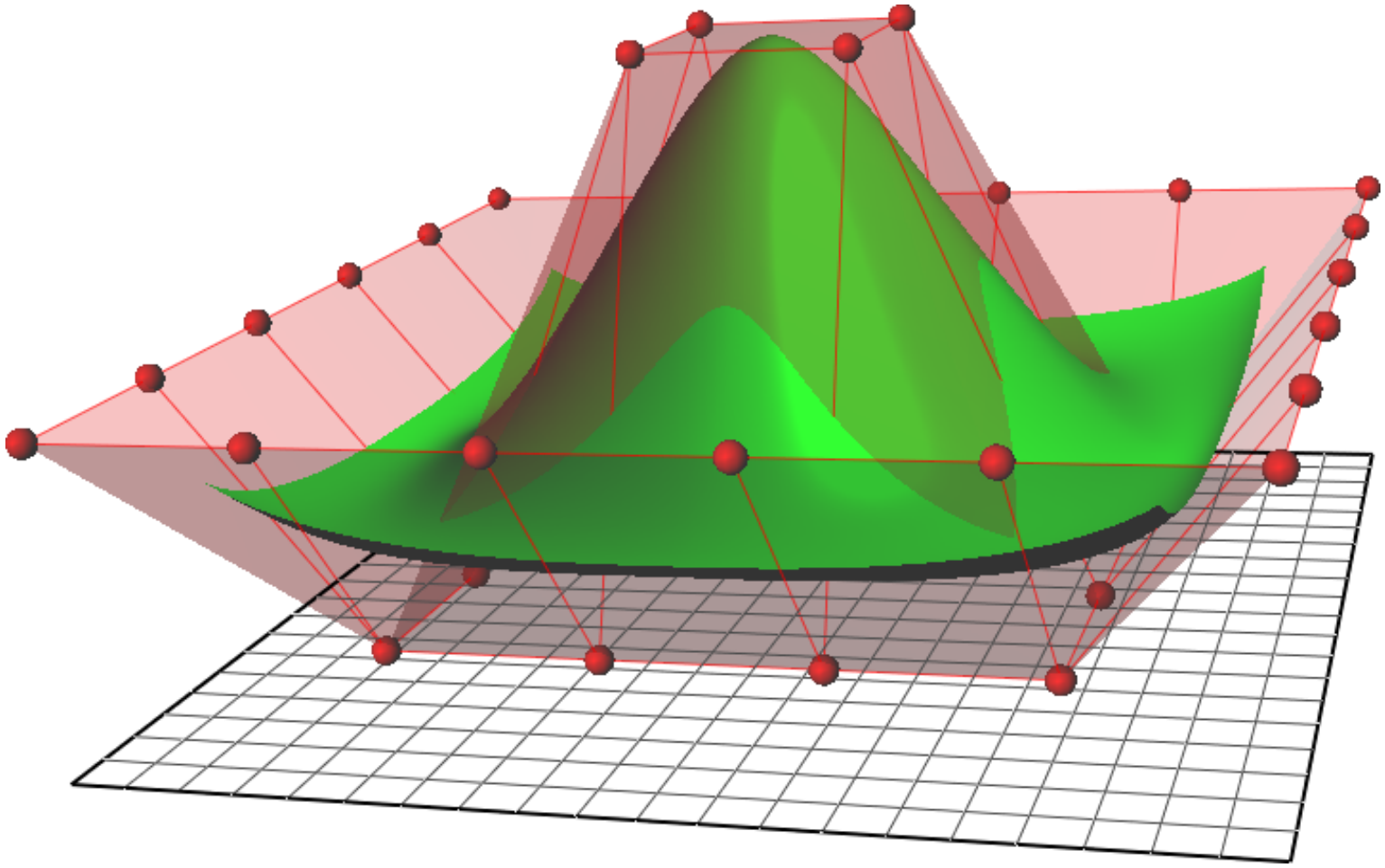




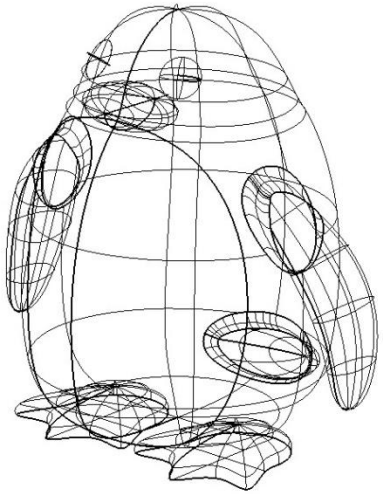
Parametric Surfaces

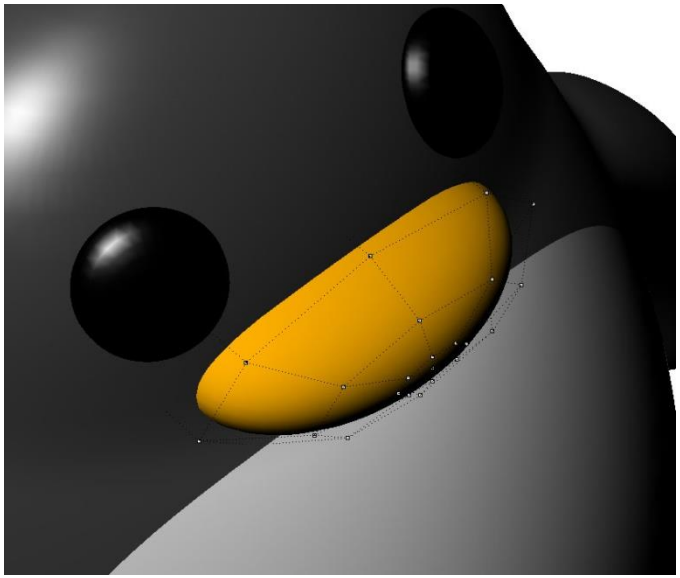
- Surface in Euclidean space defined by a parametric equation with two parameters
- A set of weighted control points determine the location of individual surface points
- The come in several flavors including Bezier, B-Spline, NURBS

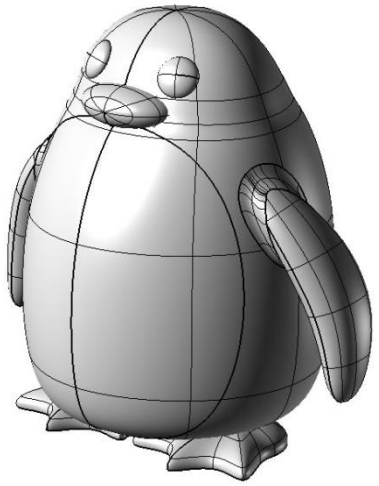
NURBS surface with control points

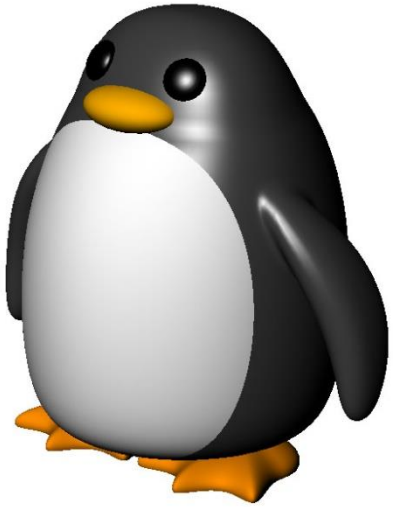


NURBS Surfaces

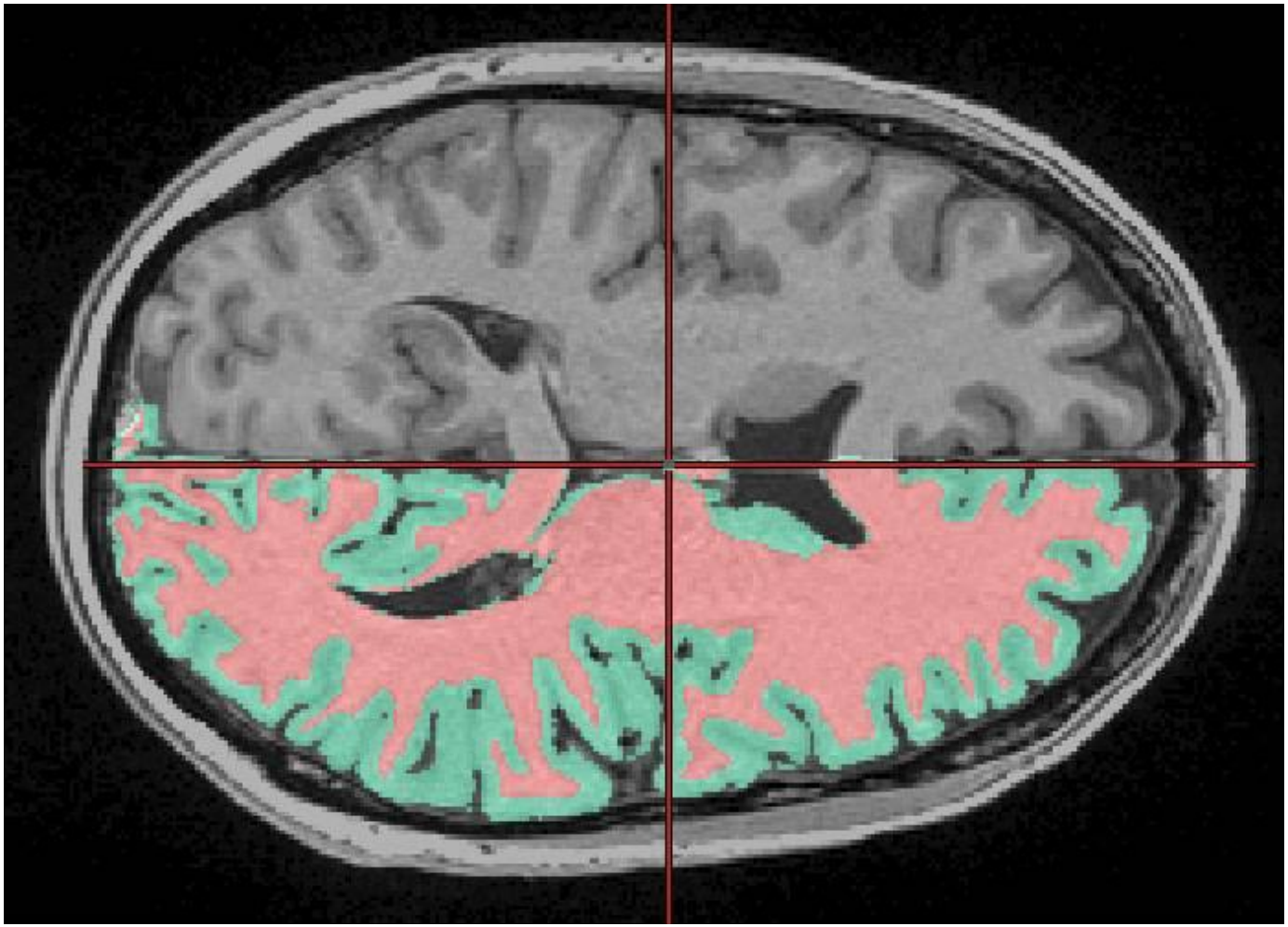








Isosurface extraction
or
Where to put the surface



Isosurfaces

- A 3-D surface corresponding to points with a single scalar value (or narrow range of values).
- The scalar value corresponds to an interface between voxels of different properties.

The Surface is Only as Good as the Tissue Classification

- Bias Correction
- Partial Volume Effect
- Classification of voxels

Isosurface Extraction Techniques

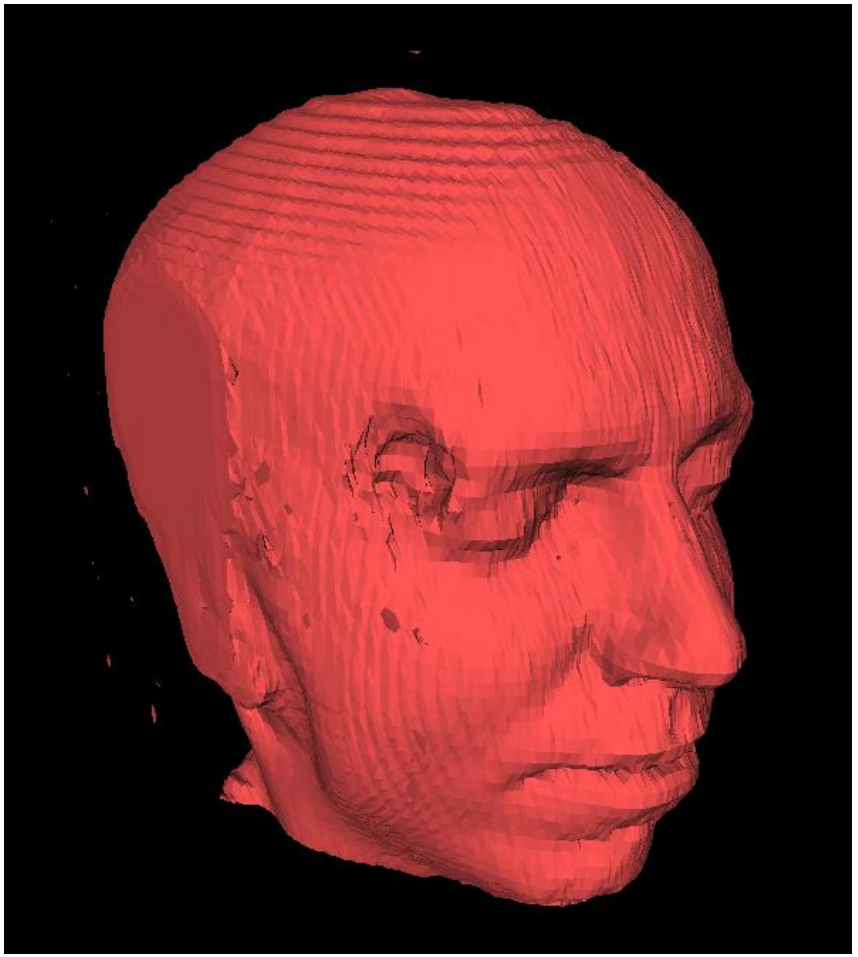
- Geometric Decomposition Techniques
  - Geometric techniques retain the original representation of the volume and partition along divisions in the voxel volume
- Span Space Decomposition Techniques
  - Span space decomposition techniques create and manipulate abstract representations of the voxels

Methods of Isosurface Extraction

- Marching Cubes (Geometric)
- BONO - branch-on-need octree (Geometric)
- ISSUE - Isosurfacing in Span Space with Utmost Efficiency (Span Space)
- Interval Tree – (Span Space)

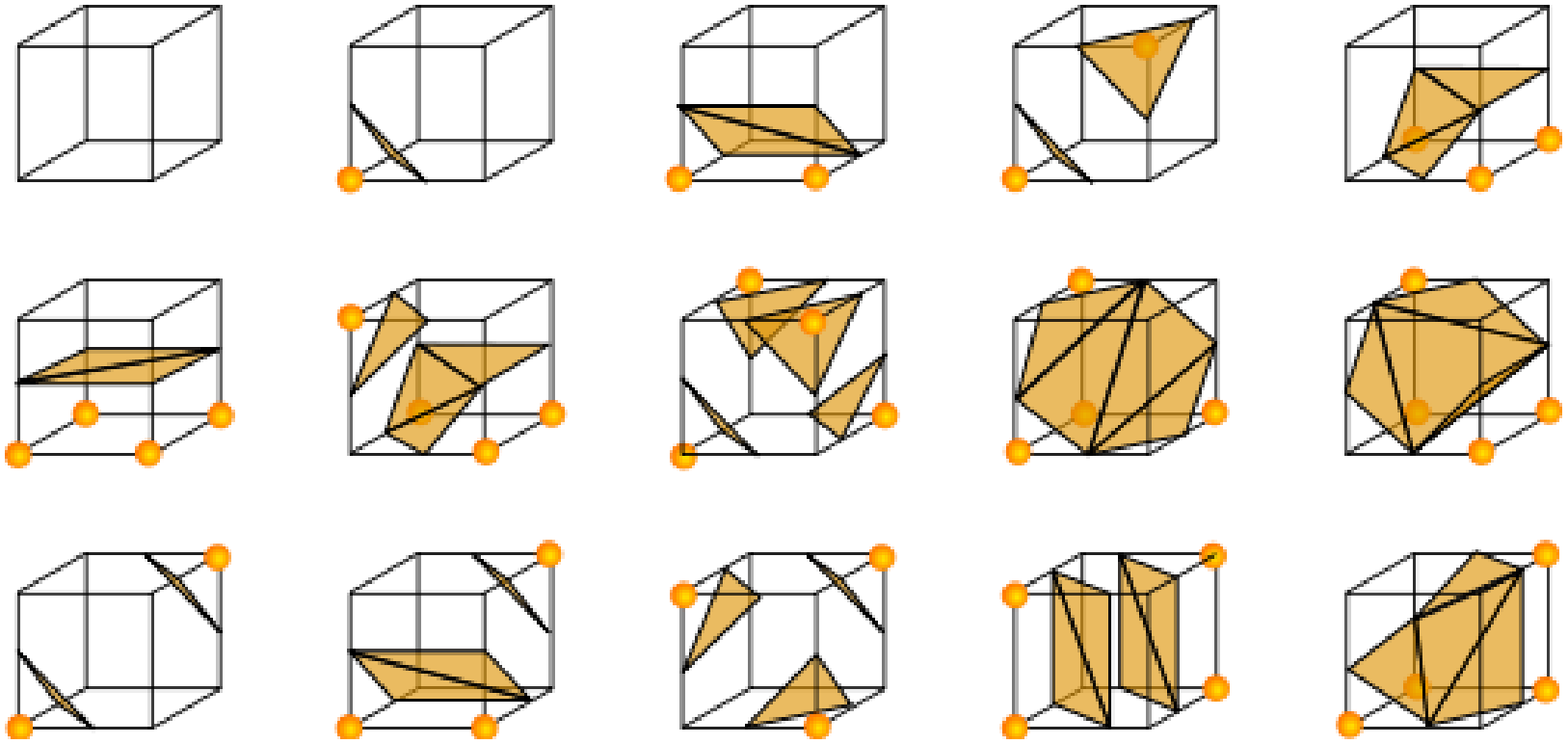
Marching Cubes

- William E. Lorensen, Harvey E. Cline: *Marching Cubes: A high resolution 3D surface construction algorithm.* In: *Computer Graphics*, Vol. 21, Nr. 4, July 1987
- Computes polygons where the isosurface passes through eight nearest neighbors
- Gradient of scalar value at each grid point used for surface normal
- Other algorithms are always compared to Marching Cubes



Marching Cubes

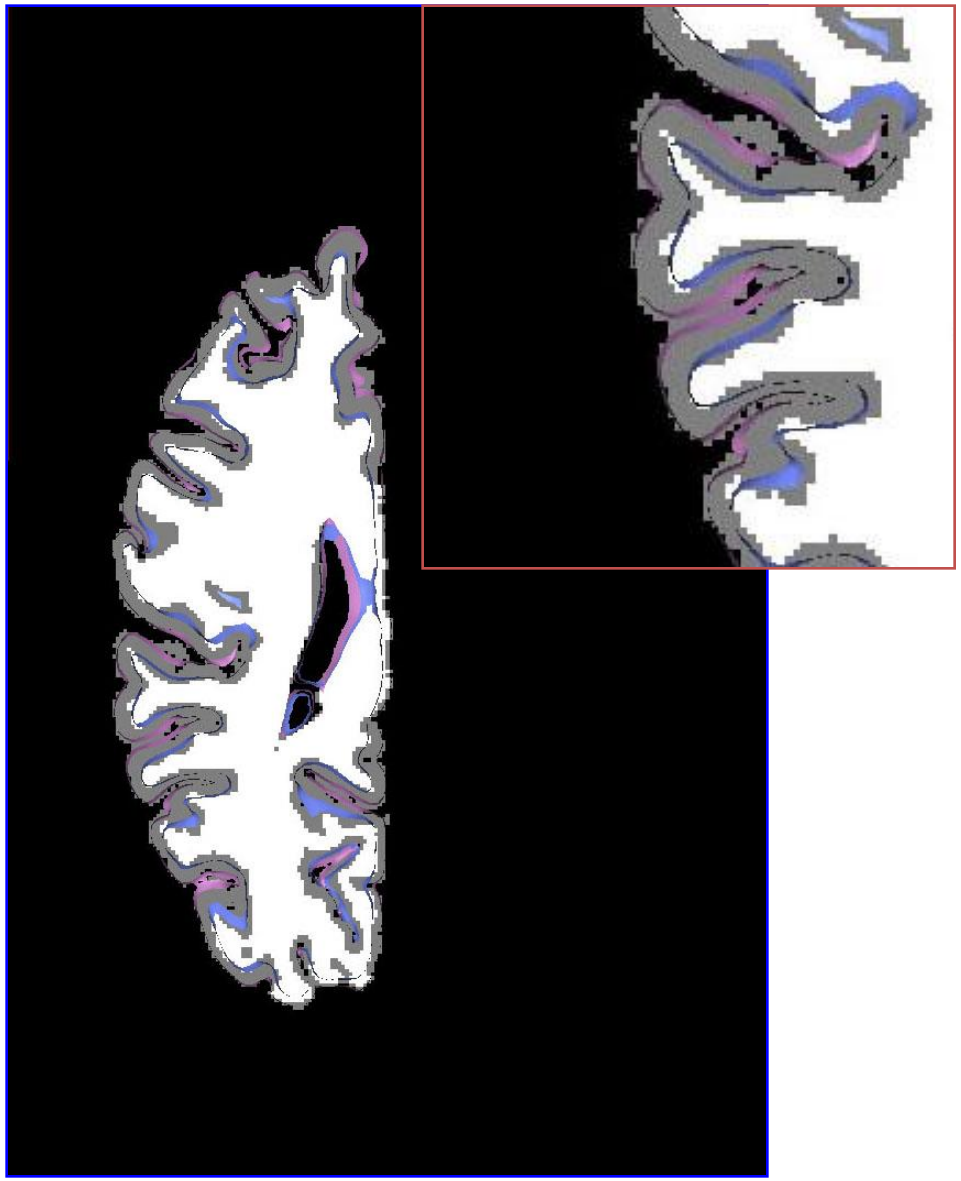
- 15 Unique cube configurations that can be rotated and reflected to 256 configurations

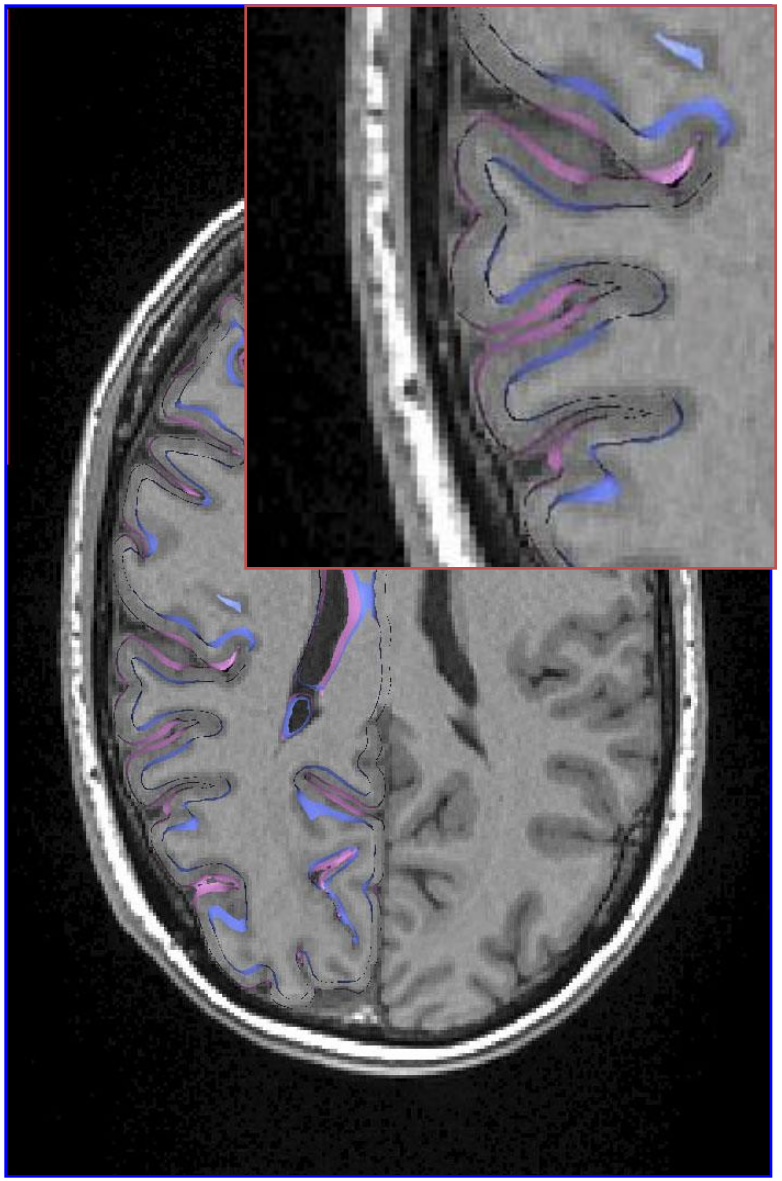


Marching Cubes Demo

Graphics cards aren’t just for games anymore

Mesh Segmentation





Contouring

5.3 Contouring

Contour line (isoline): the same scalar value, or isovalue

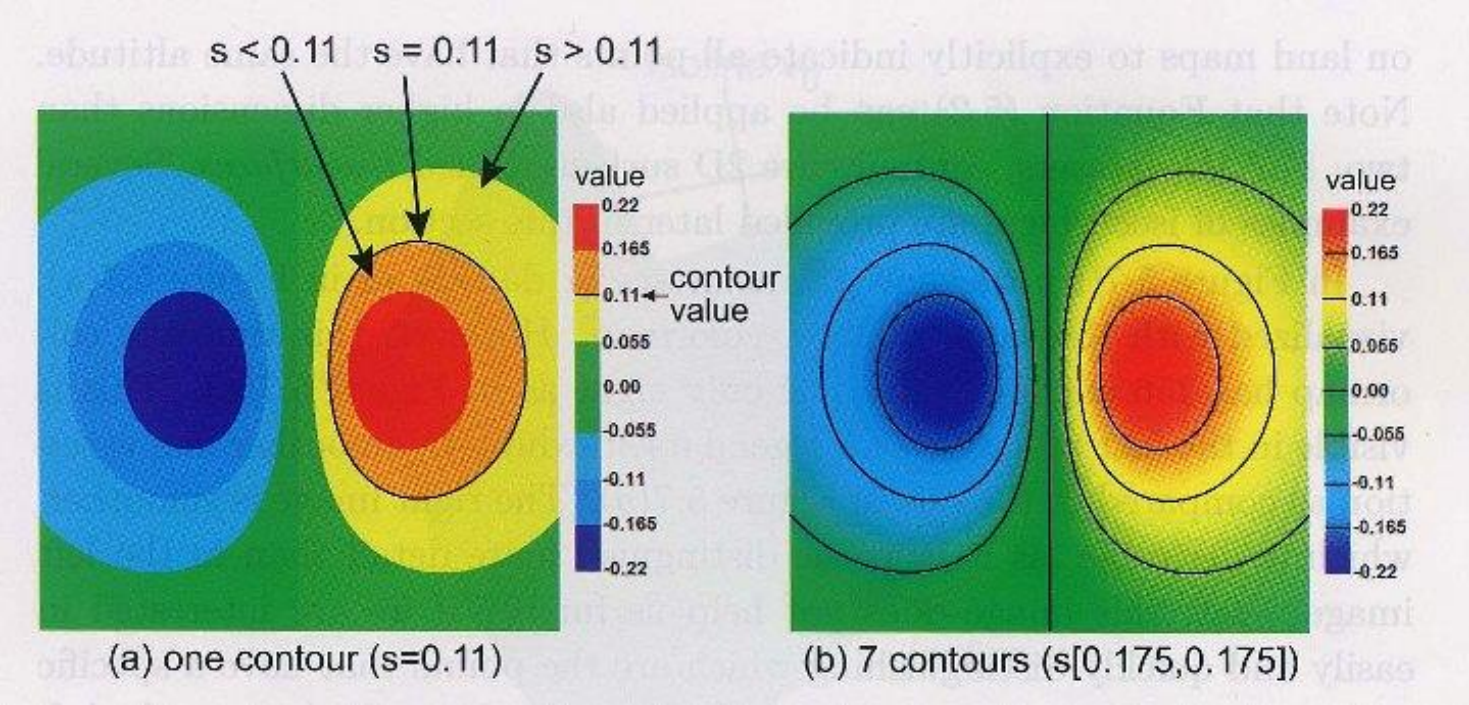


Fig 5.7. Relationship between color banding and contouring

5.3 Contouring

A contour *C* is defined as all points *p,* in a dataset *D,* that have the same scalar value *x*: s(*p*) = *x*.

For 2D dataset: contour line (isoline);

For 3D dataset: contour surface (iso-surface)

5.3 Contouring

- Contour properties:
  - Closed curve or open curves
  - Never stop inside the dataset itself
  - Never intersects itself
  - No intersect of an isoline with another scalar value
  - Contours are perpendicular to the gradient of the contoured function (**Fig 5.9**)

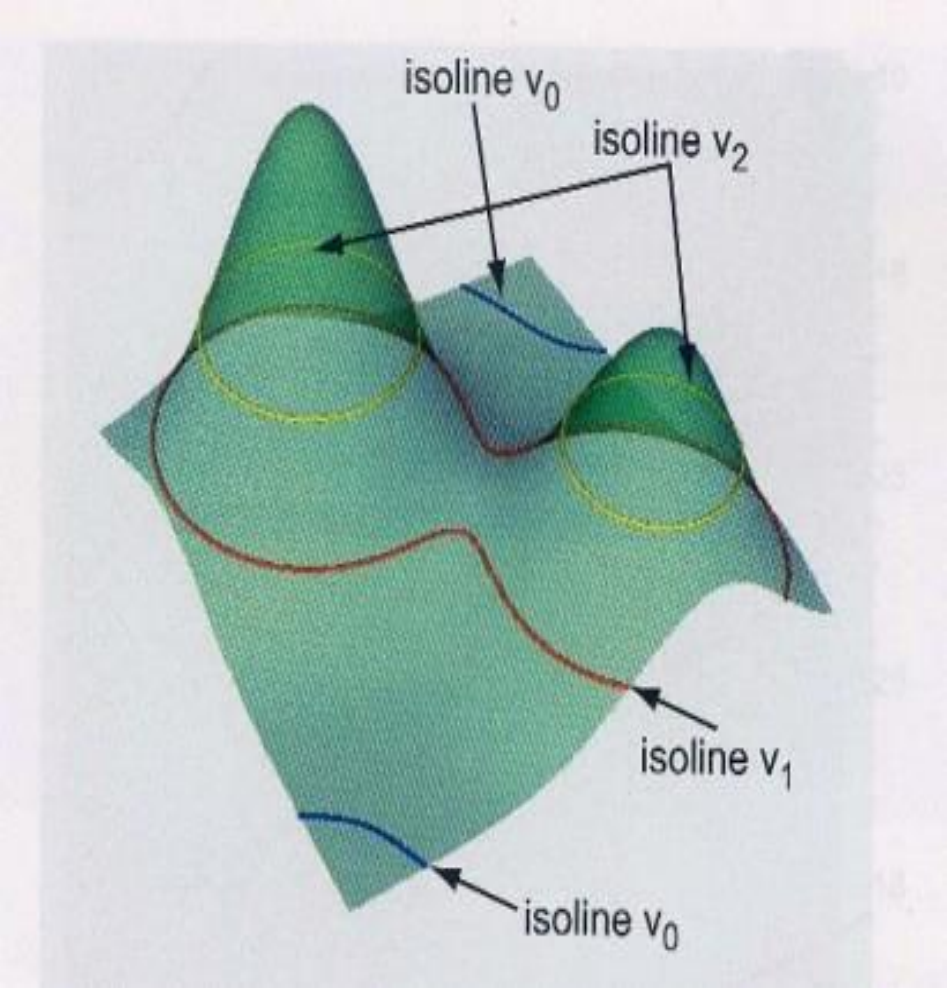


Fig 5.8. Isoline properties

5.3 Contouring

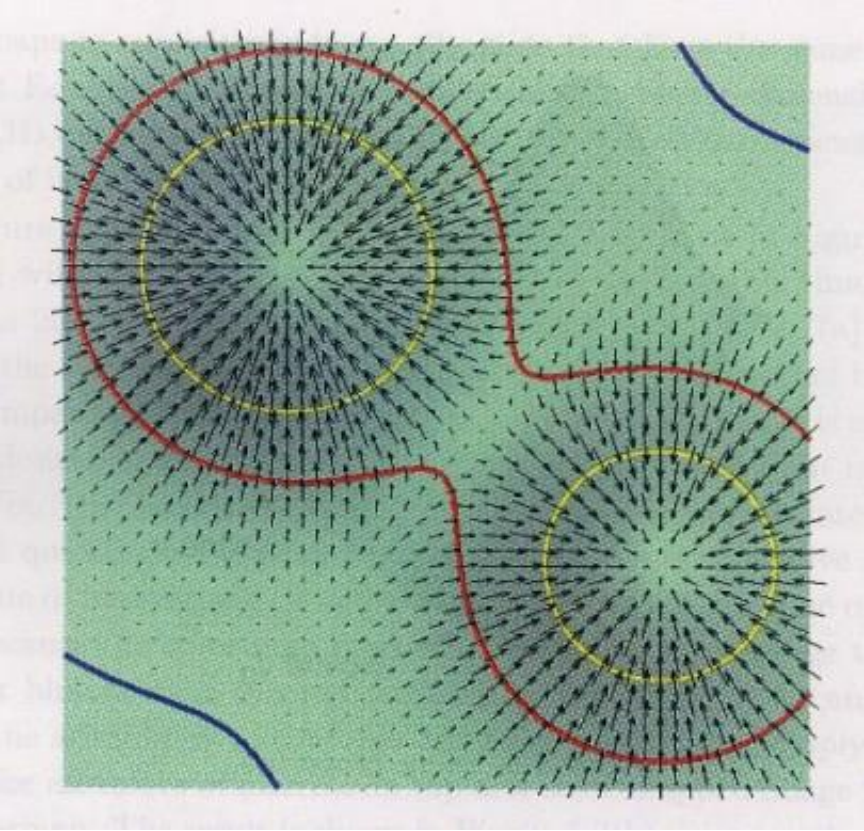


Fig 5.9. The gradient of a scalar field is perpendicular to the field's contours

5.3 Contouring

- Given a discrete, sampled dataset, compute contours

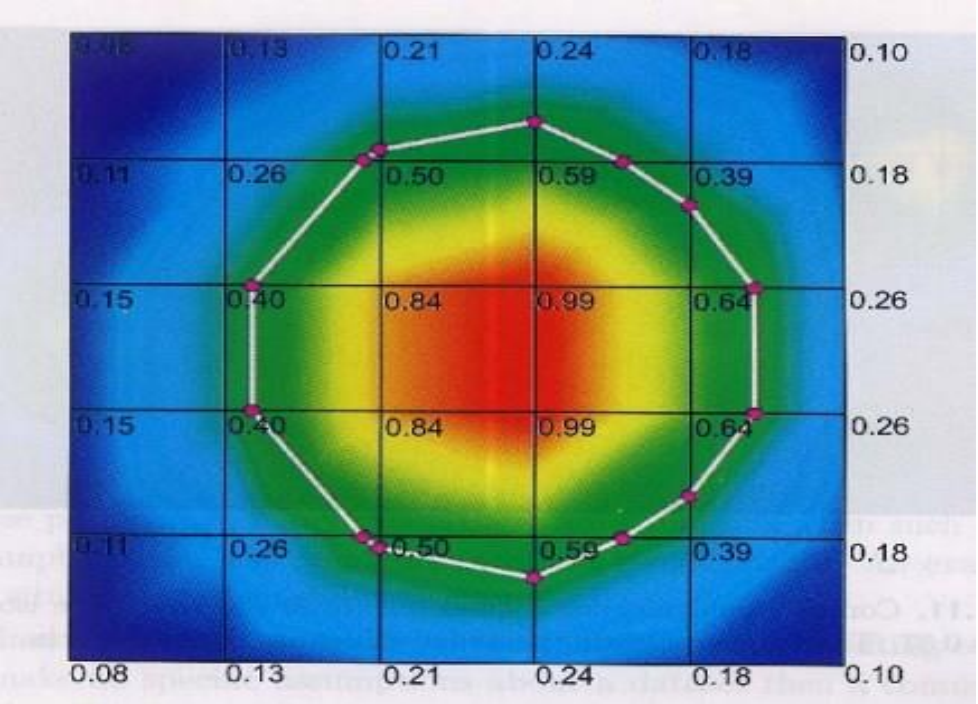


Fig 5.10. Constructing the isoline for the scalar value v = 0.48. The figure indicates scalar values at the grid vertices.

5.3 Contouring

- Contouring need
  - At least piecewise linear, C^1 dataset
  - The complexity of computing contours
- The most popular method
  - 2D: Marching Squares (§ 5.3.1)
  - 3D: Marching Cubes (§ 5.3.2)

5.3 Contouring

5.3.1 Marching Squares

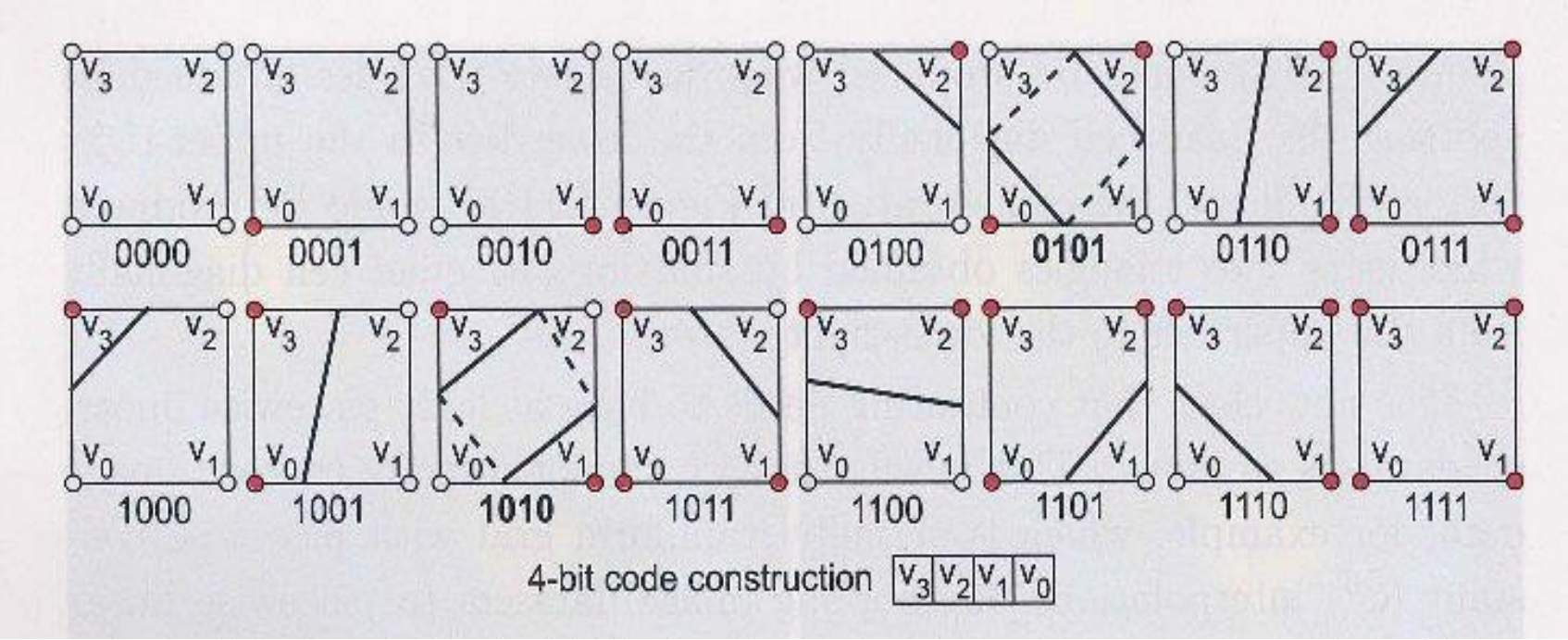


Fig5.12. Topological states of a quad cell (marching squares algorithm). Red indicates "inside" vertices. Bold indices mark ambiguous cases.

5.3.1 Marching Squares

```
for each cell c_i of the dataset
{
  int index = 0;
  for (each vertex v_j of c_i)
    store the inside/outside state of v_j in bit j of index;
  select the optimized code from the case table using index;
  for (all cell edges e_j of the selected case)
    intersect e_j with isovalue v using Equation (5.3);
  construct line segments from these intersections;
}
```

Listing 5.2. Marching squares pseudocode

5.3.2 Marching Cubes

- Similar to Marching Squares but 3D versus 2D
- 2^8= 256 different topological cases; reduced to only 15 by symmetry considerations
  - 16 topological states (**Fig 5.13**)

5.3.2 Marching Cubes

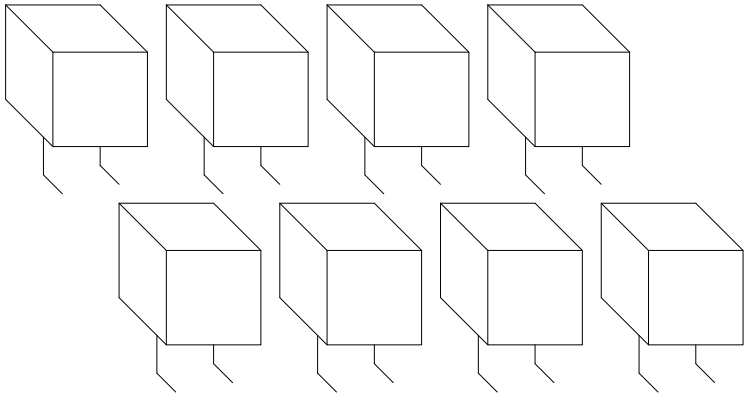
- **Marching Cubes: A High Resolution 3D Surface Construction Algorithm**

  William E. Lorensen & Harvey E. Cline

  International Conference on Computer Graphics and Interactive Techniques (ACM/SIGGRAPH 1987)

Marching Cubes: A High Resolution 3D Surface Construction Algorithm

What <u>are</u> Marching Cubes?



Marching Cubes is an algorithm which “creates triangle models of constant density surfaces from 3D medical data.”

Surface Construction

- Construction/Reconstruction of scanned surfaces or objects.
- Problem of interpreting/interpolating 2D data into 3D visuals.
- *Marching Cubes* provides a new method of creating 3D surfaces.

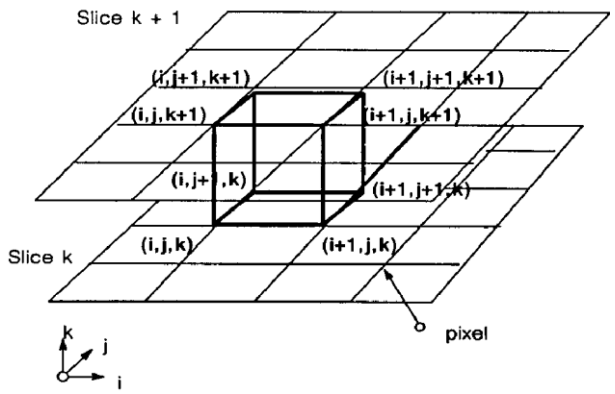
Marching Cubes Explained

- High resolution surface construction algorithm.
- Extracts surfaces from adjacent pairs of data slices using cubes.
- Cubes “march” through the pair of slices until the entire surface of both slices has been examined.

Marching Cubes Overview

- Load slices.
- Create a cube from pixels on adjacent slices.
- Find vertices on the surfaces.
- Determine the intersection edges.
- Interpolate the edge intersections.
- Calculate vertex normals.
- Output triangles and normals.

How Are Cubes Constructed



- Uses identical squares of four pixels connected between adjacent slices.
- Each cube vertex is examined to see if it lies on or off of the surface.

5.3.2 Marching Cubes

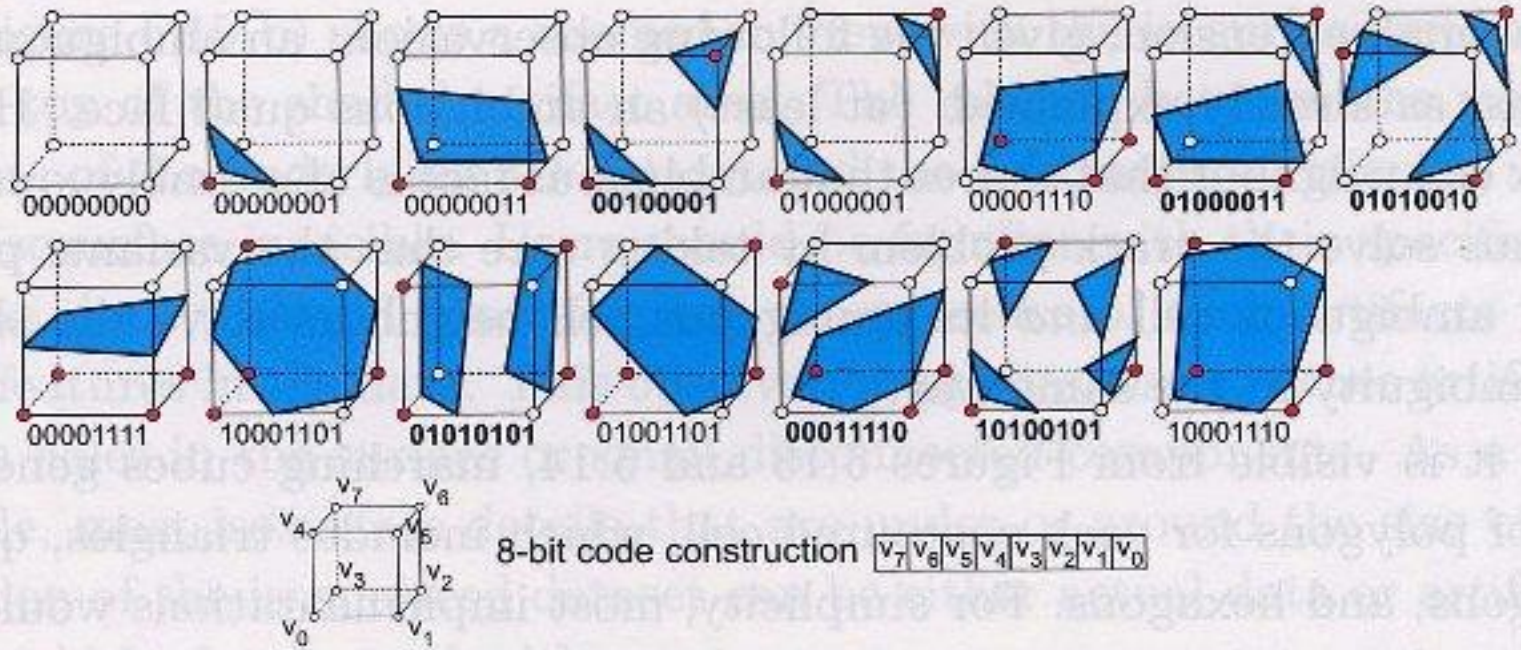


Fig 5.13. Topological states of a hex cell (marching cubes algorithm). Red indicates "inside" vertices. Bold indices mark ambiguous cases.

5.3.2 Marching Cubes -- Ambiguity

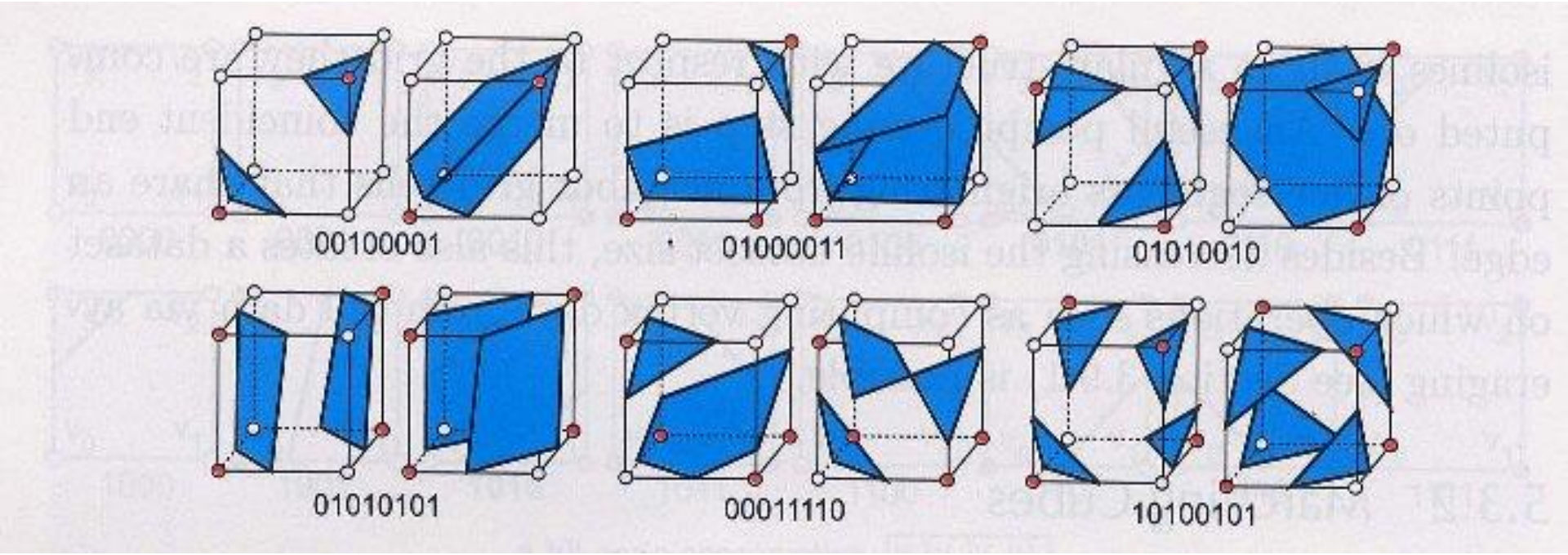
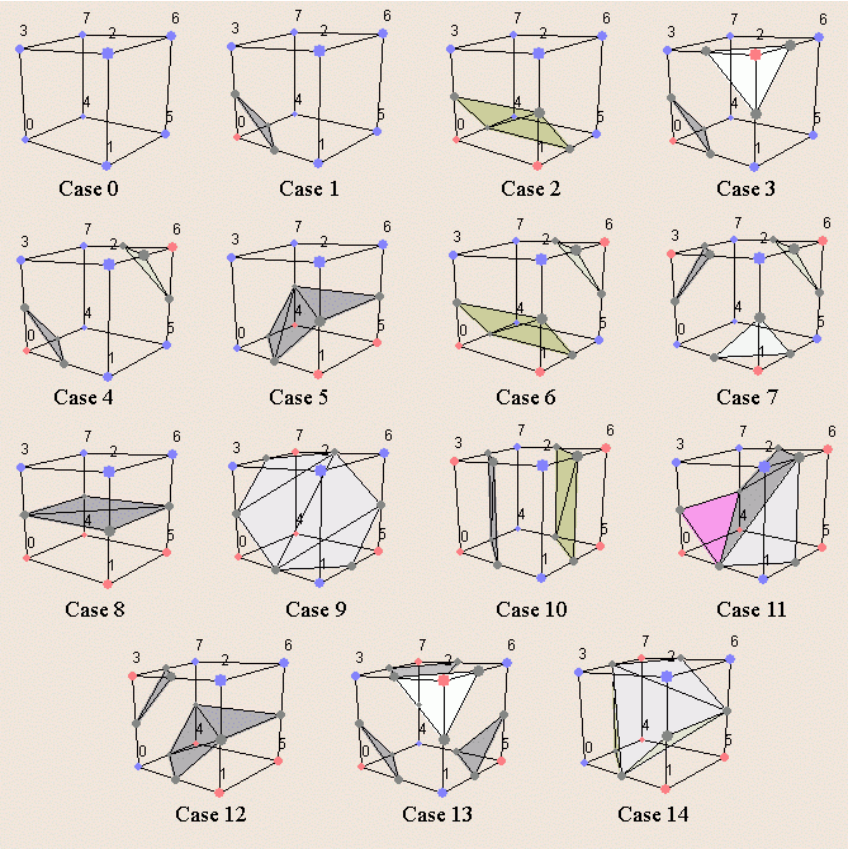


Fig 5.14. Ambiguous cases for marching cubes. Each case has two contouring variants.

How Are The Cubes Used

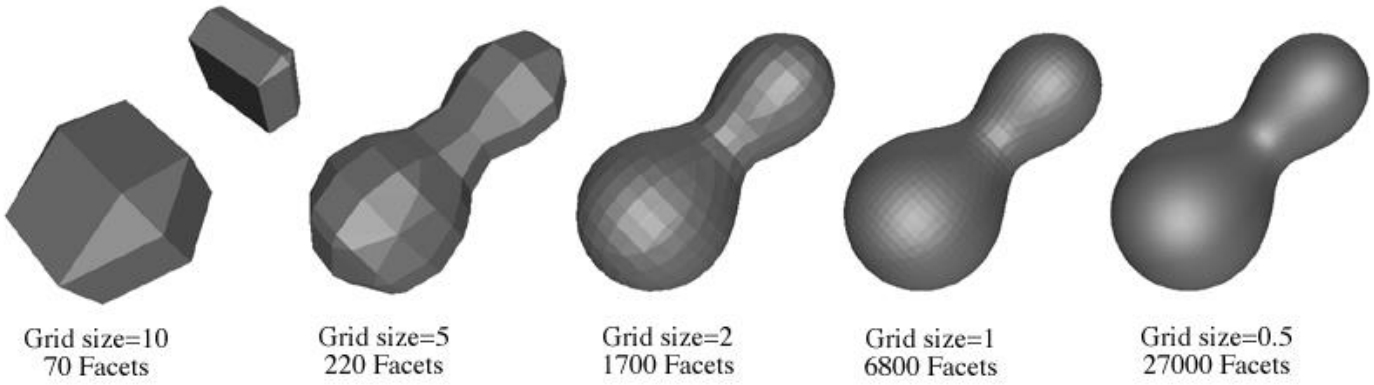
- Pixels on the slice surfaces determine 3D surfaces.
- 256 surface permutations, but only 14 unique patterns.
- A normal is calculated for each triangle vertex for rendering.



Triangle Creation

- Determine triangles contained by a cube.
- Determine which cube edges are intersected.
- Interpolate intersection point using pixel density.
- Calculate unit normals for each triangle vertex using the gradient vector.

Grid Resolution



- Variations can increase/decrease surface density.

5.3.2 Marching Cubes

Isosurfaces and isolines are strongly related

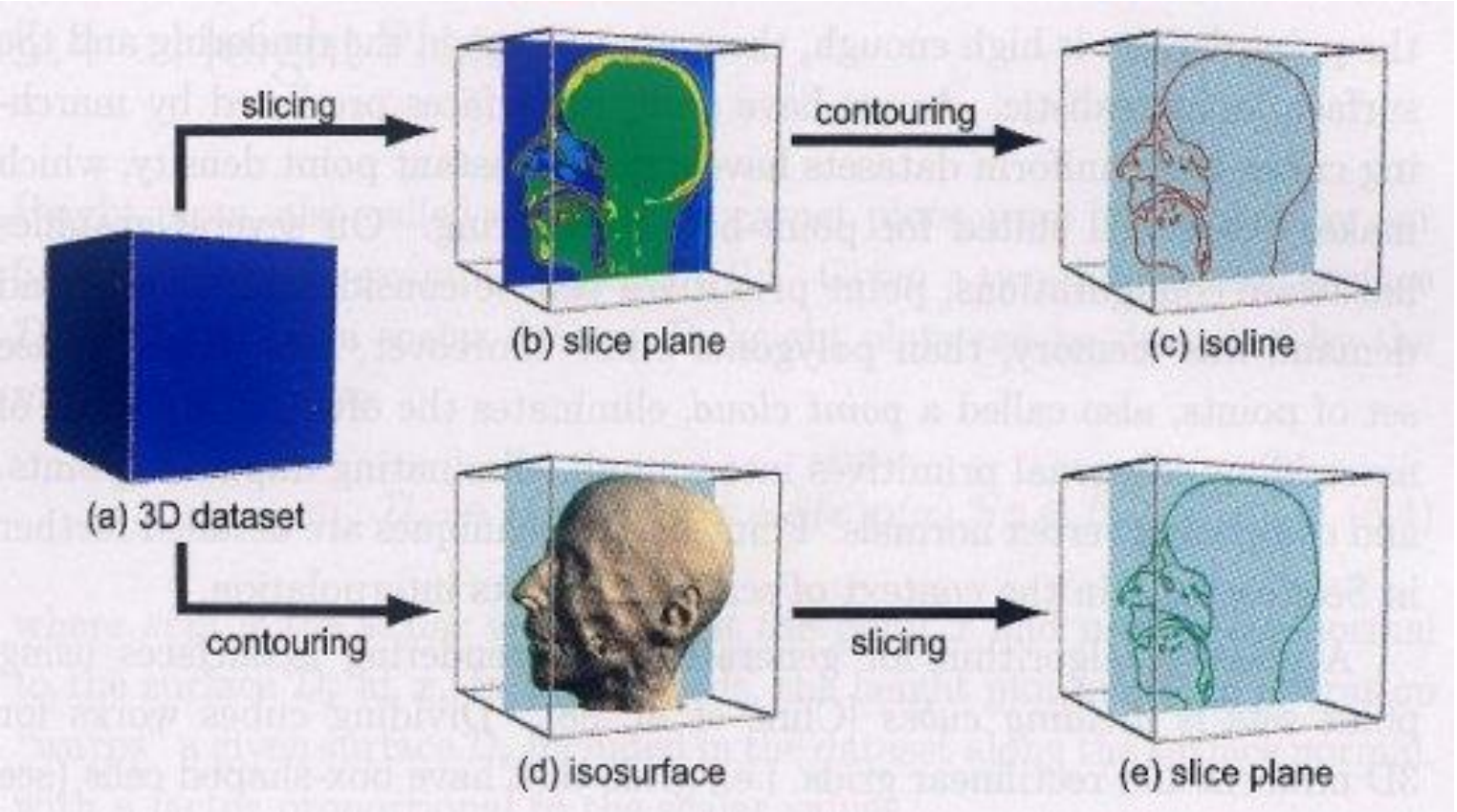


Fig 5.17. Isosurfaces, isolines, and slicing

5.3.2 Marching Cubes

- Marching Cubes provides a simple algorithm to translate a series of 2D scans into 3D objects
- Marching Squares and Marching Cubes have many variations to address:
  - Generalcity in terms of input dataset type
  - Speed of execution
  - Quality of obtained contours
- Isosurface can also be generated and rendered using point-based techniques
  - 3D surface can be rendered using large numbers of (shaded) point primitives
  - Point primitive can be considerably faster and demand less memory than polygonal ones on some graphics hardware
  - Point cloud