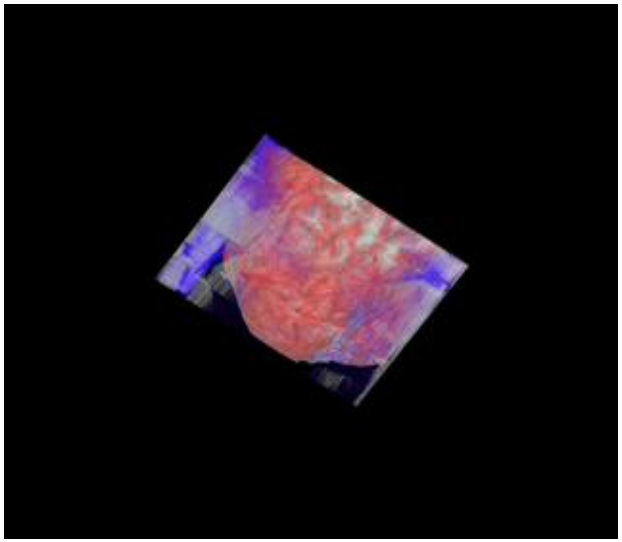
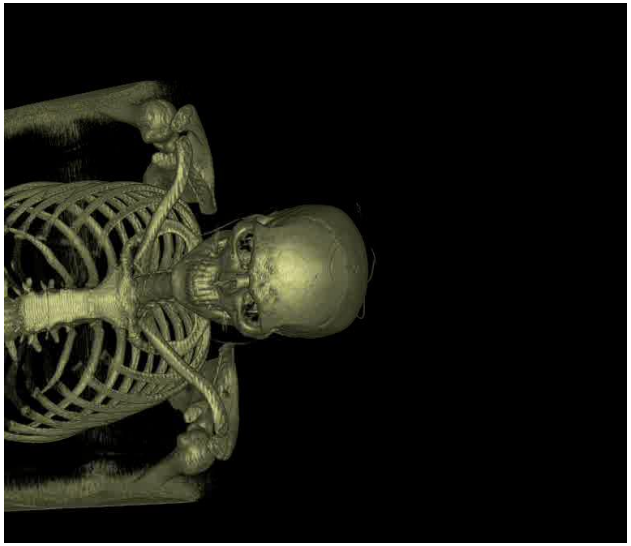
Visualization

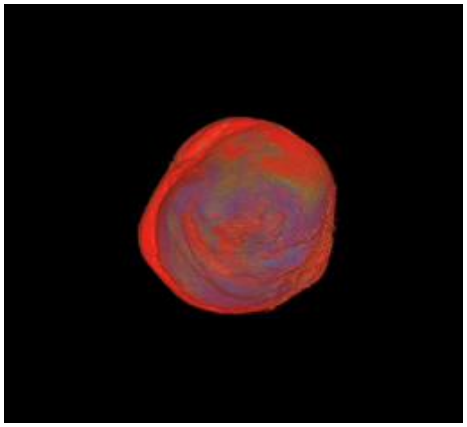
Prof. Vaibhav Vasani

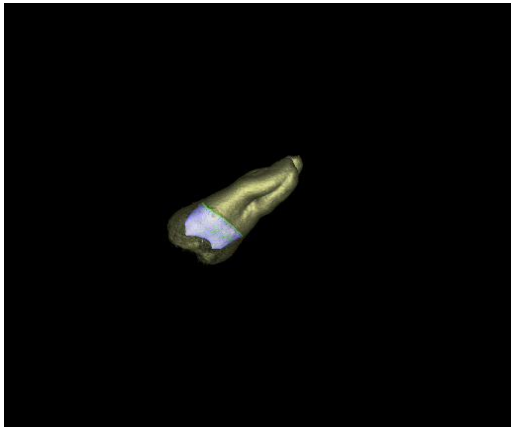
Volume Rendering

Example Volume Renderings



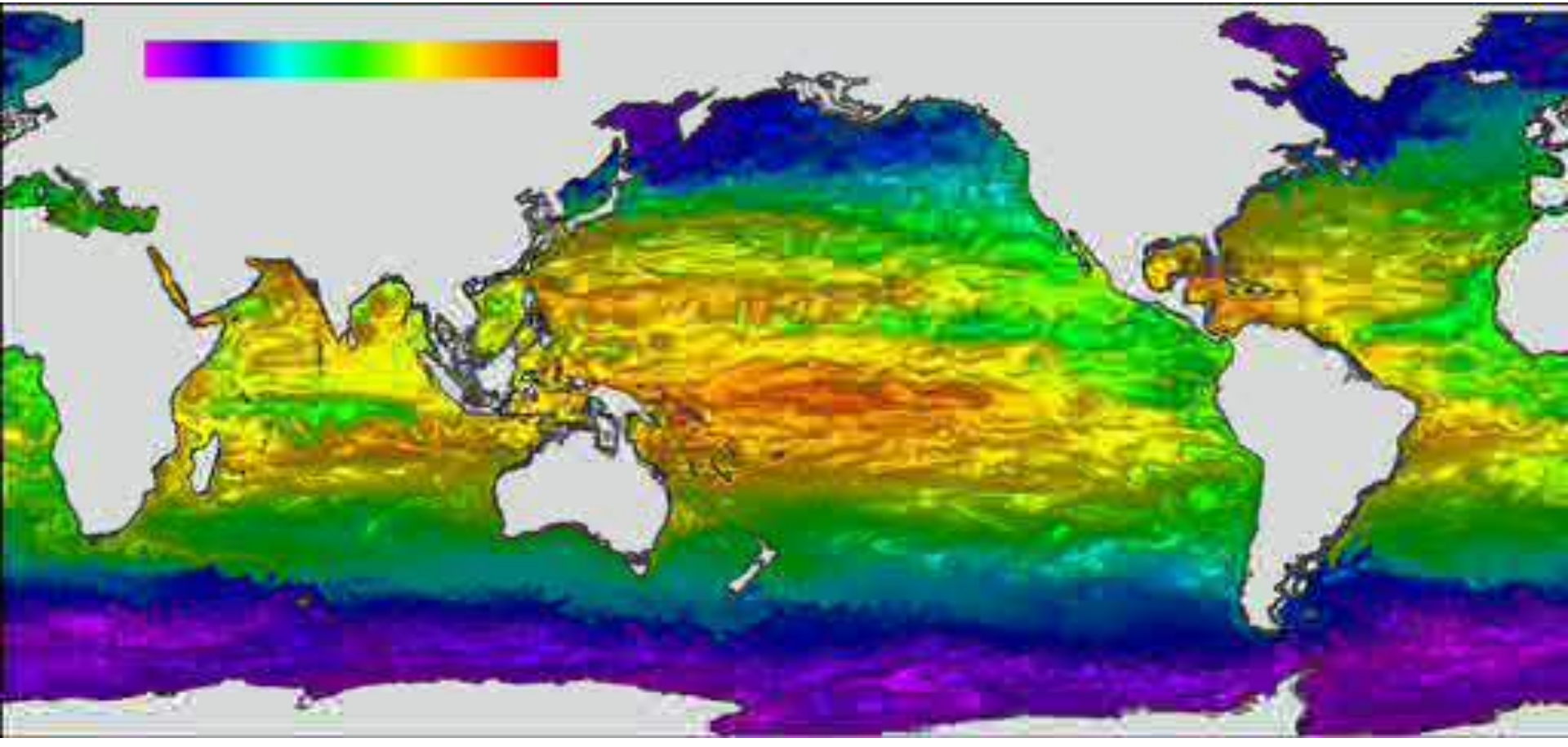






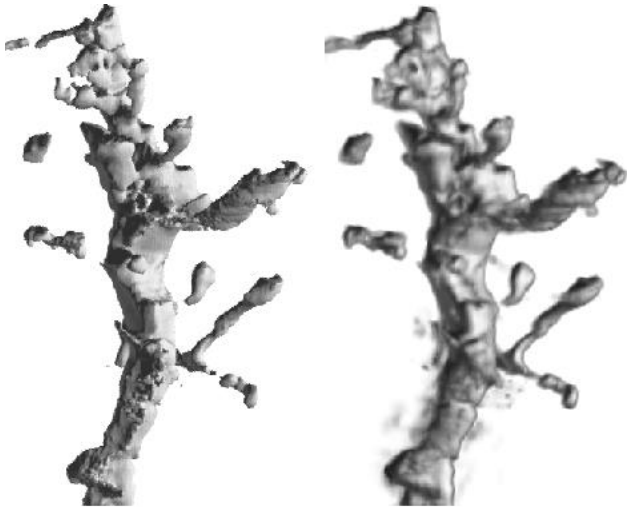
Oceanographic Simulations

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



Volume Rendering Algorithm

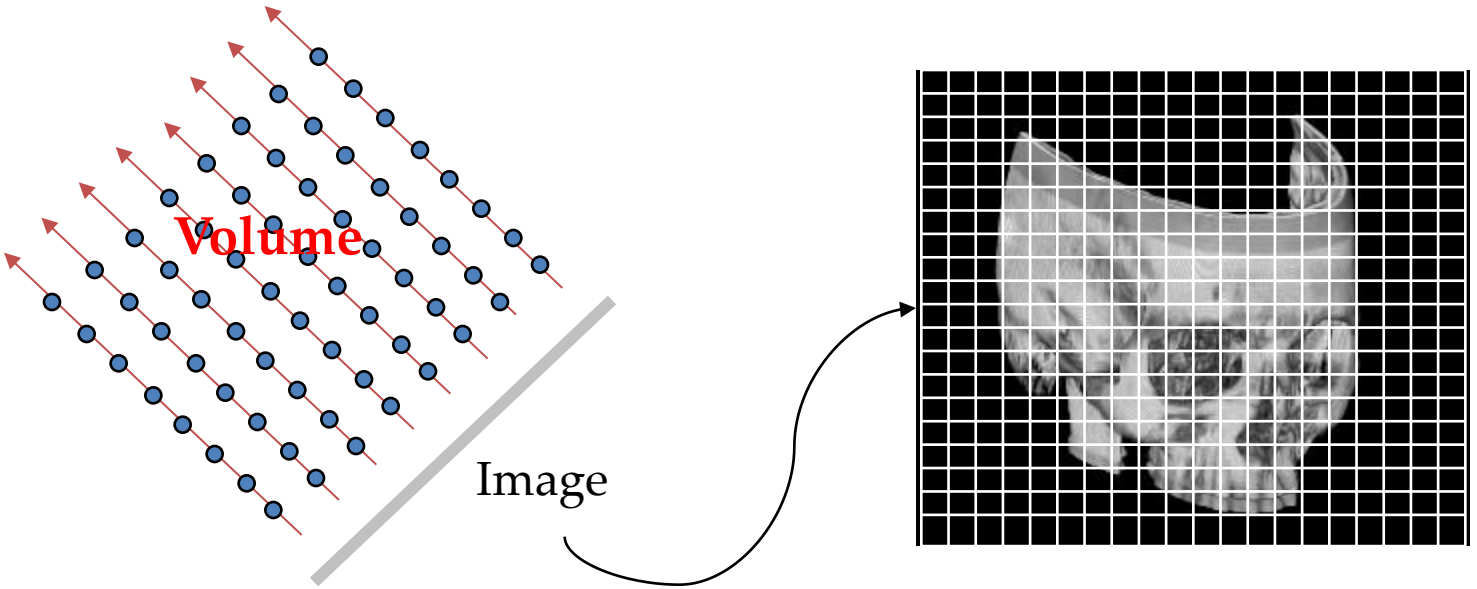
- Direct volume rendering
  - Ray-casting
  - Splatting
- Indirect volume rendering
  - Fourier
- Texture based volume rendering
  - 3D Texture mapping hardware



(a) Isosurface Rendering (b) Direct Volume Rendering

Ray-Casting

View dependent



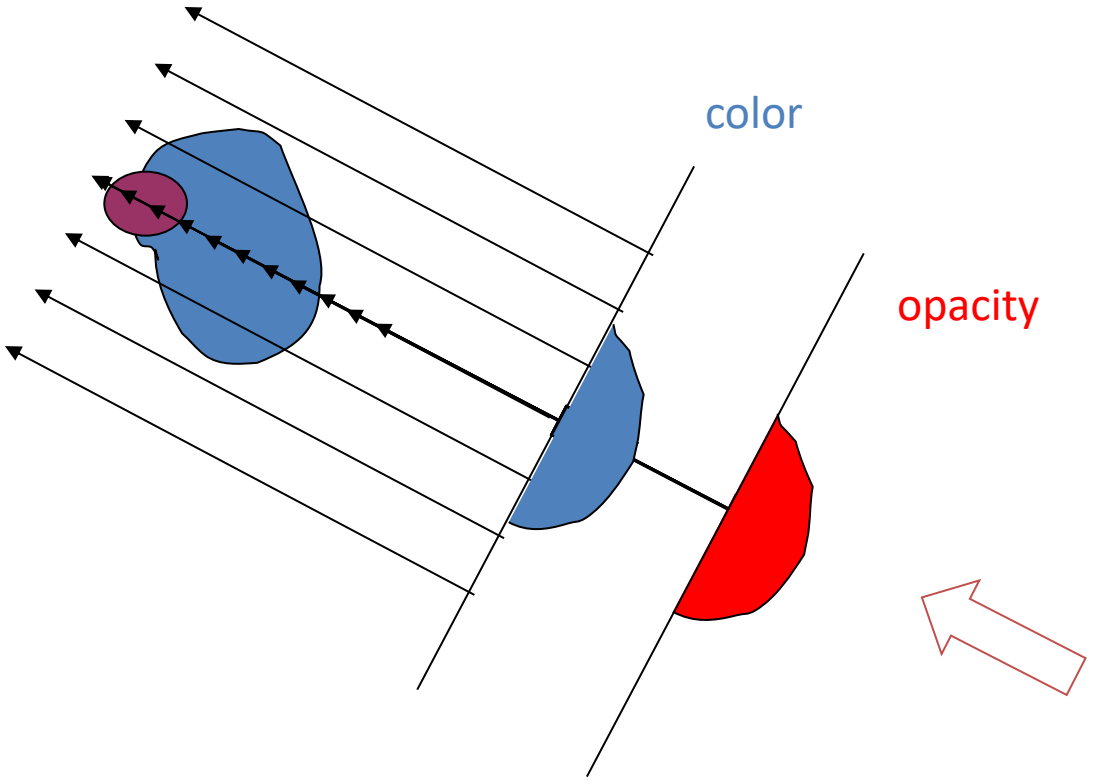
Ray-Casting (cont)

- Advantages
  - Not necessary to explicitly extract surfaces from volume when rendering
  - Can change the transfer functions to make various surfaces stand out within the volume

Ray-Casting (cont)

- Disadvantages
  - Do not have explicit representations for surfaces, therefore not straightforward to compute integral/differential properties
  - Much more computationally intensive to render volume since not dealing directly with the efficient polygon pipeline

Volumetric Ray Integration



Given Colors or Shade Before Resampling

Data comes as color/opacity

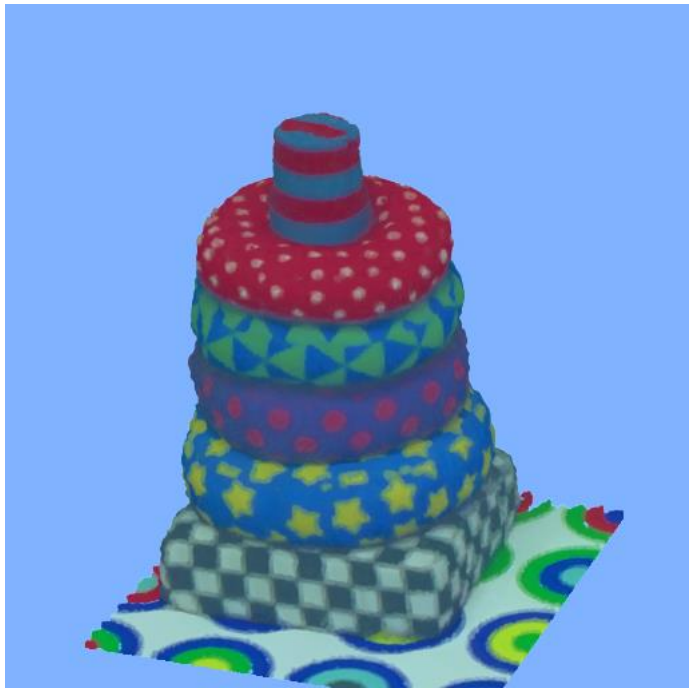
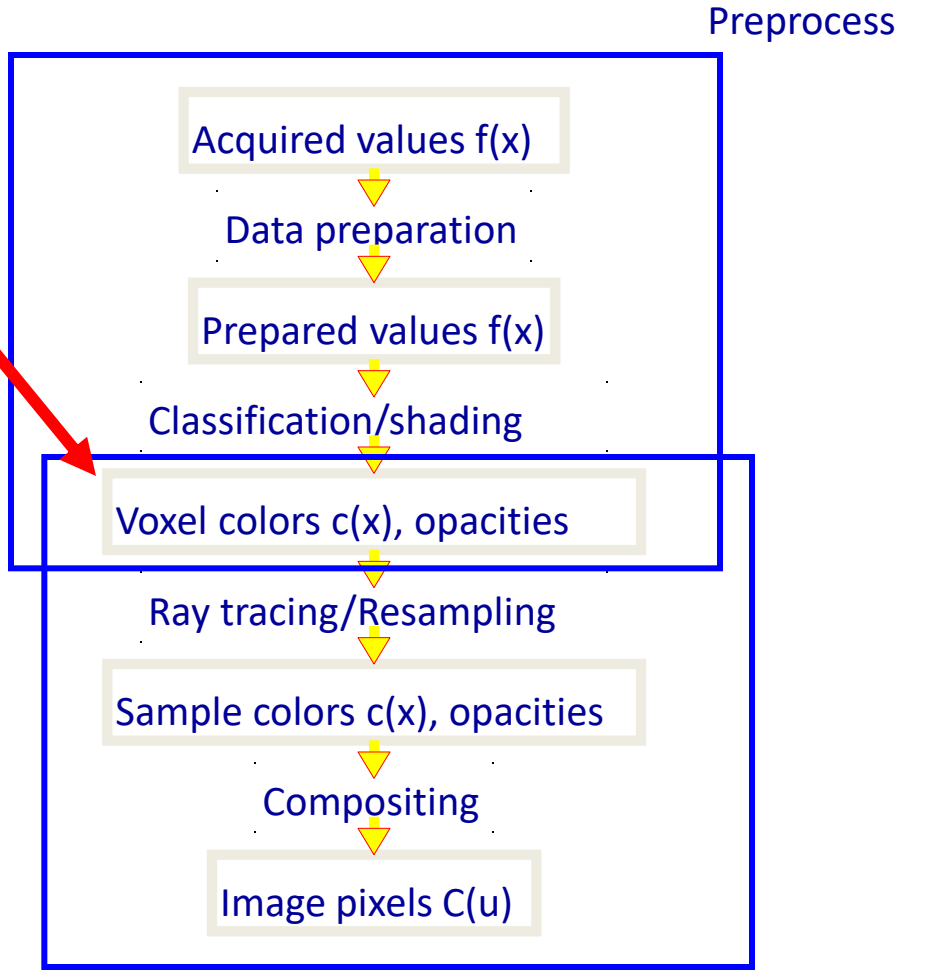


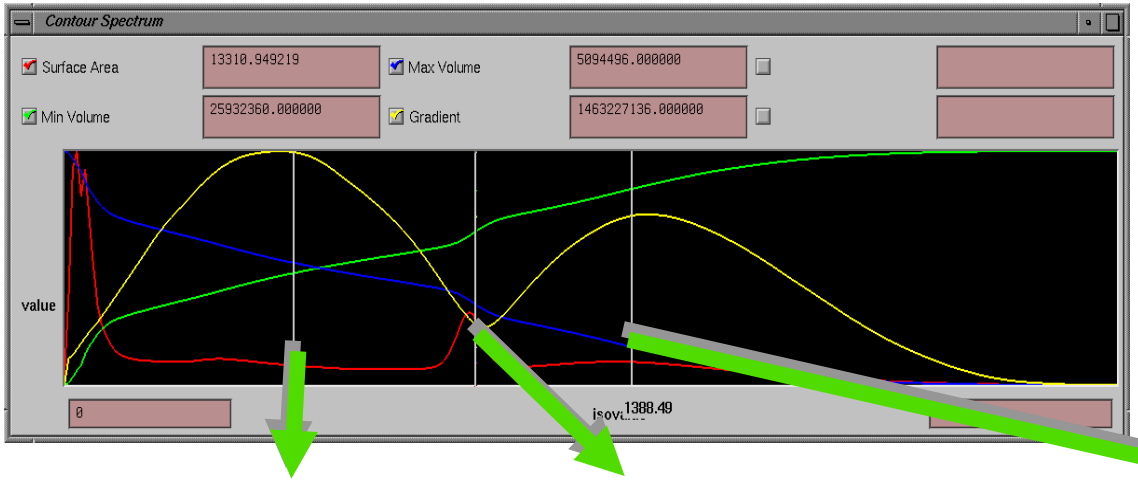
Image based rendering outputs colors



Transfer Functions

- Mapping from data values to renderable optical properties
  - Density
  - Gradient

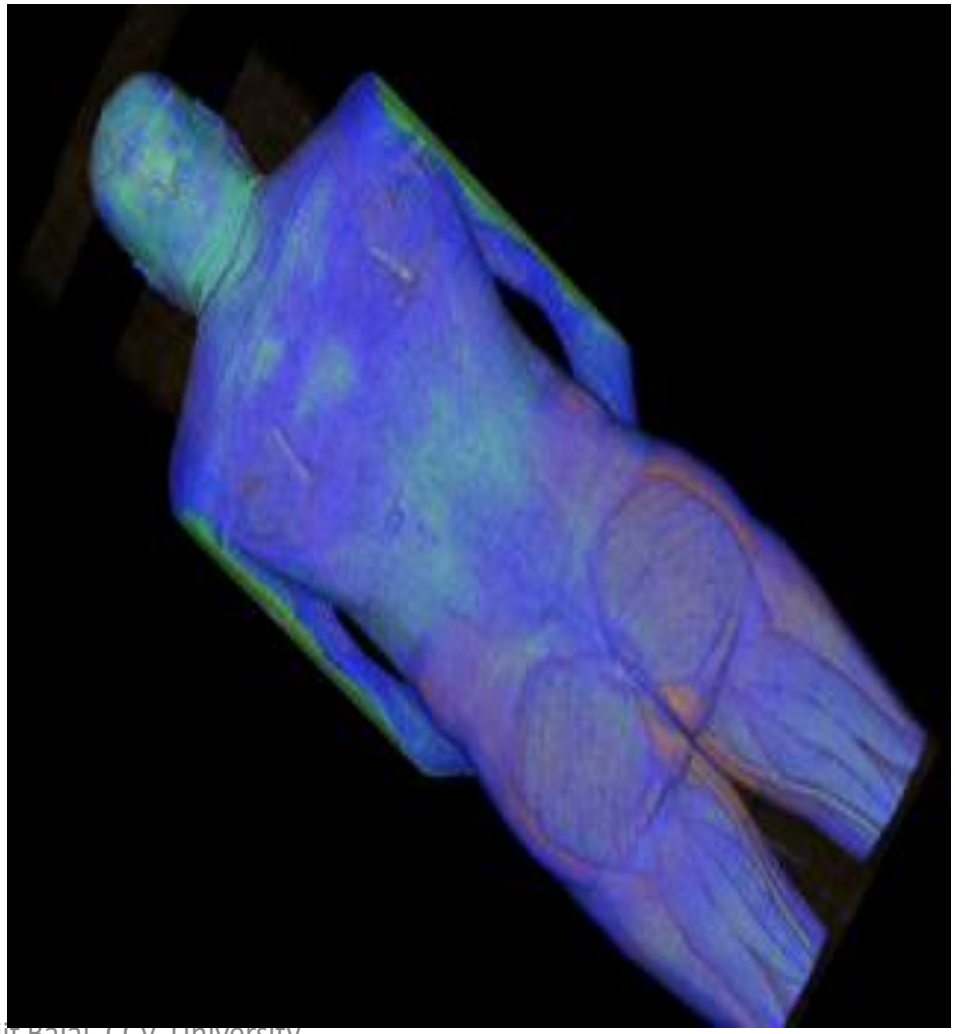
The Contour Spectrum



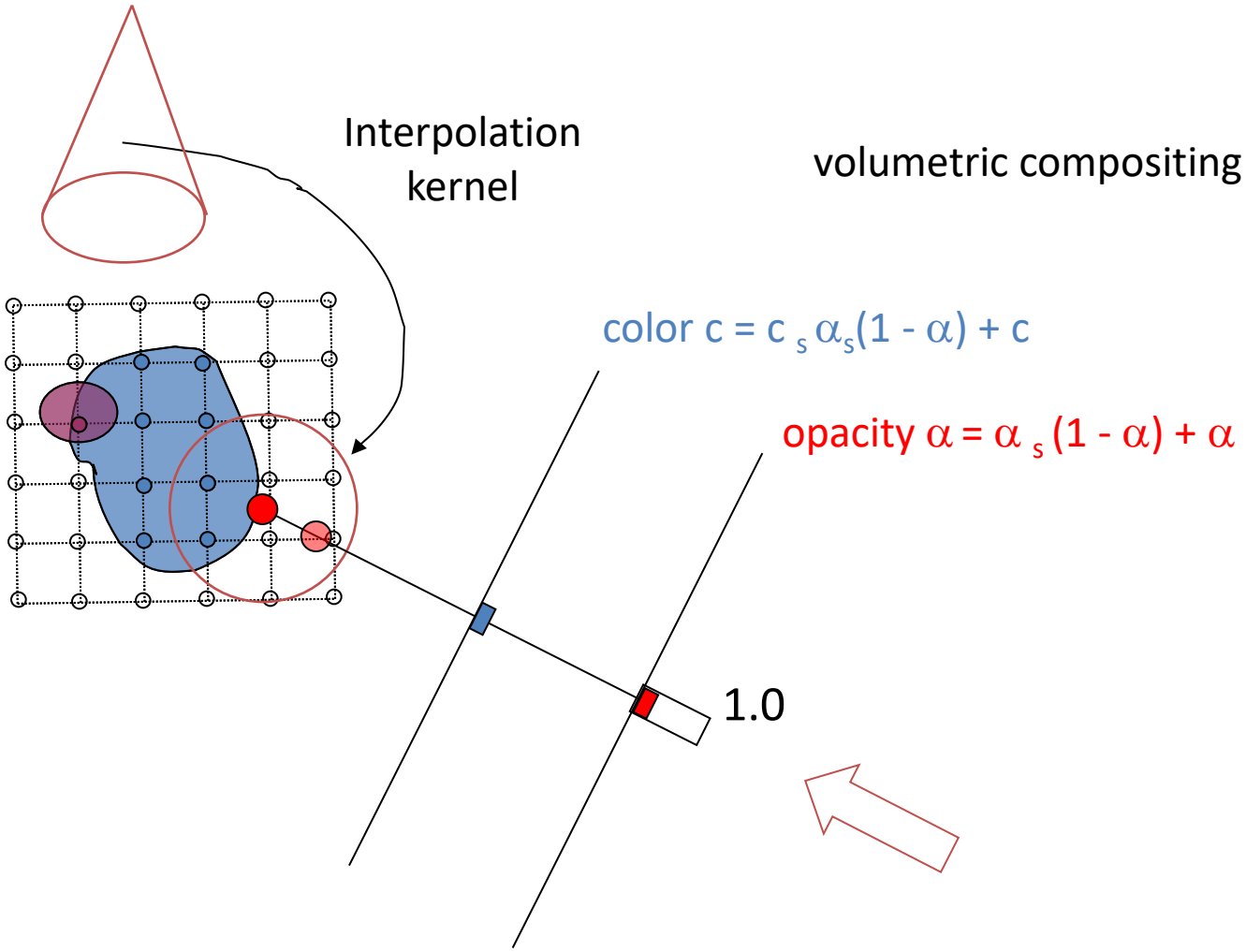
- ***The contour spectrum allows the selection of transfer functions***

Medical Data

(512×512×1871×2(bytes) = 936 MB)



Ray-casting - revisited

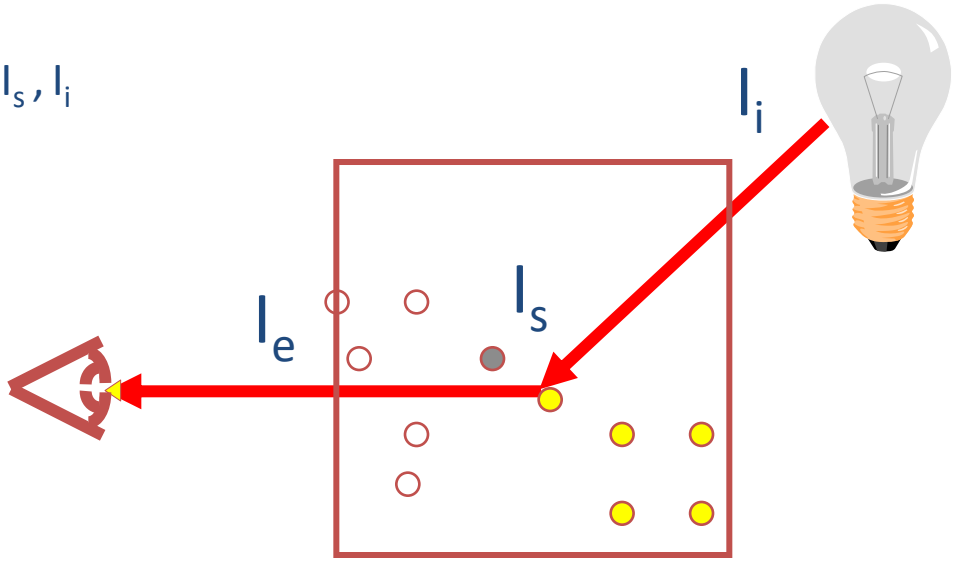


Opacity-Weighted Color

1. From first principles, emitted intensity different from shaded intensity
2. From Blinn, Opacity-Weighting before interpolation helps quality
3. From short cut, cannot do separate interpolation

Derivation from First Principles of Volume Rendering

- Actually change notation I_e, I_s, I_i



1 region in volume

Ray intensity by line integral

$$I_{ray} = \int t(l) I_s(l) \alpha(l) dl$$

$$I_e(l) = I_s(l) \alpha(l)$$

Blinn's Associated Colors

- Associated color, opacity associated or multiplied
- Generalized to Volume Rendering $\tilde{C} = \alpha C$
- Compositing Equations

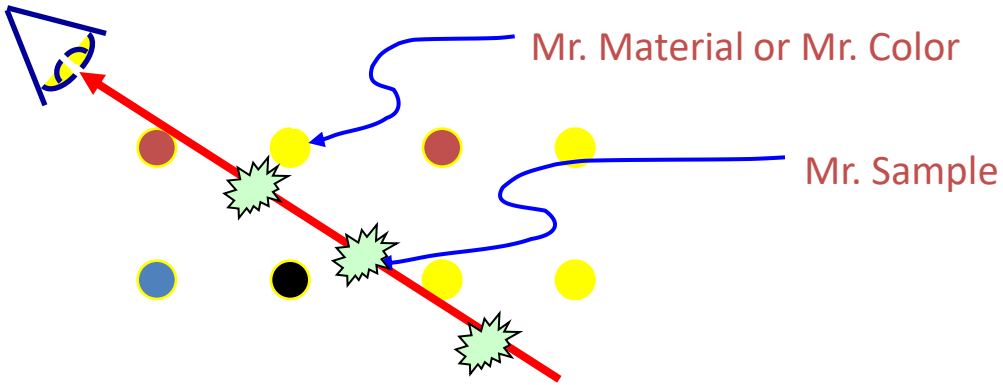
$$\tilde{C}_{\text{new}} = (1 - \alpha_{\text{front}})\tilde{C}_{\text{back}} + \tilde{C}_{\text{front}}$$
$$\alpha_{\text{new}} = (1 - \alpha_{\text{front}})\alpha_{\text{back}} + \alpha_{\text{front}}$$

See Blinn, SIGGRAPH'82,
Porter and Duff, SIGGRAPH'84
Blinn IEEE CGA, Sep. 1994.
See Drebin et al. SIGGRAPH'88

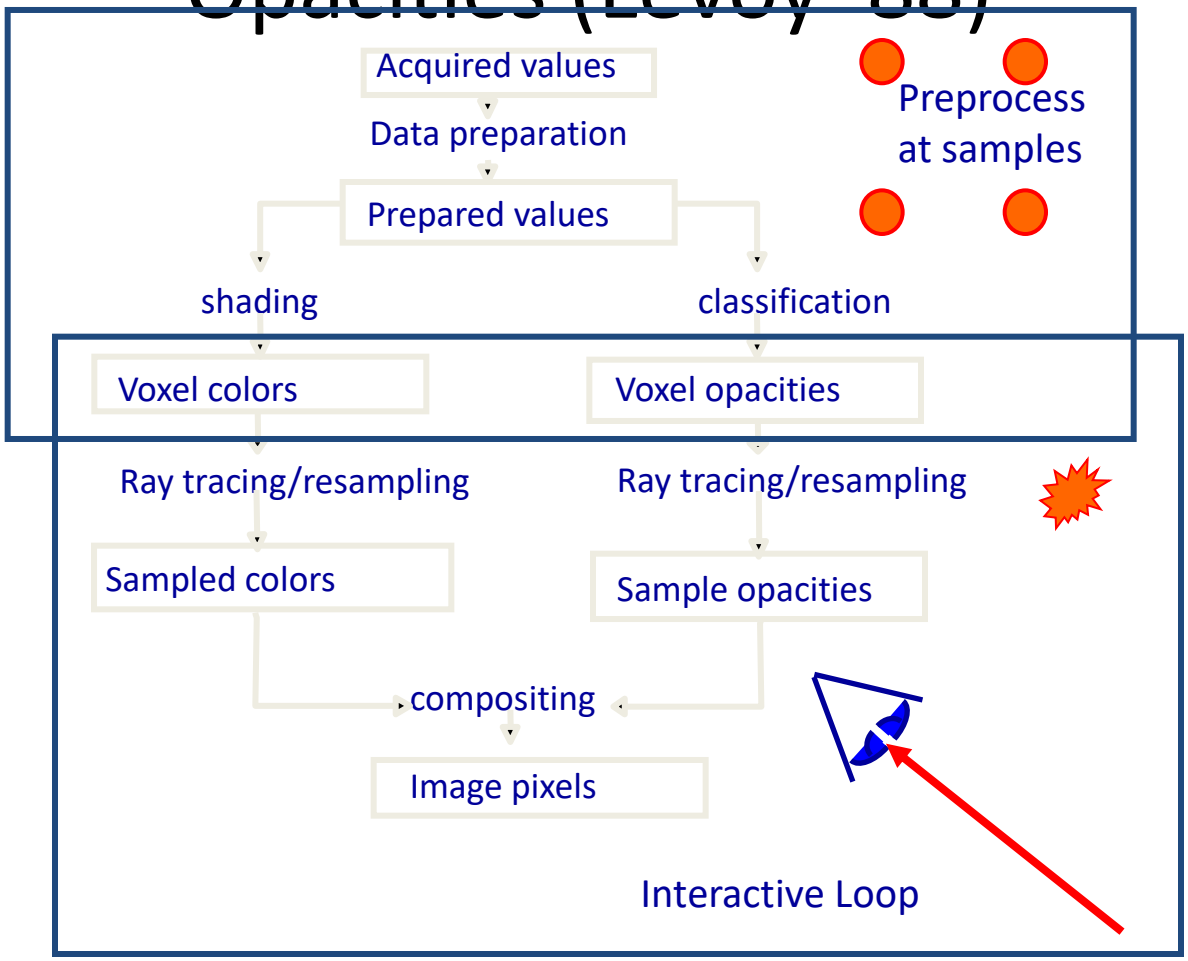
Works for back-to-front, front-to-back, parallel, etc.

A Shortcut to Represent Materials and Shading

- Assume that shading at material samples will give good results
- Levoy: separate interpolation of colors and opacities
- Pre-shade



Separate Interpolation of Colors and Opacities (Levoy '88)



$$C_{out} = C_{in} \cdot \alpha_{in} \cdot (1-\alpha) + C$$

$$\alpha_{out} = \alpha_{in} \cdot (1-\alpha) + \alpha$$

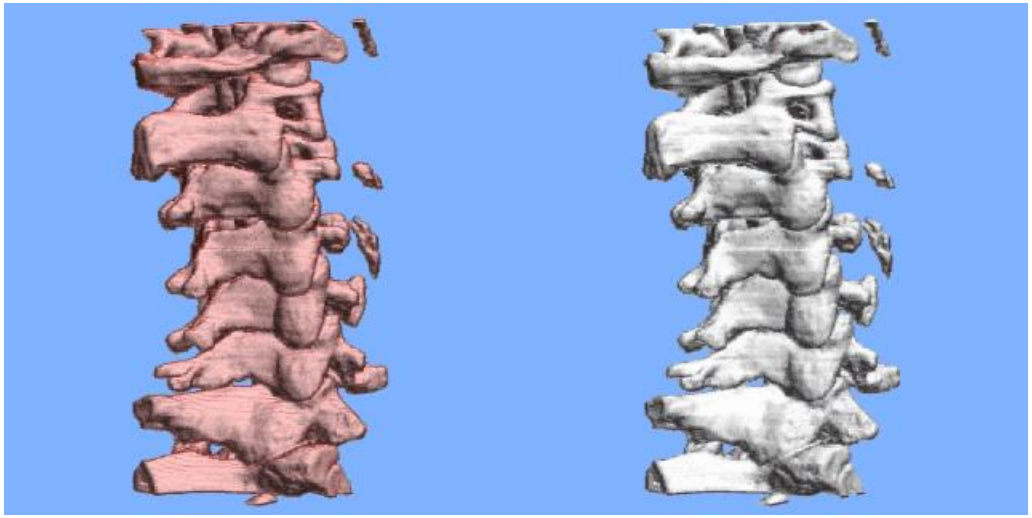
Which one?

$$C_{out} = C_{in} \cdot (1-\alpha) + C \cdot \alpha$$

$$\alpha_{out} = \alpha_{in} \cdot (1-\alpha) + \alpha$$

Opacity-Weighted Color Interpolation

C. M. Wittenbrink, T. Malzbender, and M. E. Goss, Opacity-Weighted Color Interpolation for Volume Sampling, Volume Visualization Symposium ’98, Research Triangle Park, NC, 1998.



Opacity-Weighted Interpolation (Wittenbrink et. al. 98)

FTB color:

$$\tilde{C}_{\text{front(new)}} = (1-\alpha_{\text{front}})C_{\text{back}}\alpha_{\text{back}} + \tilde{C}_{\text{front(old)}}$$

BTF color:

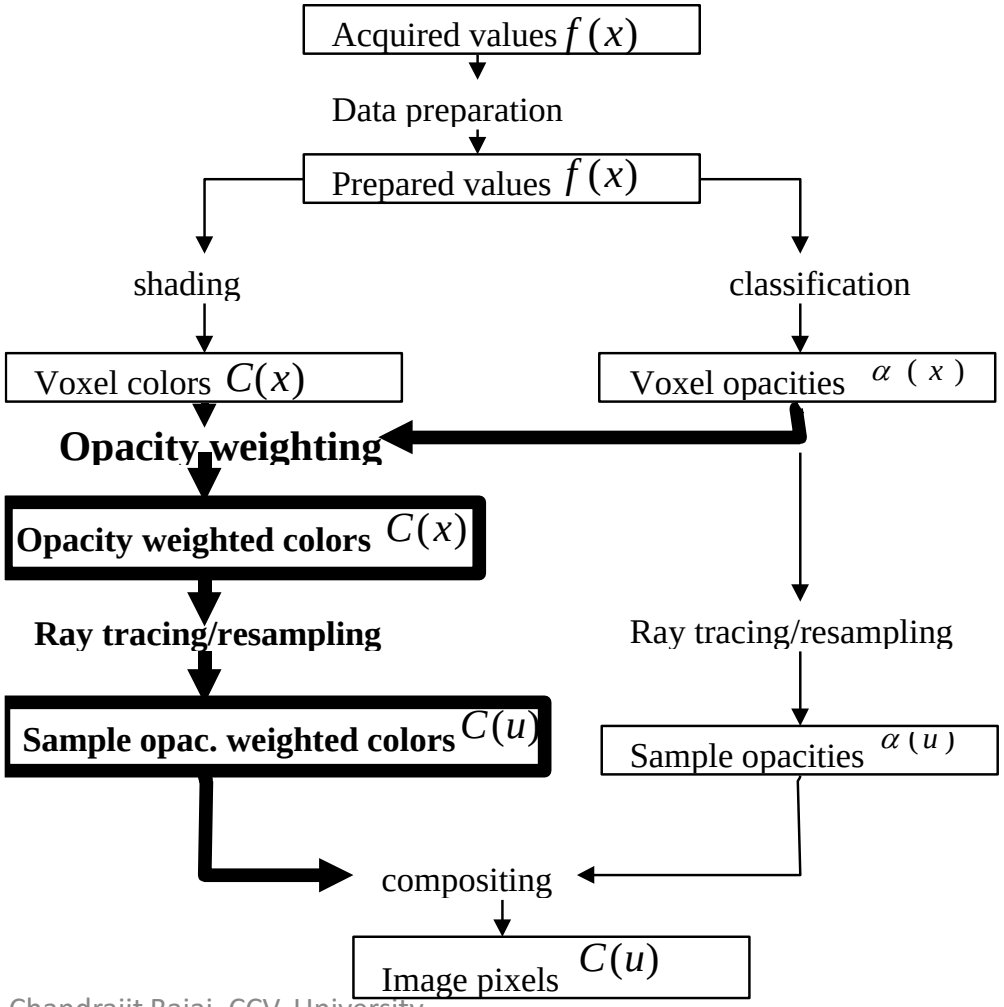
$$\tilde{C}_{\text{back(new)}} = (1-\alpha_{\text{front}})\tilde{C}_{\text{back(old)}} + C_{\text{front}}\alpha_{\text{front}}$$

Opacity:

$$\alpha_{\text{new}} = (1-\alpha_{\text{front}})\alpha_{\text{back}} + \alpha_{\text{front}}$$

The colors that are composited must be pre-weighted with opacity, i.e. associate color:

$$C' = \alpha C$$



Example Calculation

Separate

$$\tilde{C}_{12} = (1-\alpha_1)C_2\alpha_2 + C_1\alpha_1$$
$$= (1-0)\times 0.5\times 0.5 + 0\times 0 = 0.25$$

$$\alpha_{12} = (1-\alpha_1)\alpha_2 + \alpha_1$$
$$= (1-0)\times 0.5 + 0 = 0.5$$

$$\tilde{C}_{123} = (1-\alpha_{12})C_3\alpha_3 + \tilde{C}_{12}$$
$$= (1-0.5)\times 1\times 1 + 0.25 = 0.75$$

Opacity-weighted

$$\tilde{C}_{12} = (1-\alpha_1)\tilde{C}_2 + \tilde{C}_1$$
$$= (1-0)\times 0.5 + 0 = 0.5$$

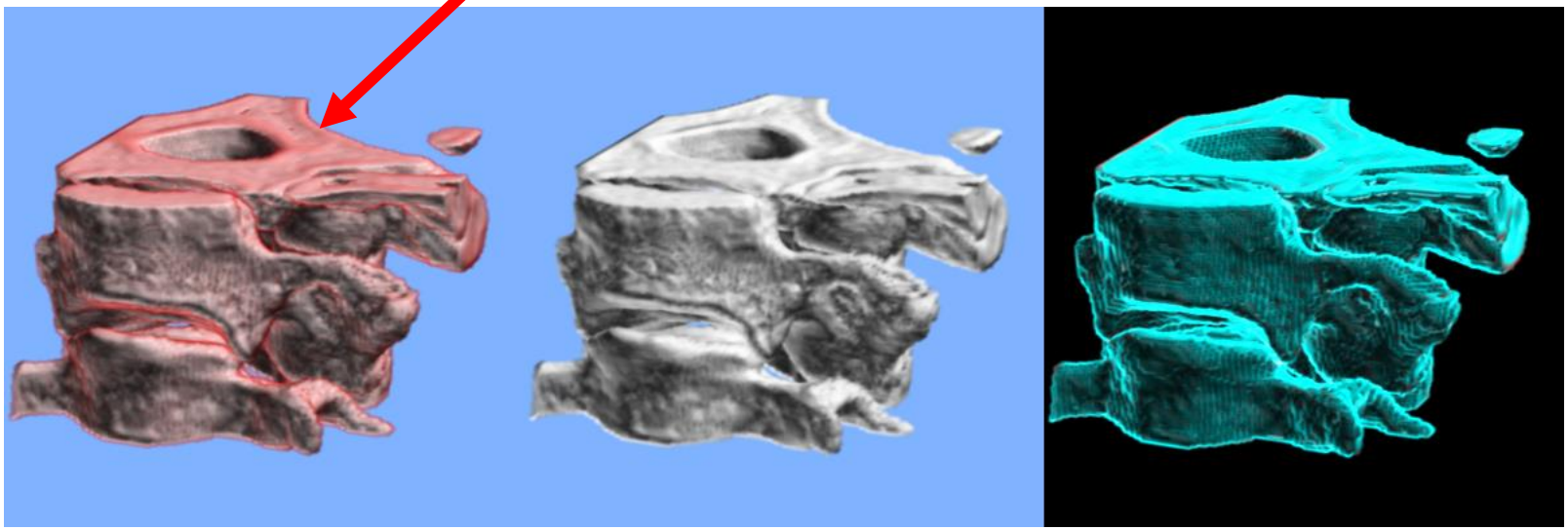
Different color

$$\tilde{C}_{123} = (1-\alpha_{12})\tilde{C}_3 + \tilde{C}_{12}$$
$$= (1-0.5)\times 1 + 0.5 = 1$$

Rendering Comparison

Red tissue bleeds onto white bone

Color errors



Separate

Opacity-weighted

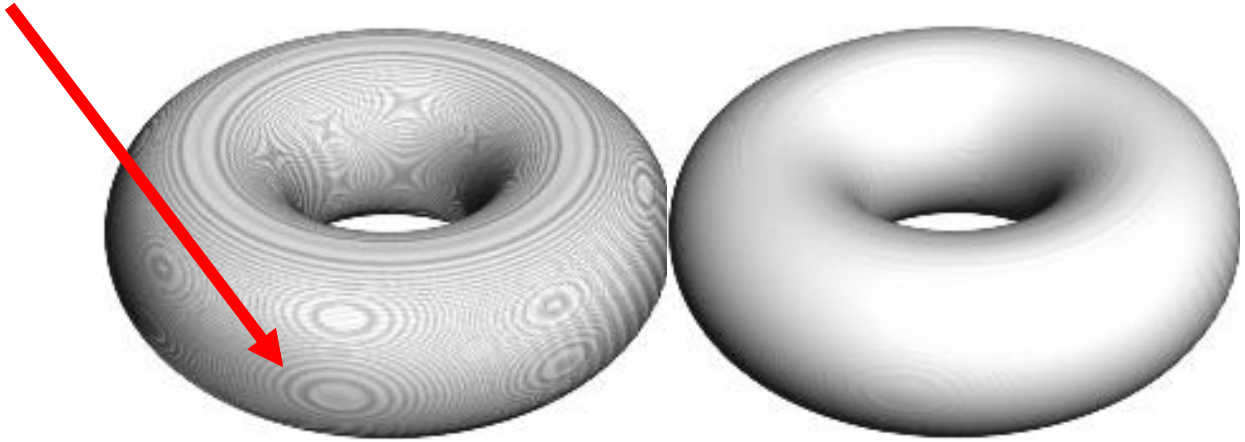
Difference

100x96x249 spiral CT dataset, classified to 8 bit

Rendering Comparison (cont)

Intensity errors

Banding results from
black air marking surface

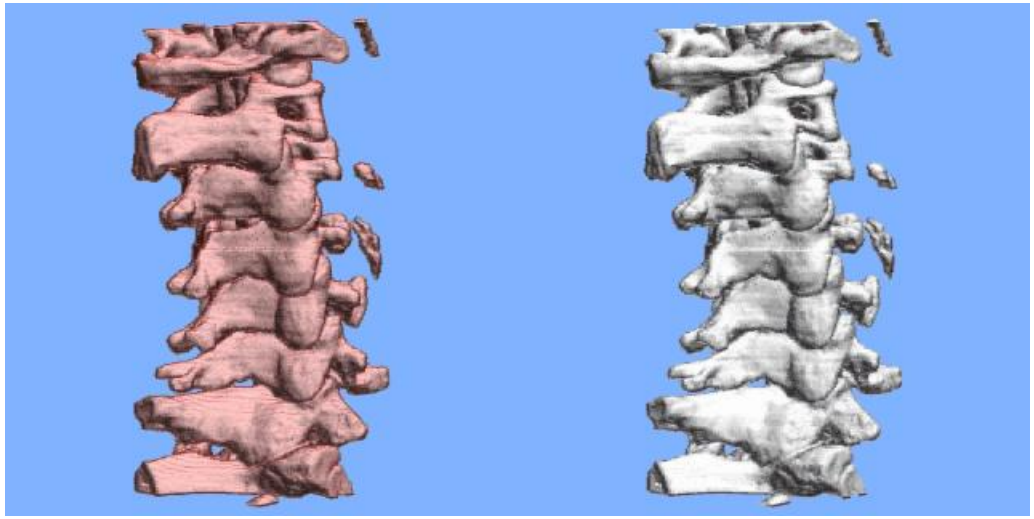


Separate Opacity-weighted

Torus volume, pre-antialiased

Spiral CT Rendering Comparison

Artifact appears to be aliasing Color & intensity errors



Separate Opacity-weighted

Summary: Opacity-Weighted Color Interpolation

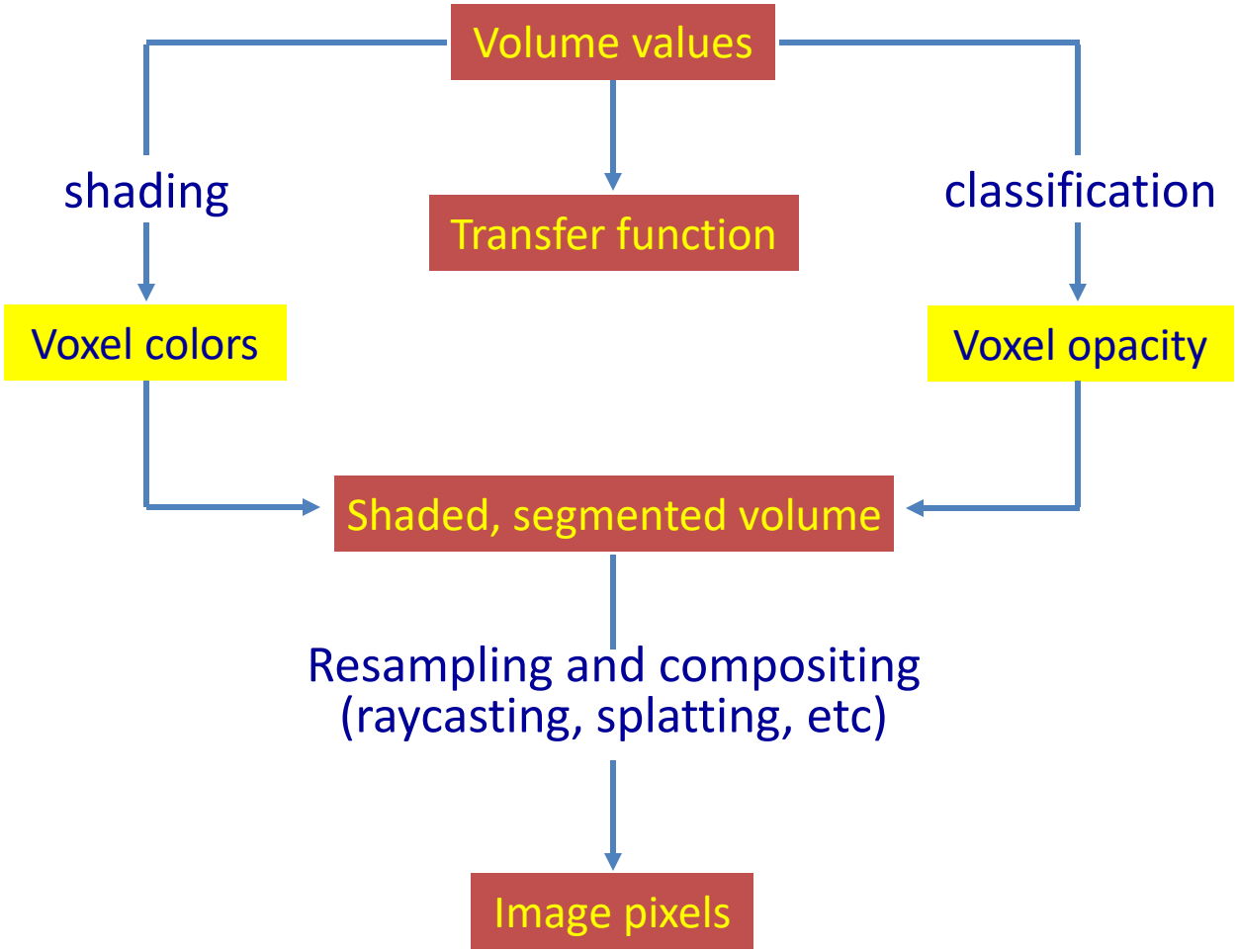
- Opacity-weight $\omega_i = w_i \alpha_i$
- Compute ray sample opacity $\alpha = \sum_i \omega_i$
- Compute ray sample color $\tilde{C} = \sum_i \omega_i C_i$
- Composite

$$\tilde{C}_{\text{new}} = (1 - \alpha_{\text{front}})\tilde{C}_{\text{back}} + \tilde{C}_{\text{front}}$$

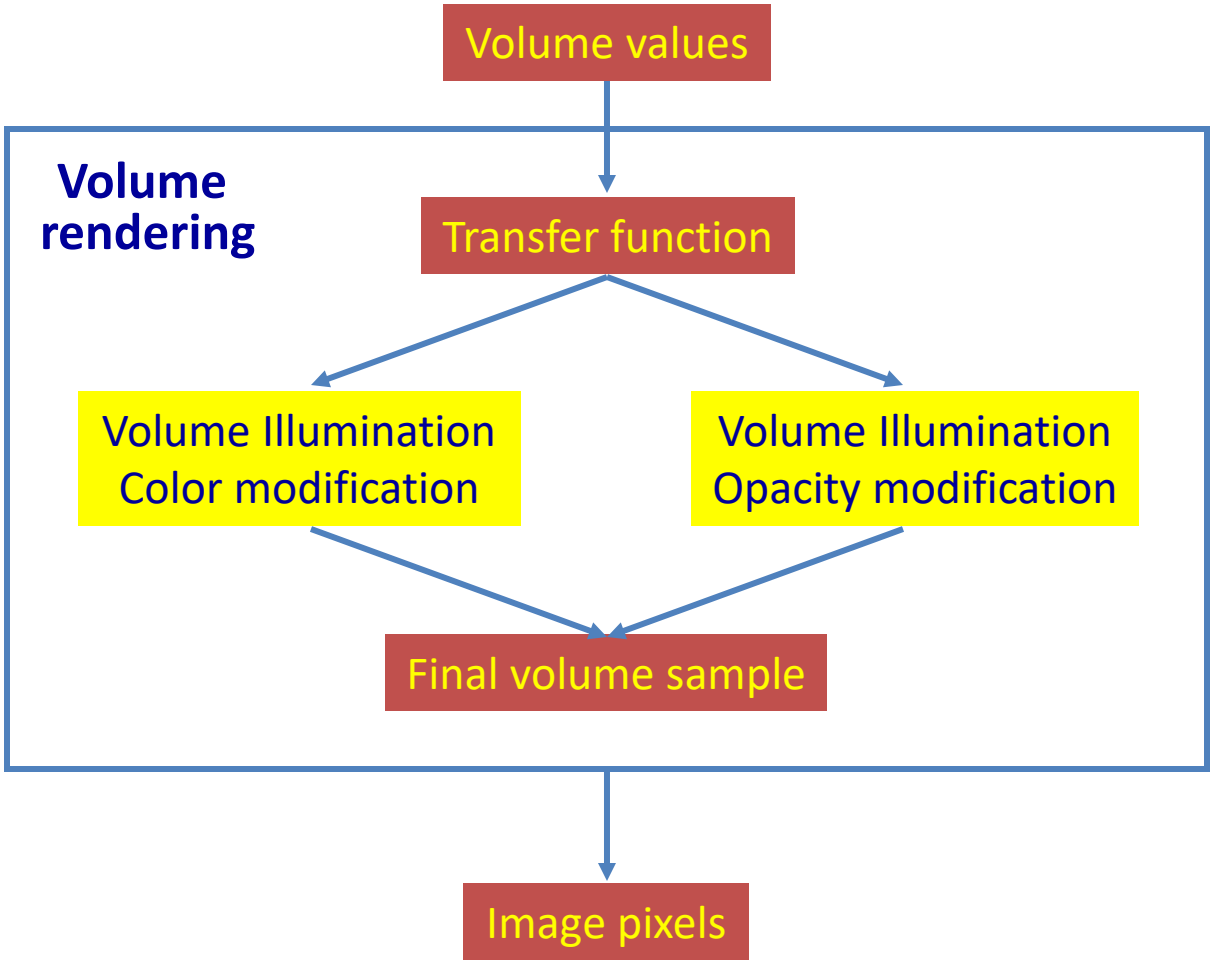
Volume Illustration

- Non-photorealistic rendering of volume models
- Properties
  - Volume sample location and value
  - Local volumetric properties, such as gradient and minimal change direction
  - View direction
  - Light information

Traditional Volume Rendering Pipeline



Volume Illustration Rendering Pipeline



Feature Enhancement

- Boundary enhancement
  - Gradient-based opacity

$$o_g = o_v(k_{gc} + k_{gs}(\|\nabla_f\|)^{k_{ge}})$$

Original opacity

Value gradient of the volume at the sample

Feature Enhancement (cont)

- Boundary enhancement example

Original volume rendering

Boundary enhancement

$k_{gc} = 0.7, k_{gs} = 10, k_{ge} = 2.0$

Feature Enhancement (cont)

- Oriented feature enhancement
  - Silhouette enhancement

$$o_s = o_v(k_{sc} + k_{ss}(1 - abs(\nabla_{fn} \cdot V))^{k_{se}})$$

gradient

View direction

Feature Enhancement (cont)

- Silhouettes enhancement example

Original volume rendering

Silhouette and boundary enhancement

$$k_{gc} = 0.8, k_{gs} = 5.0, k_{ge} = 1.0;$$

$$k_{sc} = 0.9, k_{ss} = 50, k_{se} = 0.25$$

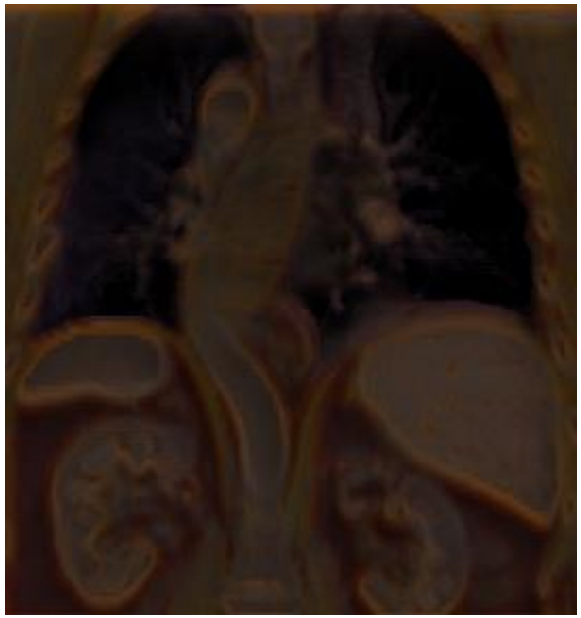
Feature Enhancement (cont)

Original volume rendering

Boundary enhancement

Silhouette and boundary enhancement

Feature Enhancement (cont)



Boundary saturation increased and value also increased

Boundary saturation increased and value decreased

Volumetric colored sketch of data

Depth and Orientation Cues

- Distance color blending
  - Depth-cued color

$$c_d = (1 - k_{ds} d_v^{k_{de}}) c_v + k_{ds} d_v^{k_{de}} c_b$$

controls the size of the color blending effect

controls the rate of application of color blending

The fraction of distance through the volume

Voxel color

Background color

Depth and Orientation Cues (cont)

- Distance color blending example

Original volume rendering

Distance coloring, boundary, and silhouette enhancement

$c_b = (0,0,0.15), k_{ds} = 1.0, k_{de} = 0.5$

Depth and Orientation Cues (cont)

- Feature halos
  - The size of halo effect

$$h_i = \left(\sum_{n}^{neighbors} \frac{h_n}{\left\| P_i - P_n \right\|^2} \right) \left(1 - \left\| \nabla_f (P_i) \right\| \right)$$

The maximum potential halo contribution of a neighbor

location

$$h_n = \left(\nabla_{fn}(P_n) \cdot \left(\frac{(P_i - P_n)}{\left\| P_i - P_n \right\|} \right) \right)^{k_{hpe}} \left(1 - \nabla_{fn}(P_n) \cdot V \right)^{k_{hse}}$$

Depth and Orientation Cues (cont)

- Feature halos example

Original volume rendering

Halos, boundary, and silhouette enhancement

$k_{hpe} = 1.0, k_{hse} = 2.0$

Depth and Orientation Cues (cont)

- Tone shading

number of lights

illuminated object color contribution

$$c = k_{ta} I_G + \sum_{i}^{N_L} \left(I_t + k_{td} I_o \right)$$

controls the amount of gaseous illumination

tone contribution to volume sample color

controls the amount of directed illumination

$$I_t = ((1.0 + \nabla_{fn} \cdot L)/2) c_w + (1 - (1.0 + \nabla_{fn} \cdot L)/2) c_c$$

warm tone color ($k_{ty}, k_{ty}, 0$)

cool tone color (0, 0, k_{tb})

$$I_o = \begin{cases} k_{td} I_i (\nabla_{fn} \cdot L) : \nabla_{fn} \cdot L > 0 \\ 0 : \nabla_{fn} \cdot L \le 0 \end{cases}$$

Depth and Orientation Cues (cont)

- Tone shading example

Original volume rendering

Tone shading, boundary, and silhouette enhancement

$k_{ty} = 0.3, k_{tb} = 0.3, k_{ta} = 1.0, k_{td} = 0.6$

Depth and Orientation Cues (cont)

Distance coloring, boundary, and silhouette enhancement

Halos, boundary, and silhouette enhancement

Tone shading, boundary, and silhouette enhancement

Depth and Orientation Cues (cont)

- Gray scale data

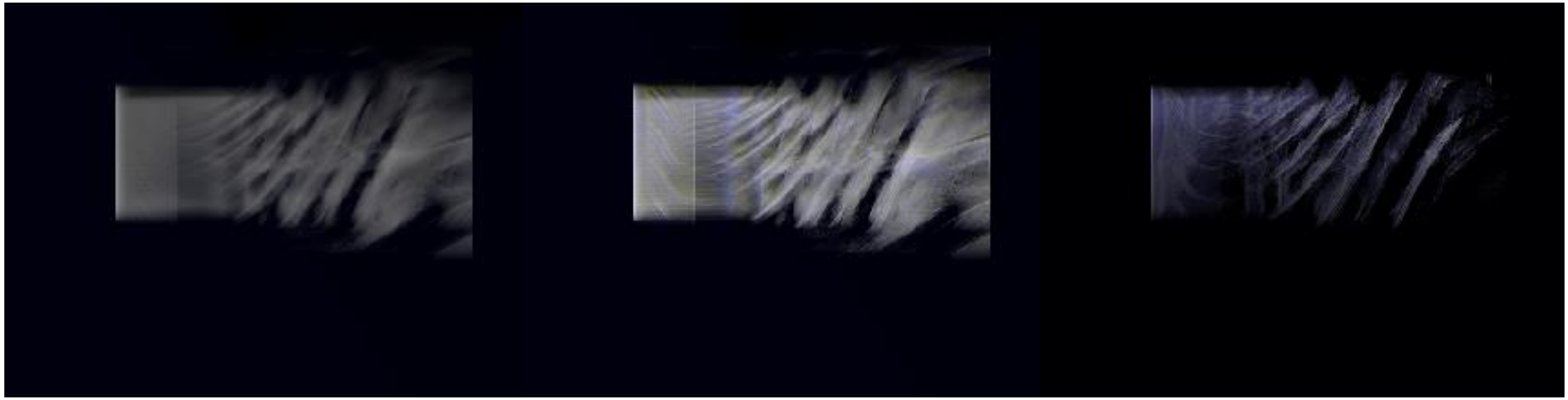
Original volume rendered image

Tone enhancement of image data

Boundary volumetric sketch of data

Depth and Orientation Cues (cont)

- 2D square vortex results



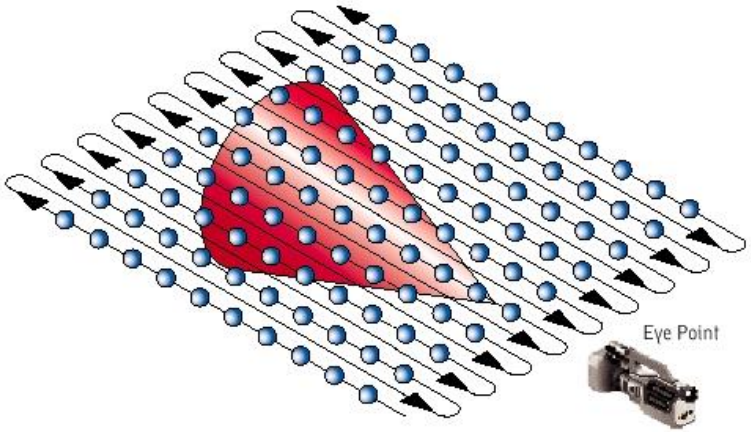
Original gaseous rendering of jet

Tone shading, boundary, silhouette enhancement added

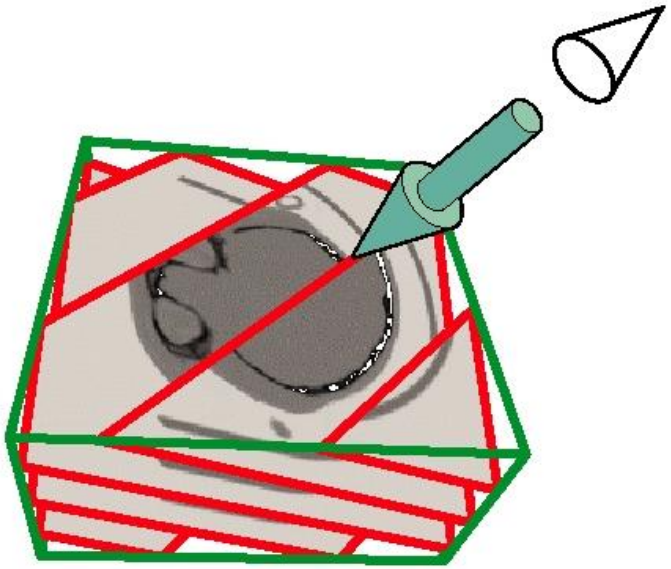
White silhouette color fading added to blue gaseous volume

Texture Based Volume Rendering

- 3D Texture mapping hardware

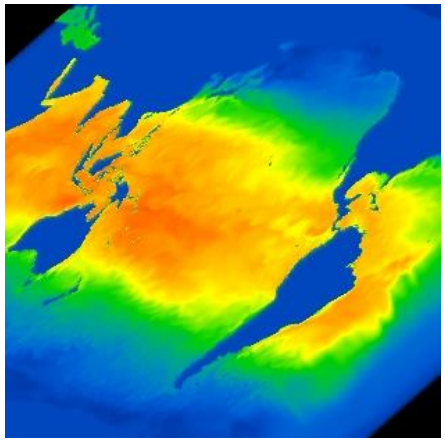


Ray Casting



3D Texture Mapping

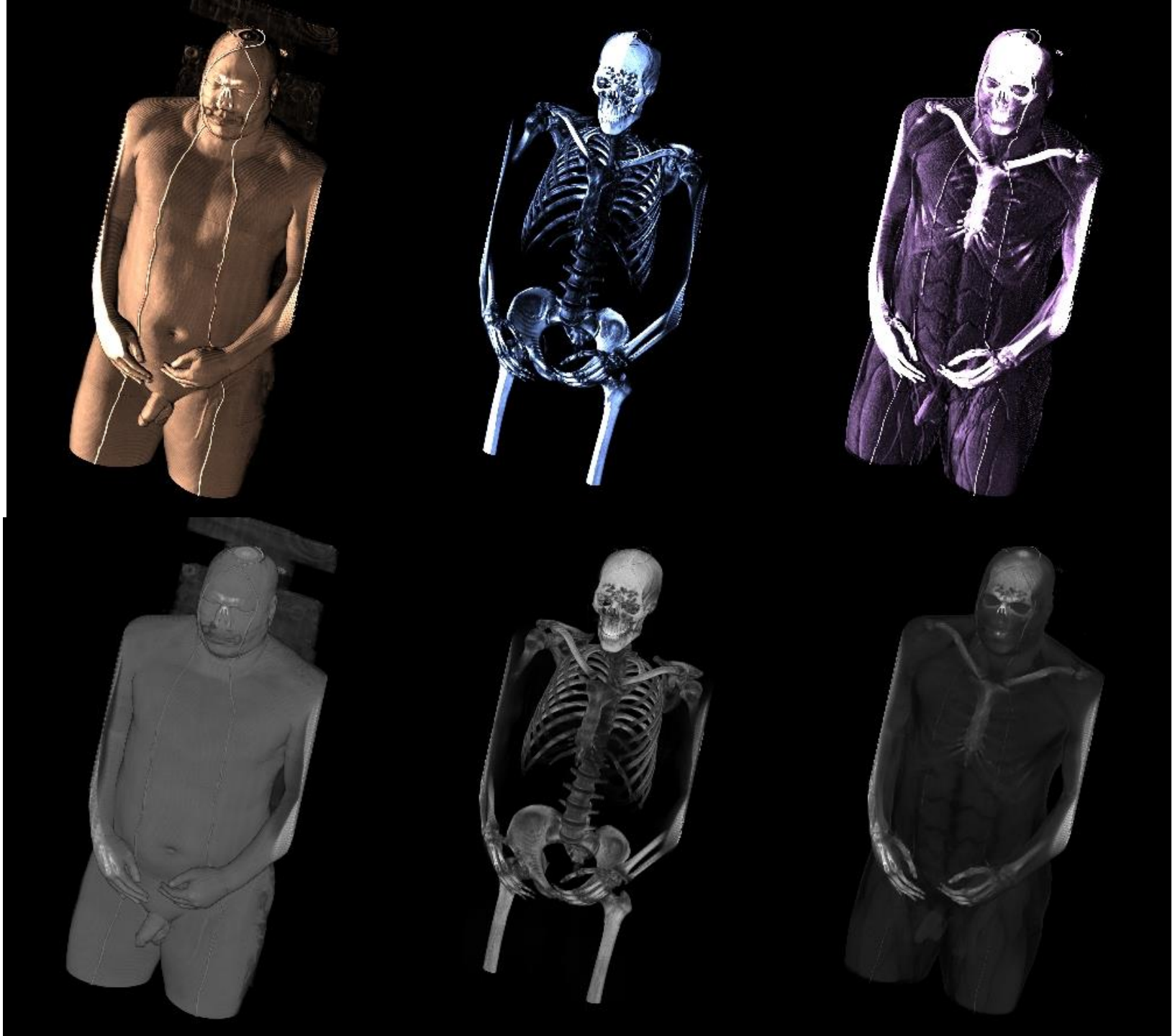
Parallel Texture Based Volume Rendering



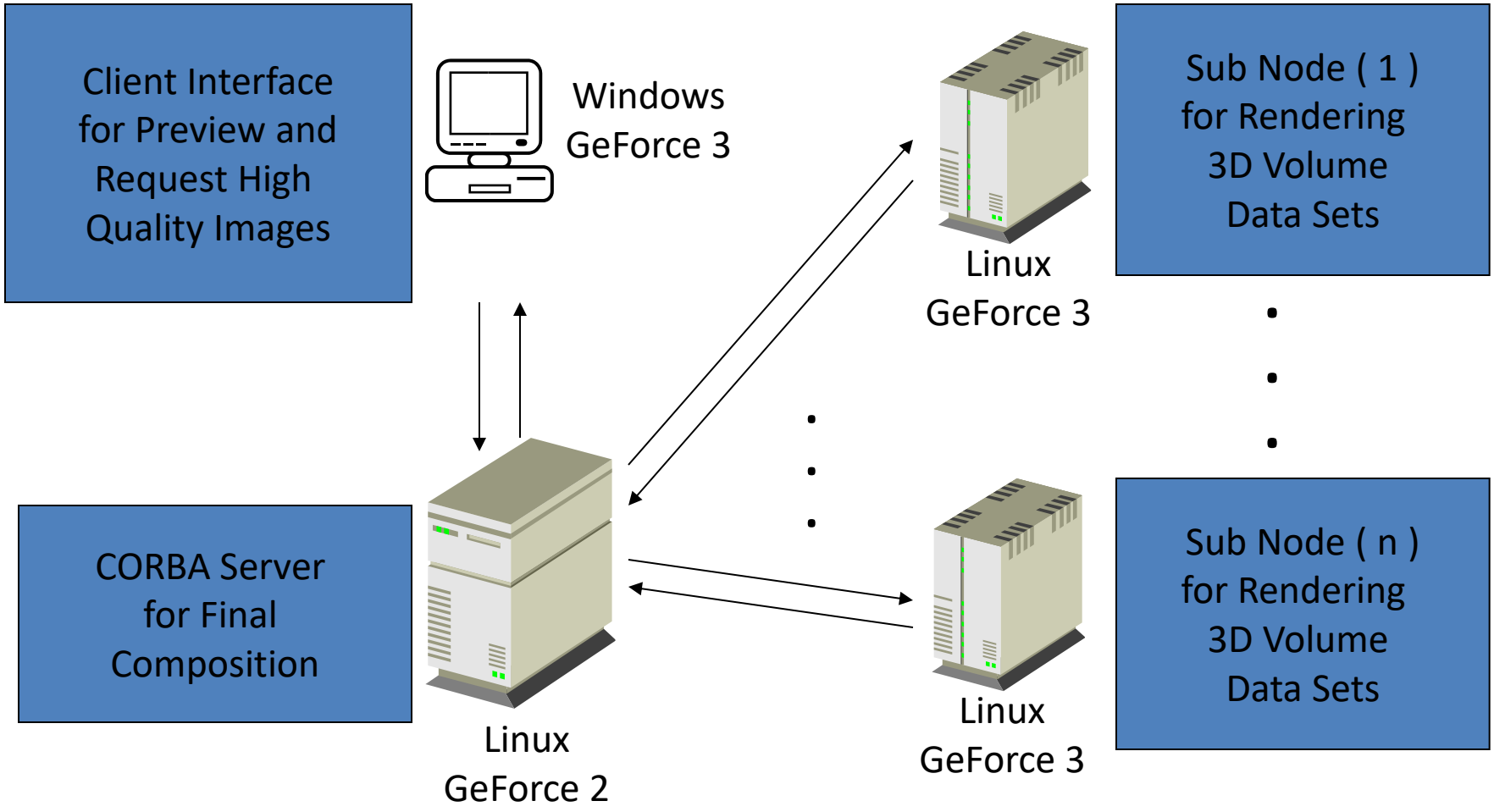
Real-time multipipe texture based volume rendering of the time-varying oceanography temperature data.

Visualization of seismic simulation data on the CCV Visualization Lab's front multi projection system.

Shaded image of the Visible Human female data using texture hardware.



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

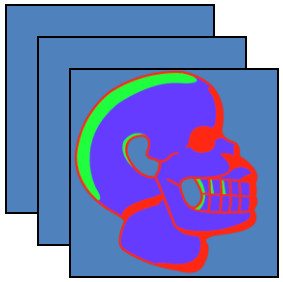
Color Look-up Table

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

Front-to-back Composition

- Texture-mapped planes blending

$$C_d = C_d + (1-\alpha_d)\alpha_s C_s \ , \quad \alpha_d = \alpha_d + (1-\alpha_d)\alpha_s$$

C_d : Destination color C_s : Source color

α_d : Destination alpha α_s : Source alpha

- Final composition of sub-images

$$C_d = C_d + (1-\alpha_d)C_s \ , \quad \alpha_d = \alpha_d + (1-\alpha_d)\alpha_s$$

- OpenGL Commands and notes
  - glBlendFunc (GL_ONE_MINUS_DST_ALPHA , GL_ONE)
  - $\alpha_s C_s$ should be pre-multiplied using a color table or register combiners of GeForce3

Image Enhancement

- Bilateral Filter

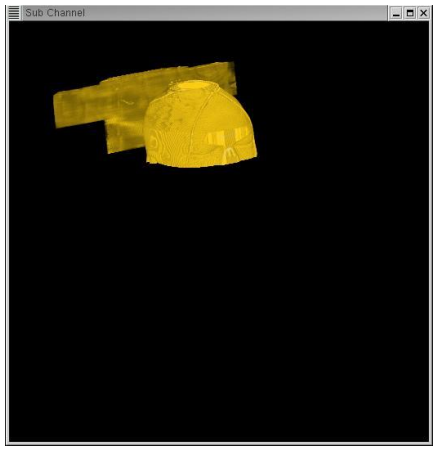
$$w_{ijk} = \exp\left[-\frac{\left(f(x,y,z) - f(i,j,k) \right)^2}{2\delta^2} \right]$$

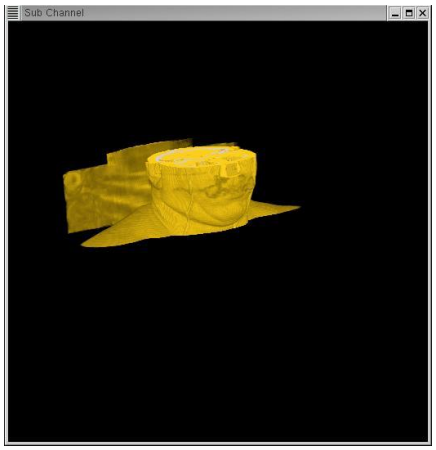
$$f_{new}(x,y,z) = \sum\sum\sum w_{ijk} \cdot f(i,j,k)$$

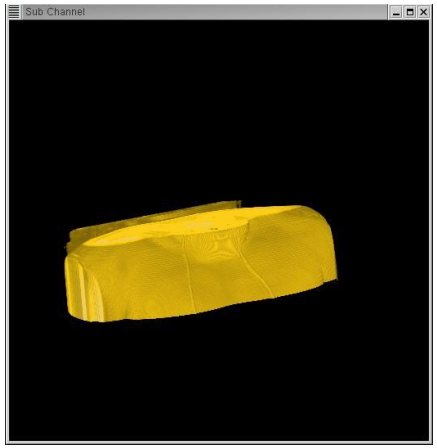
$f(i,j,k)$: Original image, $f_{new}(x,y,z)$: New image

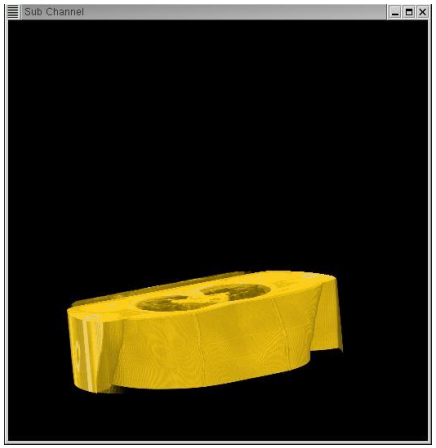
- Normal Calculation
  - (Multi-Linear Centroid Averaging) MLCA

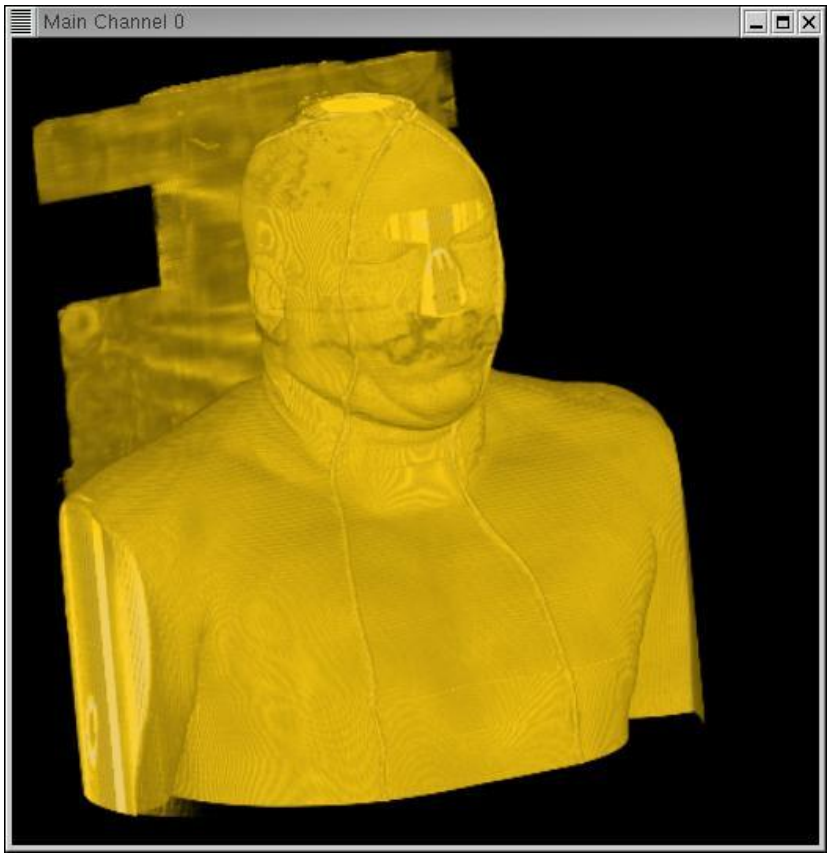
Unshaded Images of Each Node and A Final Image - Skin





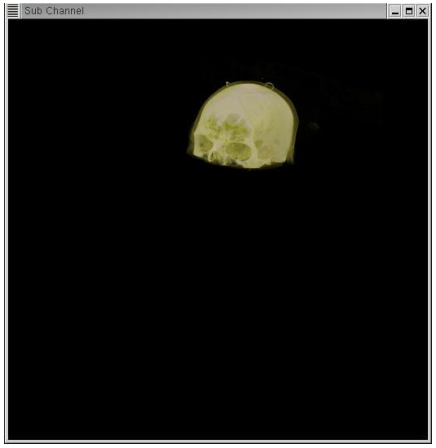


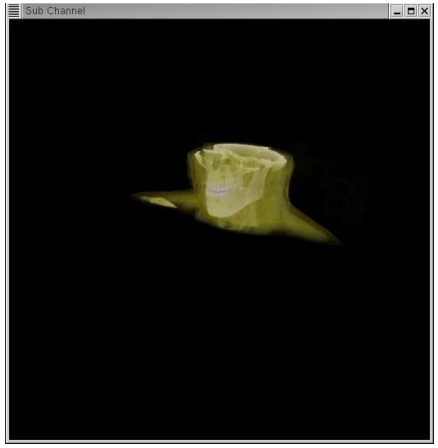


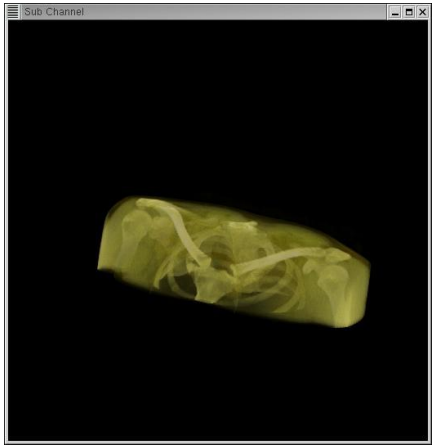


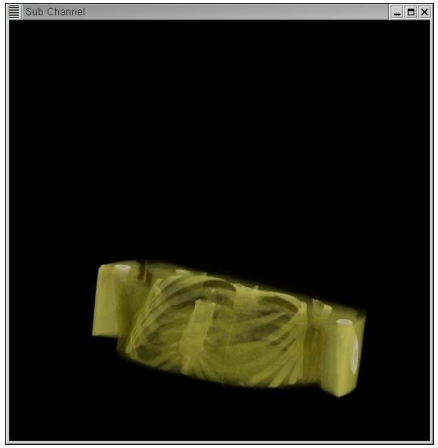
- Data Size : 512^3
- Perfomance : 4.01fps

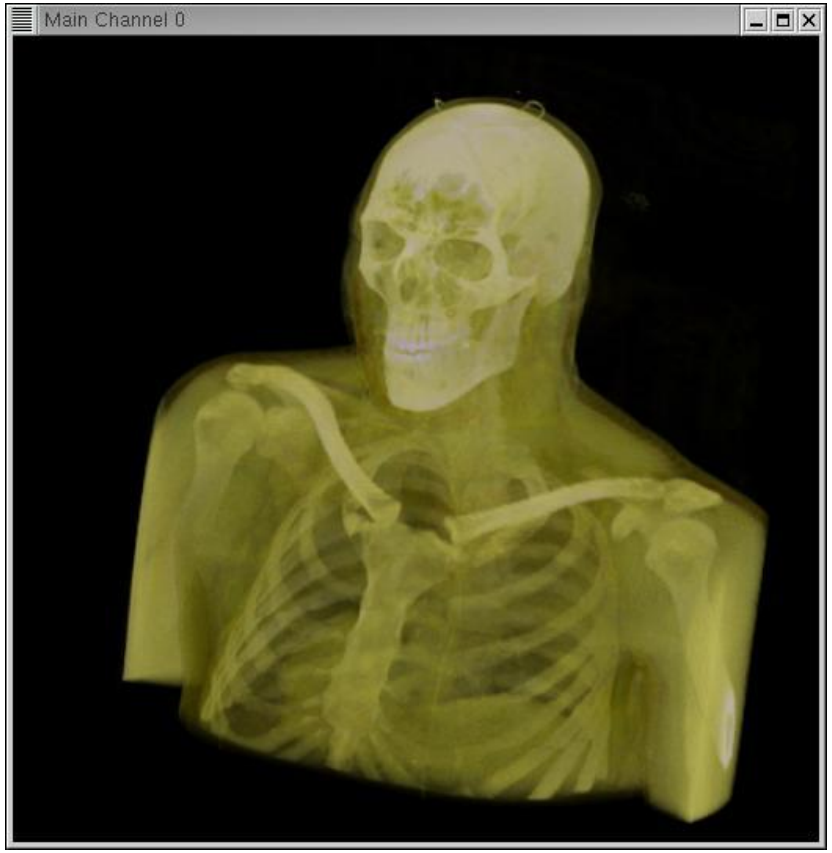
Unshaded Images of Each Node and A Final Image - Bones





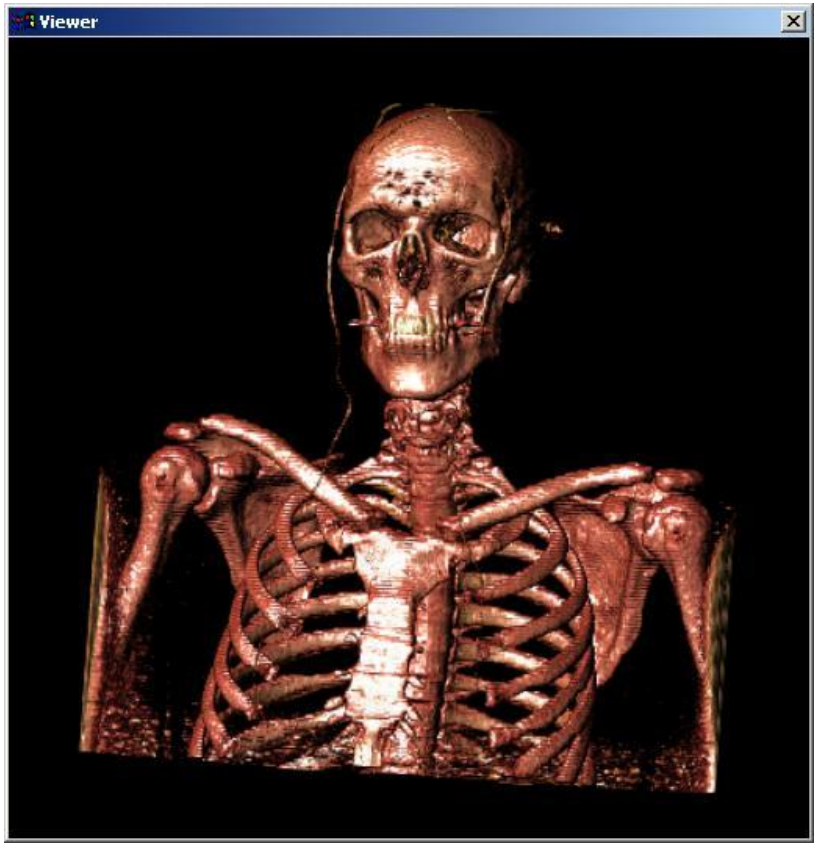


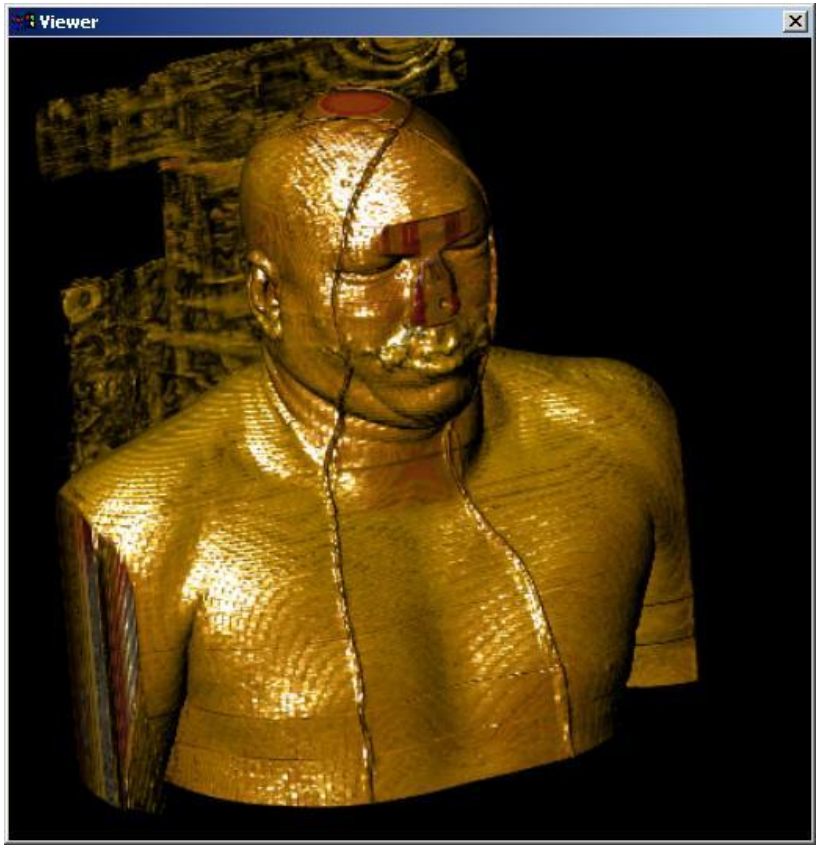




- Data Size : 512^3
- Performance 4.01fps

Shaded Images of Visible Human Male Data Set

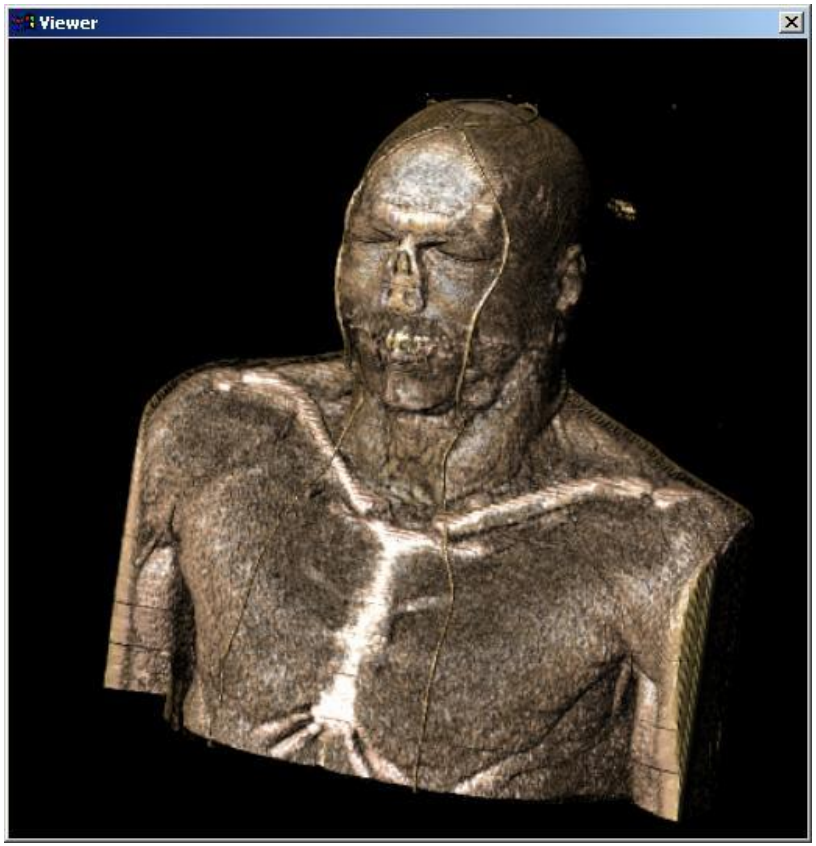


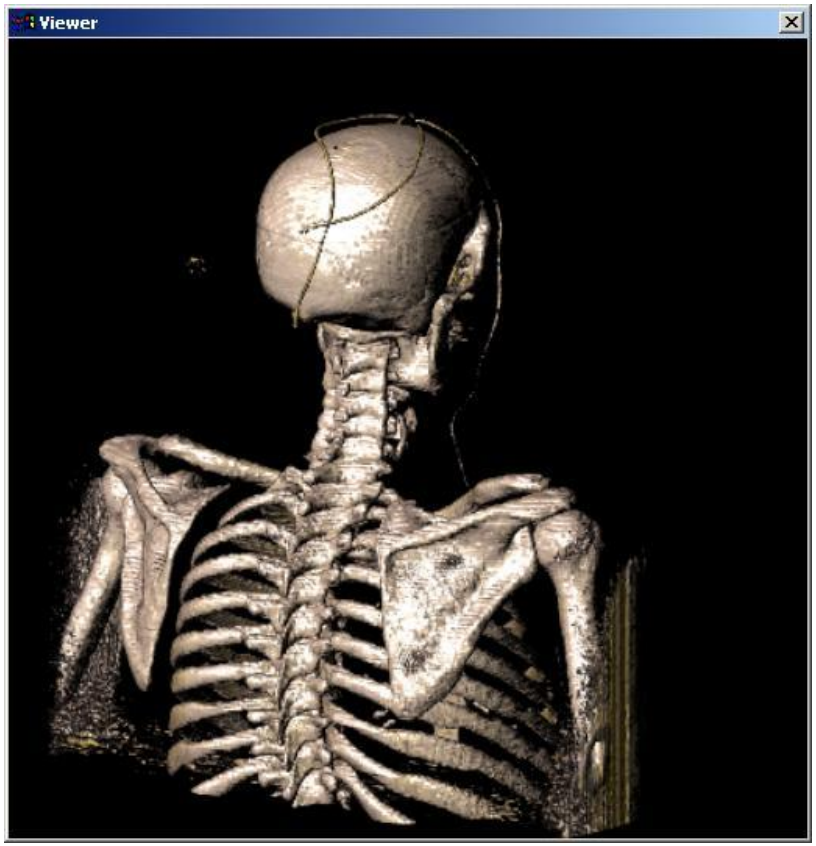


Visualization of bones and skin

Data size : 512^3

Shaded Images of Visible Human Male Data Set

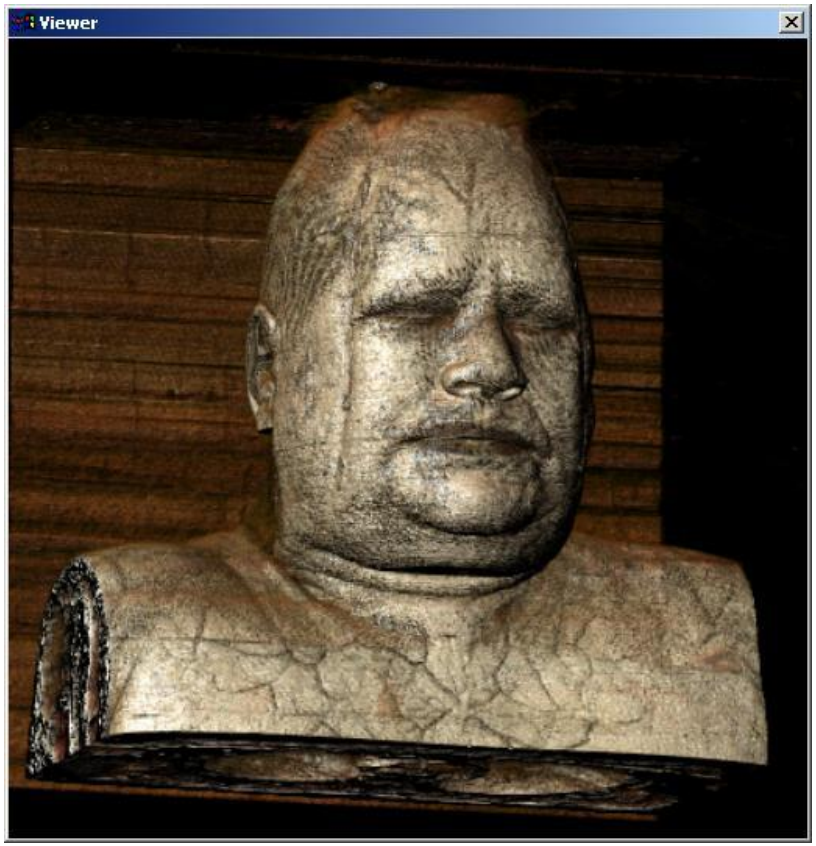


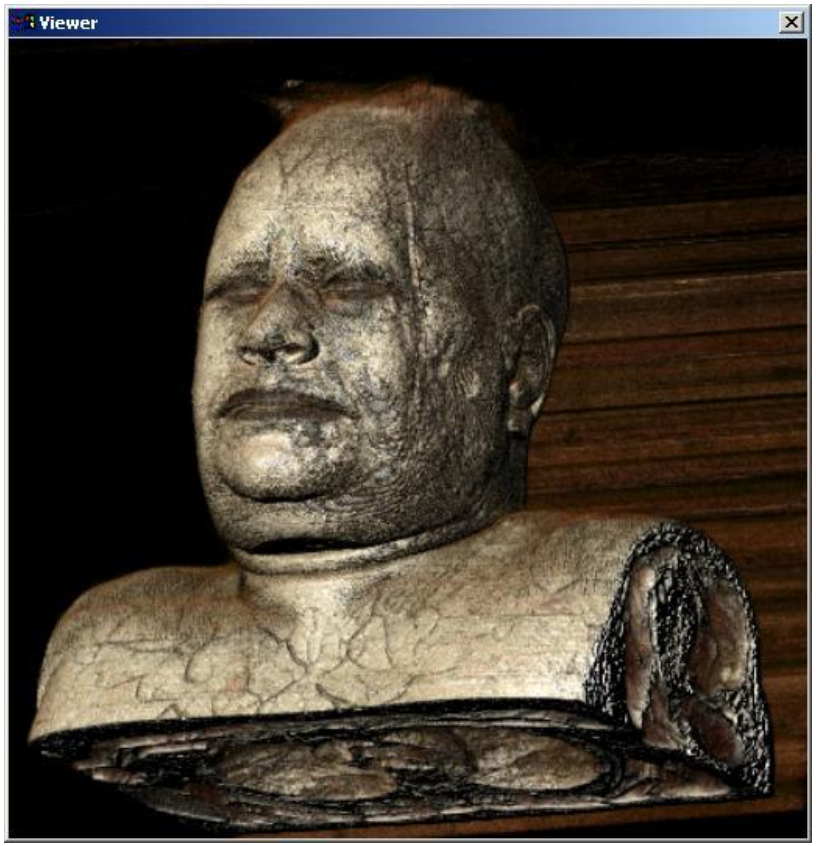


Visualization of muscles and bones

Data size : 512^3

Shaded Images of Visible Human Female Data Set

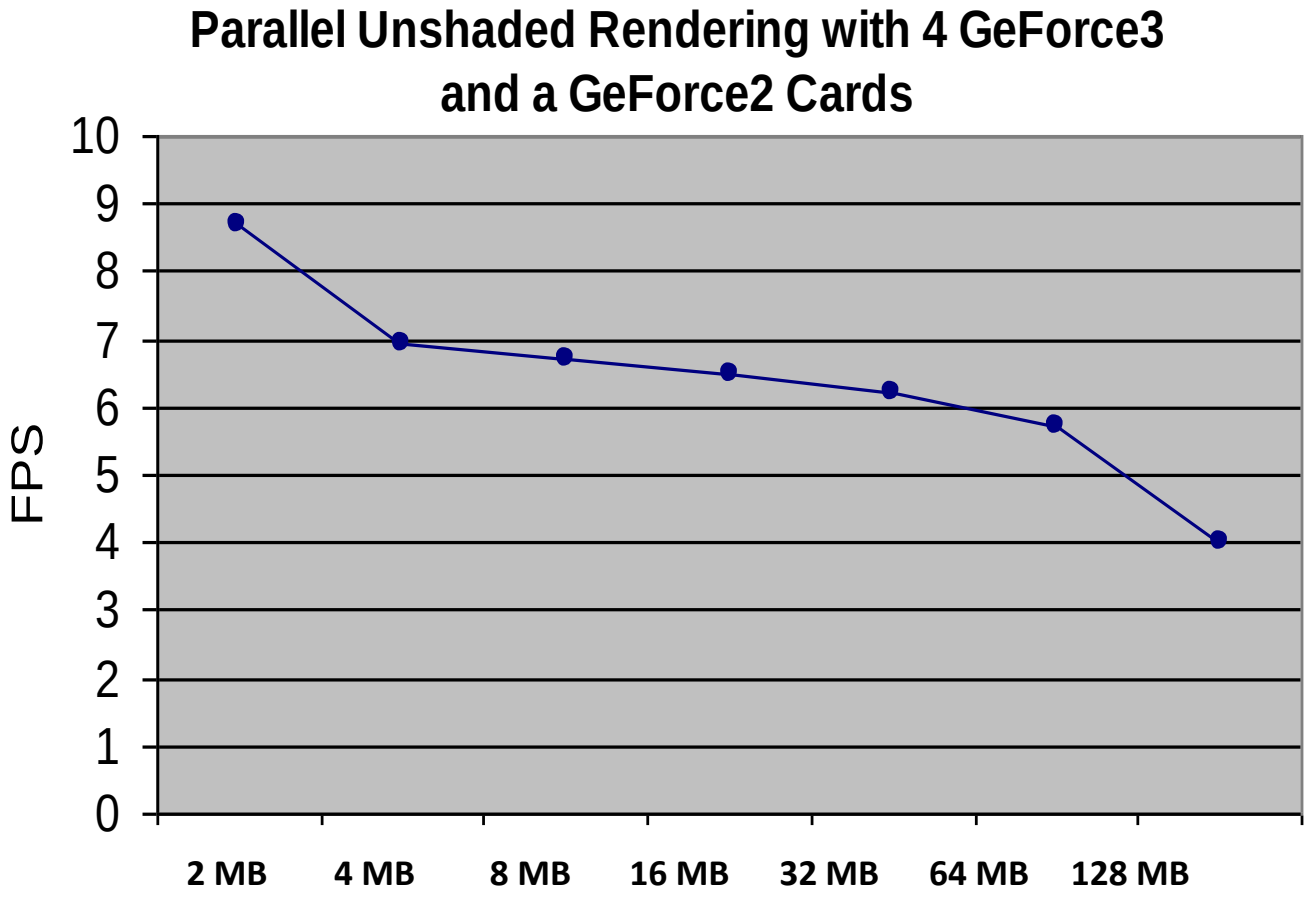




Visualization of skin

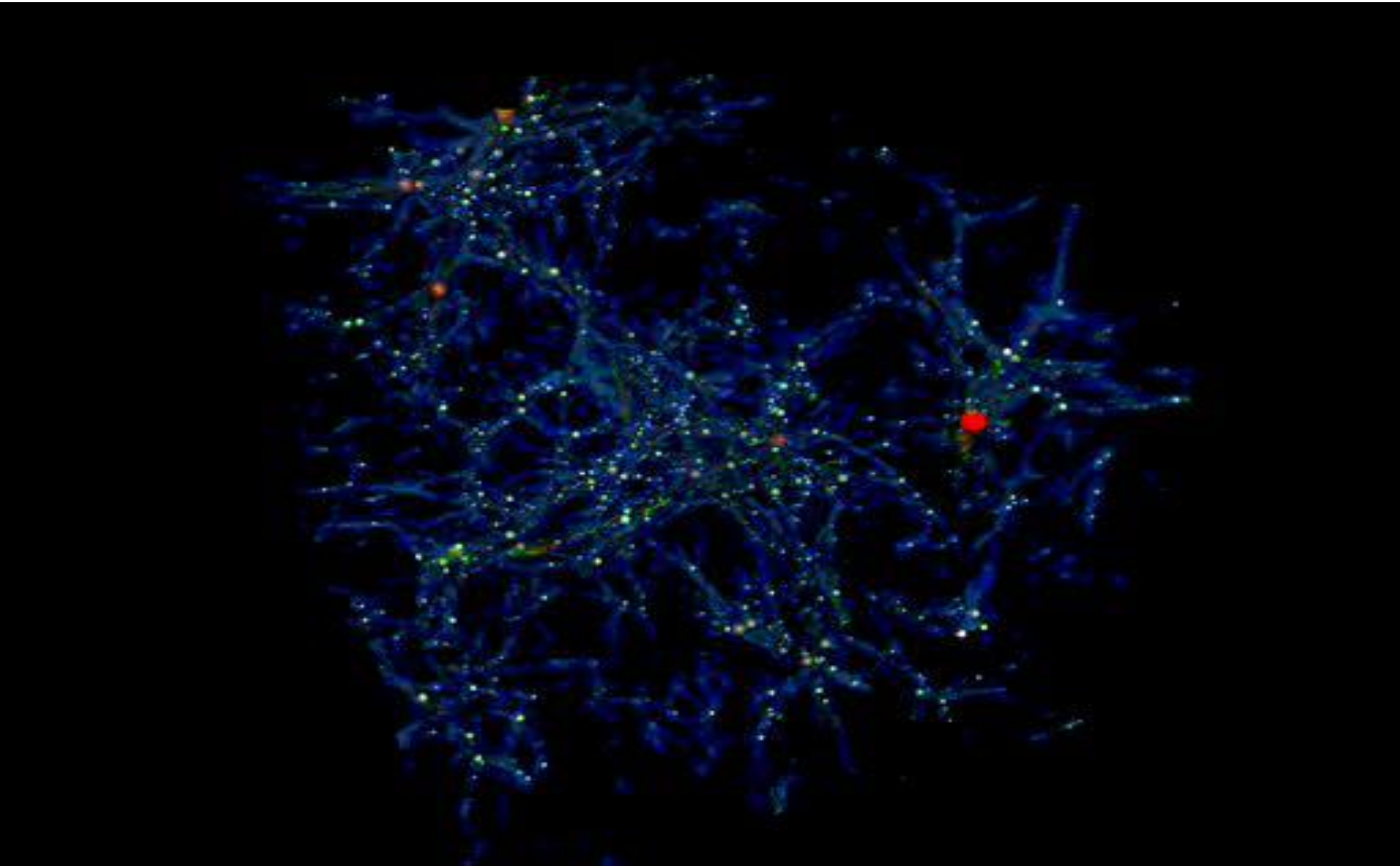
Data size : 512^3

Performance



X	*Data Size (MB)*	*FPS*
1	2	8.68
2	4	6.92
3	8	6.73
4	16	6.47
5	32	6.22
6	64	5.74
7	128	4.01

Mini-Halos Simulation



Optical Models

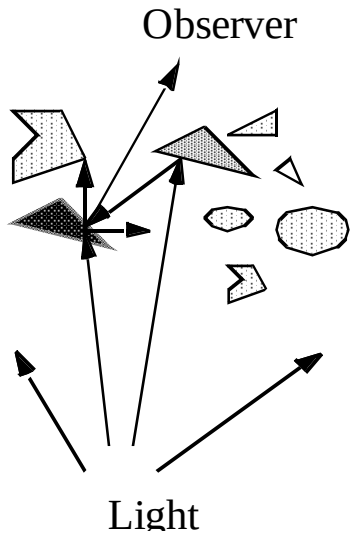
- Jim Blinn's 1982 SIGGRAPH paper on light scattering
- Nelson Max, "Optical Models", IEEE Transactions on Visualization and Computer Graphics, Vol. 1, No. 2, 1995.

- The mathematical framework for light transport in volume rendering based on

  S. Chandrasekhar "Radiative Transfer", Oxford Universtiy Press, 1950

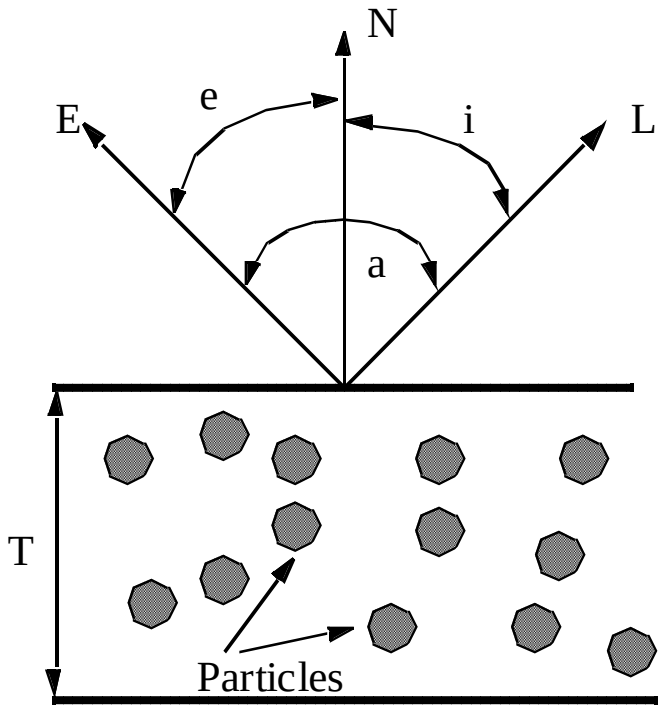
Transport of Light

- Determination of Intensity
- Local - Diffuse and Specular
- Global - Radiosity, Ray Tracing
- Mechanisms in Ultimate Model
  - Emittance
  - Absorption
  - Scattering (single vs. multiple)



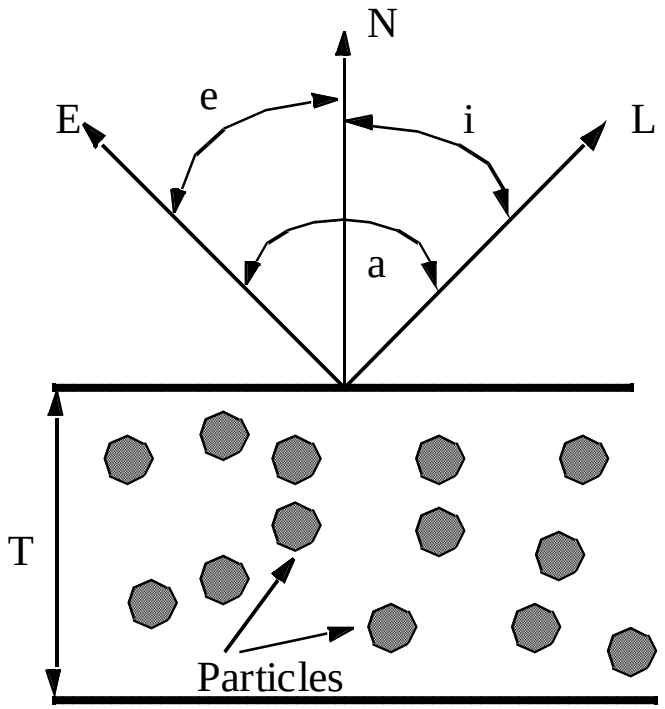
Blinn gaseous model- 1982

- Assumptions:
  - *N* - surface normal
  - *E* - eye vector
  - *L* - light vector
  - *T* - surface thickness
  - *e* - angle btw. *E* and *N*
  - *a* - angle btw. *E* and *L* aka phase angle
  - i - angle btw. *N* and *L*



Blinn model (contd.)

- Assumptions (contd.):
  - particles are little spheres with radius *p*
  - *n* - number density (number of particles per unit volume)
  - μ - cosine of angle e, (*N.E)*
  - *D* - proportional volume of the object occupied by particles



$$D = n\frac{4}{3}\pi p^3$$

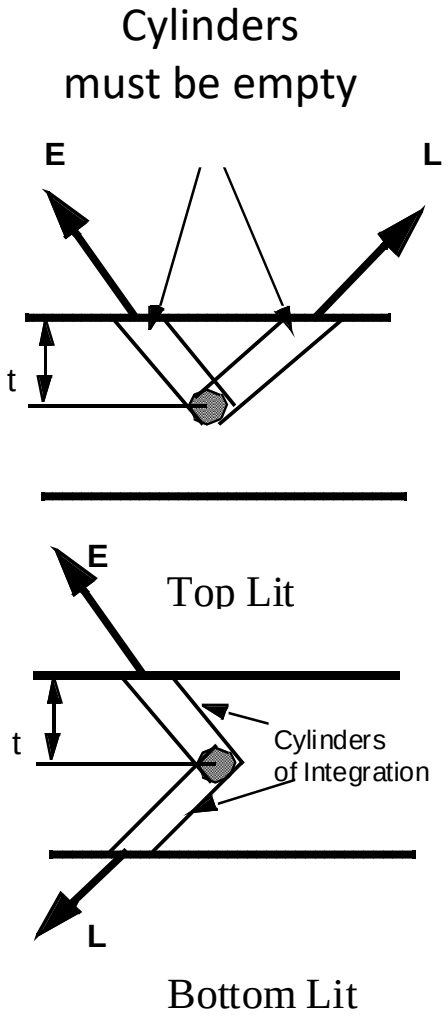
Blinn model – transparency (1)

- Expected particles in a volume will be *nV*
- Probability that there are no particles in the way can be modeled as a Poisson process:

$$P(0,V) = e^{-nV}$$

- Hence the probability that the light is making it through those tubes is:

$$P(0,V) = e^{-n\pi p^2 T'/\mu_0} e^{-n\pi p^2 T'/\mu}$$



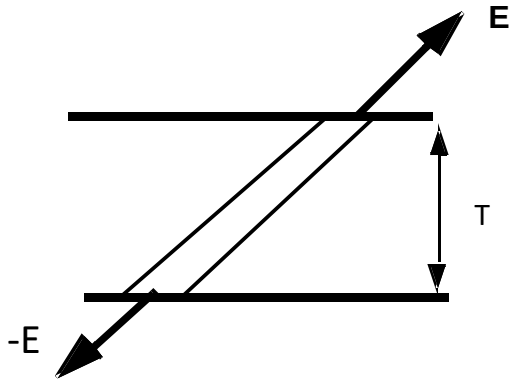
Blinn model – transparency (2)

- Transparency through the medium:

$$Tr = e^{-\tau/\mu}$$

- τ is called the optical depth:

$$\tau = n\pi p^2 T$$



Max model - 1995

- Several cases:
  - Completely opaque or transparent voxels
  - Variable opacity correction
  - Self-emitting glow
  - Self-emitting glow with opacity along viewing ray
  - Single scattering of external illumination
  - Multiple scattering

Max model - absorption only

- $I(s)$ = intensity at distance s along a ray
- $\tau(s)$ = extinction coefficient

$$\frac{dI}{ds} = -\tau(s)I(s)$$

$$I(s) = I(0)\exp\left(-\int_0^s \tau(t)dt\right)$$

$$= I_0 T(s)$$

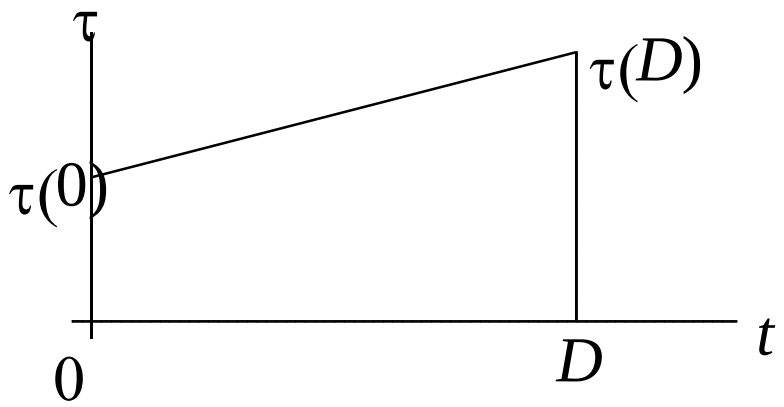
- $T(s)$ = transparency between 0 and s

Max - absorption only

- Linear variation of τ:

$$T(s) = \exp\left(-\int_0^D \tau(t)\,dt \right)$$

$$= \exp\left(-D\,\frac{\tau(0) + \tau(D)}{2} \right)$$



Max model - absorption only

- On the opacity α:

$$\alpha = 1 - T(s) = 1 - \exp\left(- \int_0^D \tau(t) dt \right)$$

$$= 1 - \exp(-\tau D)$$

$$= \tau D - (\tau D)^2 / 2 + \ldots$$

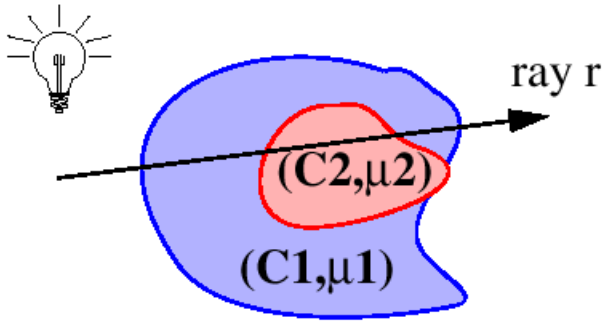
- assuming τ to be constant in the interval

Volume Ray Integration (1)

- The continuous form:

$$I(D) = I_0 \exp\left(-\int_0^D \tau(t)dt \right) + \int_0^D g(s) \exp\left(-\int_s^D \tau(t)dt \right) ds$$

- In general , cannot compute analytically



Volume Ray Integration (2)

- Practical Computation Method:

$$I(D) = I_0 \exp\left(-\int_0^D \tau(t)dt \right) + \int_0^D g(s)\exp\left(-\int_s^D \tau(t)dt \right) ds$$

$$t_i = \exp\left(-\tau(i\Delta x)\Delta x\right) \approx 1 - \tau(i\Delta x)\Delta x$$

$$I(D) = I_0 \prod_{i=1}^{n} t_i + \sum_{i=1}^{n} \left(\prod_{j=i+1}^{n} t_j \right) g_i$$

$$= g_n + t_n\left(g_{n-1} + t_{n-1}\left(g_{n-2} + \ldots\left(g_1 + t_1 I_0\right)\ldots\right)\right)$$

which leads to the familiar BTF or FTB compositing

$g(s)$

- $g(s)$ could be:
  - Self-emitting particle glow
  - Reflected color, obtained via illumination

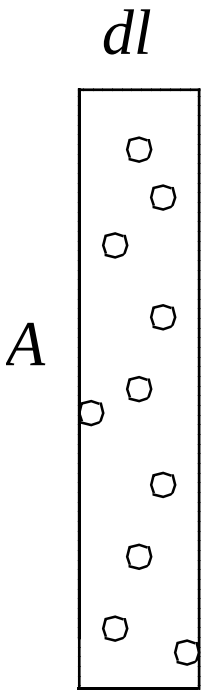
- The color is usually the sum of emitted color E and reflected color R

Max - self-emitting glow

- Identical glowing spherical particles:
- projected area $a = \pi r^2$
- surface glow color = C
- number per unit volume = N

$$\frac{\text{occluded area}}{\text{total area}} = \frac{aNAdl}{A}$$

- extinction coefficient $\tau = aN$
- added glow intensity per unit length $g = CaN = C\tau$



Max - self-emitting glow

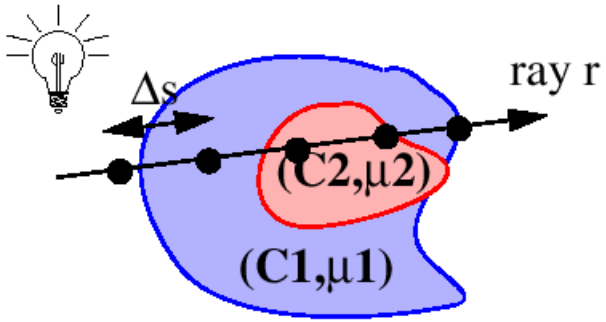
- Special Case $g=C\tau$: (and C constant)

$$\int_0^D g(s)\exp\left(-\int_s^D \tau(t)dt\right)ds = \int_0^D C\tau(s)\exp\left(-\int_s^D \tau(t)dt\right)ds$$

$$= C\left(1-\exp\left(-\int_0^D \tau(t)dt\right)\right)$$

$$I(D) = I_0 T(D) + C(1-T(D))$$

- This is compositing color C on top of background I_0

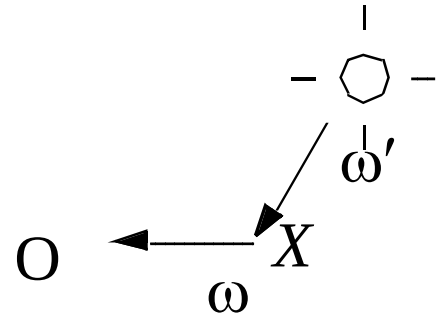


Max - self-emitting glow

- For I_0=0 and τ: varying according to f :

Max - reflection

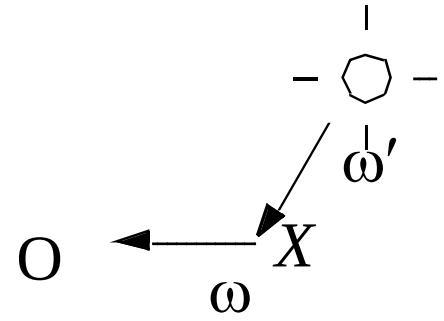
$$g(x) = r(x, \omega, \omega')i(x)$$



- $i(x)$ = illumination reaching point x
- ω = unit reflection direction vector
- ω' = unit illumination direction vector
- $r(x,\omega,\omega')$: BRDF

  $|\nabla f(x) \cdot \omega|$ for conventional surface shading effects

Max - reflection



- For particle densities:

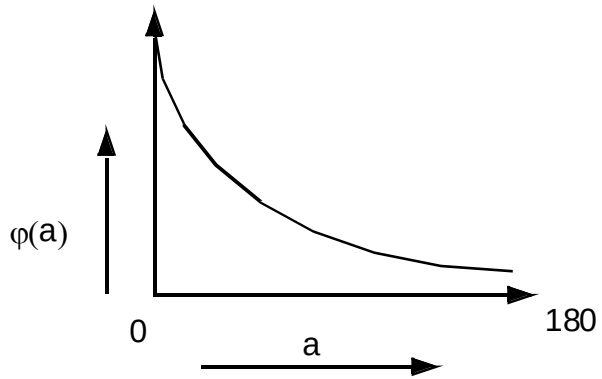
$$r(x,\omega,\omega') = w(x)\tau(x)p(\omega,\omega')$$

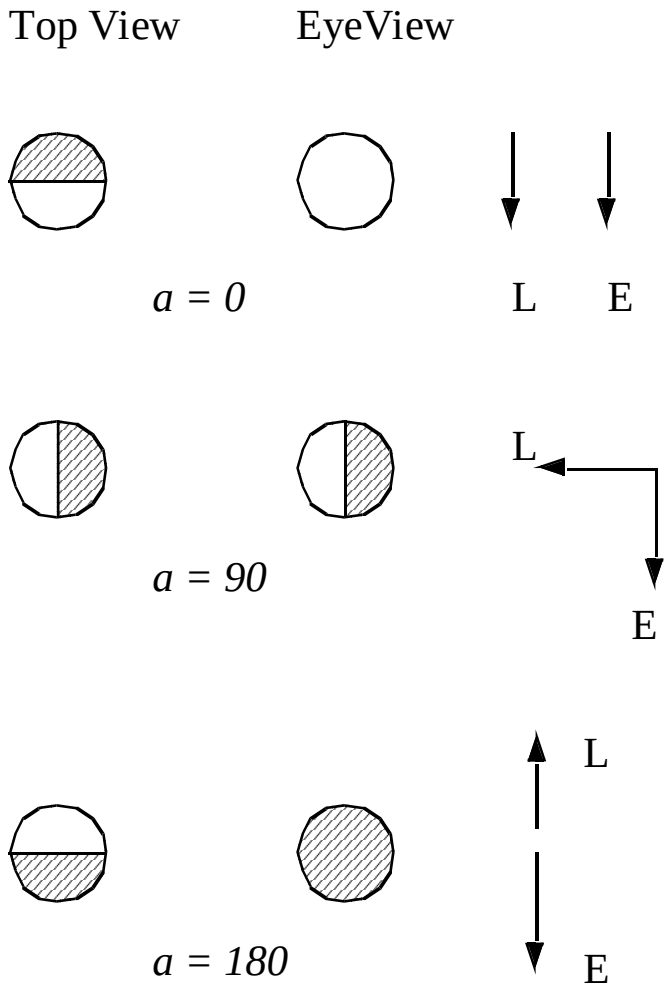
  - $w(x)$ = albedo
    - Blinn: assumes that the primary effect is from interaction of light with one **single** particle
    - albedo - proportion of light reflected from a particle: in the range of 0..1
  - $p(\omega,\omega')$ = phase function

- still unrealistic external reflection of outside illumination

Blinn - Phase Function

- "how" we see the particles
- depends on the angle of eye *E* and light vector *L*
- smooth drop off …





Blinn - Phase Function

- Many different models possible
- Constant function $\varphi(a) = 1$
  - size of particles much less than wavelength of visible light
- Anisotropic $\varphi(a) = 1 + x\cos(a)$
  - more light forward then backward - essentially our diffuse shading
- Lambert surfaces $\varphi(a) = 8\pi/3\,(\sin(a) + (\pi - a)\cos(a))$
  - spheres reflect according to Lambert's law
  - physically based

Blinn - Phase Function

- Rayleigh Scattering $\varphi(a) = 3/4\left(1+\cos^2(a)\right)$
  - diffraction effects dominate
- Henyey-Greenstein $\varphi(a) = \left(1-g^2\right)/\left(1+g^2-2g\cos(a)\right)^{3/2}$
  - general model with good fit to empirical data
- Empirical Measurments
  - tabulated phase function
- sums of functions
  - weighted sum of functions - model different effects in parallel

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

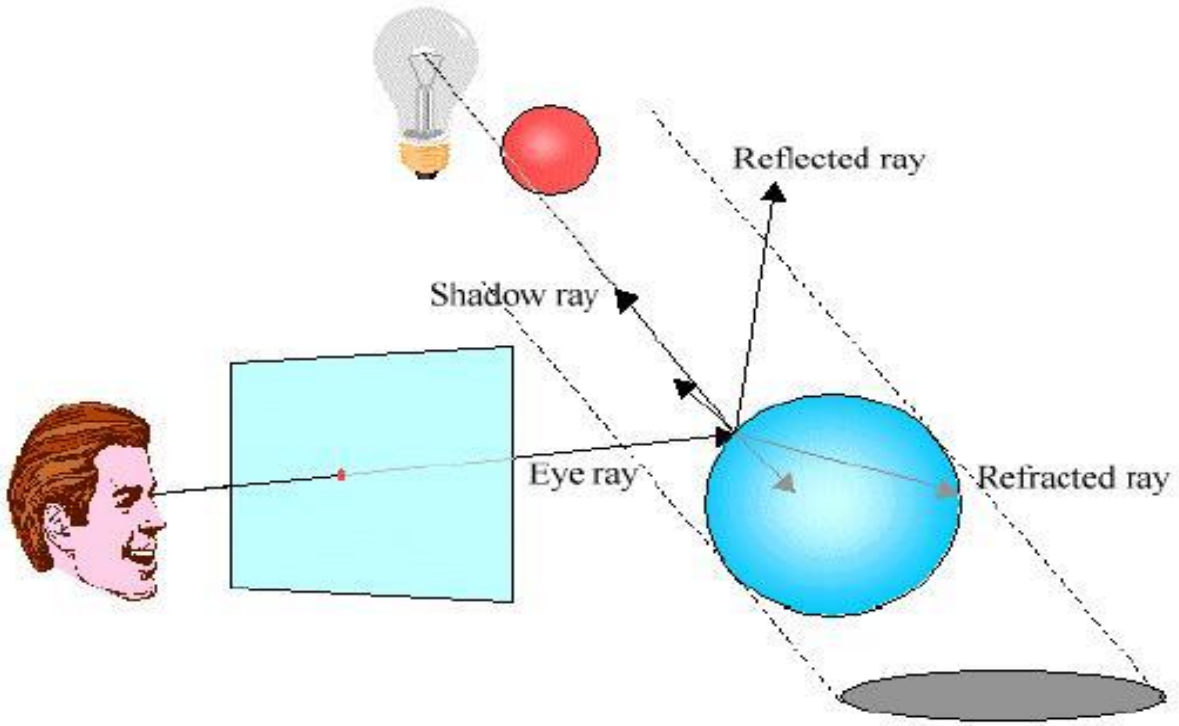
Forward Ray Tracing

- 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

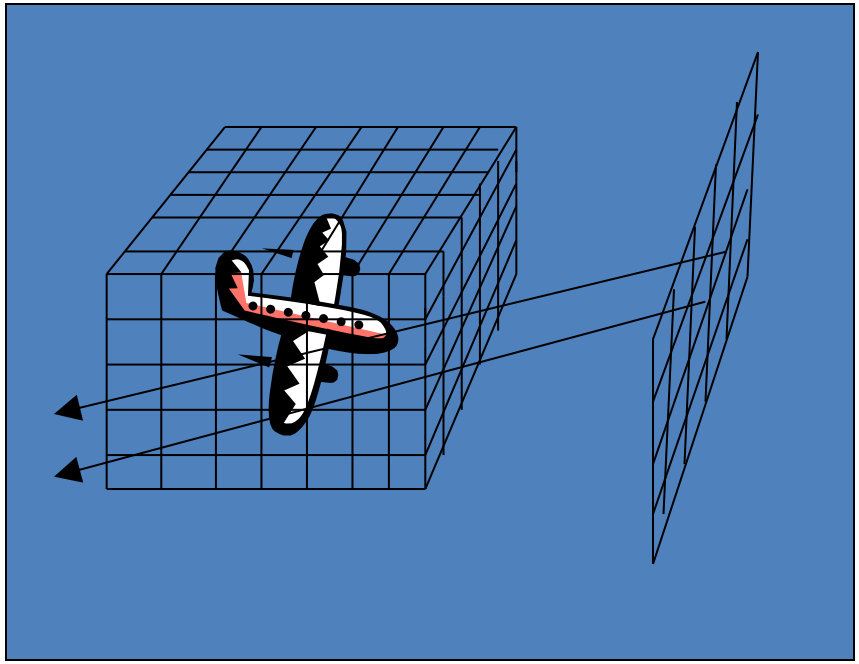


Backward Ray Tracing

- 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

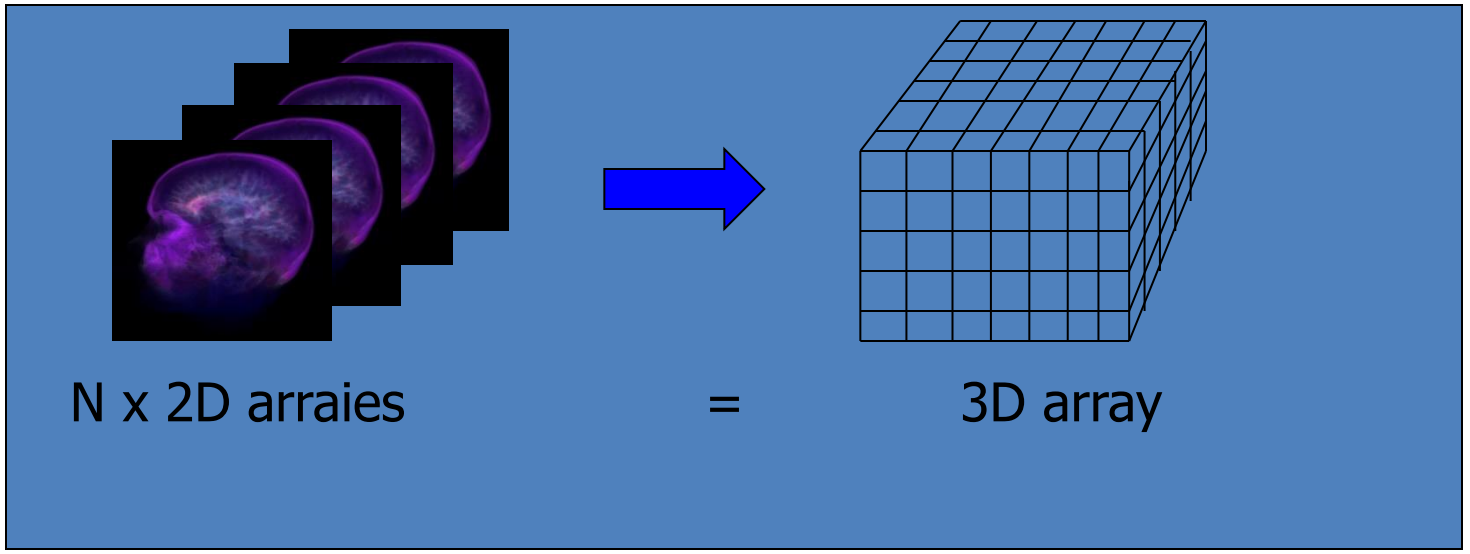
as a ray into object space

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

Data Representation

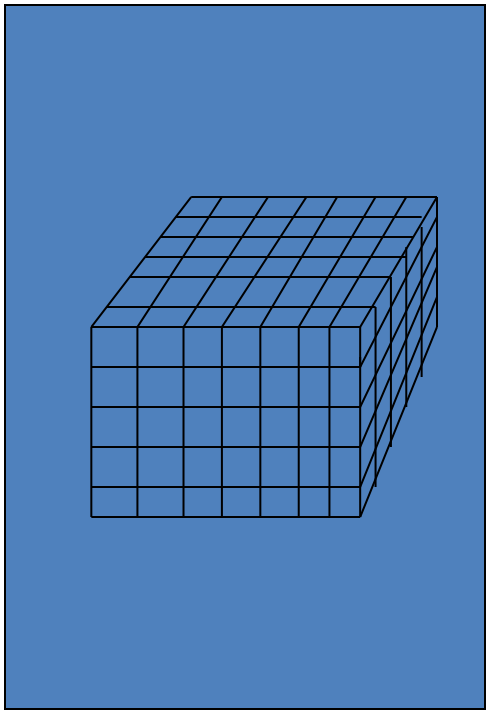
•3D volume data are represented by a finite number of cross sectional slices (hence a 3D raster)

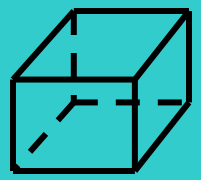
•On each volume element (voxel), stores a data value (if it uses only a single bit, then it is a binary data set. Normally, we see a gray value of 8 to 16 bits on each voxel.)



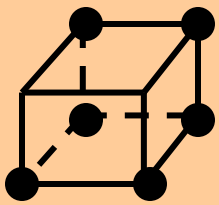
Data Representation (2)

What is a Voxel? – Two definitions





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

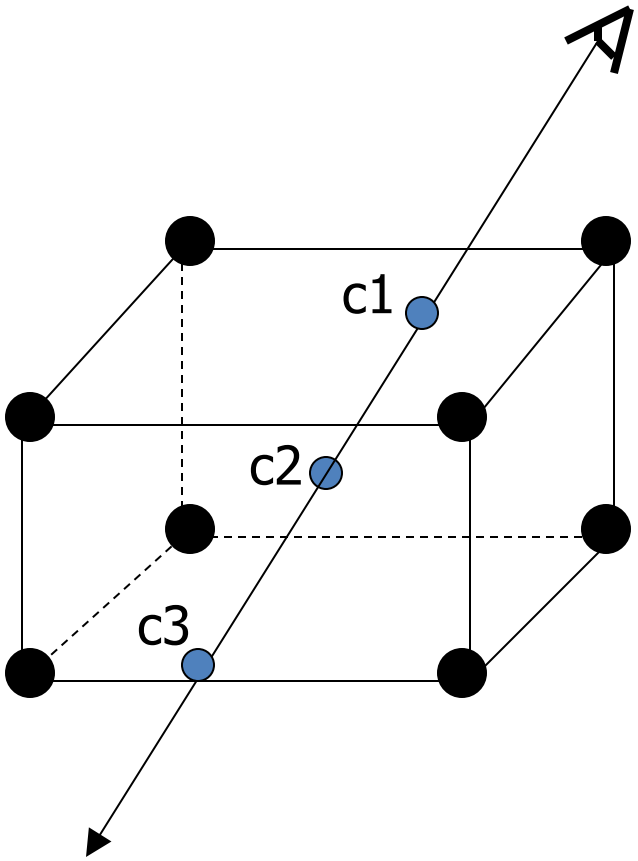


A voxel is a data point at a corner of the cubic cell
The value of a point inside the cell is determined by interpolation

Ray Casting

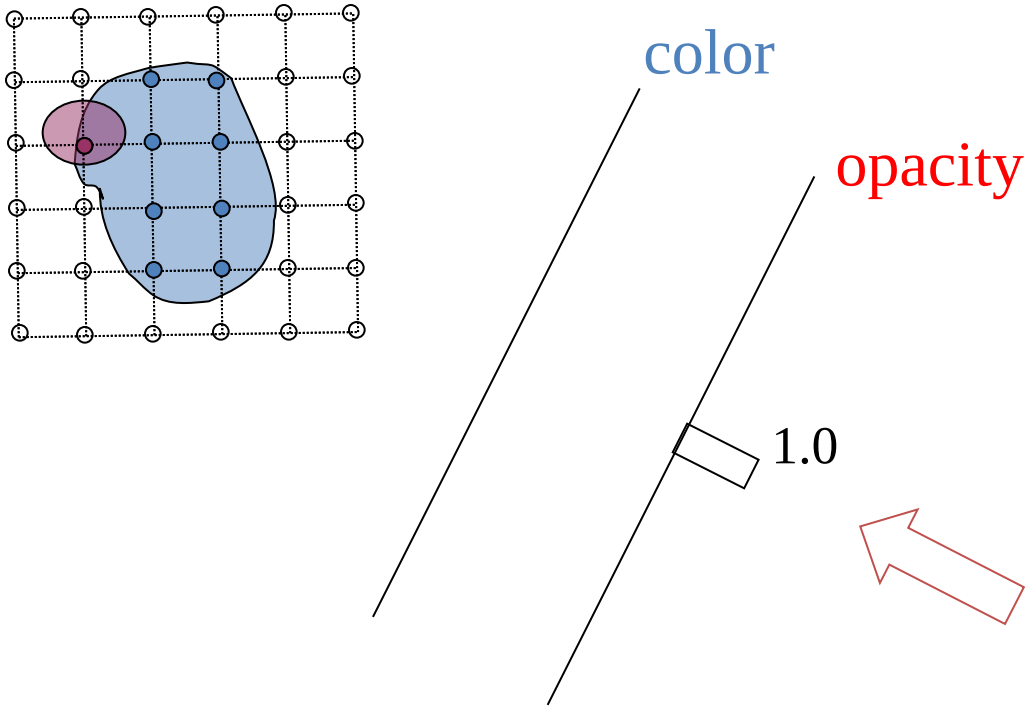
- Stepping through the volume: a ray is cast into the volume, sampling the volume at certain intervals
- The sampling intervals are usually equi-distant, but don’t have to be
- At each sampling location, a sample is interpolated / reconstructed from the grid voxels
- popular filters are: nearest neighbor (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

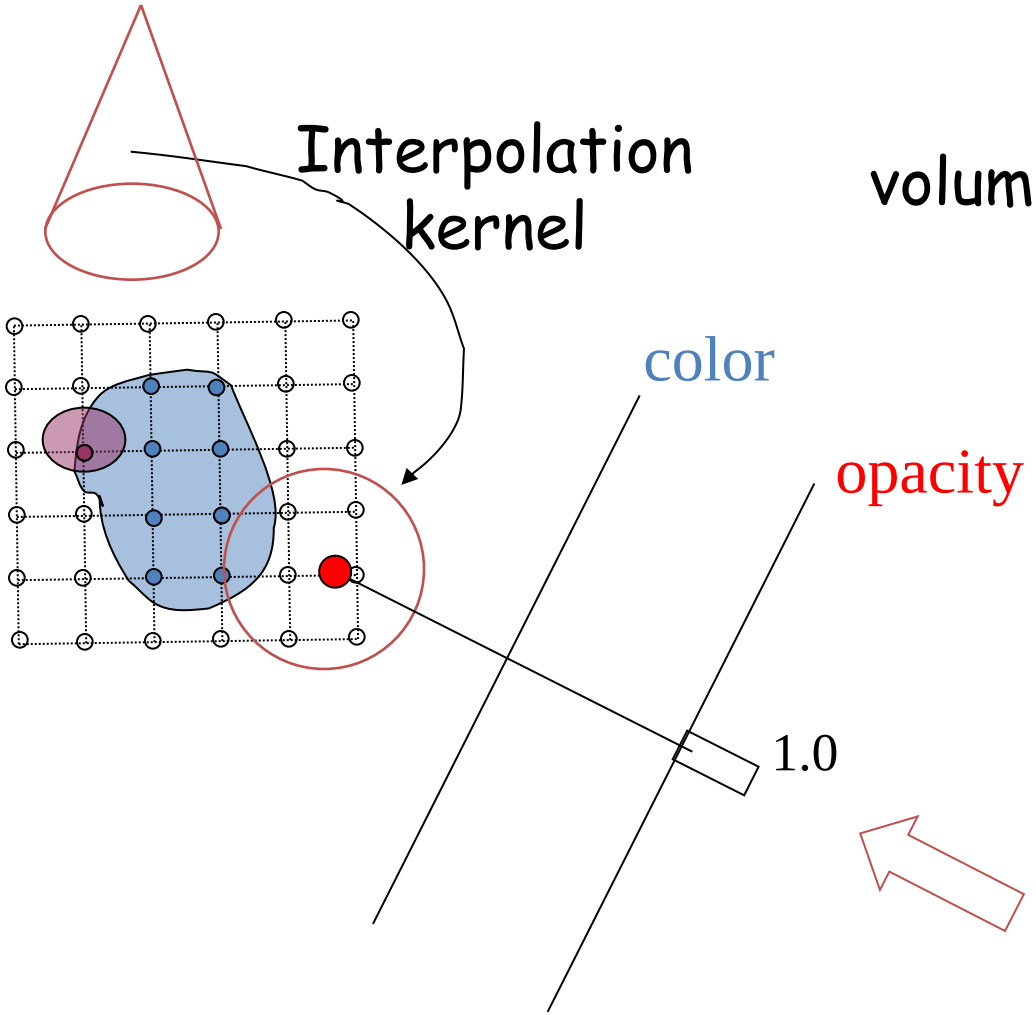


Raycasting

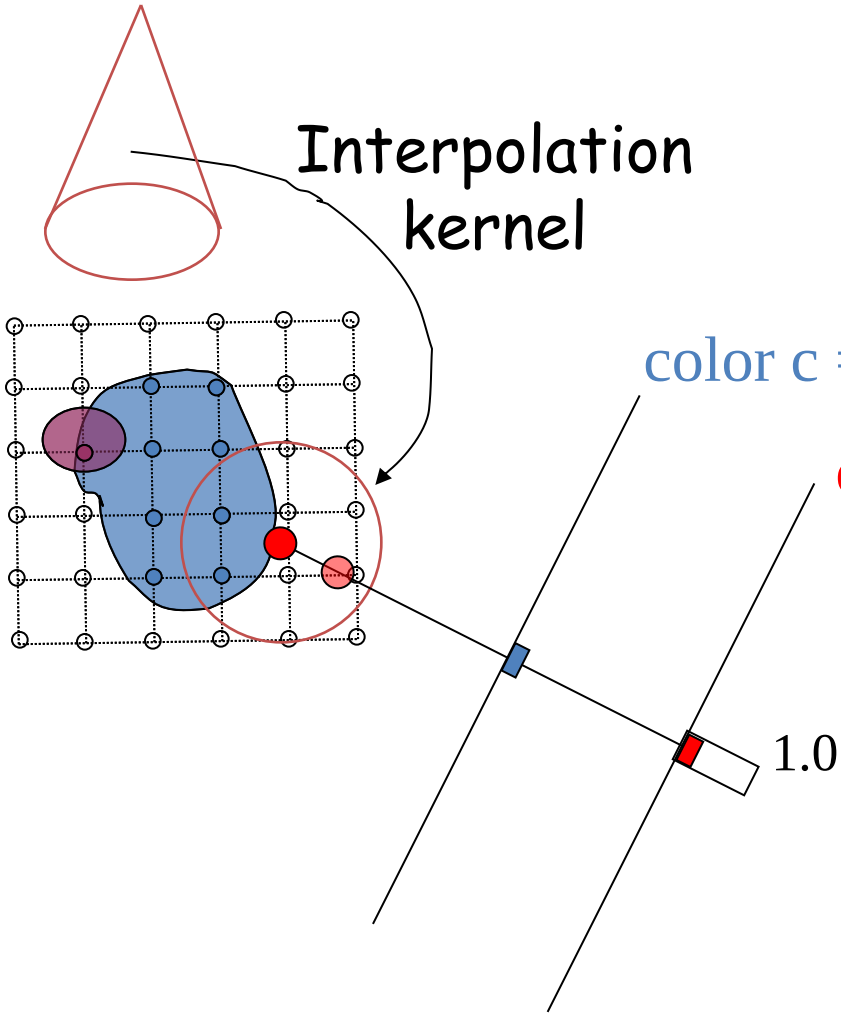
volumetric compositing



Raycasting


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

Raycasting



volumetric compositing

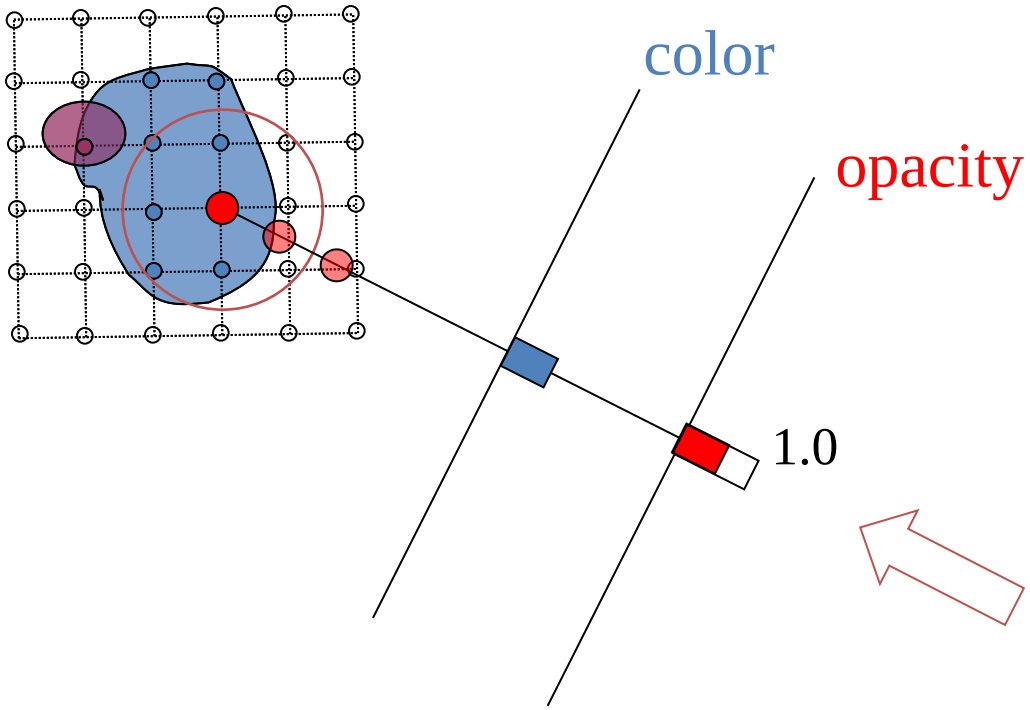
color $c = c_s \alpha_s(1 - \alpha) + c$

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

object (color, opacity)

Raycasting

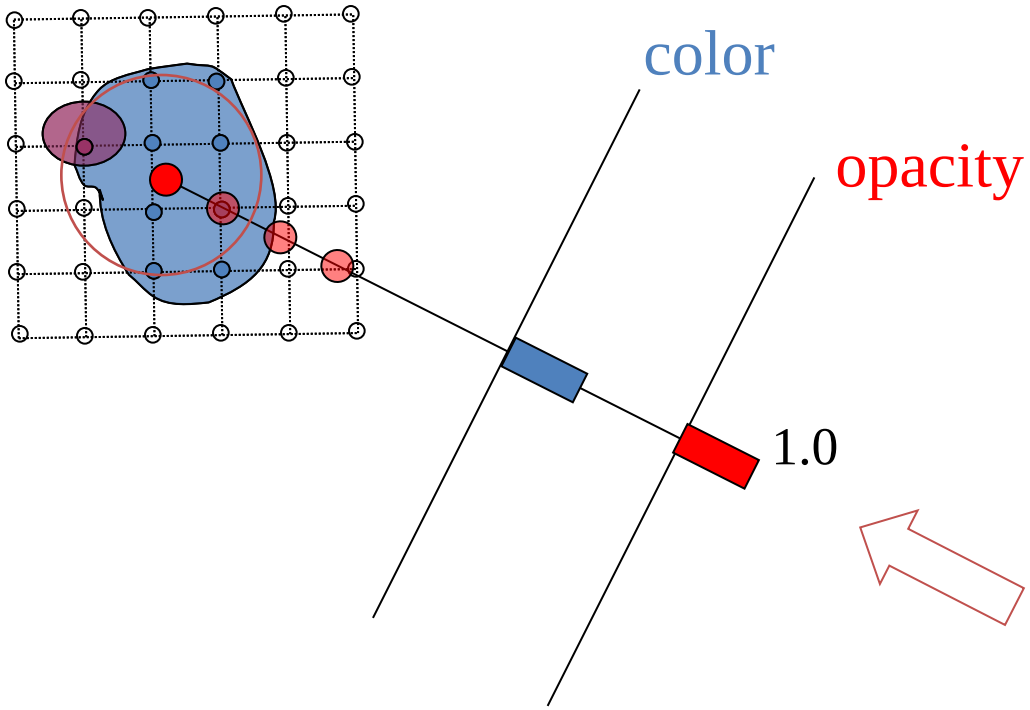
volumetric compositing



object (color, opacity)

Raycasting

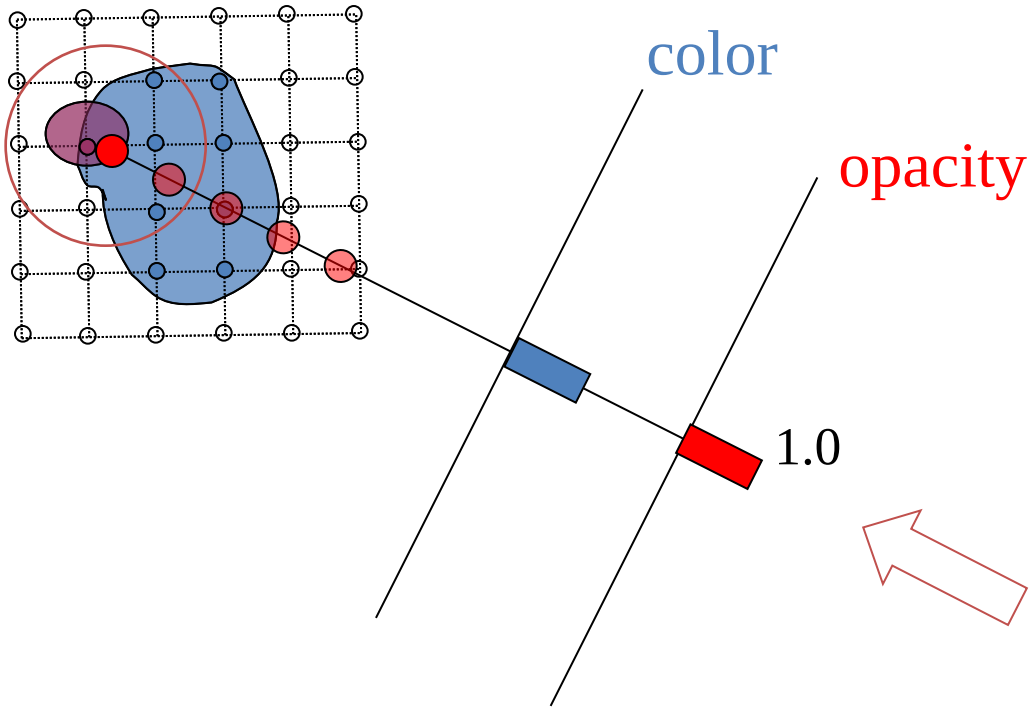
volumetric compositing



object (color, opacity)

Raycasting

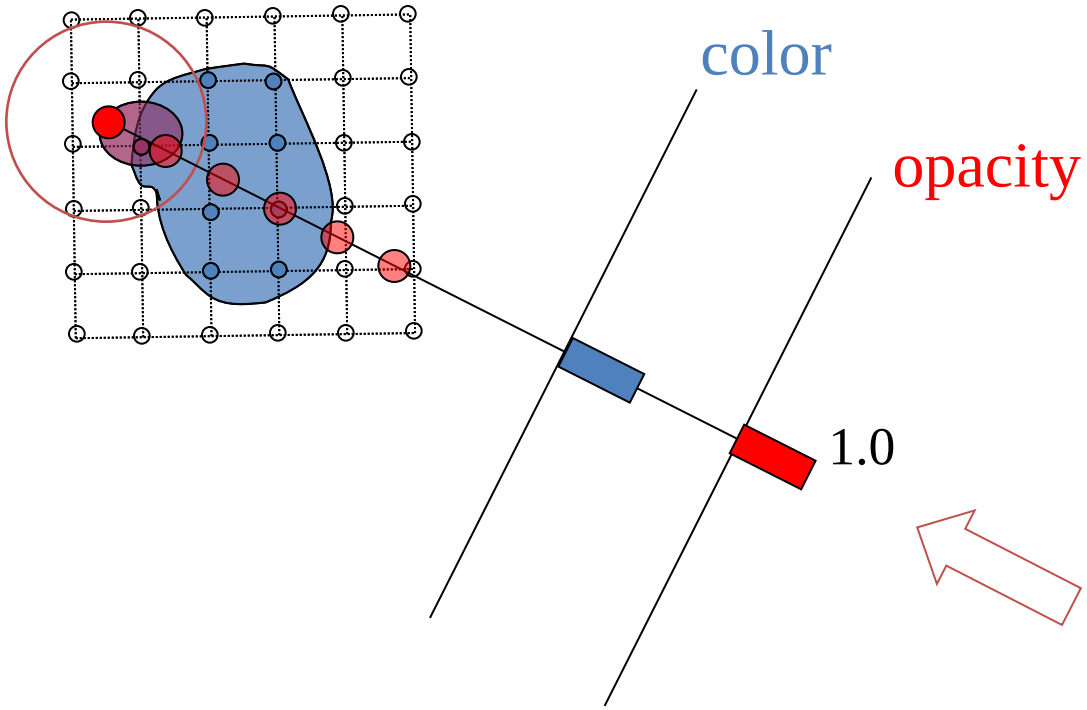
volumetric compositing



object (color, opacity)

Raycasting

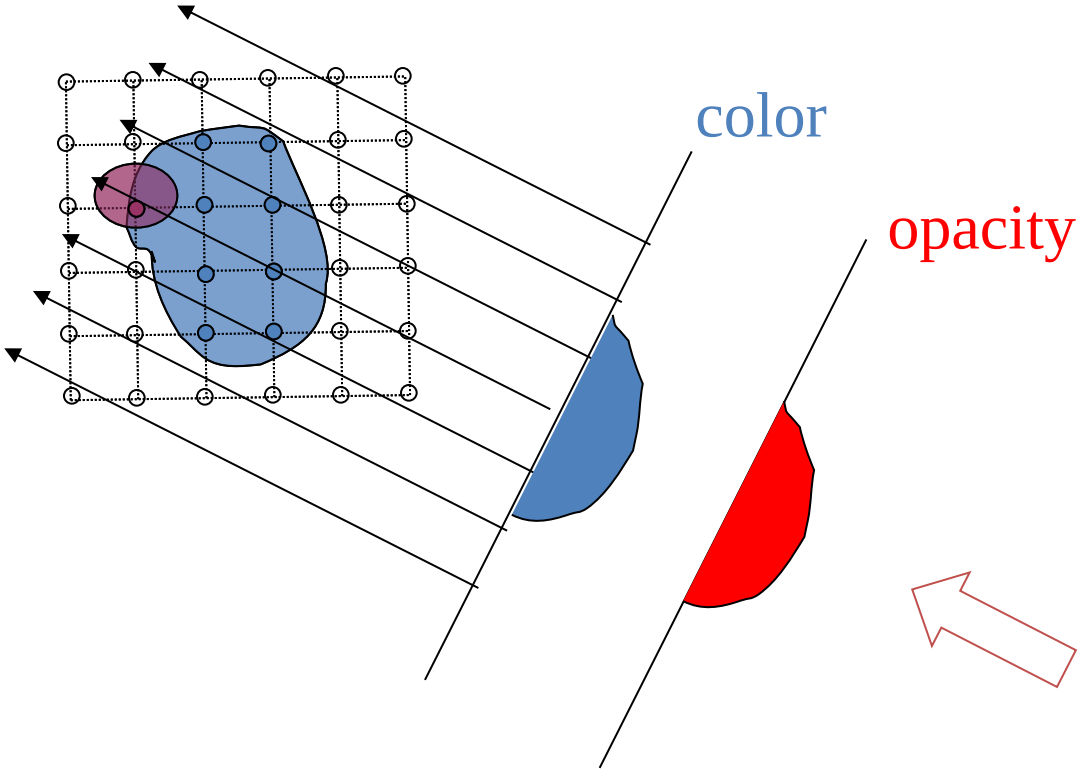
volumetric compositing



object (color, opacity)

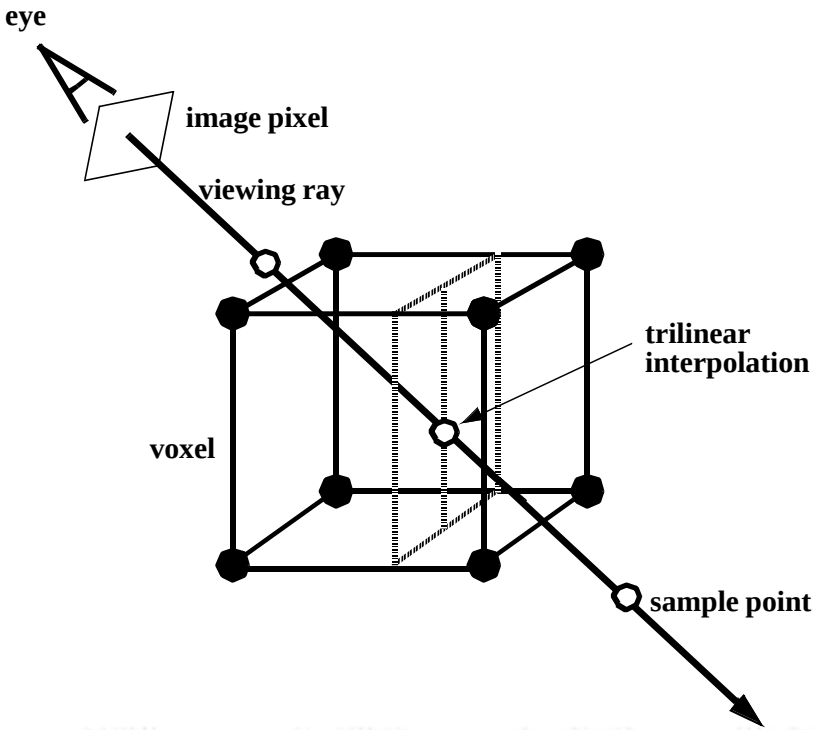
Raycasting

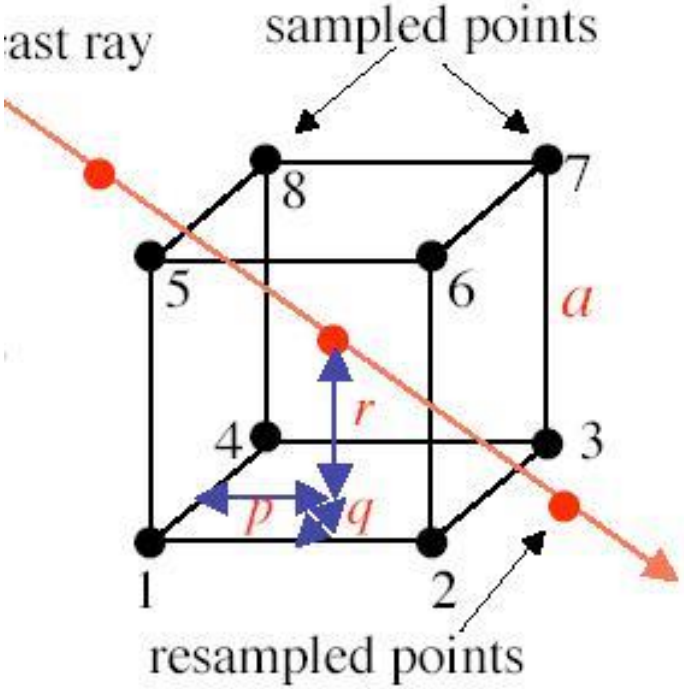
volumetric compositing



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.

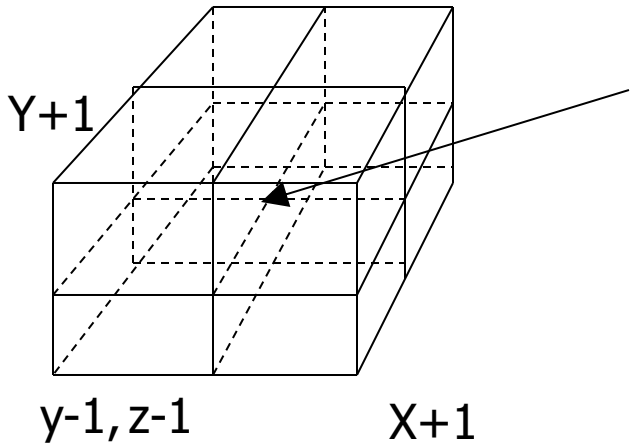
  Raw-data $\rightarrow$ material (R, G, B, α, K_a, K_d, K_s, .)

- 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

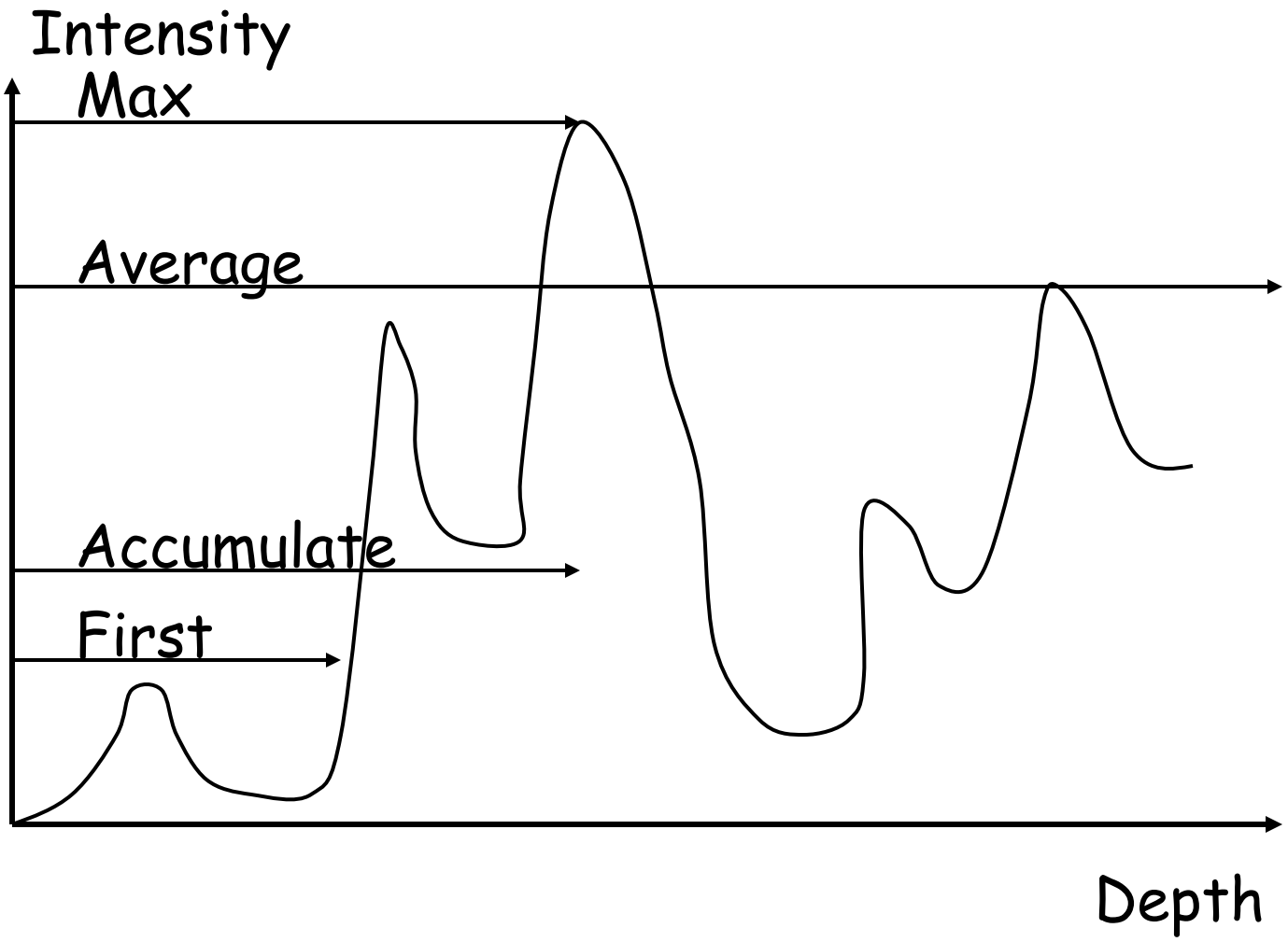
- back-to-front

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

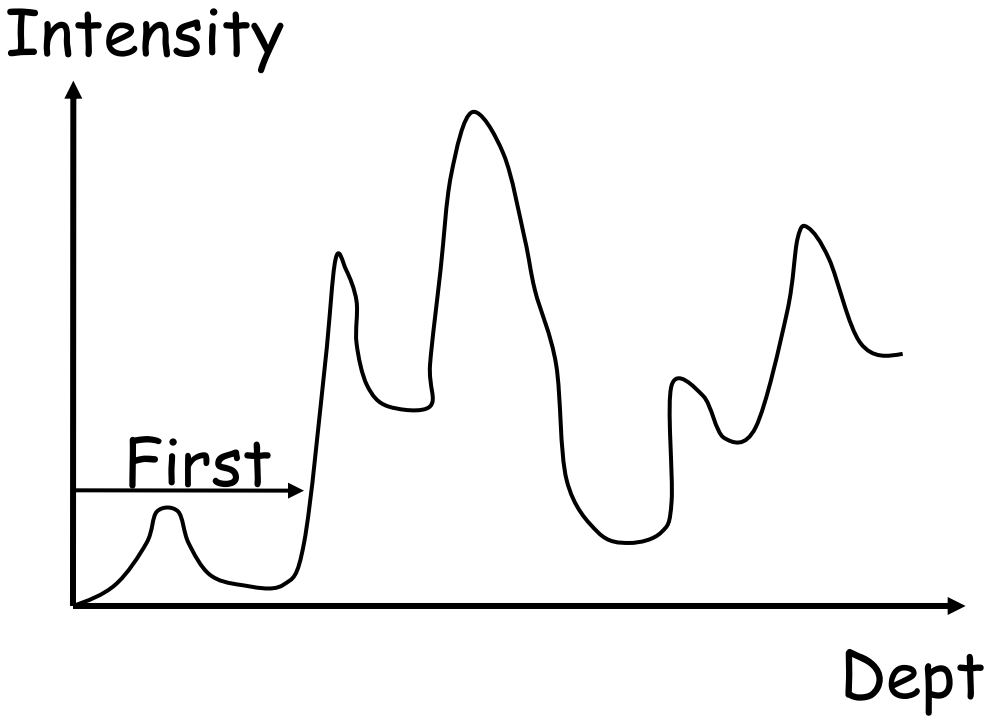
$$\alpha = \alpha \cdot (1 - \alpha_s) + \alpha_s$$

Advantage: object approach suitable for hardware implementation

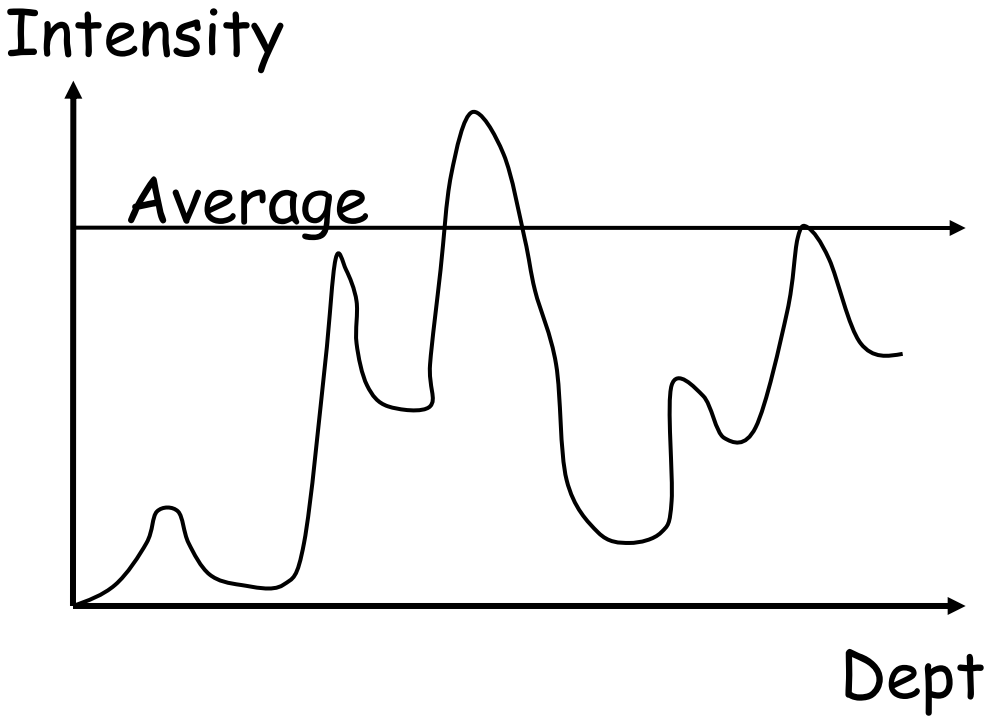
Ray Traversal Schemes

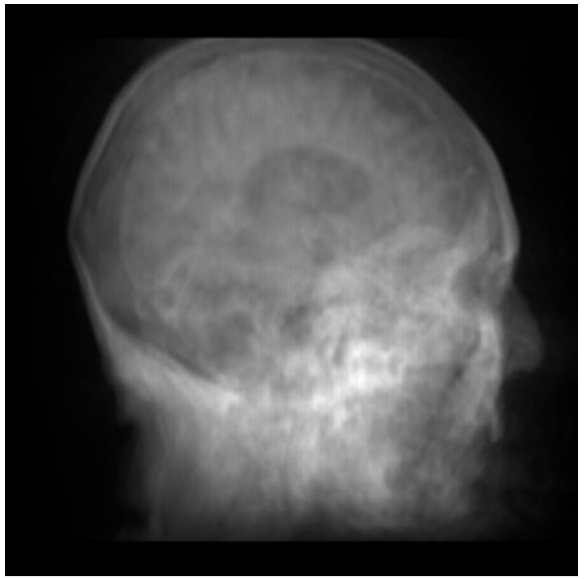


Ray Traversal - First



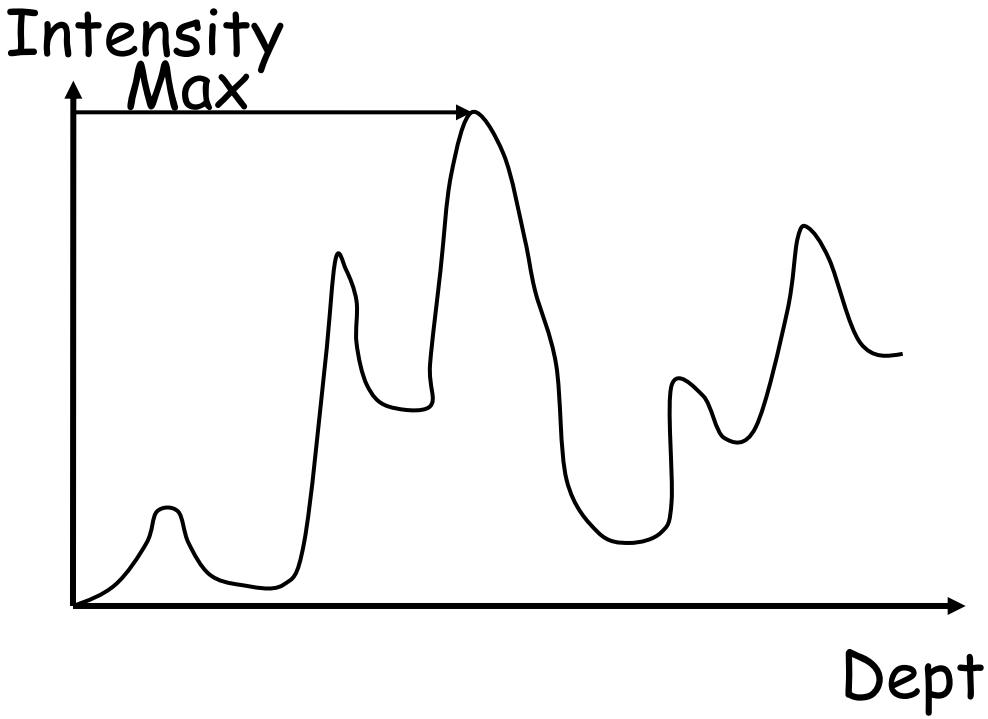
Ray Traversal - Average

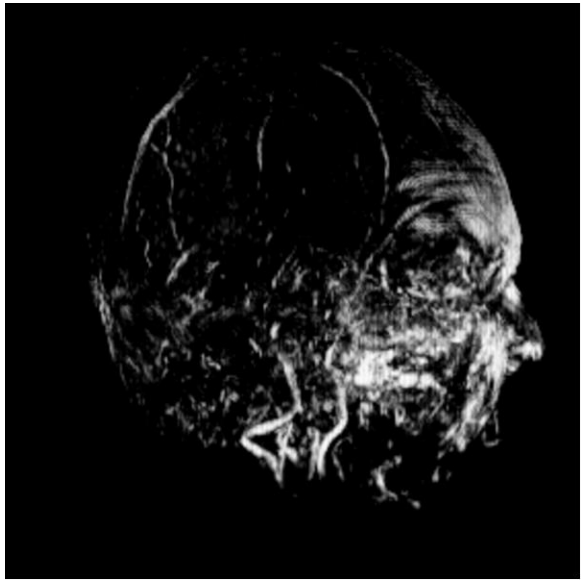




- **Average**: produces basically an X-ray picture

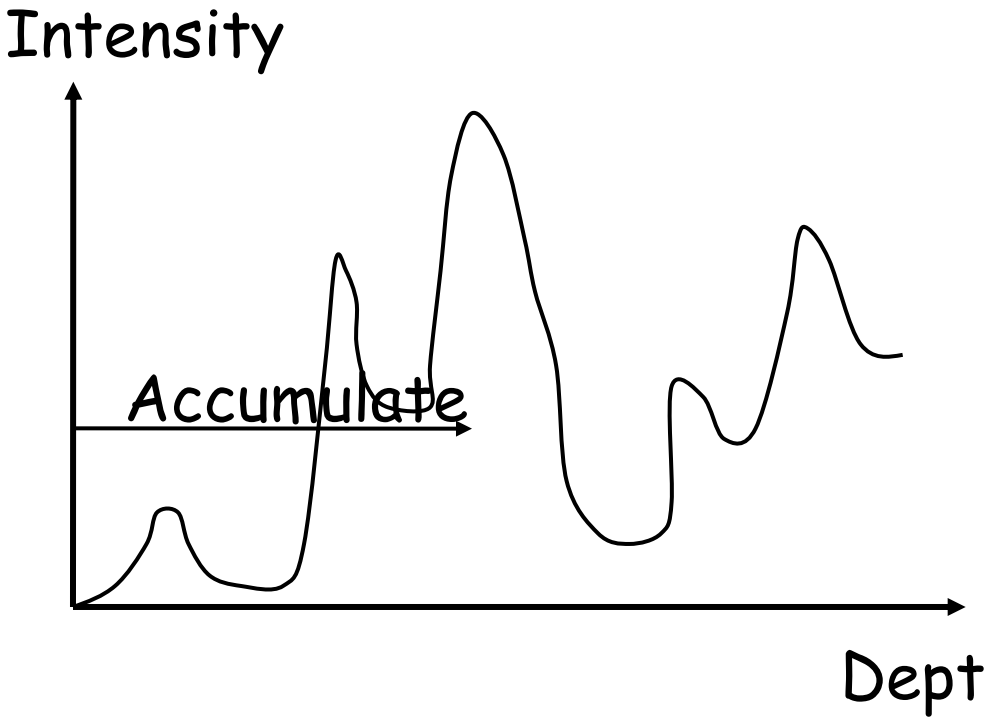
Ray Traversal - MIP

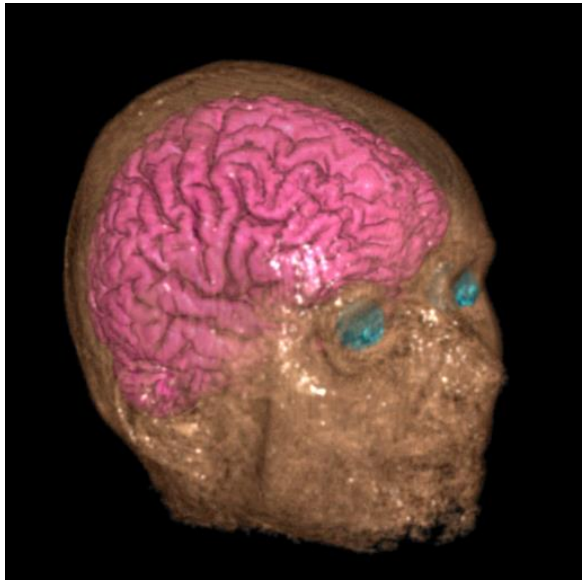




- **Max**: Maximum Intensity Projection used for Magnetic Resonance Angiogram

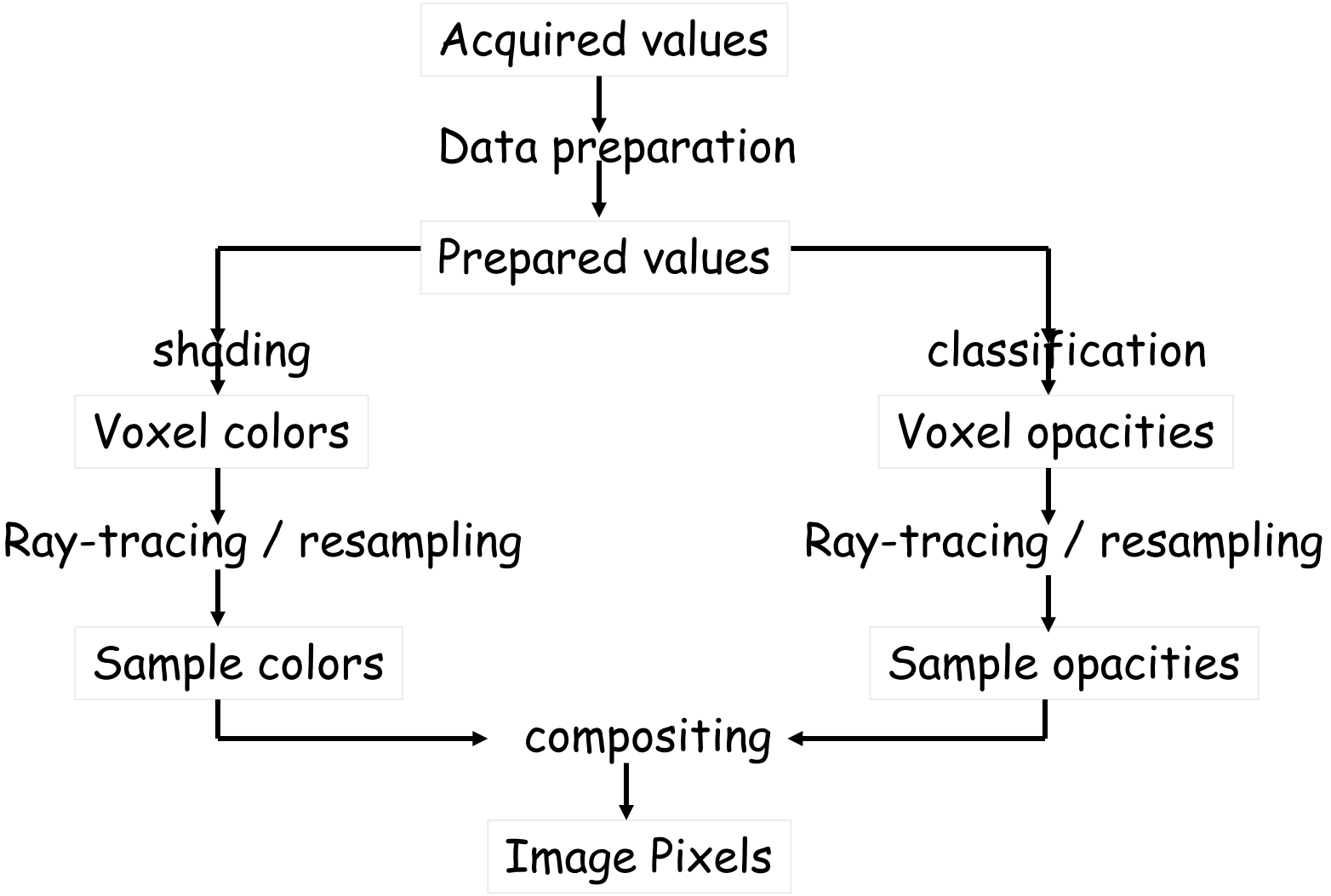
Ray Traversal - Accumulate



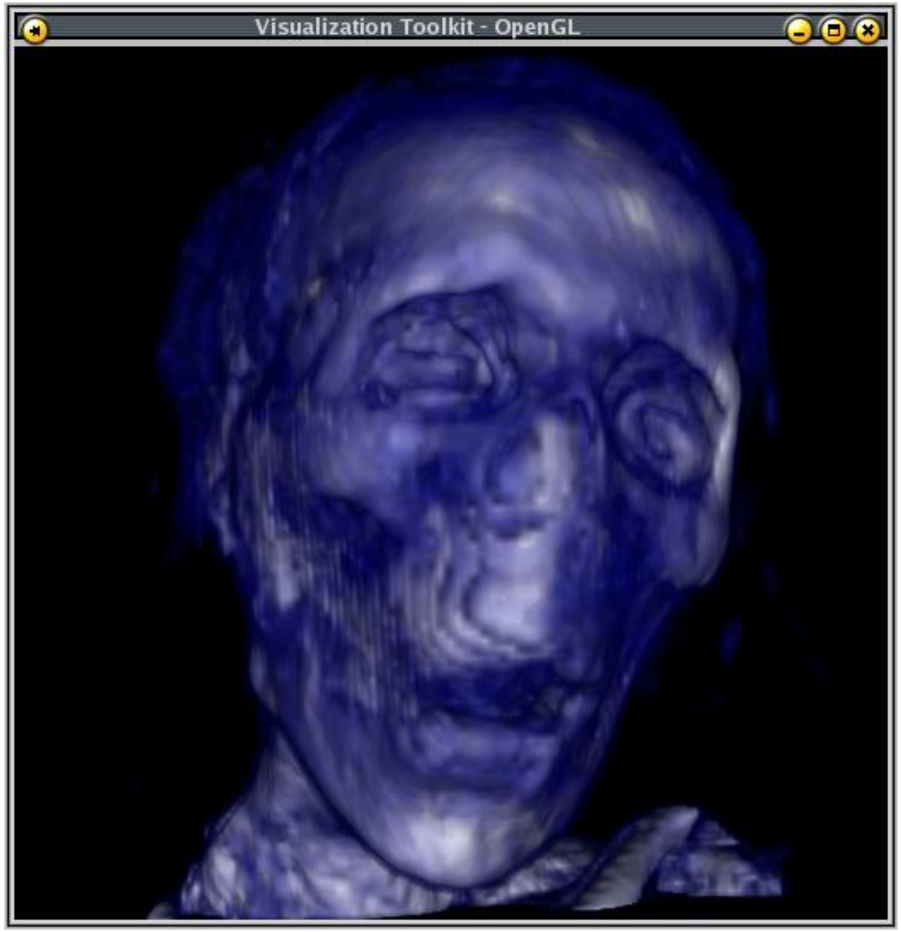


- **Accumulate opacity while compositing colors**: make transparent layers visible!
  Levoy '88

Volume Rendering Pipeline



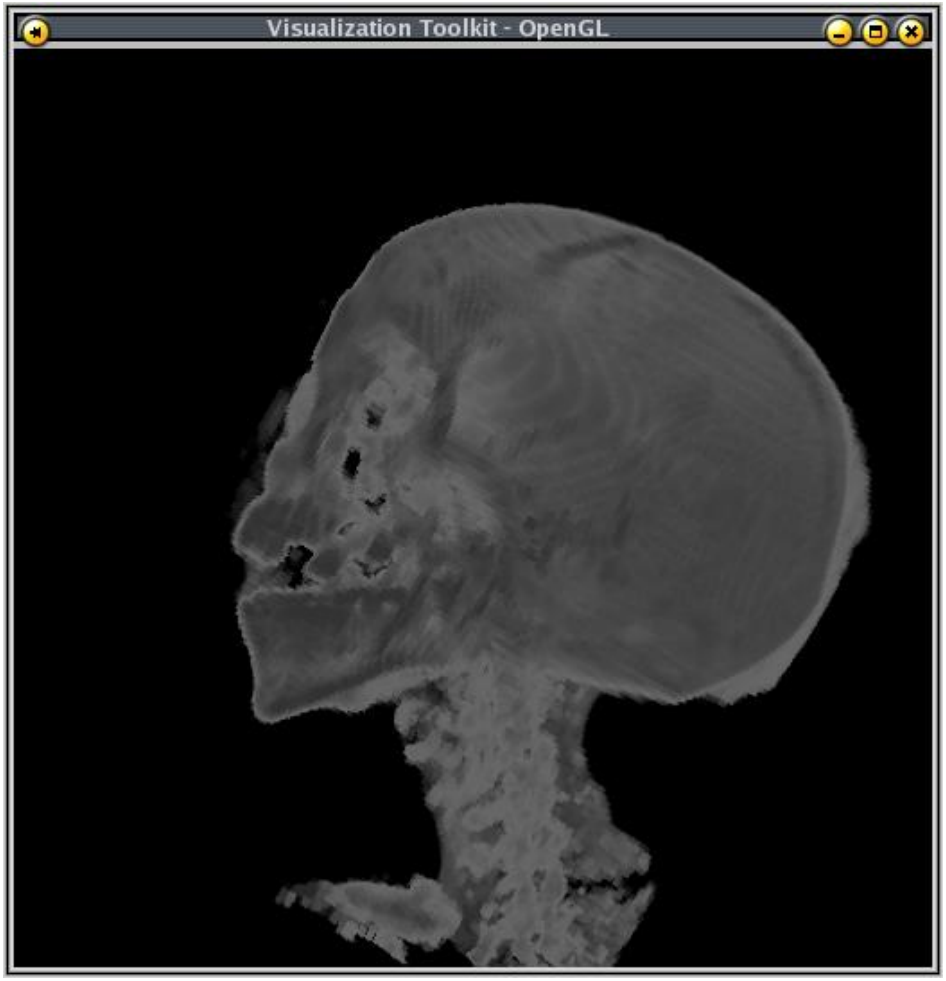
VTK Ray Casting Example



- 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**

Adaptive Termination

- The goal of adaptive termination is to quickly identify the last sample along a ray that significantly changes the color of the ray. We define a significant color change as one in which C_out – C_in > Epsilon for some small Epsilon > 0. Thus, Once opacity first exceeds 1-Epsilon, the color will not change significantly. This is the termination criteria.
- Higher values of Epsilon reduce rendering time, while lower values of Epsilon reduce image artifacts producing a higher quality image.

Four Steps Summary

- The volume ray casting algorithm comprises :
- **Ray casting.** For each pixel of the final image, a ray of sight is shot ("cast") through the volume. At this stage it is useful to consider the volume being touched and enclosed within a *bounding primitive*, a simple geometric object — usually a cuboid — that is used to intersect the ray of sight and the volume.
- **Sampling.** Along the part of the ray of sight that lies within the volume, equidistant *sampling points* or *samples* are selected. As in general the volume is not aligned with the ray of sight, sampling points usually will be located in between voxels. Because of that, it is necessary to **trilinearly interpolate** the values of the samples from its surrounding voxels.
- **Shading.** For each sampling point, the gradient is computed. These represent the orientation of local surfaces within the volume. The samples are then *shaded*, i. e. coloured and lighted, according to their surface orientation and the source of light in the scene.
- **Compositing.** After all sampling points have been shaded, they are *composited* along the ray of sight, resulting in the final colour value for the pixel that is currently being processed. The composition is derived directly from the rendering equation and is similar to blending acetate sheets on an overhead projector.

http://en.wikipedia.org/wiki/Volume_ray_casting

Exercises

- The scalar values at eight vertices of the voxel of unit length (**a=1**) are

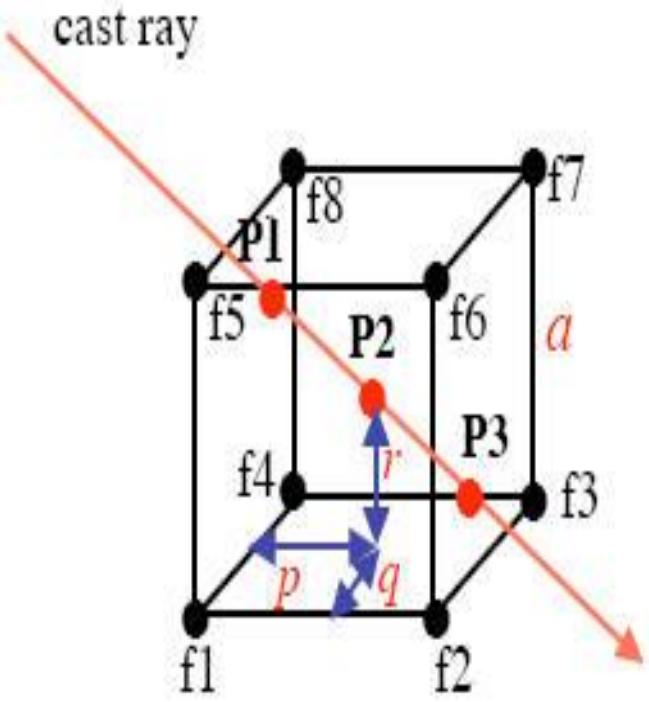
  f1=5.0, f2=5.1, f3=5.2, f4=4.6

  f5=4.9, f6=5.1, f7=5.3, f8=4.7

  The ray is sampled at three points

  P1(0.2,0.3,0.8), P2(0.5,0.5,0.5), P3(0.8,0.7,0.2)

  Color and opacity of the cast ray before it hits **P1** is (0,0,0,0)



Surface Modeling

Visualization using BrainVISA

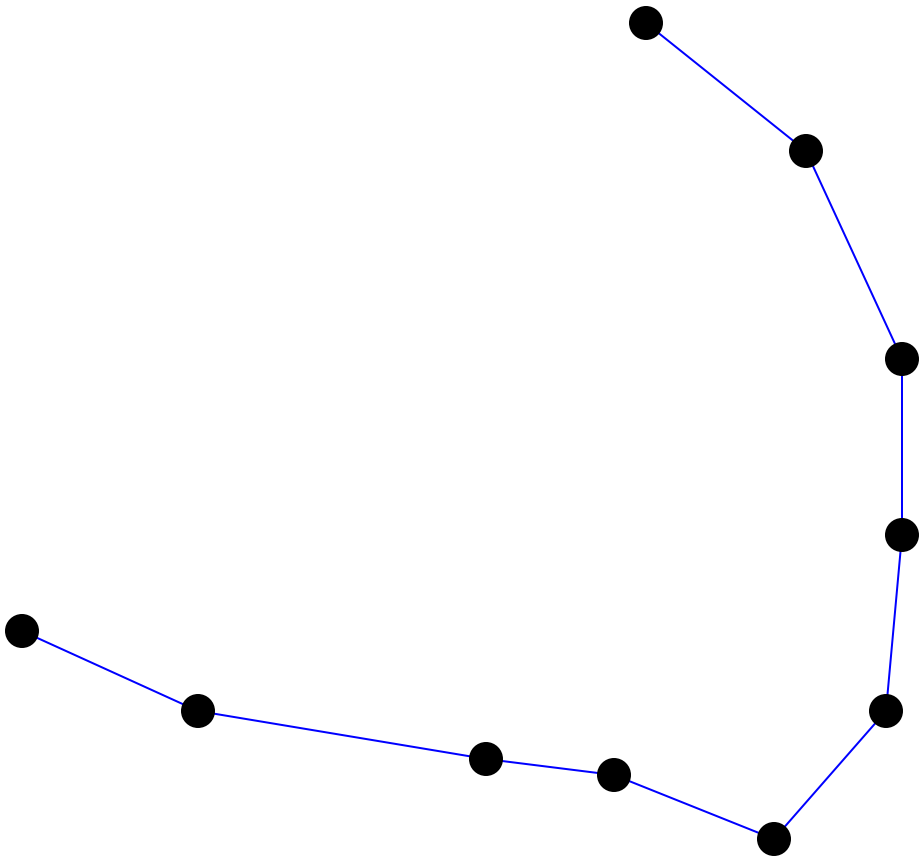
Why Use Surface Modeling

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

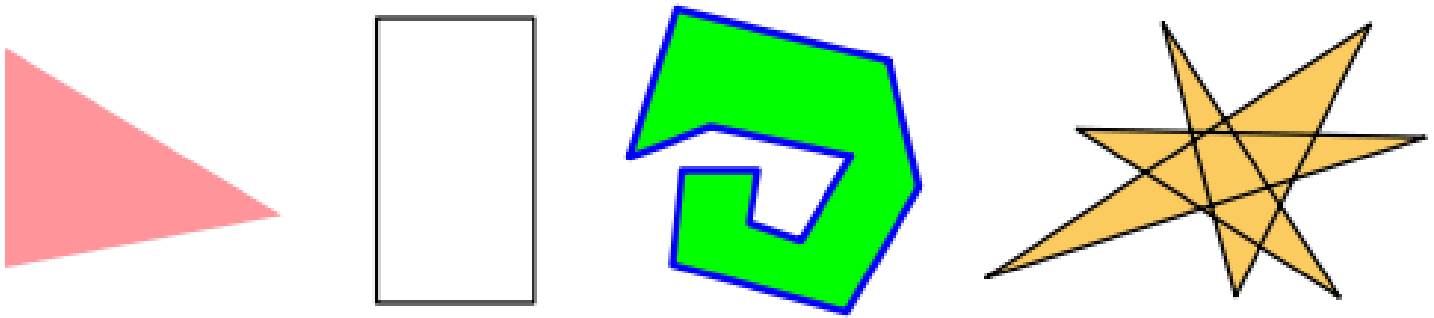
What Makes a Surface

- 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

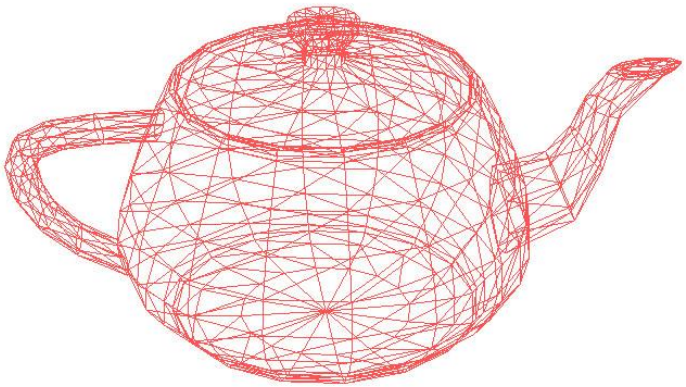
Vertices and Edges

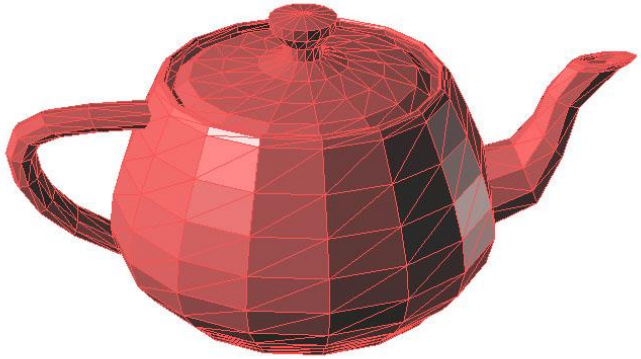


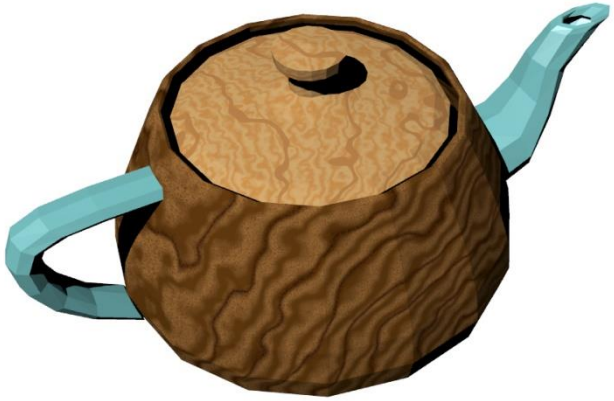
Polygons



Polygon Mesh Surface



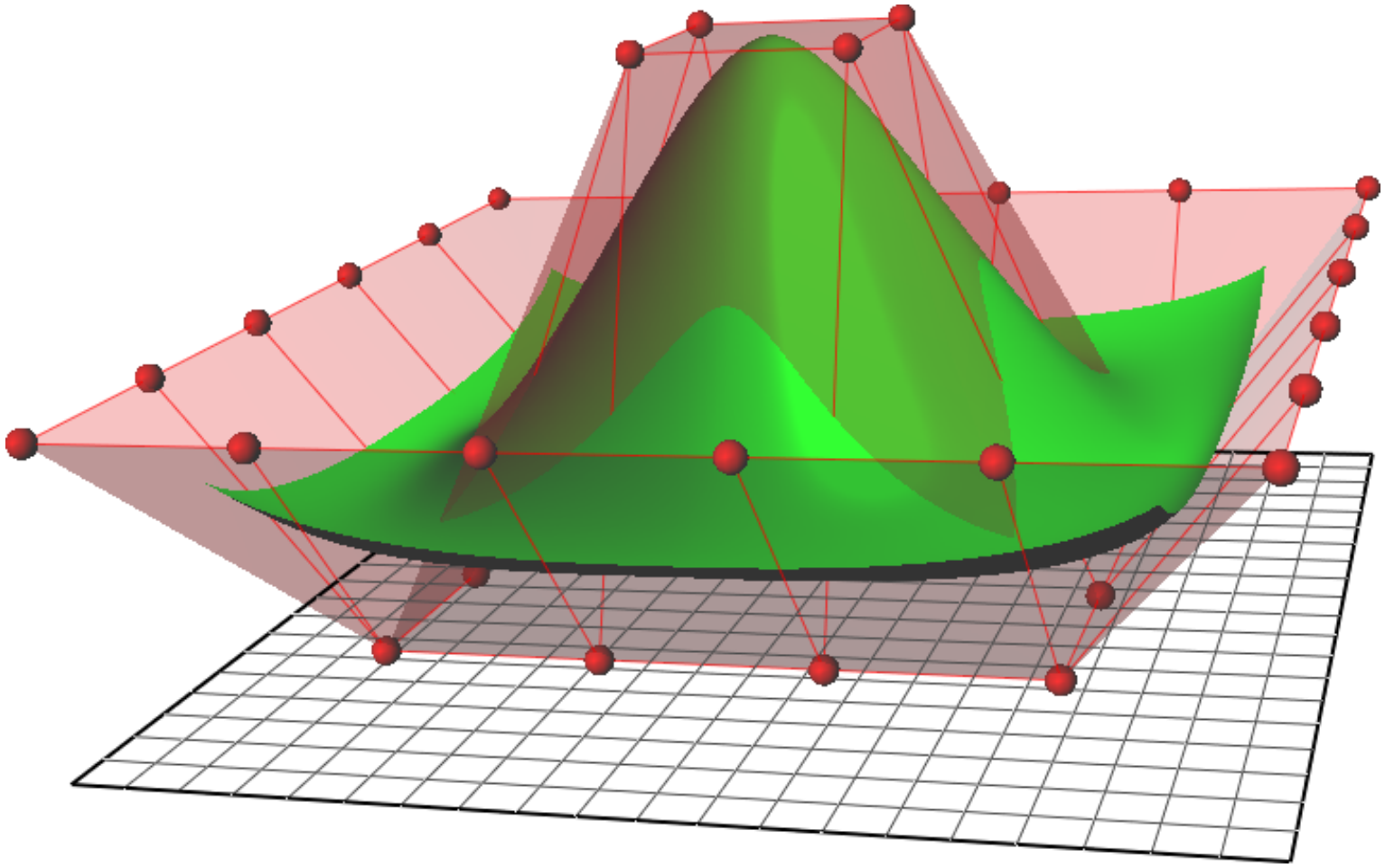




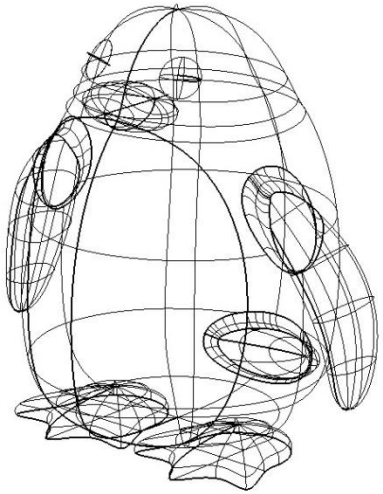
Parametric Surfaces

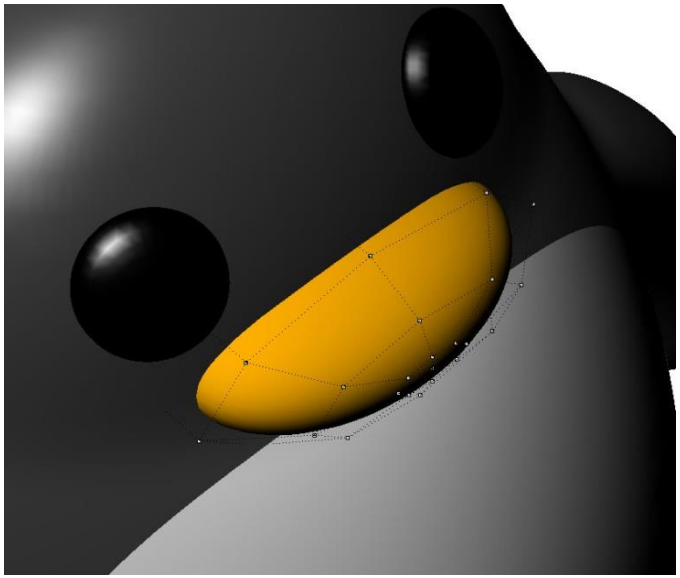
- Surface in Euclidean space defined by a parametric equation with two parameters
- A set of weighted control points determine the location of individual surface points
- The come in several flavors including Bezier, B-Spline, NURBS

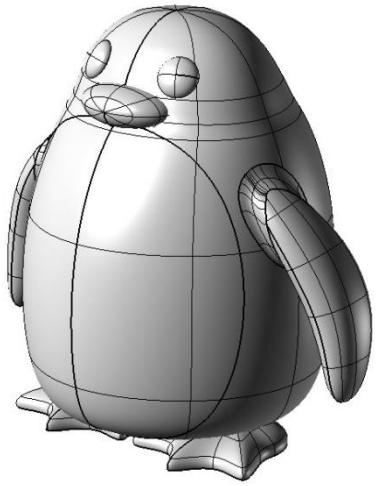
NURBS surface with control points

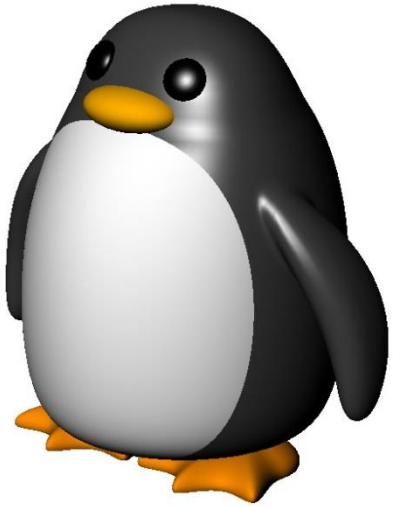


NURBS Surfaces

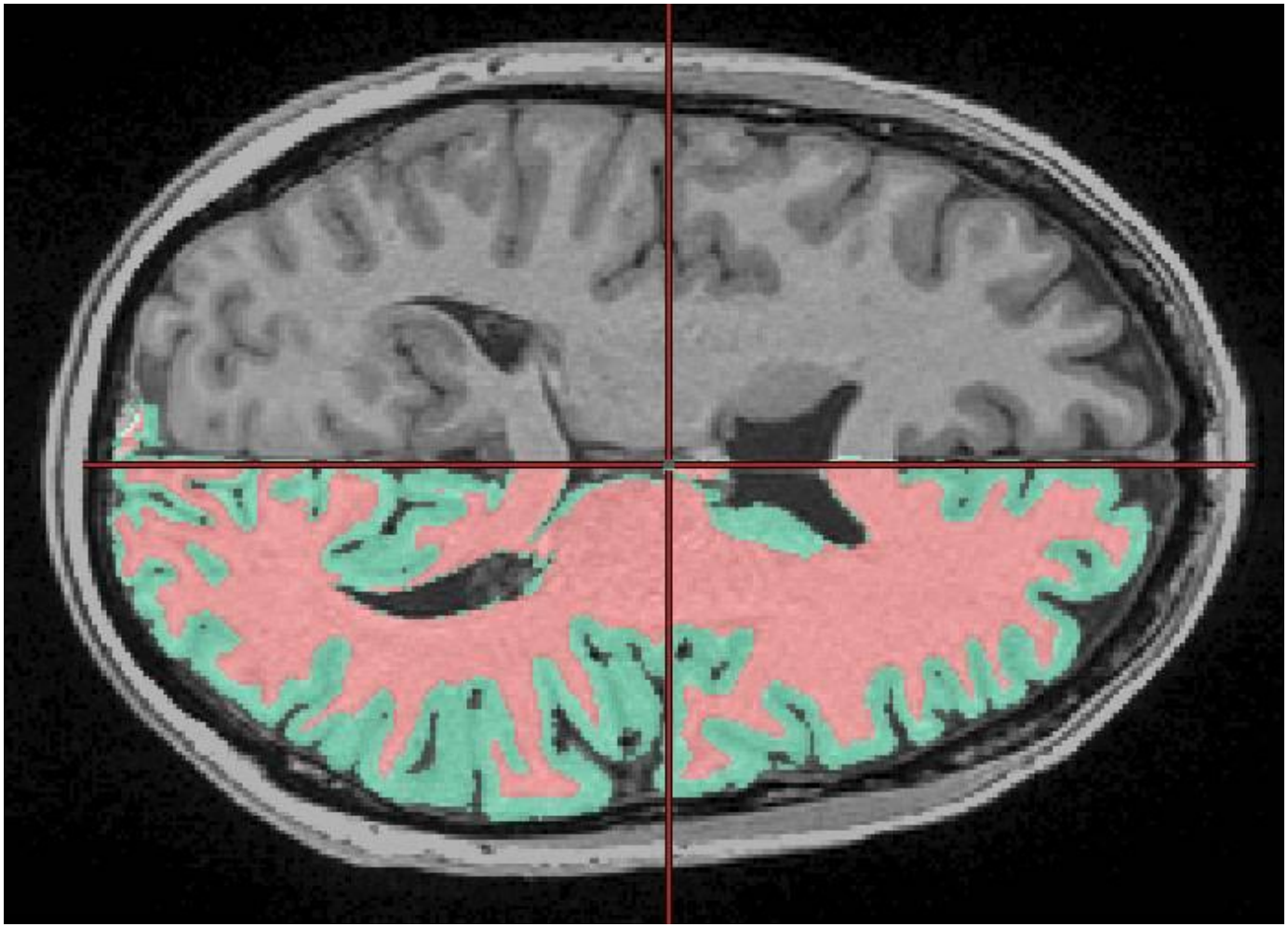








Isosurface extraction
or
Where to put the surface



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

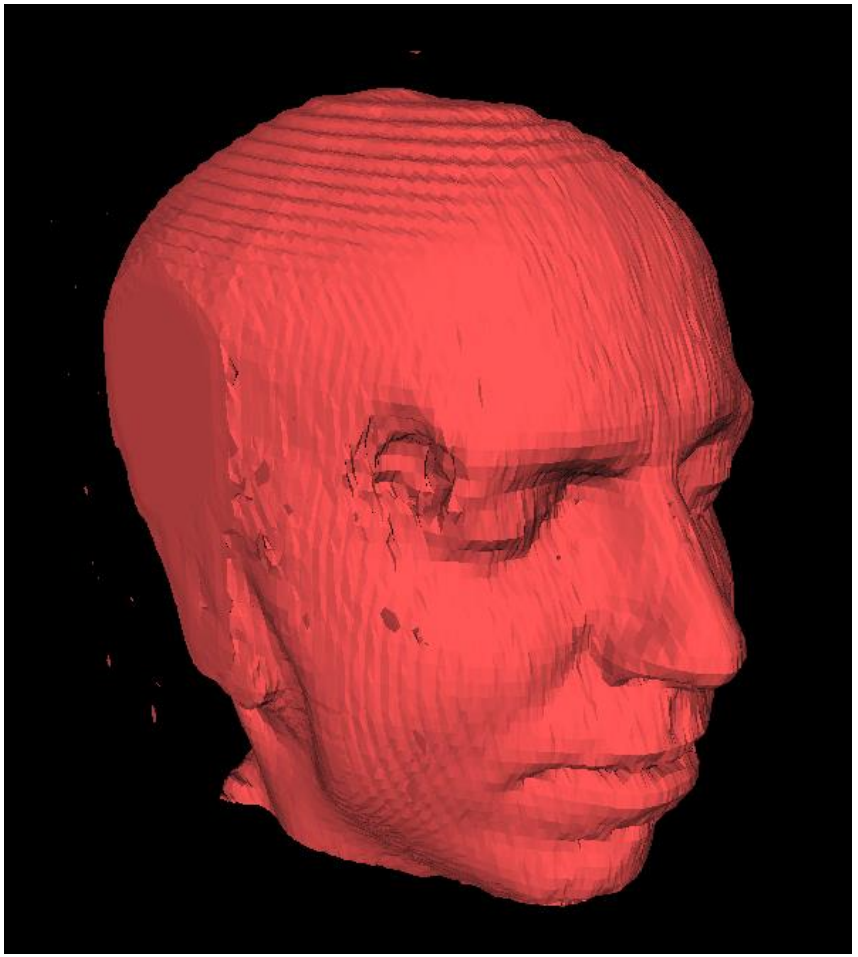
- Geometric Decomposition Techniques
  - Geometric techniques retain the original representation of the volume and partition along divisions in the voxel volume
- Span Space Decomposition Techniques
  - Span space decomposition techniques create and manipulate abstract representations of the voxels

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)

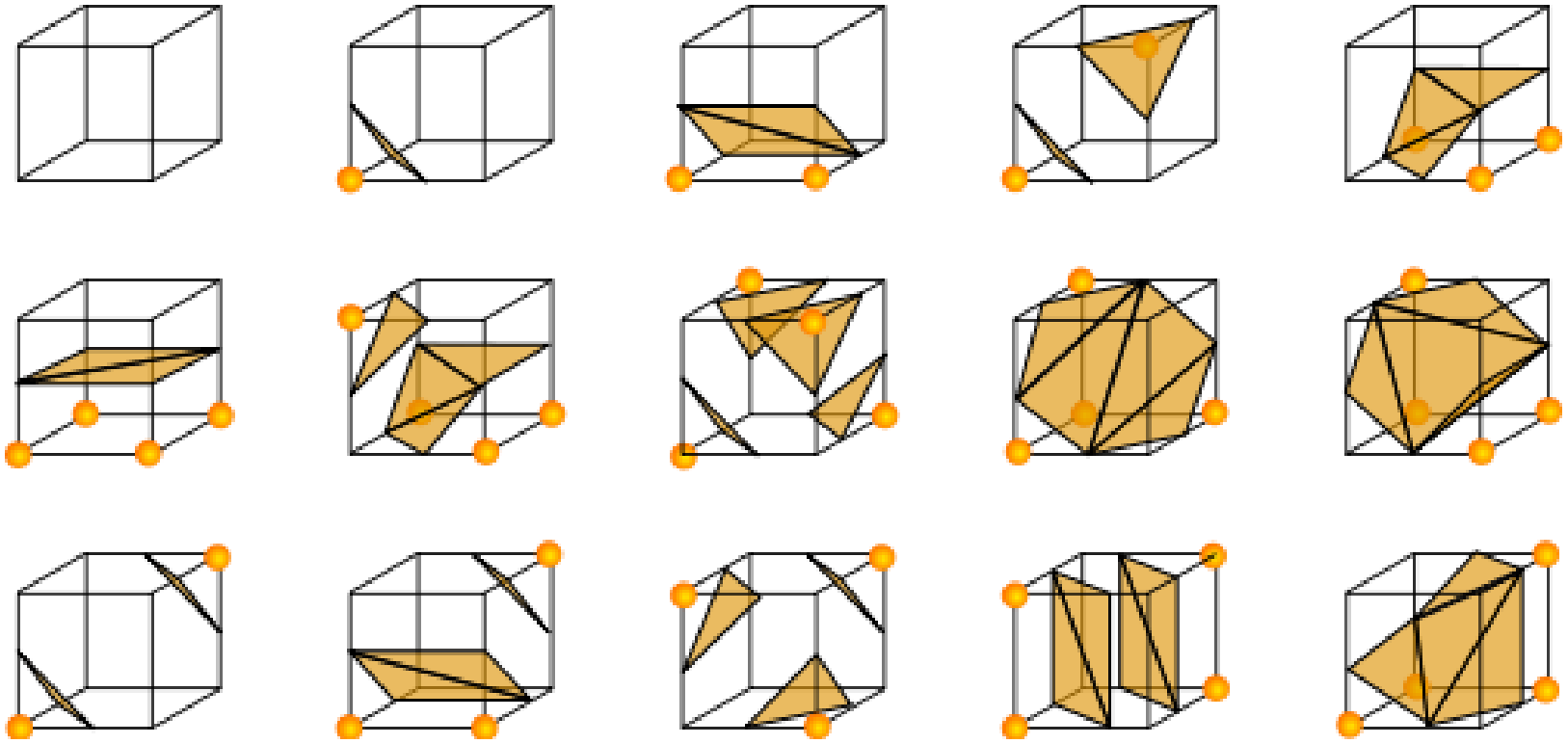
Marching Cubes

- 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

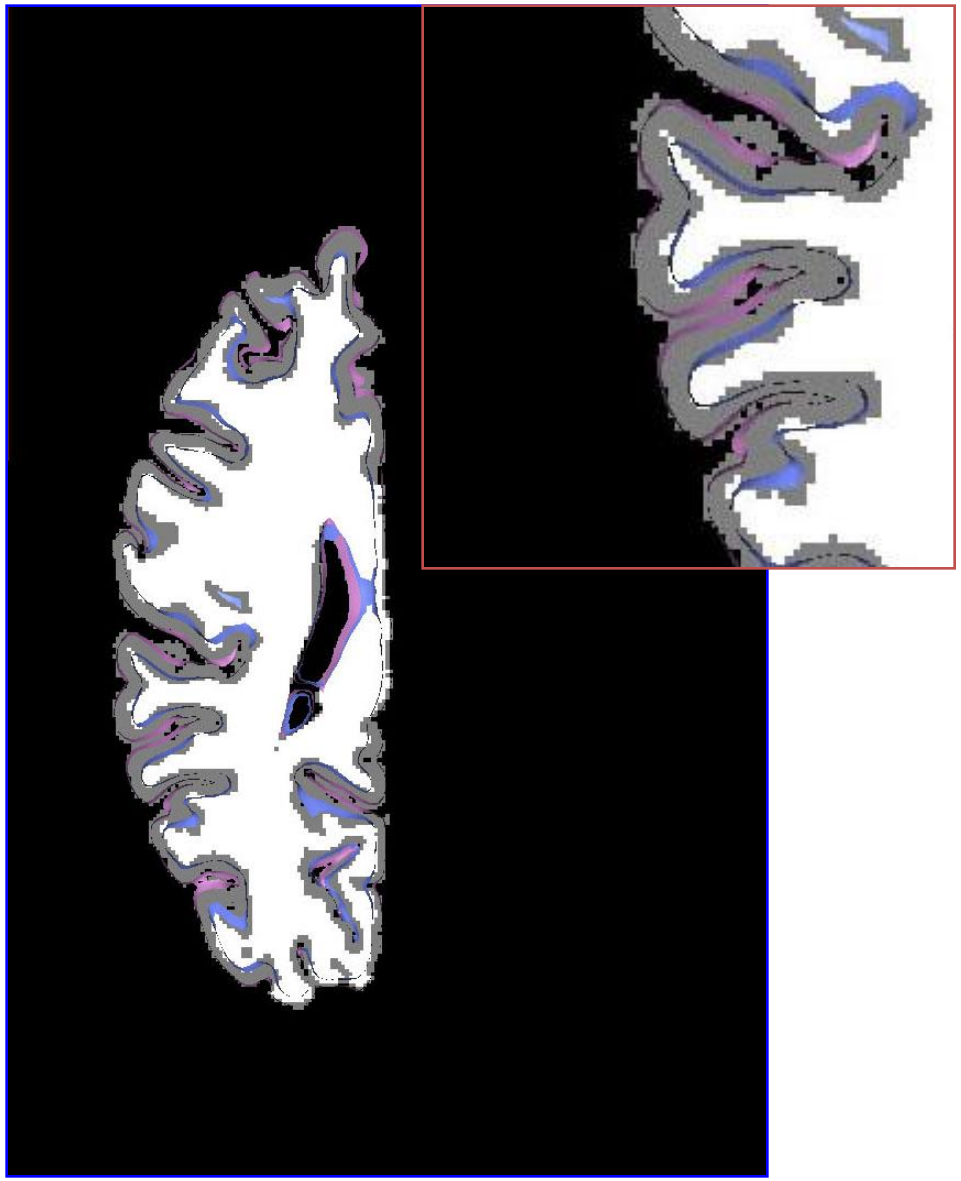
- 15 Unique cube configurations that can be rotated and reflected to 256 configurations

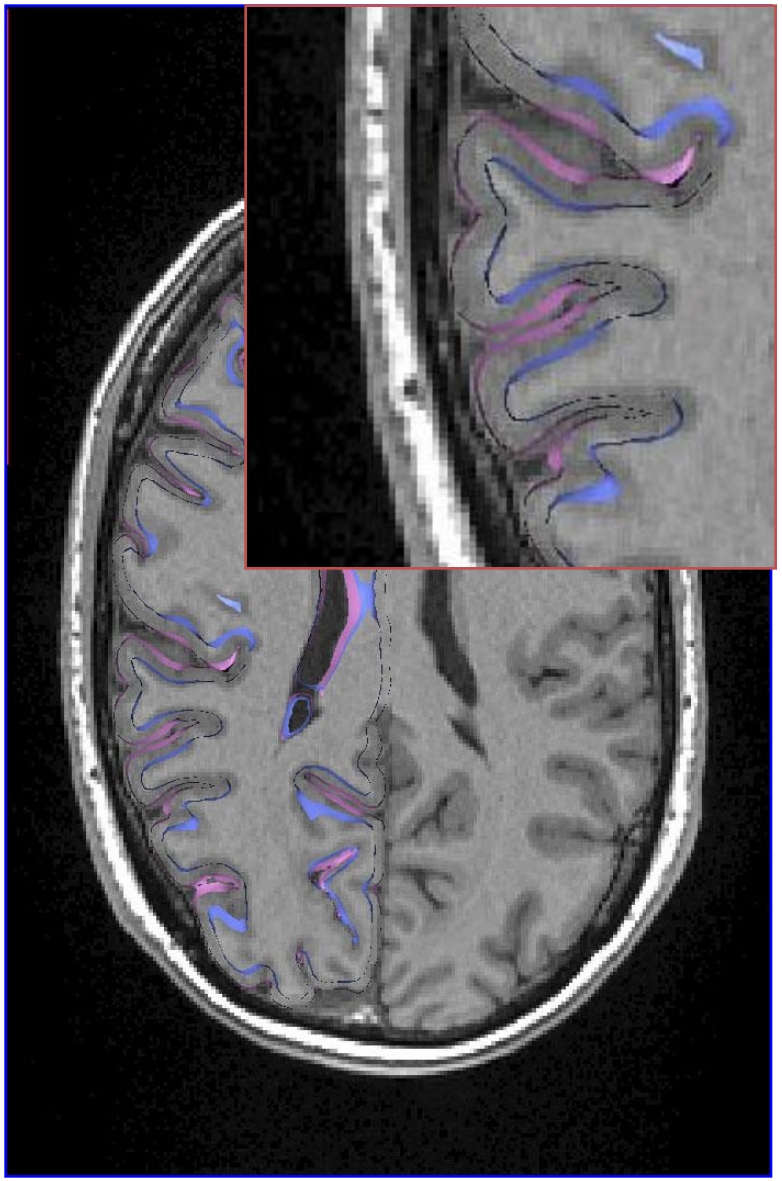


Marching Cubes Demo

Graphics cards aren’t just for games anymore

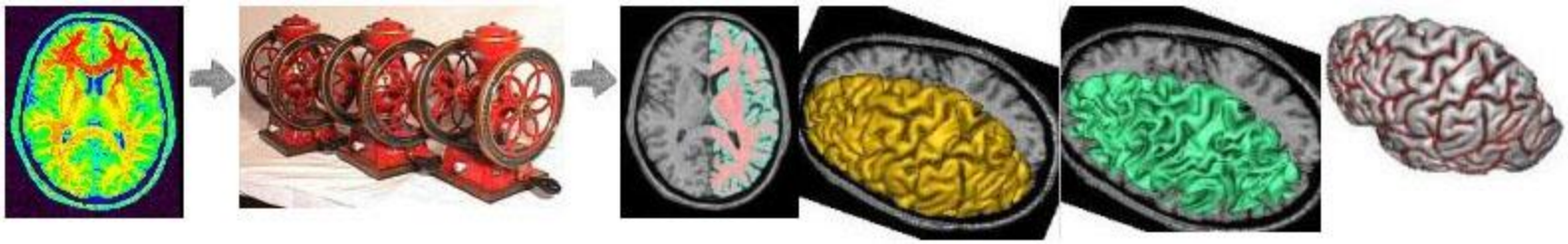
Mesh Segmentation





Introduction to BrainVisa

- Origin and Development
  - Collaborative work of methodologists of the Institut Fédératif de Recherche
  - Core development now at the Service Hospitalier Frédéric Joliot
- Framework for Image Processing
  - GUI for chaining applications together
    - GUI developed in Python
    - Command line application developed in C++
  - Database for organization of input and output files
  - Visualization package for viewing results



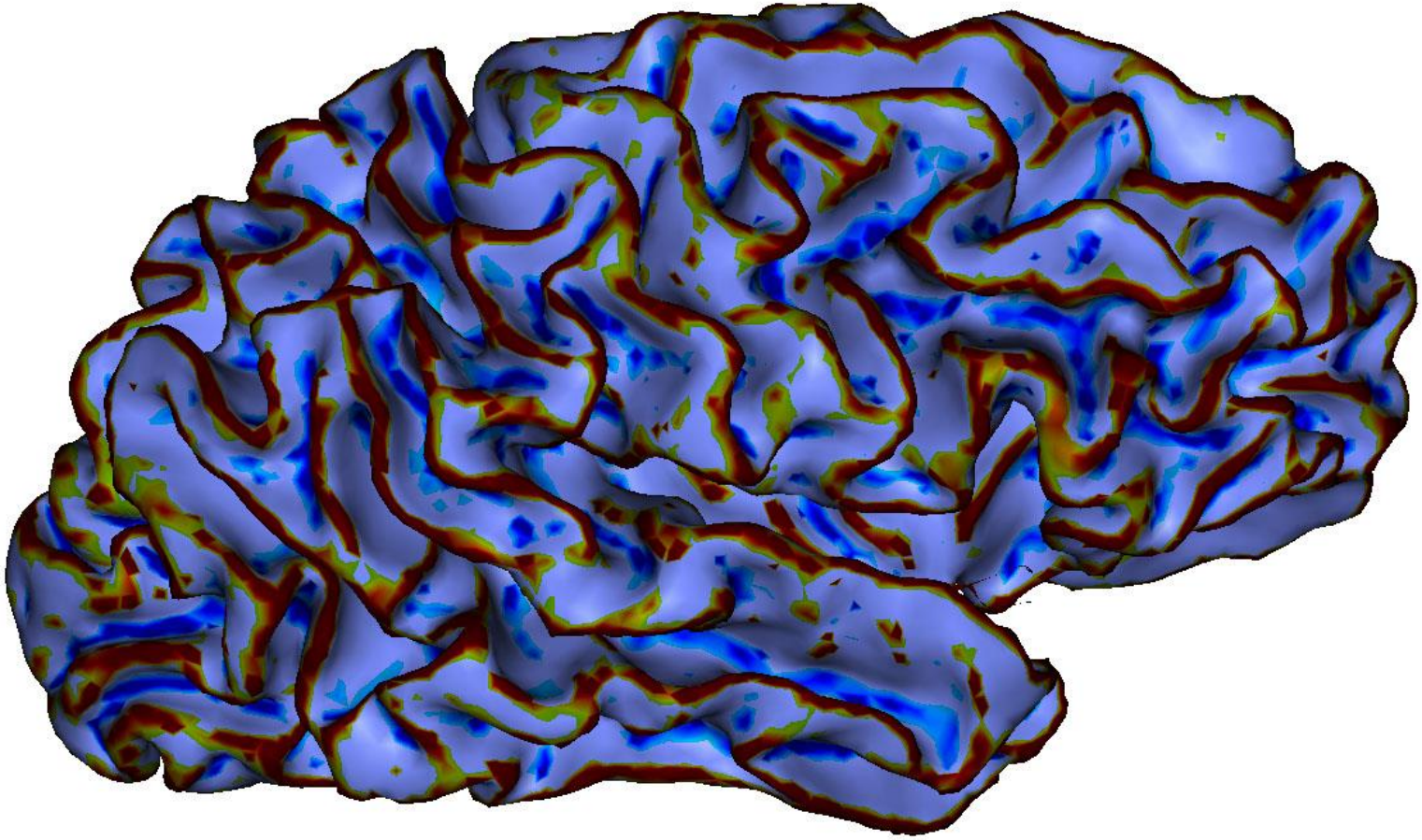
BrainVISA Availability

- Multiplatform
  - Linux (Fedora, Redhat, Mandriva)
  - Macintosh (OS X)
  - Windows (2000, XP)

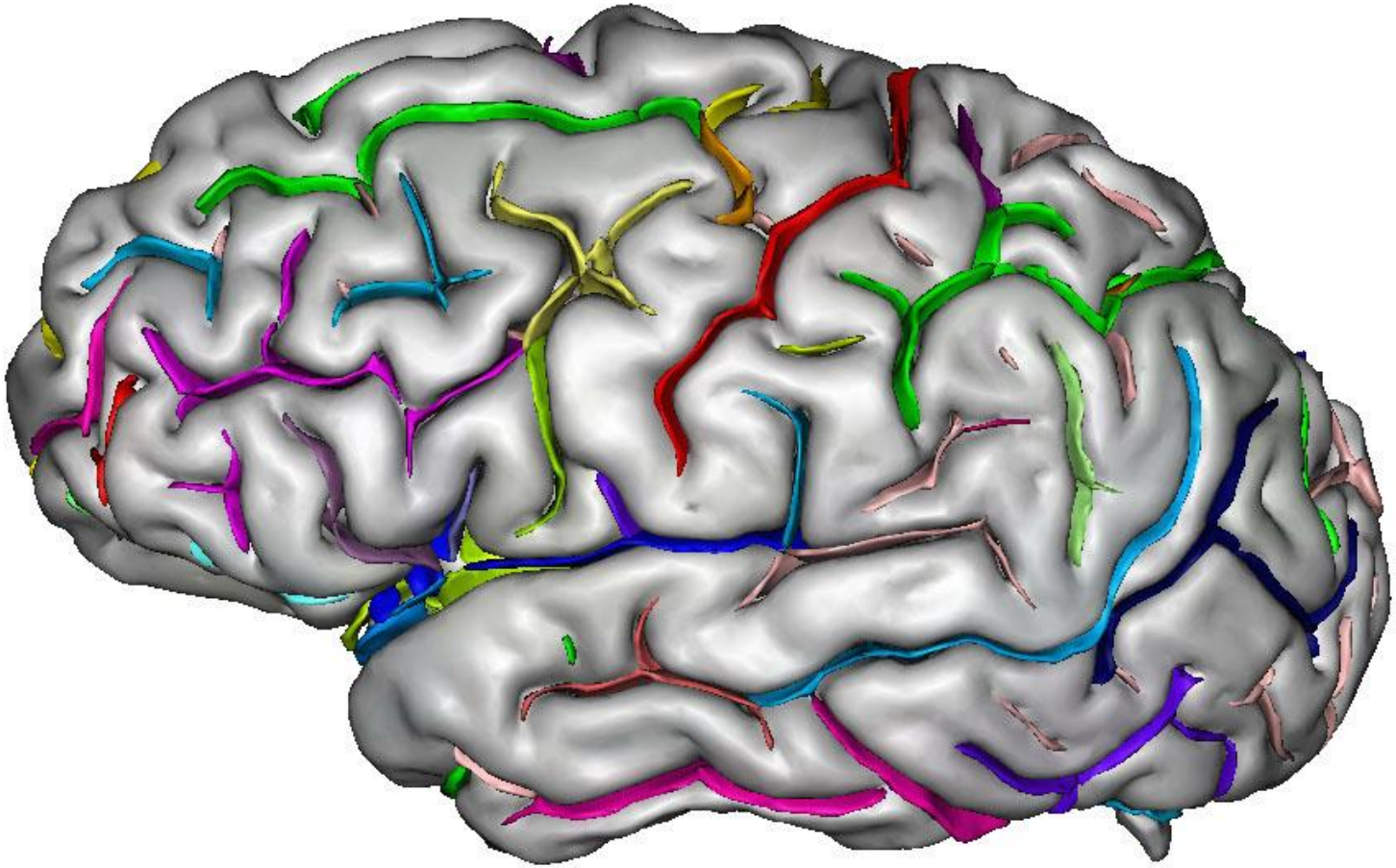
- Download at http://brainvisa.info/

BrainVISA Demo

BrainVISA Curvature Mapping

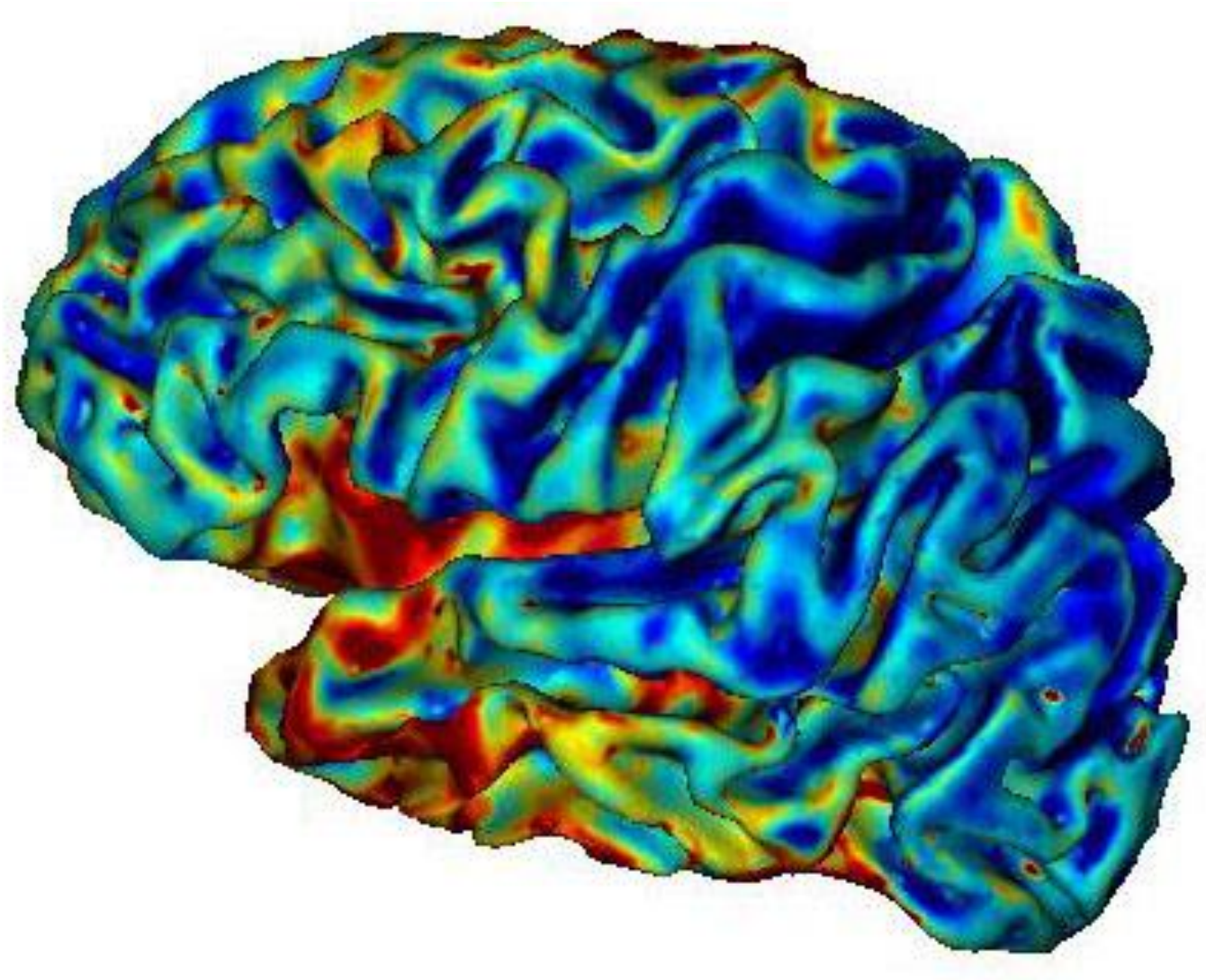


BrainVISA Cortical Folds

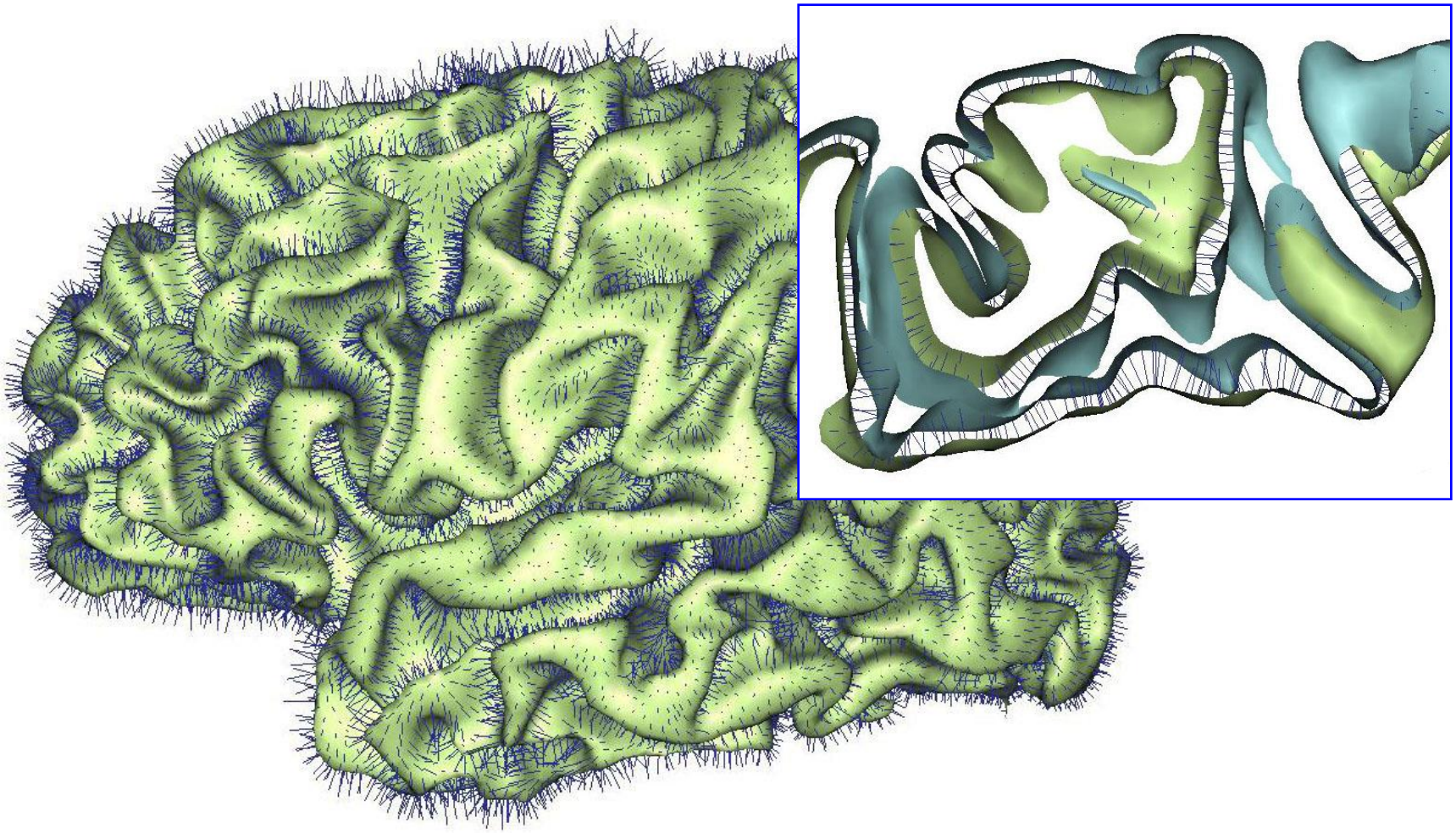


RIC BrainVisa Extensions

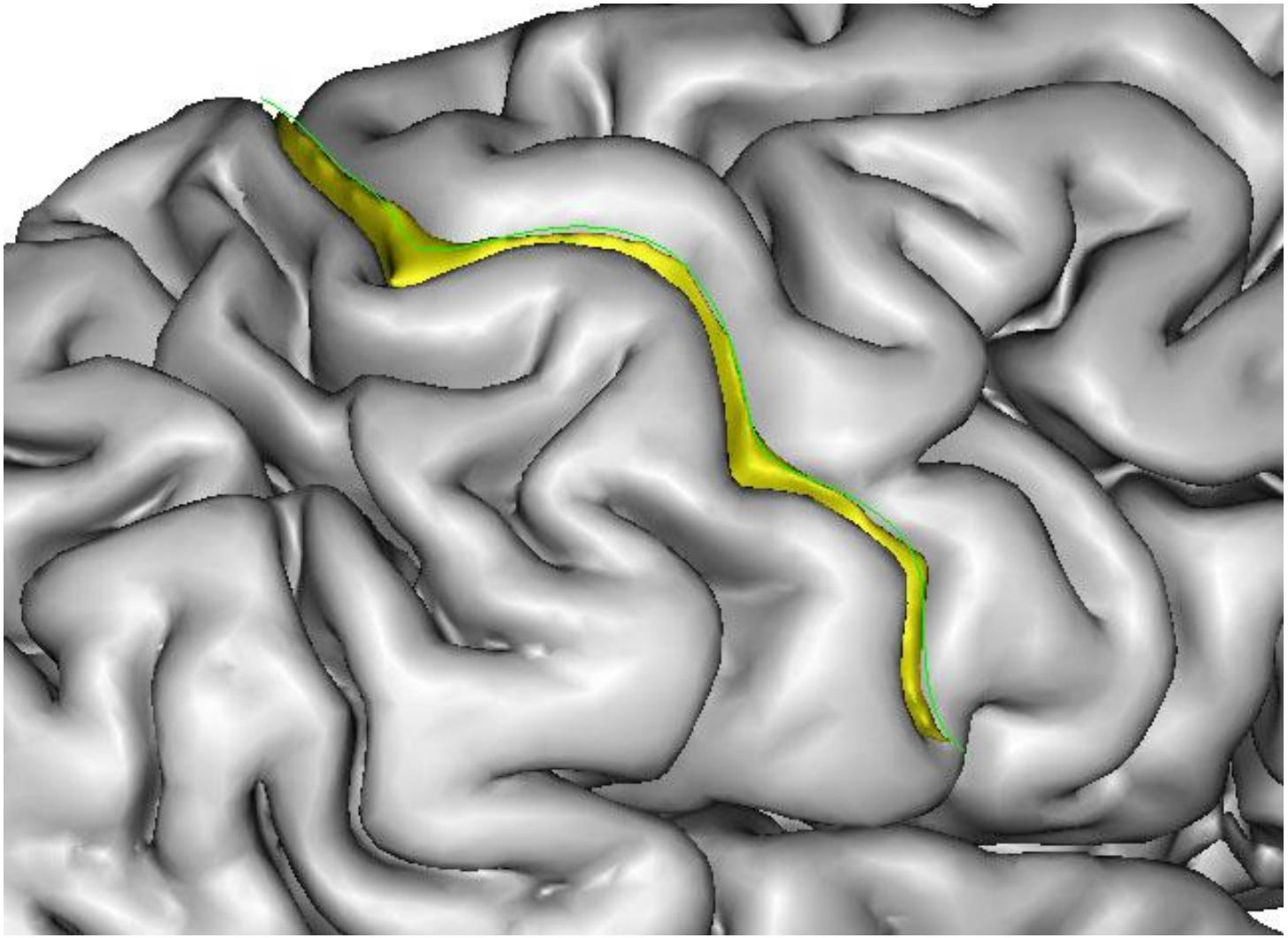
RIC Cortical Thickness



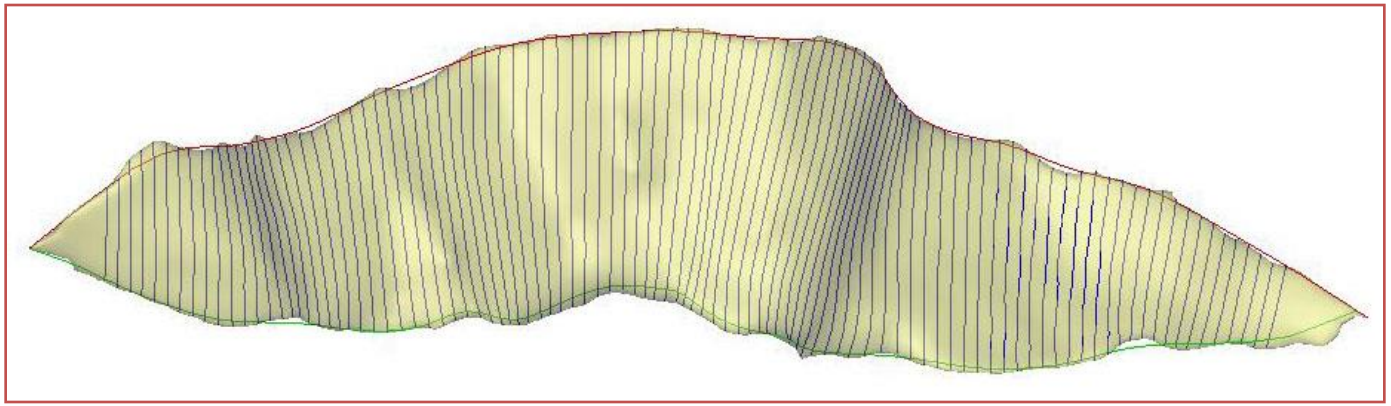
White matter surface normals



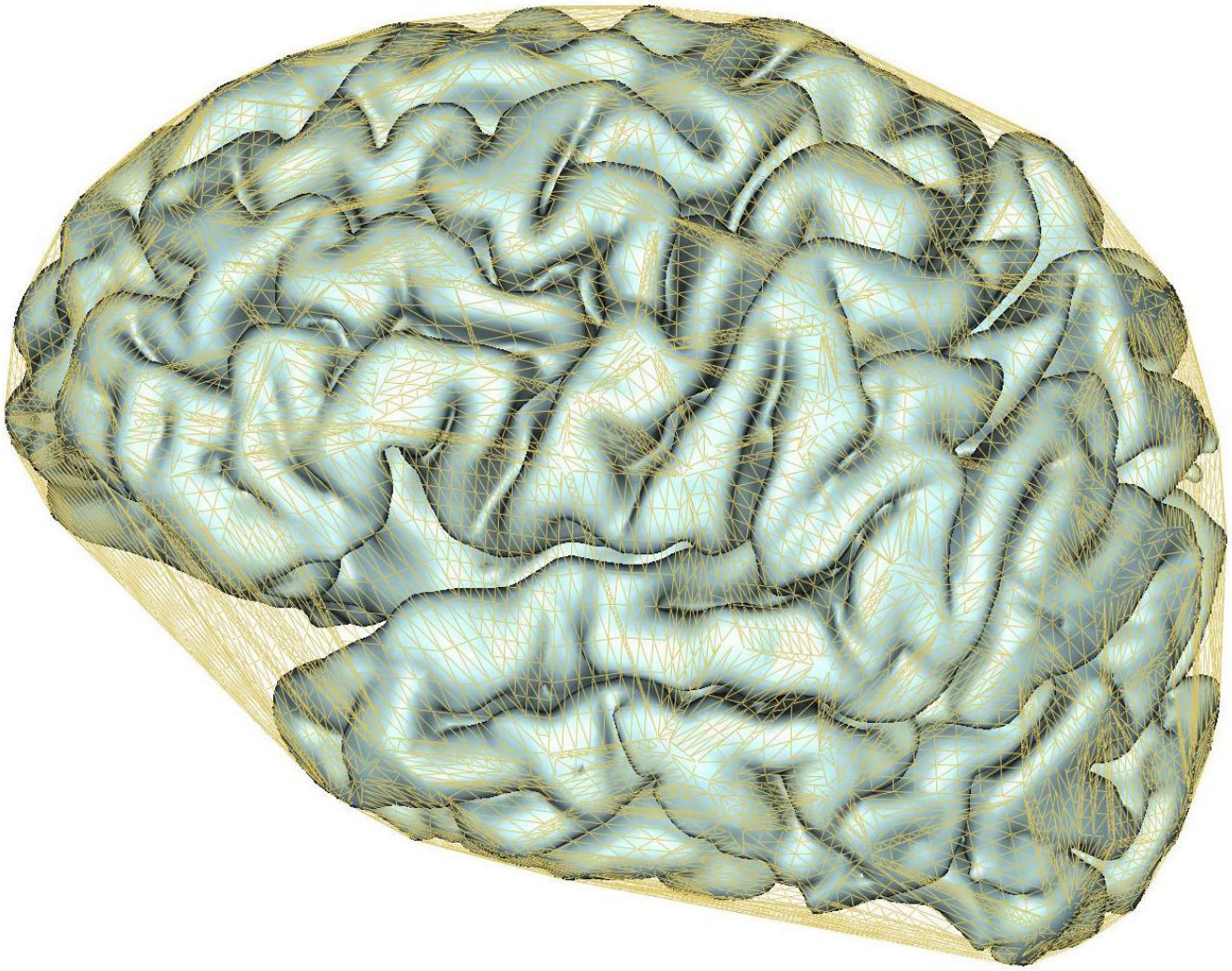
RIC Sulcal Length and Depth



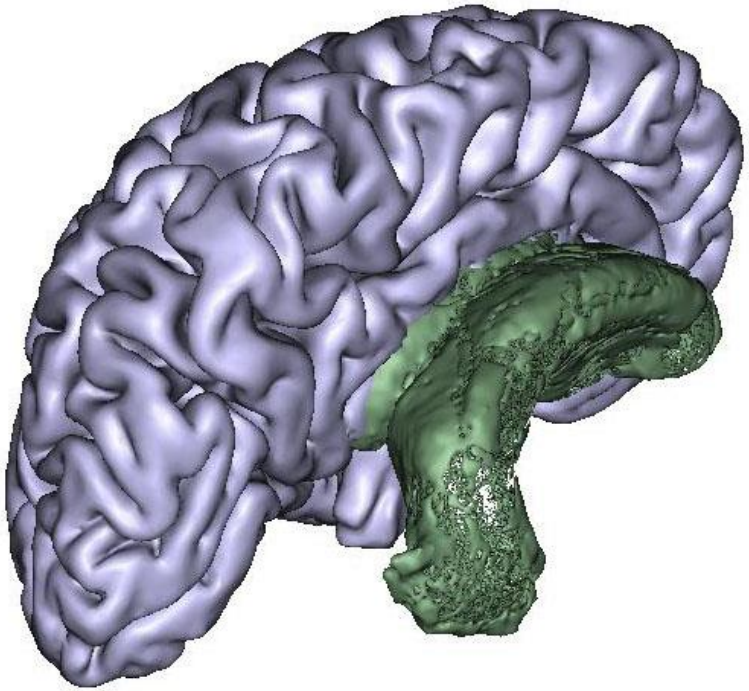
Mapping Mesh to Volume

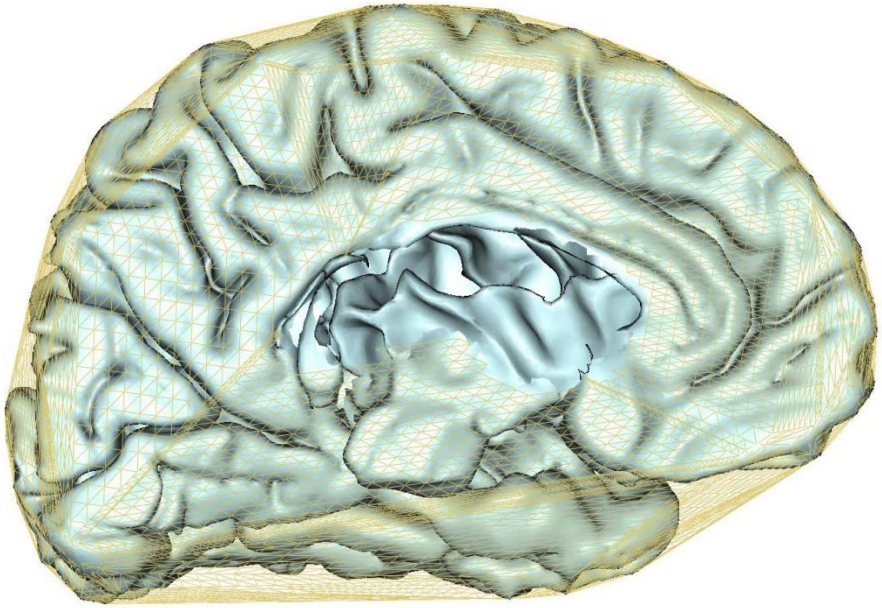


RIC 3D Gyrification Index



Removing Effect of Ventricles





Contouring

5.3 Contouring

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

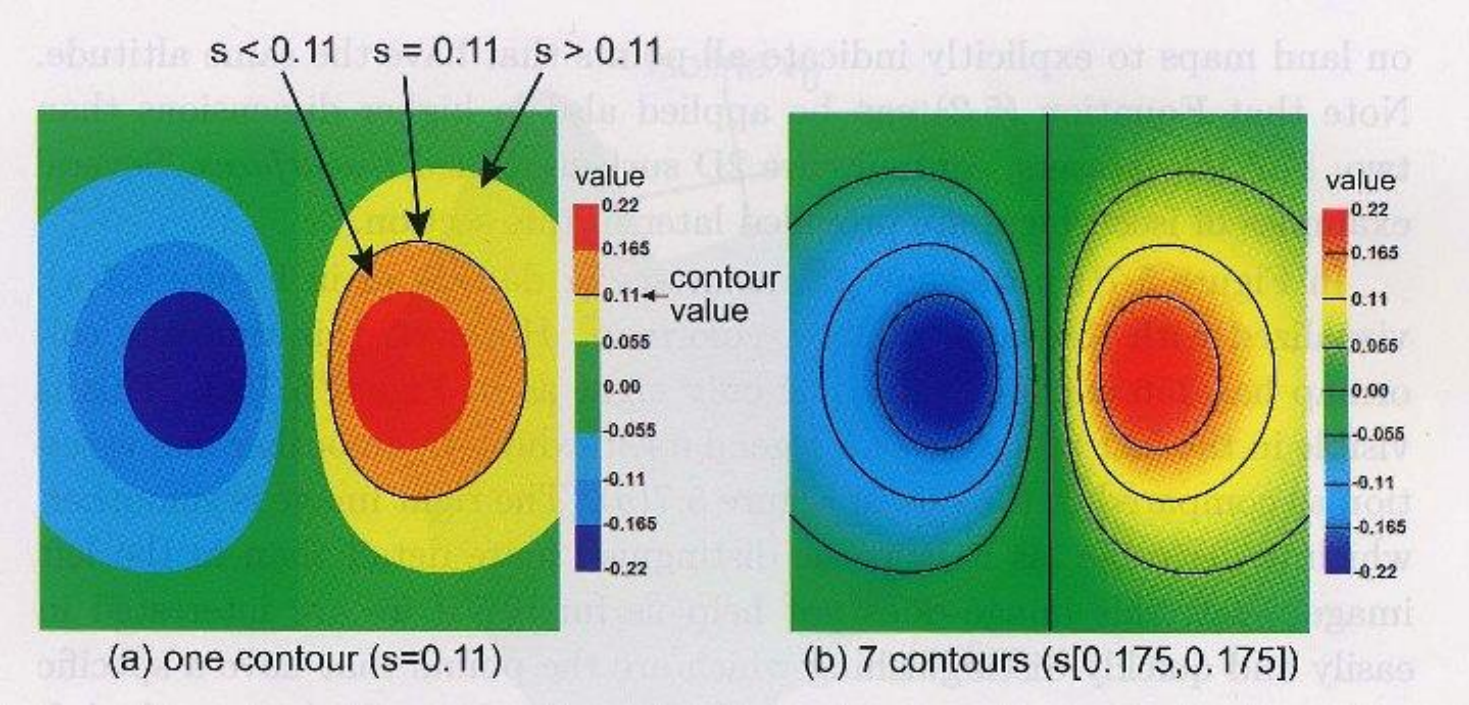


Fig 5.7. Relationship between color banding and contouring

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**)

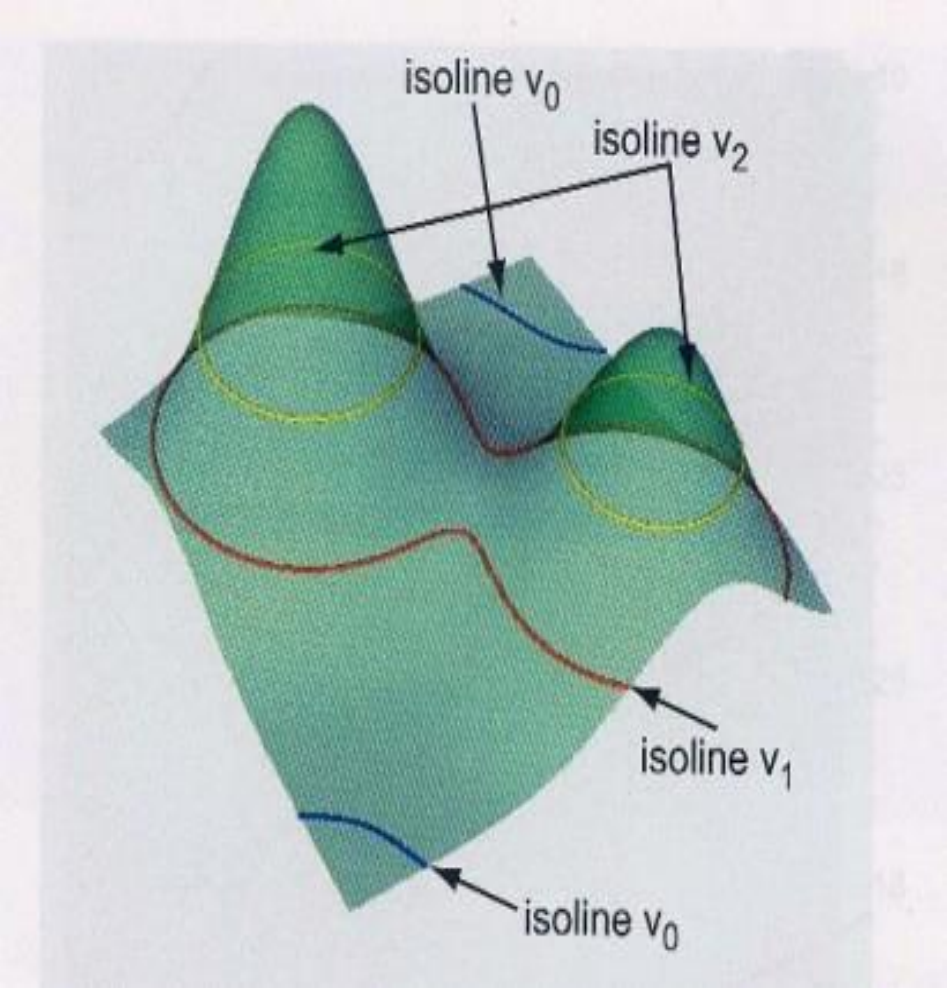


Fig 5.8. Isoline properties

5.3 Contouring

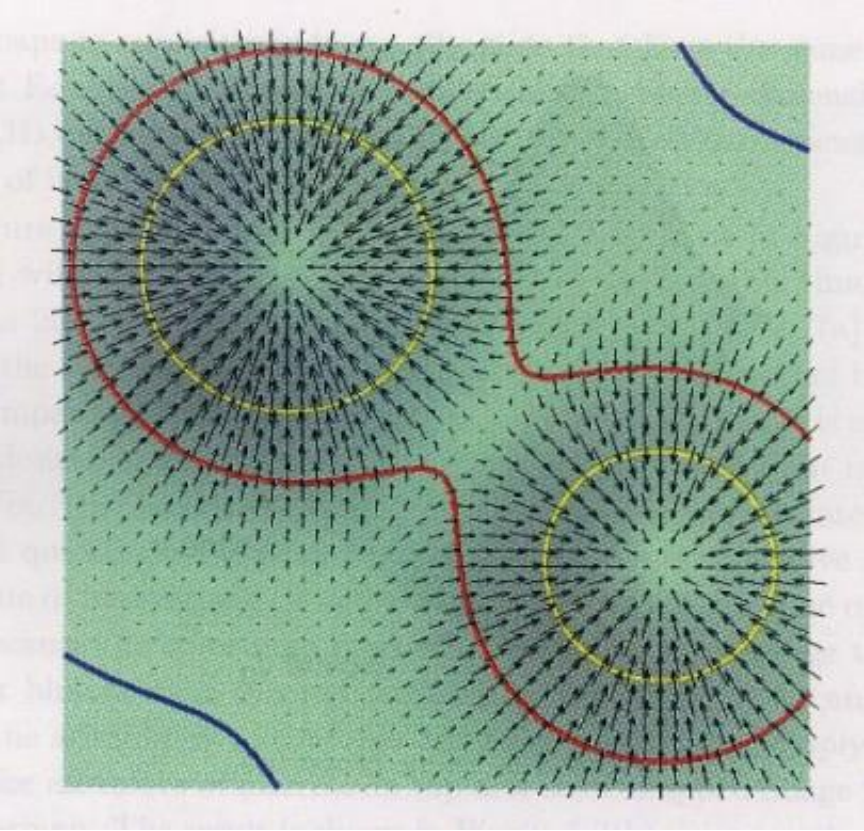


Fig 5.9. The gradient of a scalar field is perpendicular to the field's contours

5.3 Contouring

- Given a discrete, sampled dataset, compute contours

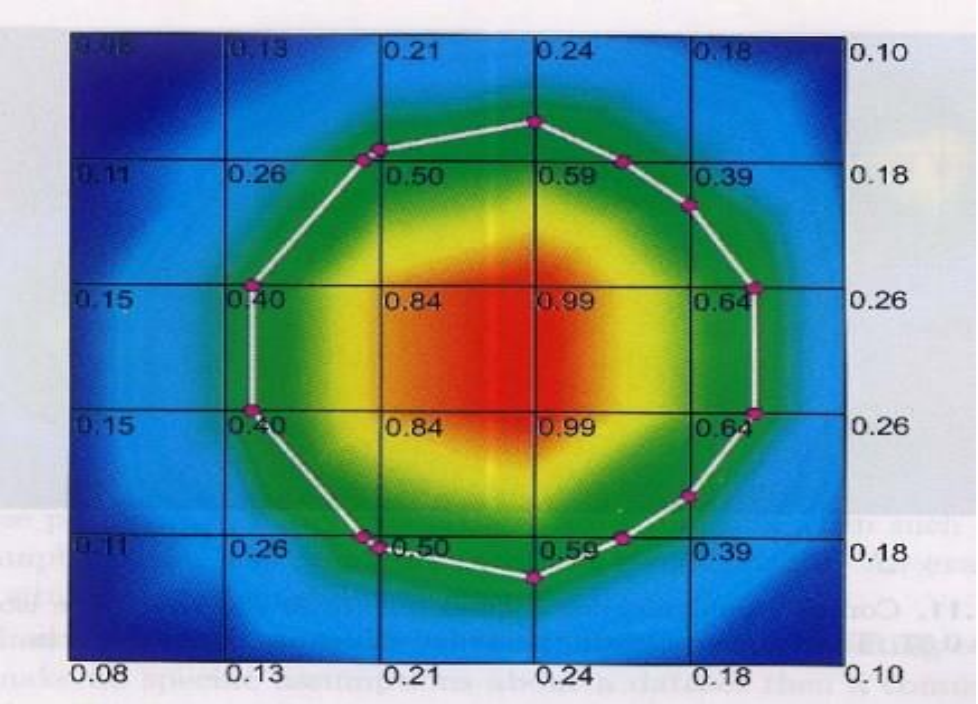


Fig 5.10. Constructing the isoline for the scalar value v = 0.48. The figure indicates scalar values at the grid vertices.

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

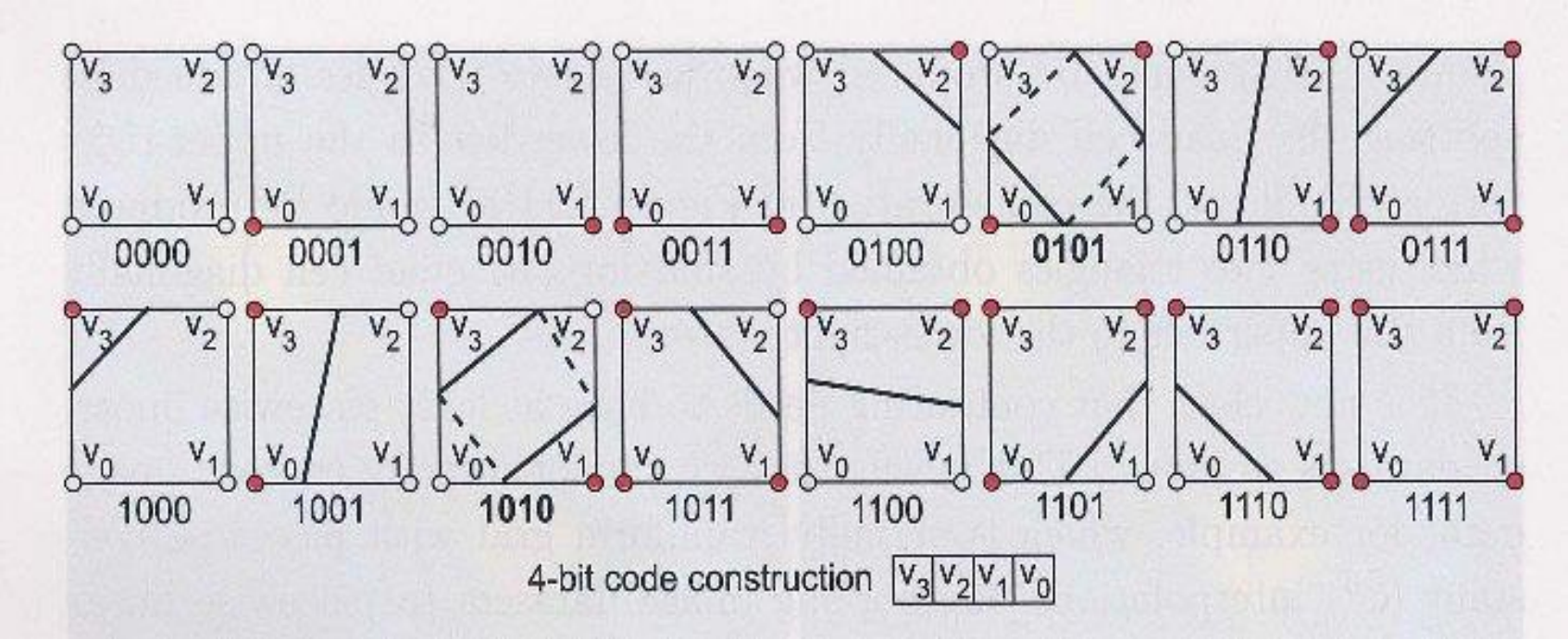


Fig5.12. Topological states of a quad cell (marching squares algorithm). Red indicates "inside" vertices. Bold indices mark ambiguous cases.

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

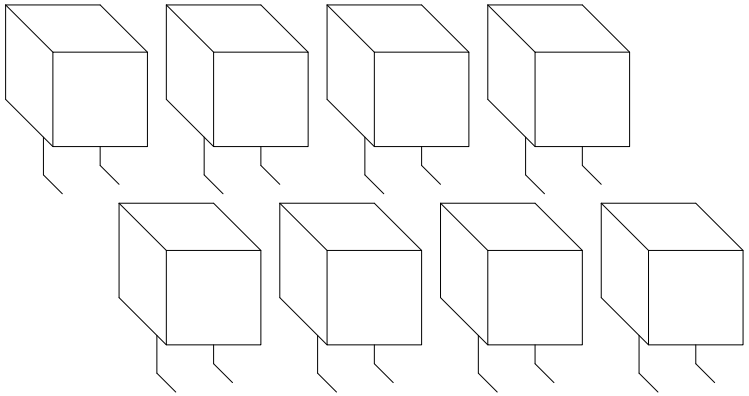
- **Marching Cubes: A High Resolution 3D Surface Construction Algorithm**

  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.”

Medical Data Acquisition

- Computed Tomography (CT)
- Magnetic Resonance (MRI)
- Single-Photon Emission Computed Tomography (SPECT)

Each scanning process results in two dimensional “slices” of 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.

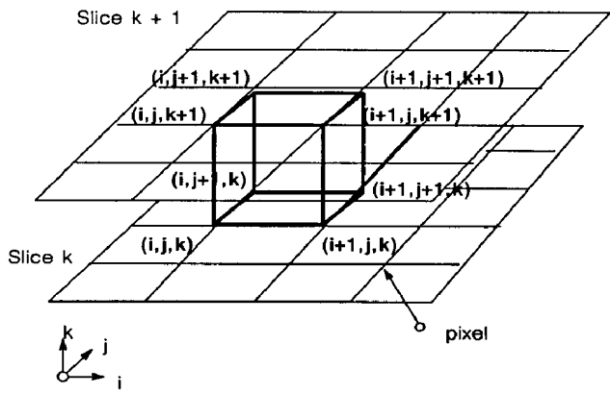
Marching Cubes Explained

- 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

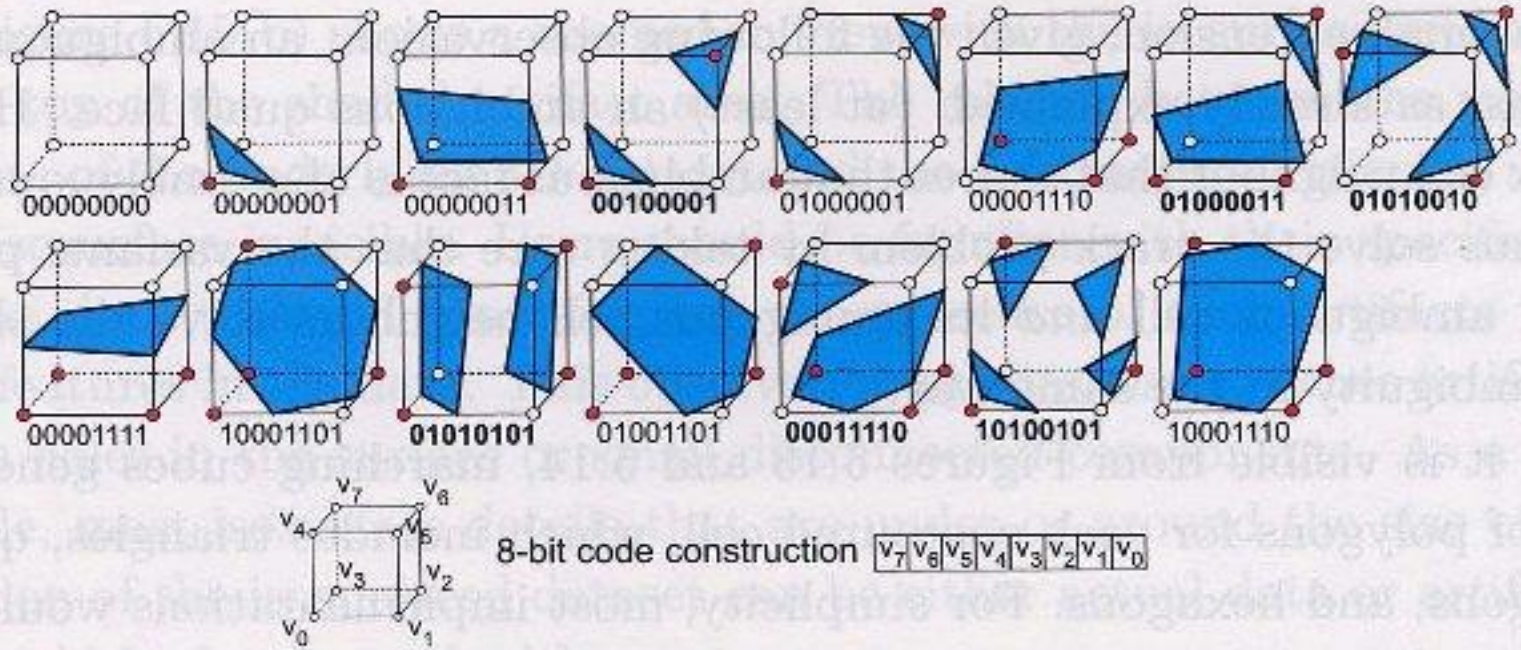


Fig 5.13. Topological states of a hex cell (marching cubes algorithm). Red indicates "inside" vertices. Bold indices mark ambiguous cases.

5.3.2 Marching Cubes -- Ambiguity

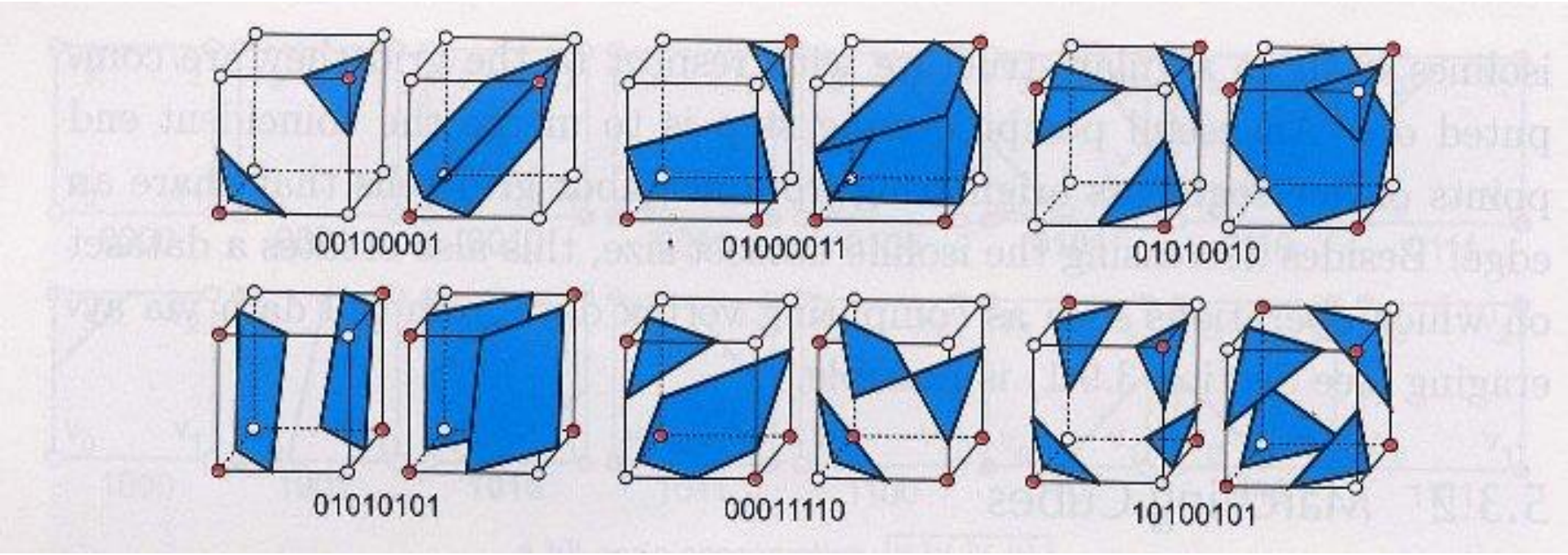
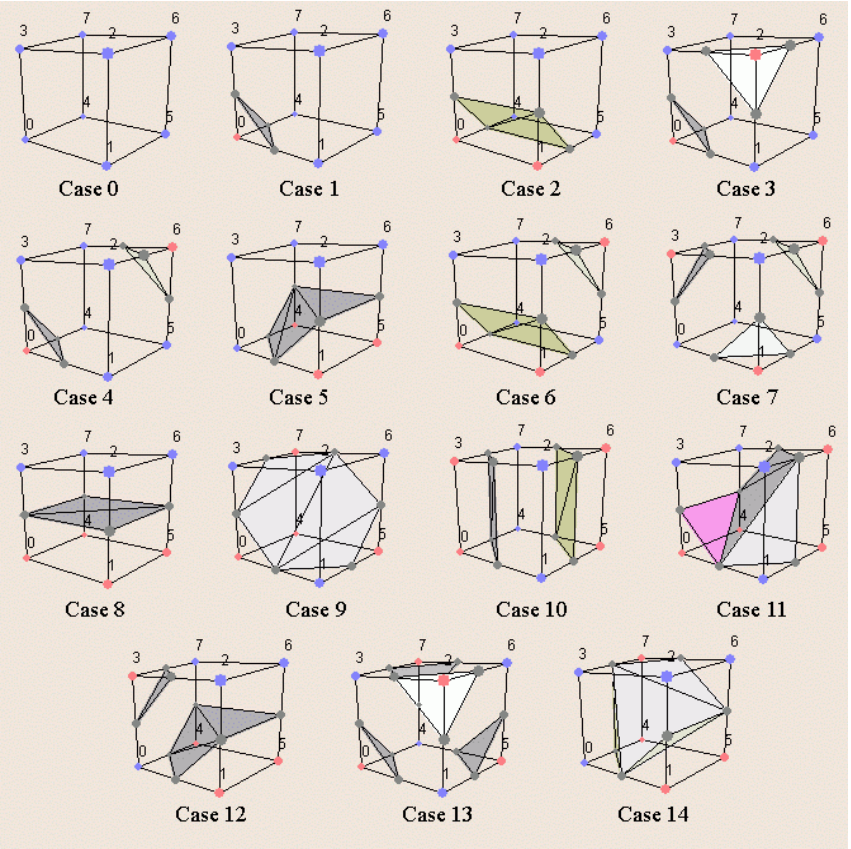


Fig 5.14. Ambiguous cases for marching cubes. Each case has two contouring variants.

How Are The Cubes Used

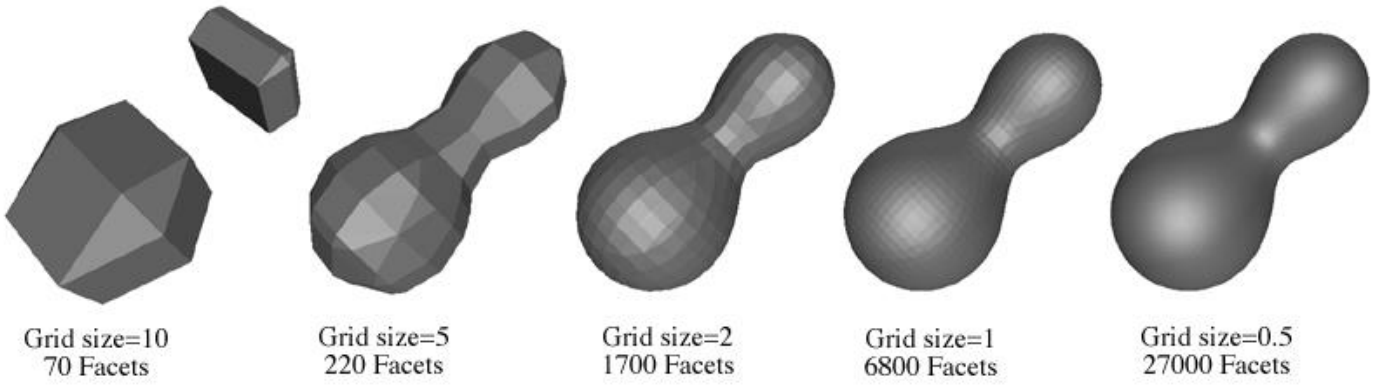
- Pixels on the slice surfaces determine 3D surfaces.
- 256 surface permutations, but only 14 unique patterns.
- A normal is calculated for each triangle vertex for rendering.



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.

Improvements over Other Methods

- Utilizes pixel, line and slice coherency to minimize the number of calculations.
- Can provide solid modeling.
- Can use conventional rendering techniques and hardware.
- No user interaction is necessary.
- Enables selective displays.
- Can be used with other density values.

5.3.2 Marching Cubes

MRI scan data

“wavy” pattern:

Caused by subsampling

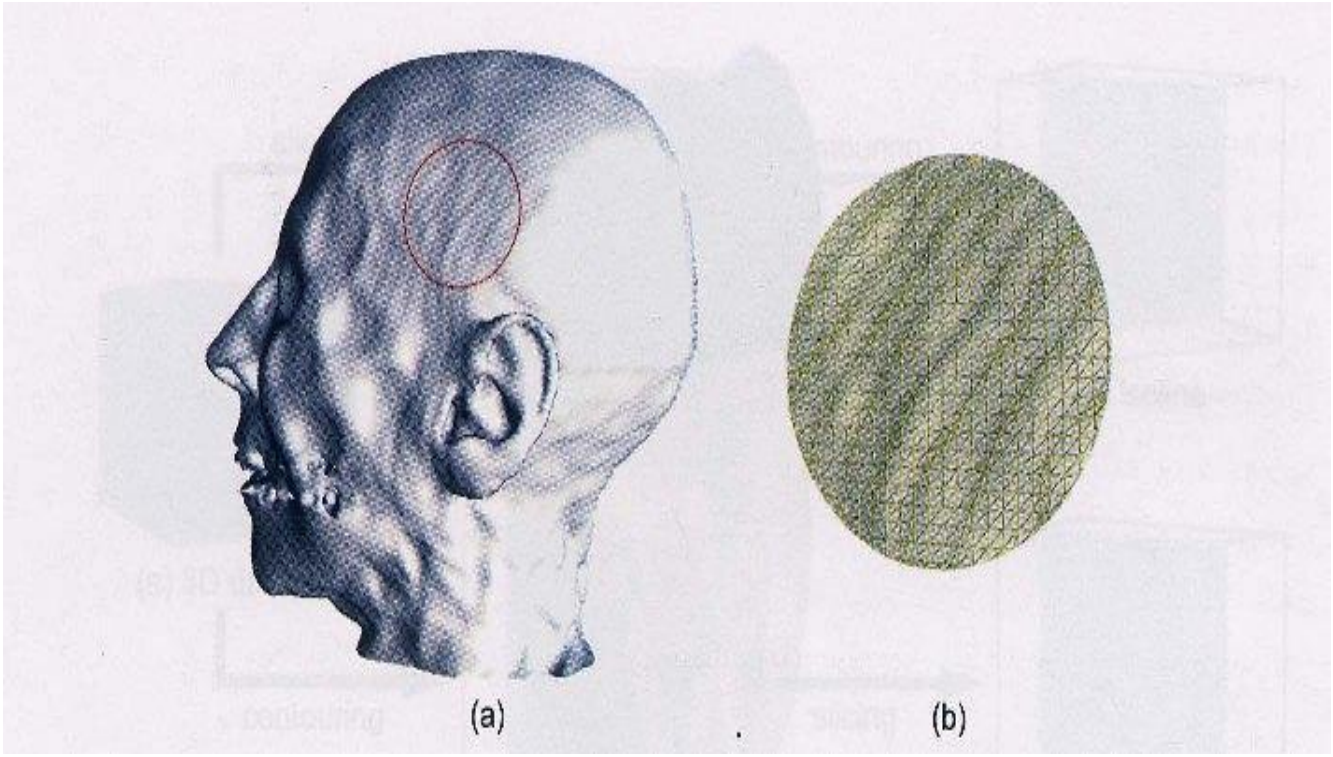
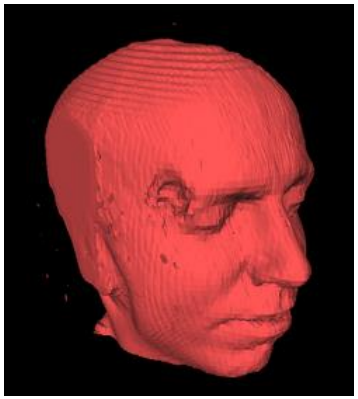
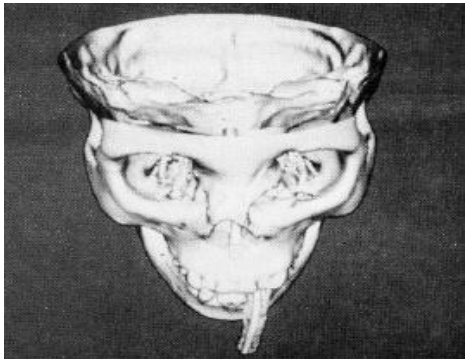
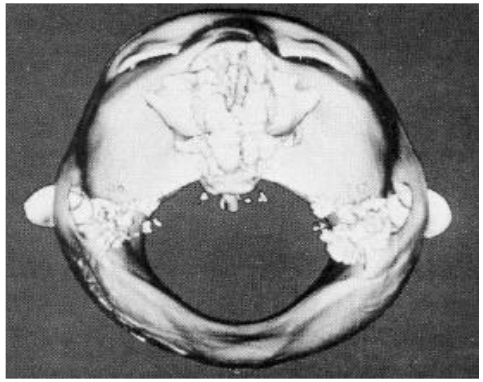


Fig 5.15. Ringing artifacts on isosurface. (a) Overview. (b) Detail mesh.

Examples







5.3.2 Marching Cubes (attentioin!)

- General rule: most isosurface details that are under or around the size of the resolution of the iso-surfaced dataset can be
  - either actual data or artifact
  - should be interpreted with great care
- Can draw more than a single iso-surface of the same dataset in one visualization (Fig 5.16)

5.3.2 Marching Cubes

Fig 5.16. Two nested isosurfaces of a tooth scan dataset

5.3.2 Marching Cubes

Isosurfaces and isolines are strongly related

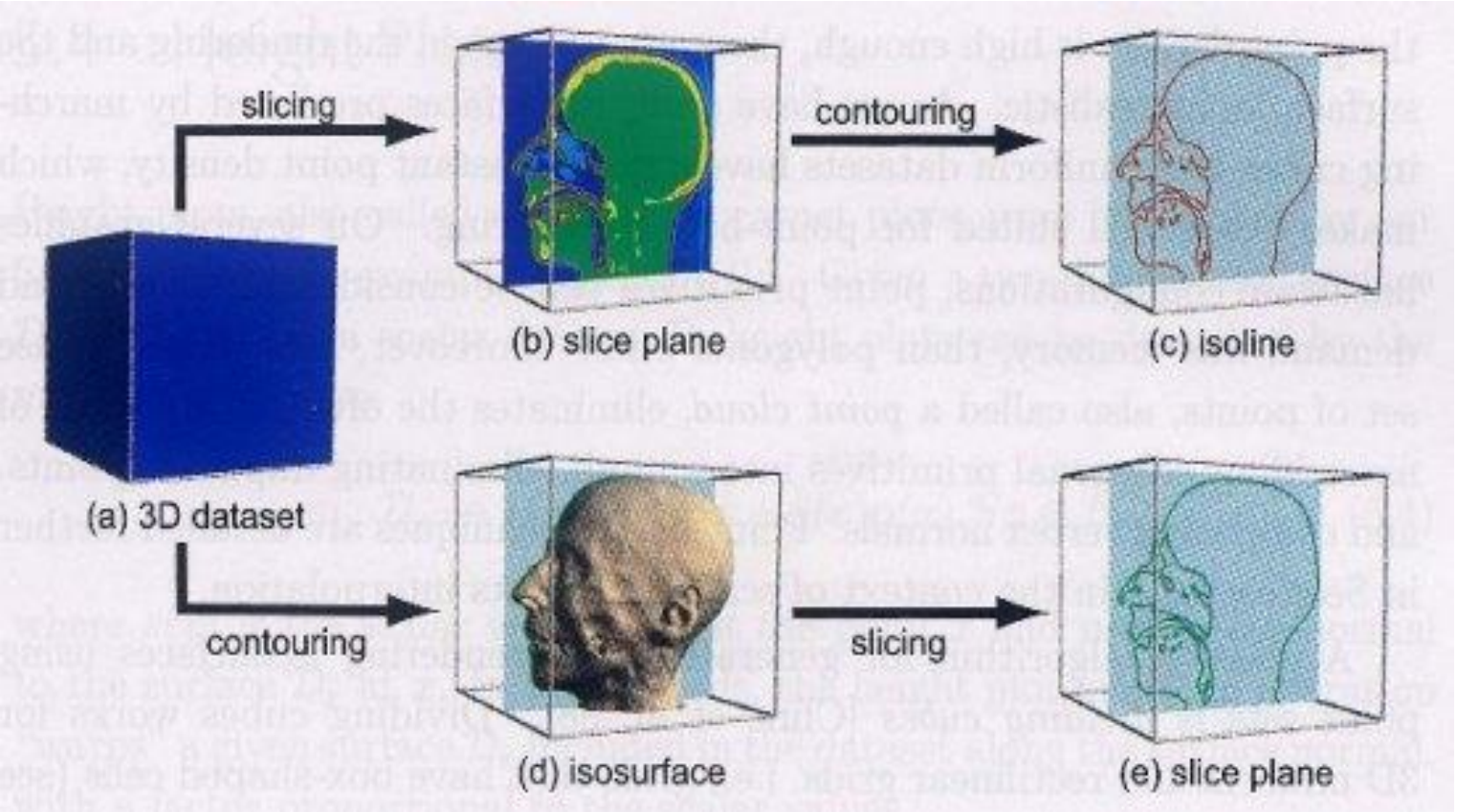


Fig 5.17. Isosurfaces, isolines, and slicing

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

Additional Resources (cont.)

- Miller, J., et al. “Geometrically Deformed Models” 1991. Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques.
- Livnat, Y., Han-Wei Shen, and C. Johnson. “A Near Optimal Isosurface Extraction Algorithm Using the Span Space” 1996. IEEE Transactions on Visualization and Computer Graphics, Vol. 2-1.

5.4 Height plots

- Height plots (elevation or carpet plots)

$$m:\ D_s \rightarrow D\ ,\ m(x) = x + s(x)\ n(x),\ \ \forall x \in D_s$$

  - S(x) is the scalar value of D at the point x
  - n(x) is the normal to the surface D_s at x

- The height plot mapping operation "warp" a given surface D_s included in the dataset along the surface normal, with a factor proportional to the scalar values.
- Height plots are a particular case of displacement, or warped plots

5.4 Height plots

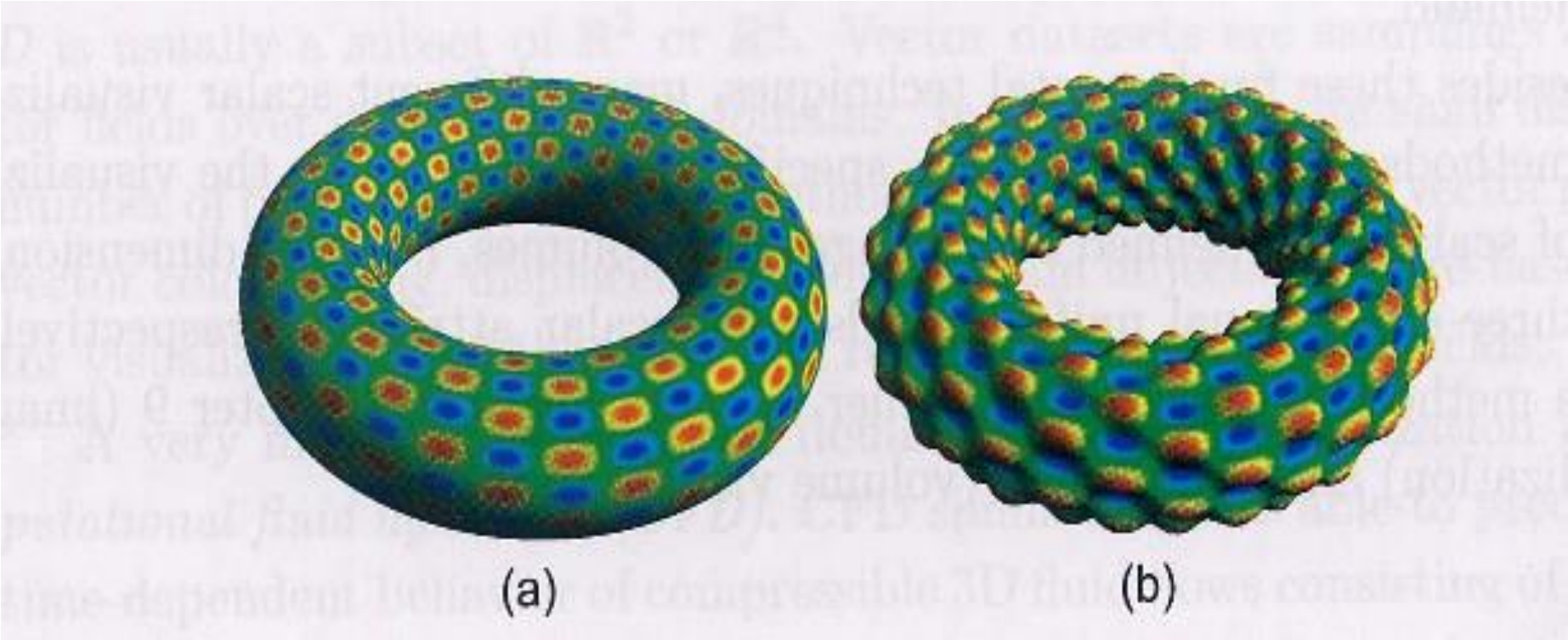


Fig 5.18. Height plot over a non-planar surface

5.5 Conclusion

- Visualizing scalar data
  - Color mapping
    - Assign a color as a function of the scalar value at each point of a given domain
  - Contouring
    - Displaying all points with a given 2D or 3D domain that have a given scalar value
  - Height plots
    - Deform the scalar dataset domain in a given direction as a function of the scalar data
- Advantage
  - Produce intuitive results
  - Easily understood by the vast majority of users
  - Simple to implement
- Disadvantage
  - A number of restrictions
    - One or two dimensional scalar dataset
    - We want to visualize the scalar values of ALL, not just a few of the data points of a 3D dataset
- Other scalar visualization method
  - Specific method for data over images or volumes