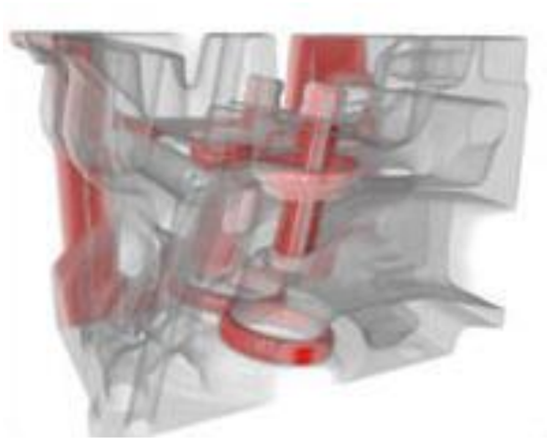
How Do We Add Color?

- We learned earlier in the term that color plays a vital role in visualization
- The skull in the right image was colored white, the skin a transparent orange color
- Where did these colors come from?
- The input is just a grayscale, 8-bit volume, after all
- Also, what happened to all the blood vessels, muscles, and other biological structures? They seem to have disappeared, but we know they were in the original data-set
- Evidently there was some sort of mapping of density to color and opacity
- Skull → white
- Skin → transparent orange
- Flesh → 100% transparent (0% opacity)

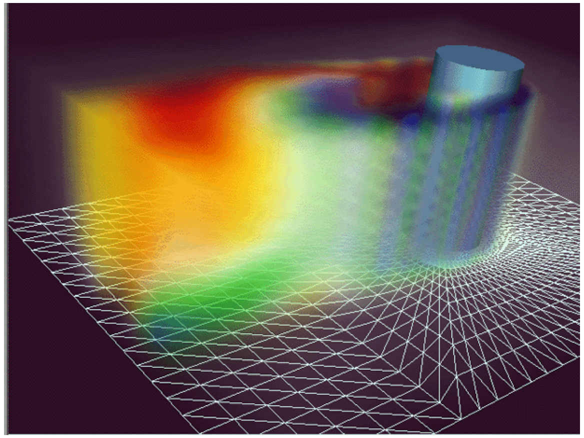
Classification

- So, in general, a voxel stores some density value
- This density can have many meanings, depending on the origin of the dataset:
  - stress, strain, temperature (finite element applications, numerical simulations)
  - X-ray absorption of a material (computed tomography (CT))
  - magnetic spin relaxation property of a material (MRI)
  - a material tag (voxelization)
- We would like to give certain aspects of the dataset meaningful visual attributes, such as colors
- For this purpose, the raw values have to be translated into colors and attenuation properties (opacity)
- These assignments are made based on the voxel's material
- This assignment process is formally called *classification*



STONY BROOK
STATE UNIVERSITY OF NEW YORK

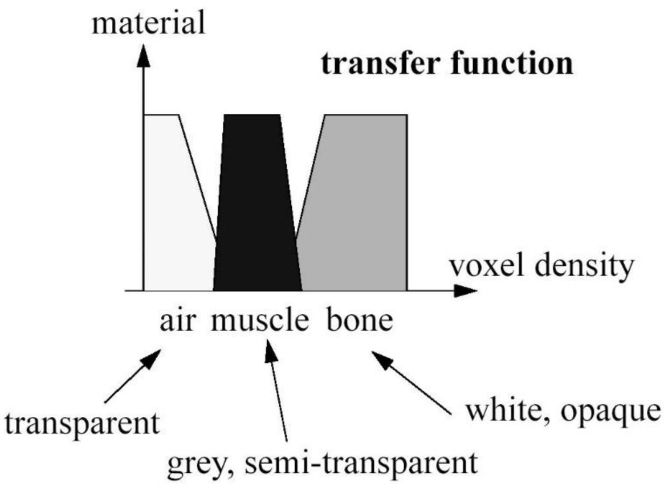
Classification

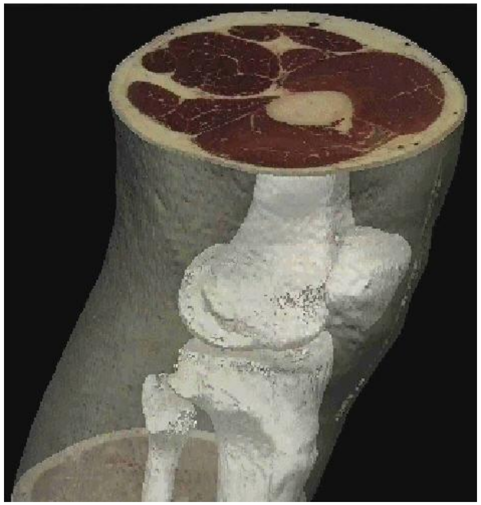


- *Classification* is the task of assigning an (R,G,B,A) tuple to an 8-bit density
- But how do we do this?
- We use a set of four *transfer functions* that map density to R, G, B and A
- In X-ray rendering, each interpolated sample along the ray was assigned the same weight; all we did was add them up
- Now we will let the user decide how much each density level will contribute to the final image
- Example: if we want to visualize high densities (e.g., bone), we would assign a high opacity to such densities, and assign lower opacities or even zero opacity (full translucency) to lower densities
- Let's see an example

Classification

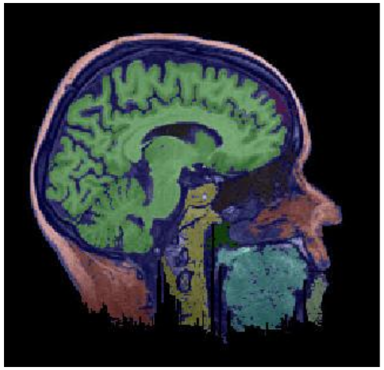
- In simple cases (such as CT), each material has a characteristic raw density range





- Some datasets (for example MRI) do not have unique density-material correspondences
- These require more sophisticated segmentation methods (seed growing, clustering)

segmentation



render

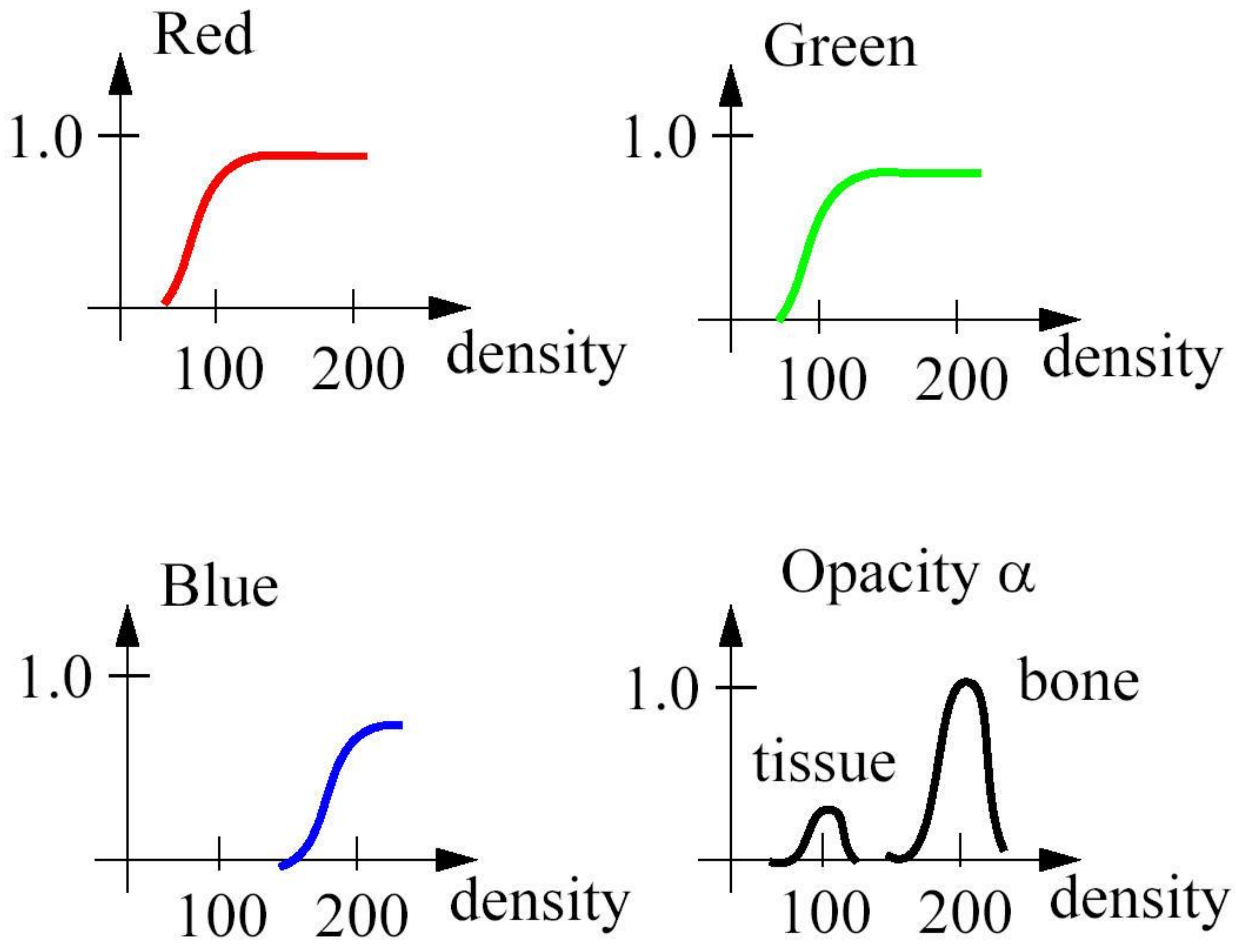
STONY BROOK
STATE UNIVERSITY OF NEW YORK

Transfer Functions

- Think of the volume as a transparent gel that we are looking through
- With X-ray rendering, each sample along the rays are assigned the same importance, in some sense
- Low densities contribute to the final image as much as high densities
- Transfer functions allow us to change the relative contributions of densities
- So, if we wanted to make high densities contribute more (e.g., to see bone), we should assign higher importance to those densities
- In volume rendering, this importance is assigned by the opacity
- So, if we want to view bone, we assign high opacity to densities corresponding to bone
- As the viewing rays traverse the volume, the system will use those opacities (and colors) to modulate the pixel value accordingly
- So even though the bone was given high opacity, since the skin was assigned non-zero opacity and the color orange, the skull has an orange cast to it

STONY BROOK
STATE UNIVERSITY OF NEW YORK

Transfer Functions



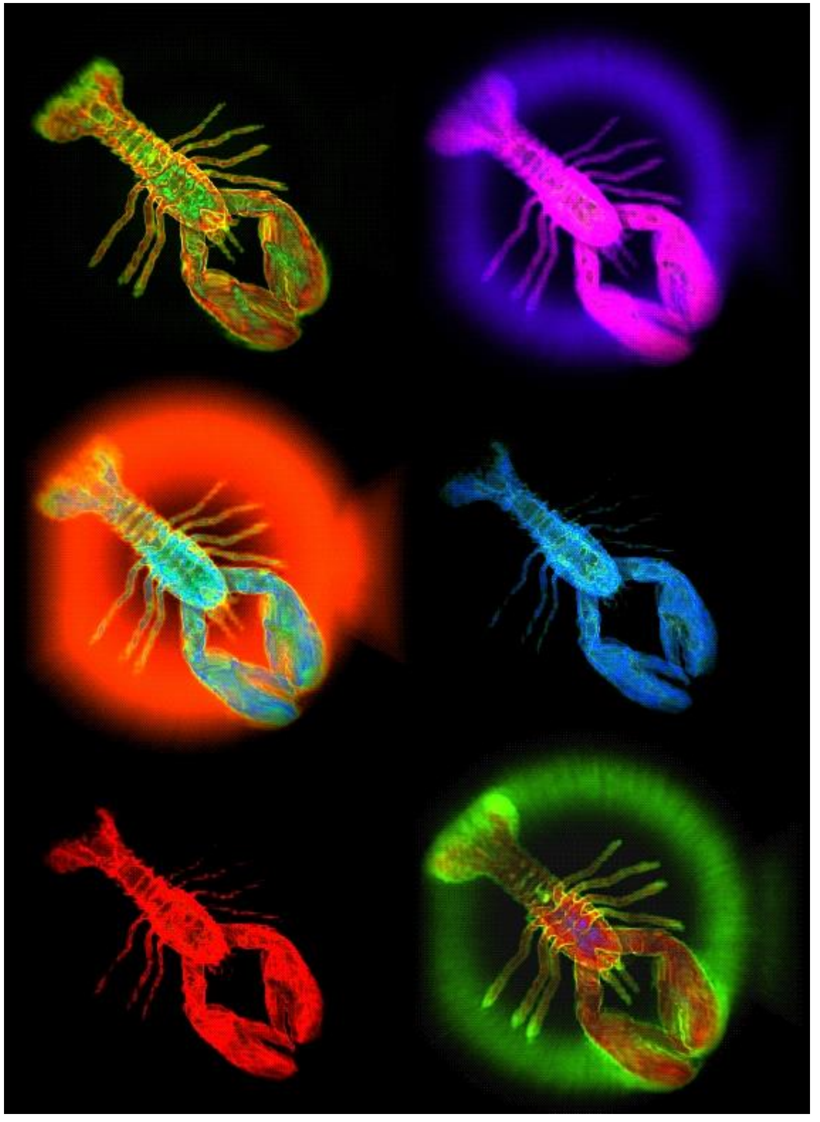
Transfer Functions

- Notice how transfer functions are similar to intensity transformations
- In both cases, the input is density (or intensity)
- For intensity transformations, the output is also intensity
- For transfer functions, the output is either color or opacity
- Unlike in X-ray rendering, can now assign arbitrary, non-negative weights to voxels and generate interesting effects
- Interpolate density then assign color
- So what happens when our viewing ray accumulates full opacity (1.0)?
- In other words, it could happen that half-way through traversing the volume, a ray hits full opacity
- What this means is every other sample we encounter along the ray will be occluded by the opaque structures in front that we have already processed
- This is why we can't see the back side of the skull
- Early ray termination optimization

STONY BROOK
STATE UNIVERSITY OF NEW YORK

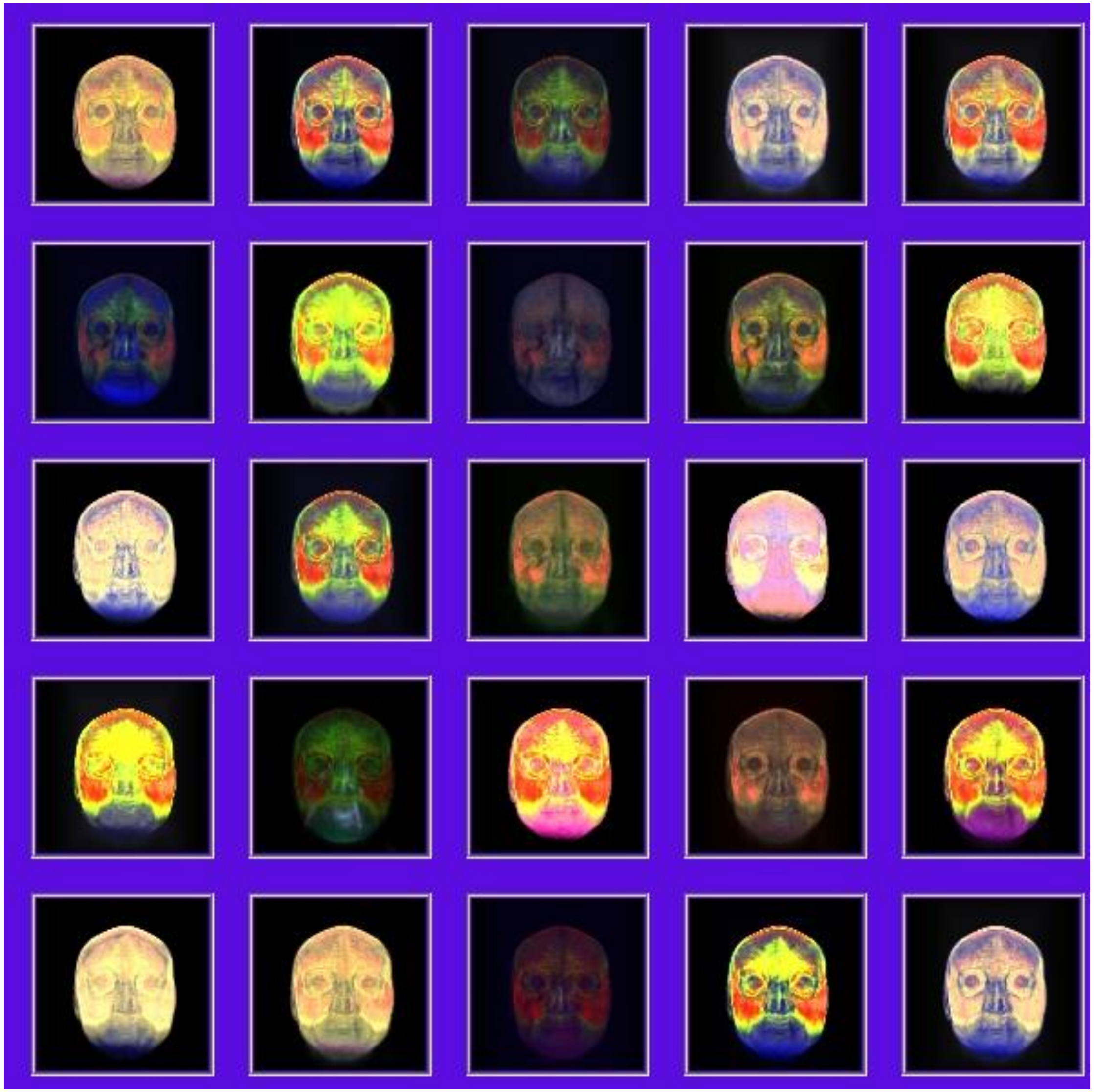
Transfer Function Design Galleries

- Transfer functions are somewhat unintuitive to use at times
- Remember our motivation for histogram equalization?
- That gave us an automatic process
- Unfortunately, there is no such automatic process for transfer function generation because of the wide variety of volumetric data-sets and the wide range of features people wish to visualize from data-sets
- But, there is one way in which a computer can help us: so-called *transfer function design galleries*
- A set of randomly-generated transfer functions that the system mutates (based on user input) to generate a new gallery



STONY BROOK
STATE UNIVERSITY OF NEW YORK

Transfer Function Design Galleries



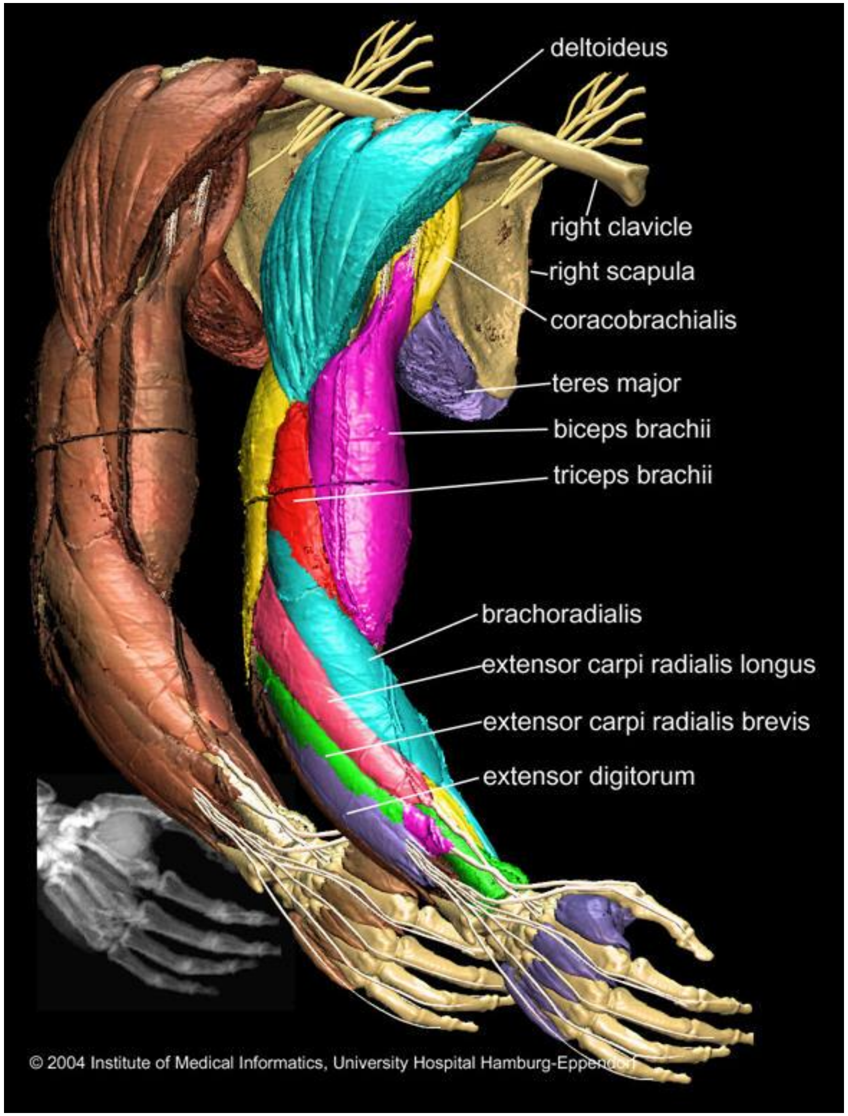
STONY BROOK
STATE UNIVERSITY OF NEW YORK

Visible Male Visualization

- Let's see a movie of transfer functions being used at real time to visualize the visible male data-set (6)
- Watch the bottom of the two graphs, which shows the α transfer function

Segmentation

- Related to the task of classification is *segmentation*, the process of extracting features from a data-set
- Basically, segmentation assigns a material label to each voxel in a data-set
- e.g., a voxel value of 100 might indicate that a particular voxel is muscle, but which muscle? Does a density of 200 indicate a tooth or the jaw in which the tooth is located?
- Classification can't answer these questions
- Segmentation – done manually or with some algorithm – partitions a data-set into logical pieces



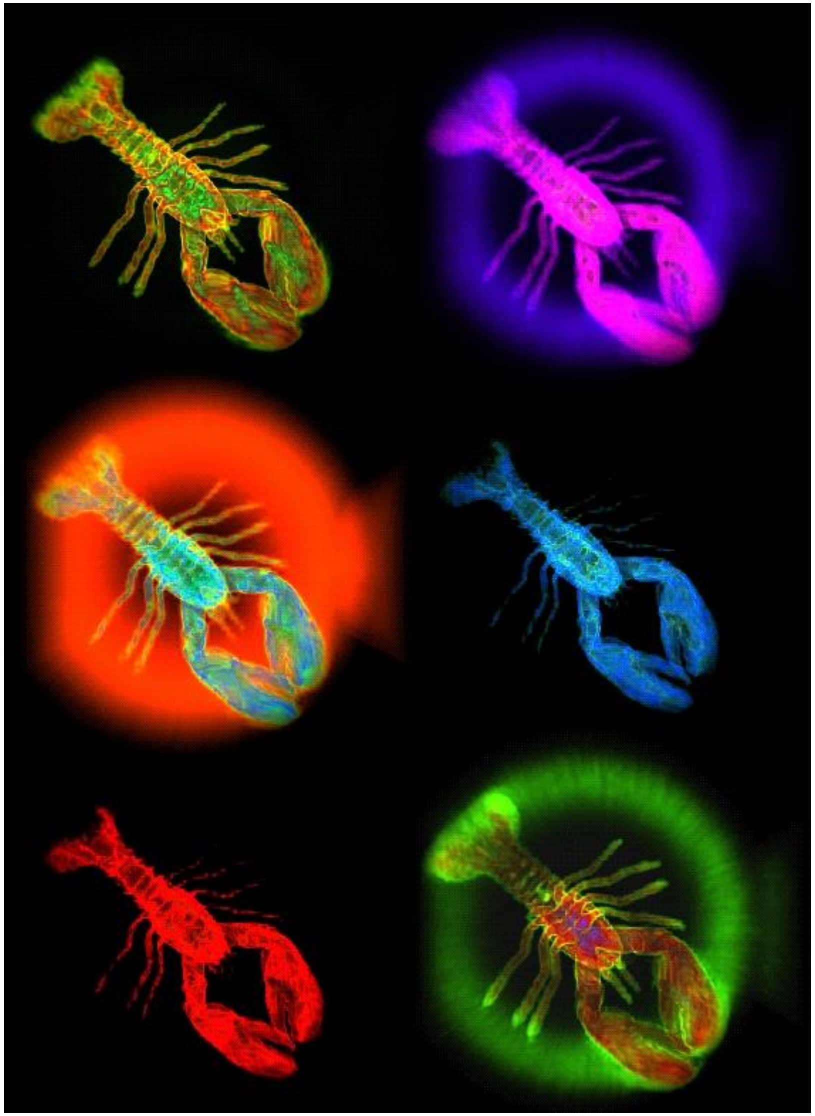
STONY BROOK STATE UNIVERSITY OF NEW YORK

Segmentation

- Segmentation is, in general, an extremely difficult and potentially time-consuming process
- There is no single segmentation algorithm that can solve all or even most segmentation problems
- Almost without exception, a certain level of human intervention is needed at some point during execution
- Numerous algorithms for each particular application
- Can you think of some reasons why there is no single, really good way of segmenting data? After all, classification, despite its faults, is actually a pretty solution to the problem it addresses (assigning color to voxels)
- Consider the many domains in which volume rendering is used
- Many sources of data
- Many, many differences in the underlying structures of data
- Just look at the human body, especially the abdomen and all the organs and complex structures (branching lungs/blood vessels, tubular gastrointestinal tract, etc.)
- Movie of brain tumor segmentation (4)

How About Shading

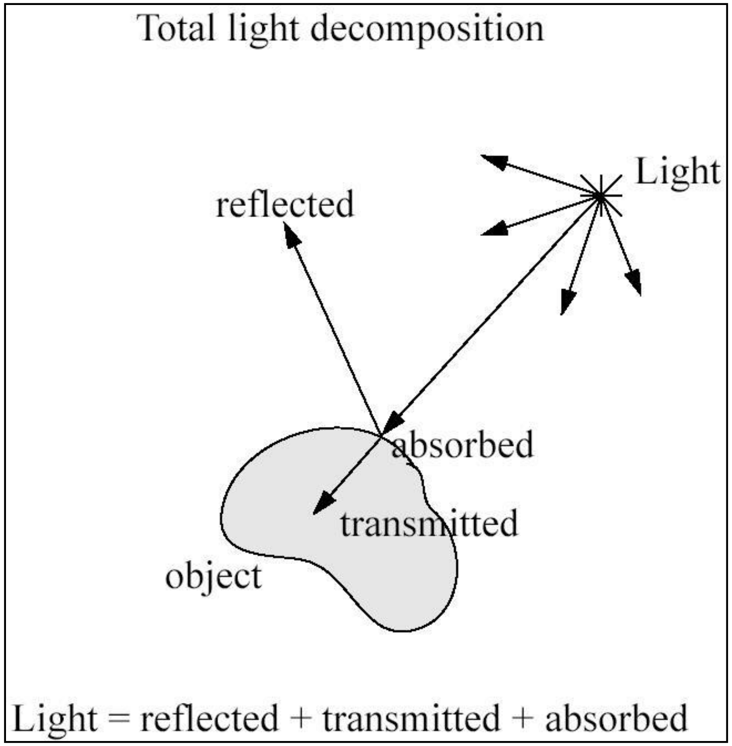
- We haven't seen yet how to employ shading with volume rendering to make objects look 3D
- In fact, we aren't required to shade our volumetric objects, but it sure helps in revealing structure
- The lobsters on the right are not shaded
- Note that we can distinguish different materials, but it's hard to discern 3D shape
- So, we can use transfer functions in volume rendering and omit shading, but let's take a look at how we *can* incorporate shading into volume rendering
- Let's look at some traditional computer graphics shading techniques and extend them to volumetric ray-casting



STONY BROOK STATE UNIVERSITY OF NEW YORK

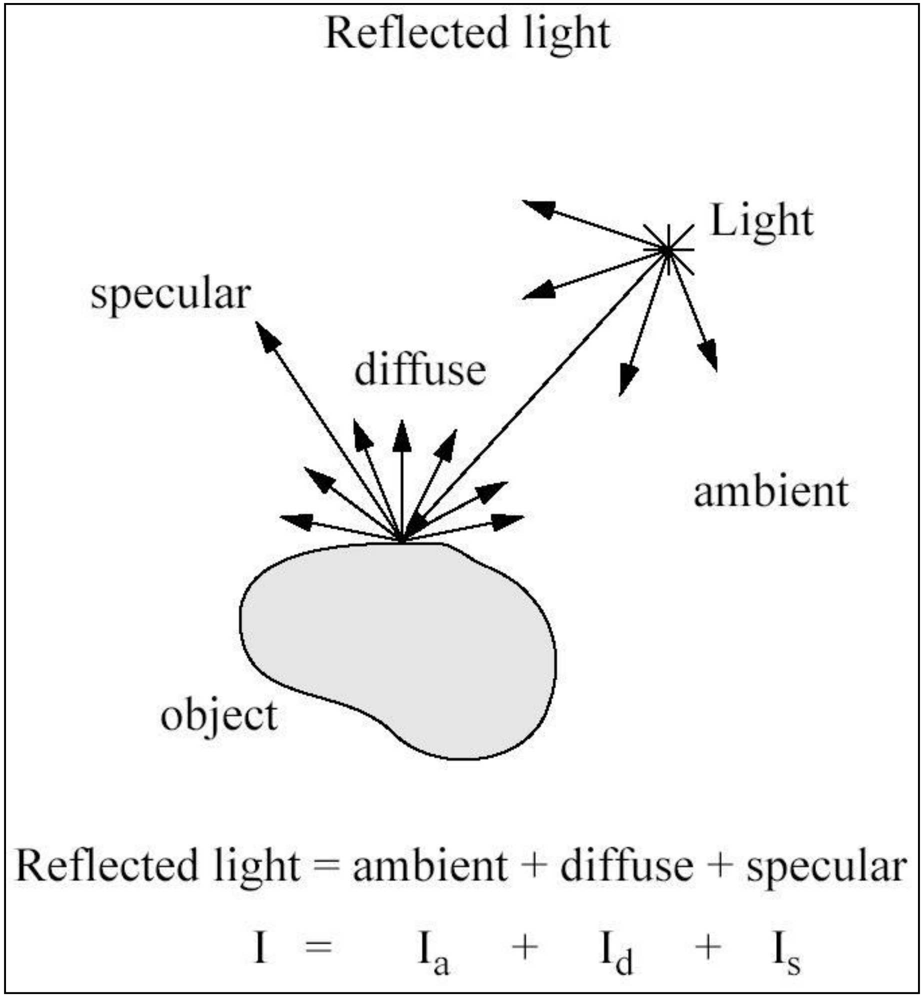
Illumination and Shading

- Now we'll look at how to shade **surfaces** to make them look 3D
- We'll see different *shading models*, or frameworks that determine a surface's color at a particular point
- These shading models can be easily modified to incorporate illumination and shading into the volume rendering pipeline
- A shading model checks what the lighting conditions are and then figures out what the surface should look like based on the lighting conditions and the surface parameters:
- Amount of light reflected (and which color(s))
- Amount of light absorbed
- Amount of light transmitted (passed through)
- Thus, we can characterize a surface's shading parameters by how much incoming light that strikes a surface is reflected to the eye, absorbed by the object, and transmitted



Reflected Light

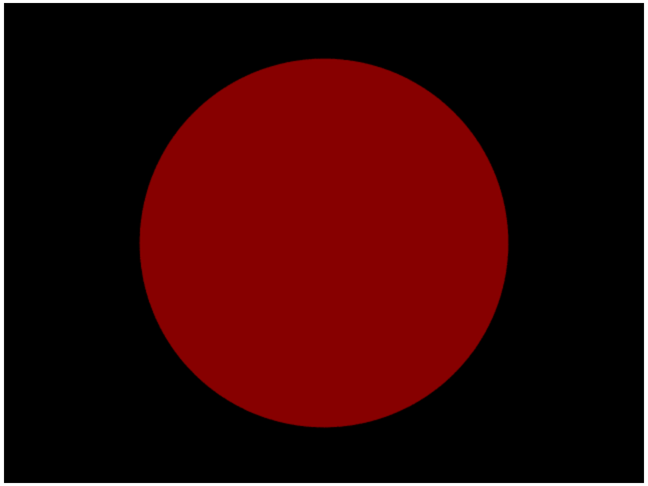
- Typically in computer graphics, unless we are trying to create effects like refraction, diffraction and translucency, we are mostly concerned with the reflected light – that light which bounces off the object and enters the eye (camera, really)
- We'll use equations to make it easy to compare one shading model to another



STONY BROOK
STATE UNIVERSITY OF NEW YORK

Ambient Reflection

- *Ambient reflection* refers to reflected light that originally came from the "background" and has no clear source
- Models general level of brightness in the scene
- Accounts for light effects that are difficult to compute (secondary diffuse reflections, etc)
- Constant for all surfaces of a particular object and the directions it is viewed from
- Directionless light
- One of many hacks or kludges used in computer graphics since every ray of light or photon has to come from somewhere!

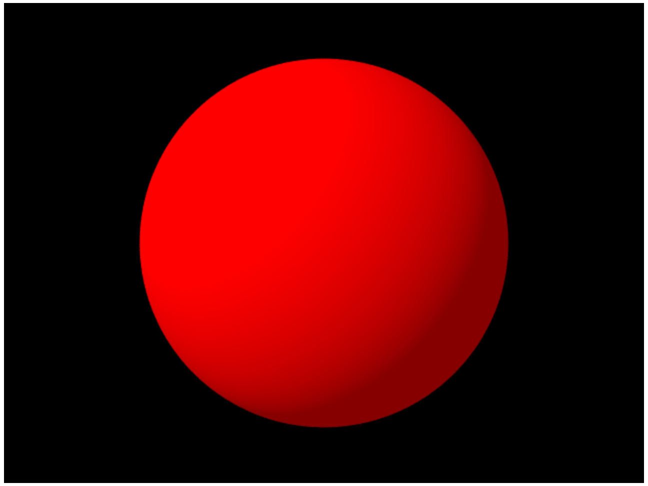


Ambient-lit sphere

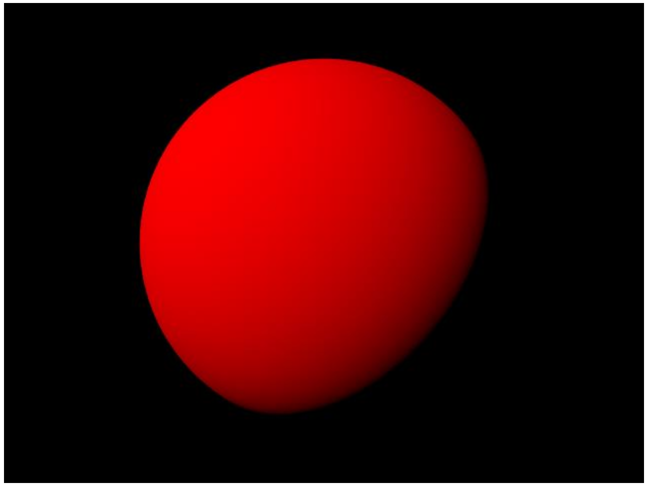
- Imagine yourself standing in a room with the curtains drawn and the lights off
- Some sunlight will still get through, but it will have bounced off many objects before entering the room
- When an object reflect this kind of light, we call it *ambient reflection*
- $I_a = k_a \cdot I_A$ I_A = ambient light k_a = material's ambient reflection coefficient

Diffuse Reflection

- Models dullness, roughness of a surface
- Equal light scattering in all directions
- For example, chalk is a diffuse reflector
- Unlike ambient reflection, diffuse reflection is dependent on the location of the light relative to the object
- So, if we were to move the light from the front of the sphere to the back, there would be little or no diffuse reflection visible on the near side of the sphere
- Compare with ambient light, which has no direction
- With ambient, it doesn’t matter where we position the camera since the light source has no true position
- Computer graphics purists don’t use ambient lights and instead rely on diffuse light sources to give some minimal light to a scene



Ambient & diffuse



Diffuse only

STONY BROOK
STATE UNIVERSITY OF NEW YORK

Diffuse Reflection

- Diffuse reflection is also called *Lambertian reflection*
- Lambertian cosine law:

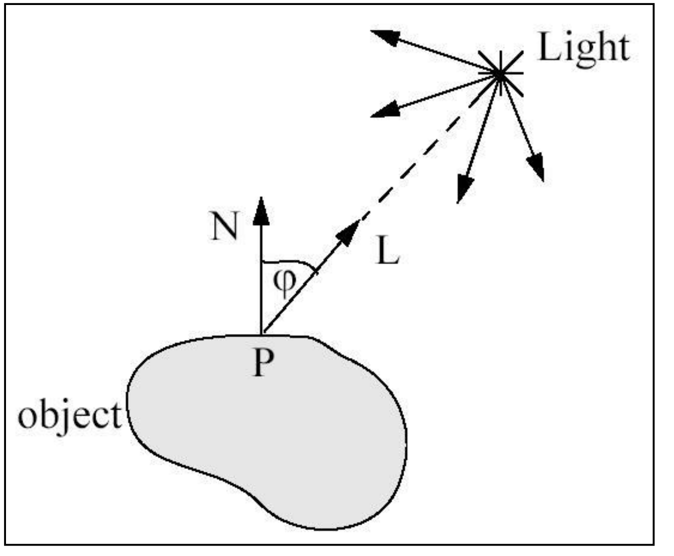
$$I_d = k_d I_L \cos\varphi = k_d I_L N \cdot L$$

dot product: $(N_x \cdot L_x + N_y \cdot L_y + N_z \cdot L_z)$

- I_L: intensity of light source
- N: surface normal vector
- L: light vector (unit length)

$$L = \frac{Light - P}{|Light - P|} = \left(\frac{Light_x - P_x}{|L'|}, \frac{Light_y - P_y}{|L'|}, \frac{Light_z - P_z}{|L'|} \right)$$

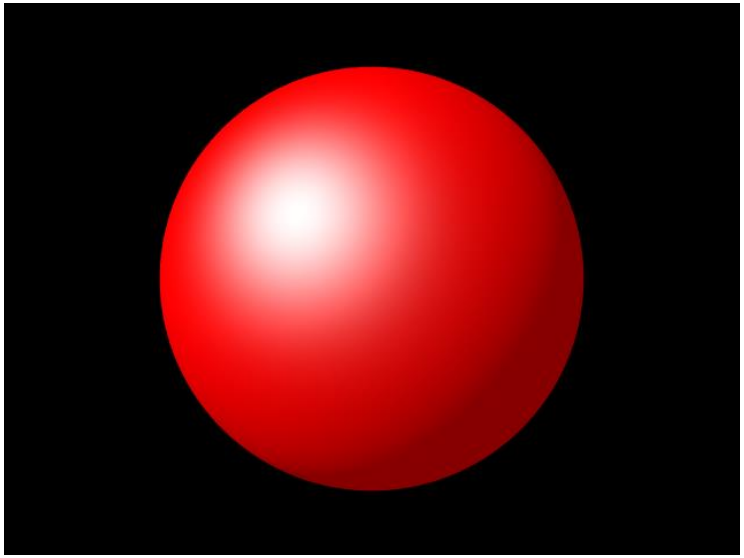
$$|L'| = \sqrt{(Light_x - P_x)^2 + (Light_y - P_y)^2 + (Light_z - P_z)^2}$$



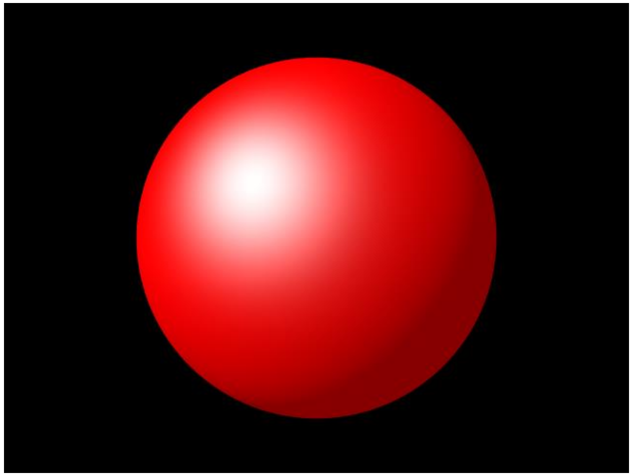
- φ: angle of light incidence
- k_d: diffuse reflection coefficient (material constant)
- Note: $I_d = 0$ for $N \cdot L < 0$
- What does this inequality mean intuitively?

Specular Reflection

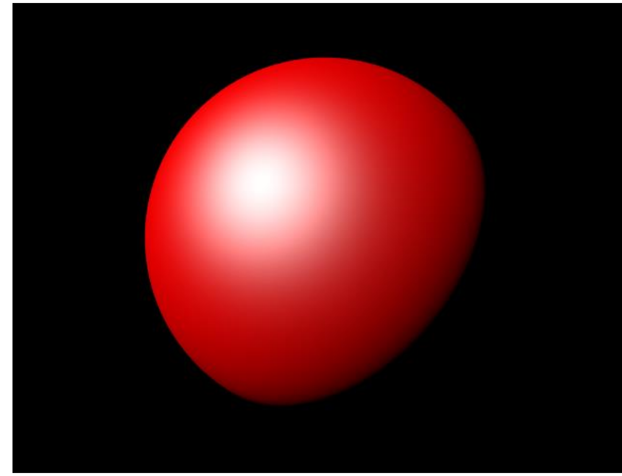
- Models reflections on shiny surfaces (polished metal, chrome, plastics, etc.)
- Specular reflection is *view-dependent* – the specular *highlight* will change as the camera's position changes
- This implies we need to take into account not only the angle the light source makes with the surface, but the angle the viewing ray makes with the surface
- Example: the image you perceive in a mirror changes as you move around
- Example: the chrome on your car shines in different ways depending on where you stand to look at it



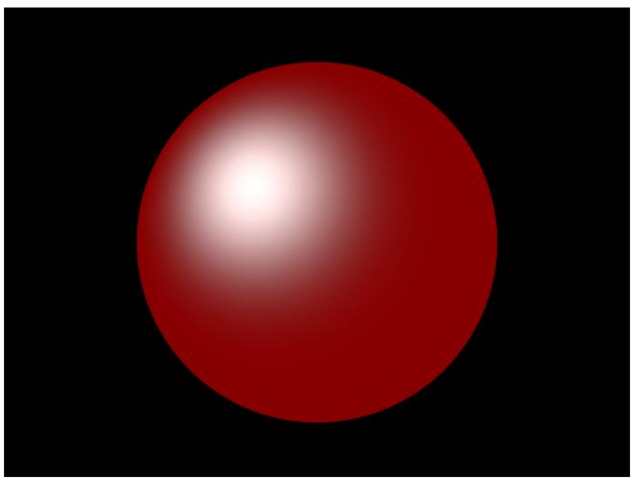
Specular Reflection



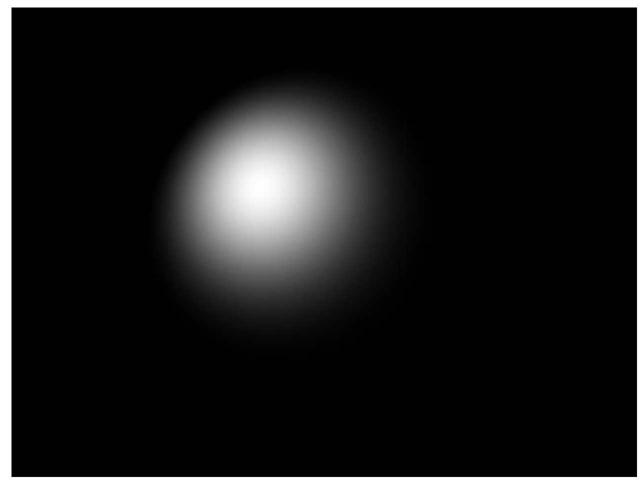
Specular & diffuse & ambient



Specular & diffuse



Specular & ambient

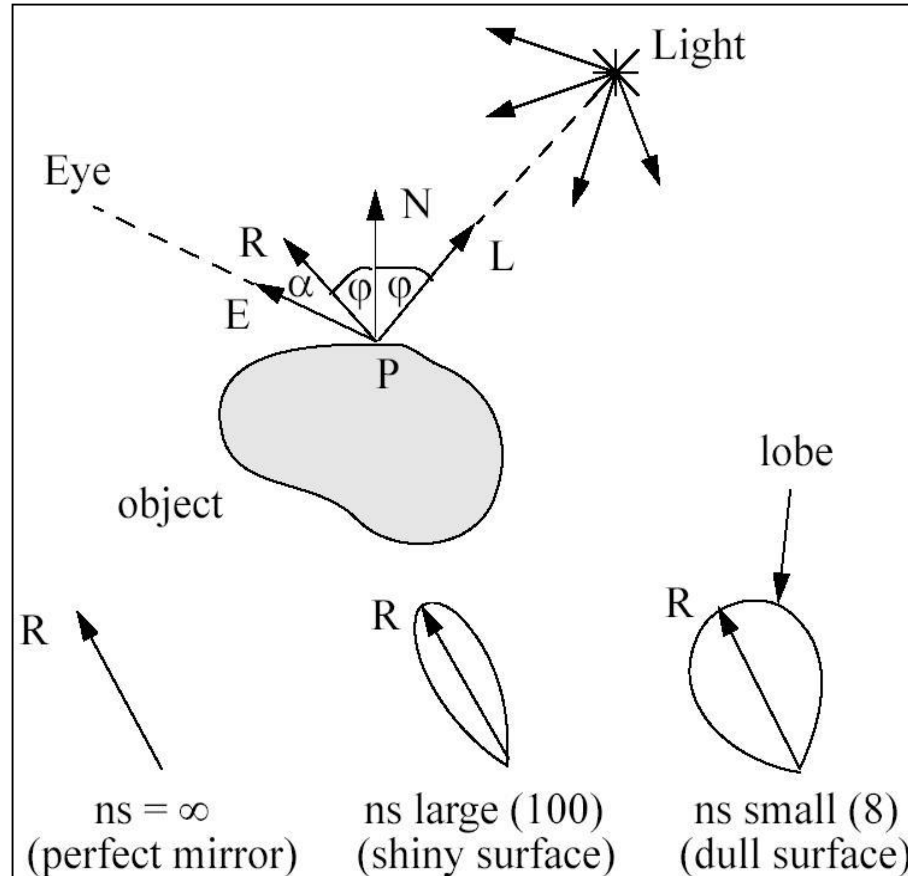


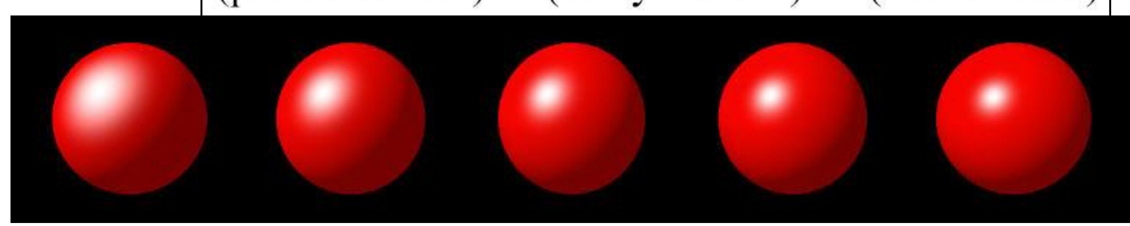
Specular only

- Ideal specular reflector (perfect mirror) reflects light only along reflection vector R
- Non-ideal reflectors reflect light in a lobe centered about R
- Phong specular reflection model:

$$I_s = k_s\, I_L\, (\cos \alpha)^{ns} = k_s\, I_L\, (E \cdot R)^{ns}$$

- cos(α) models this lobe effect
- The width of the lobe is modeled by Phong exponent ns, it scales cos(α)
- I_L: intensity of light source
- L: light vector
- R: reflection vector = 2 N (N·L) - L
- E: eye vector = (Eye-P) / |Eye-P|
- α: angle between E and R
- ns: Phong exponent
- k_s: specular reflection coefficient





increasing *ns* value →

Total Reflected Light

- Total reflected light (for a white object):

$$I = k_a I_A + k_d I_L N \cdot L + k_s I_L (E \cdot R)^{ns}$$

- Multiple light sources:

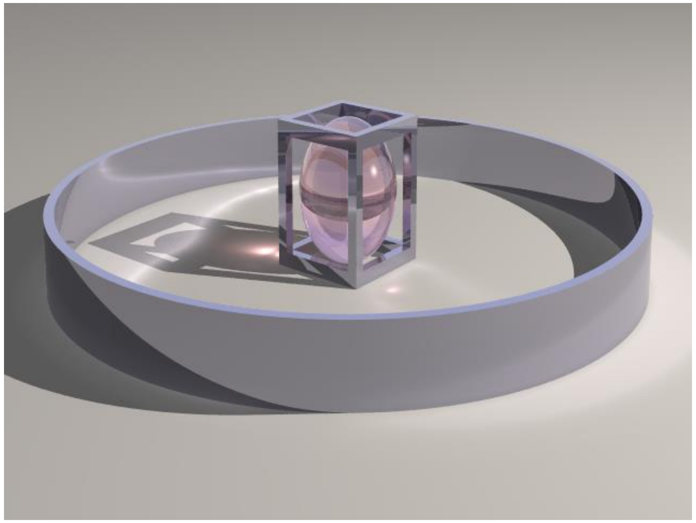
$$I = k_a I_A + \sum_i (k_d I_i N \cdot L_i + k_s I_i (E \cdot R_i)^{ns})$$

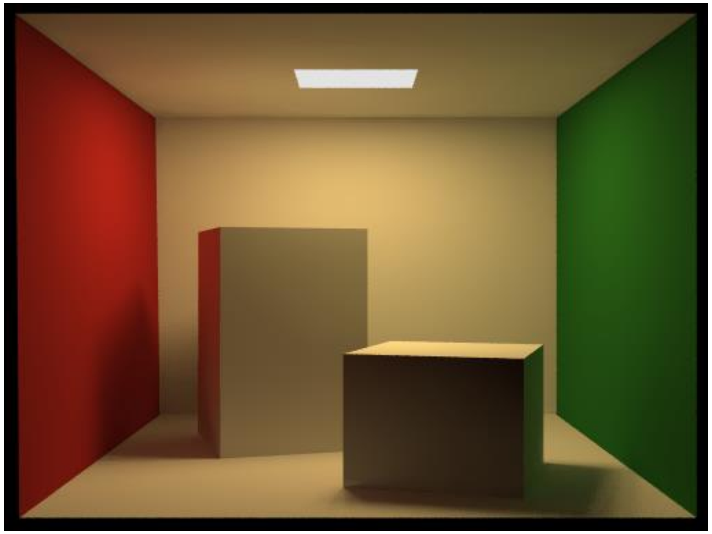
- Usually, I is a color vector of RGB
- Object has color $C_{obj} = (R_{obj}, G_{obj}, B_{obj})$
- Object reflects I, modulated by C_{obj}
- Color C reflected by object:

$$C = C_{obj}(k_a I_A + \sum_i (k_d I_i N \cdot L_i)) + \sum_i (k_s I_i (E \cdot R_i)^{ns})$$

- In many applications, the specular color is not modulated by object color
  - specular highlight has the color of the light source
- k_s, k_d, k_a in [0.0, 1.0]
- R,G,B in [0.0, 1.0]. Remapped to [0, 255] for display
- See Foley chapter 16 for the various formulae and computations

Other Shading and Illumination Concepts/Effects

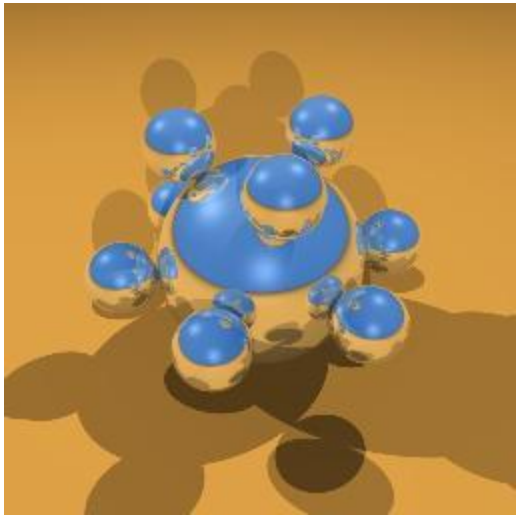


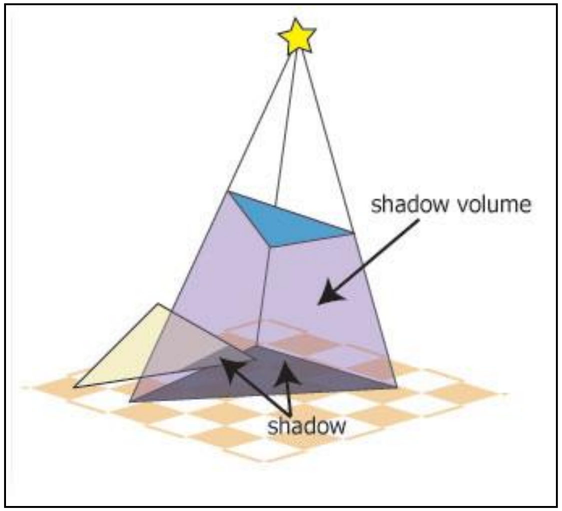


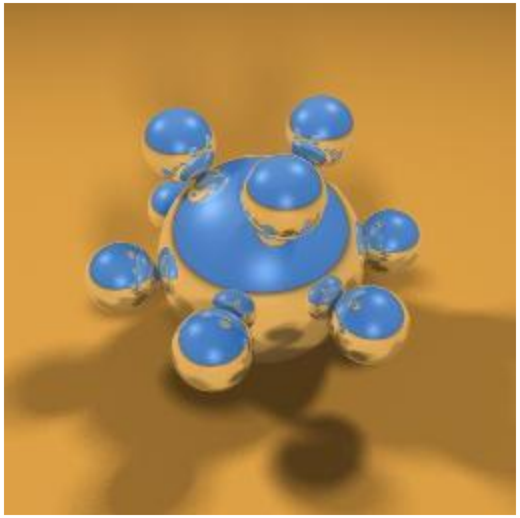
- Area lights
- Shadows
- Refraction
- Reflection
- Caustics
- Color bleeding
- Radiosity
- Camera effects
- …and many, many more!
- Most fall under the general area of *global illumination*, which is capable of generating all those *photorealistic* images you see in movies and special effects
- Most require the use of *ray-tracing*, rather than *ray-casting*, and radiosity
- Want to try it yourself? Go to www.povray.org and try out the free POV-Ray ***surface*** ray-tracing program
- Later in the term we may have time to see how these sophisticated effects can be generalized and used in volume visualization

Shadows

- Hard shadows and soft shadows
- Hard shadows: caused by very distance light sources, like the sun
- Soft shadows: caused by close light sources, usually area light sources, like light bulbs
- Different techniques for generating shadows
- Depend on whether we are using traditional graphics rendering techniques or ray-tracing
- Ray-tracing techniques often employ *shadow volumes*
- Cast rays from *light source* to the object: occluded objects lie in a volumetric region of space that is in shadow

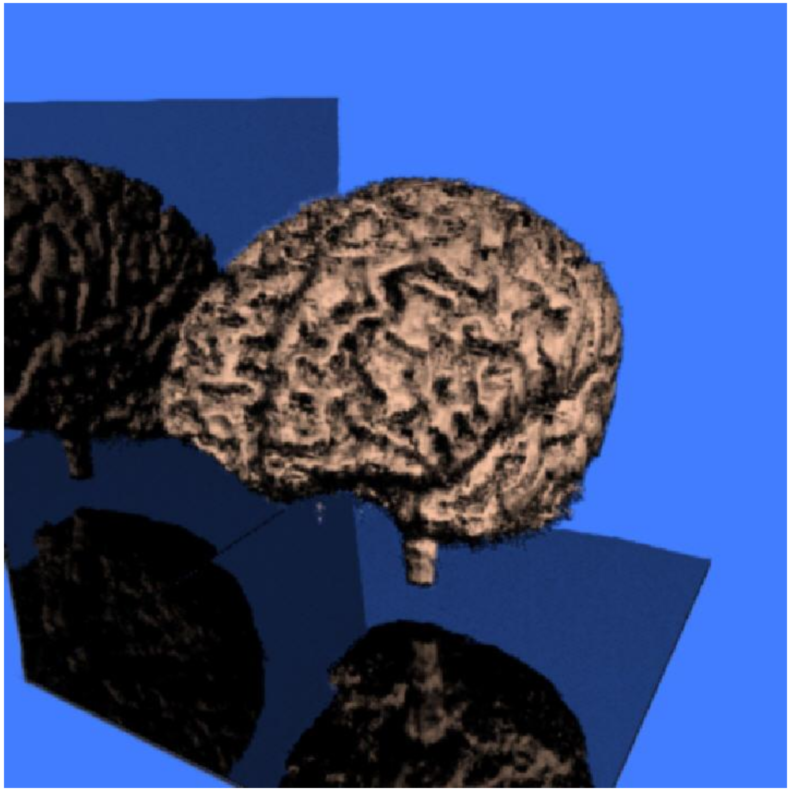


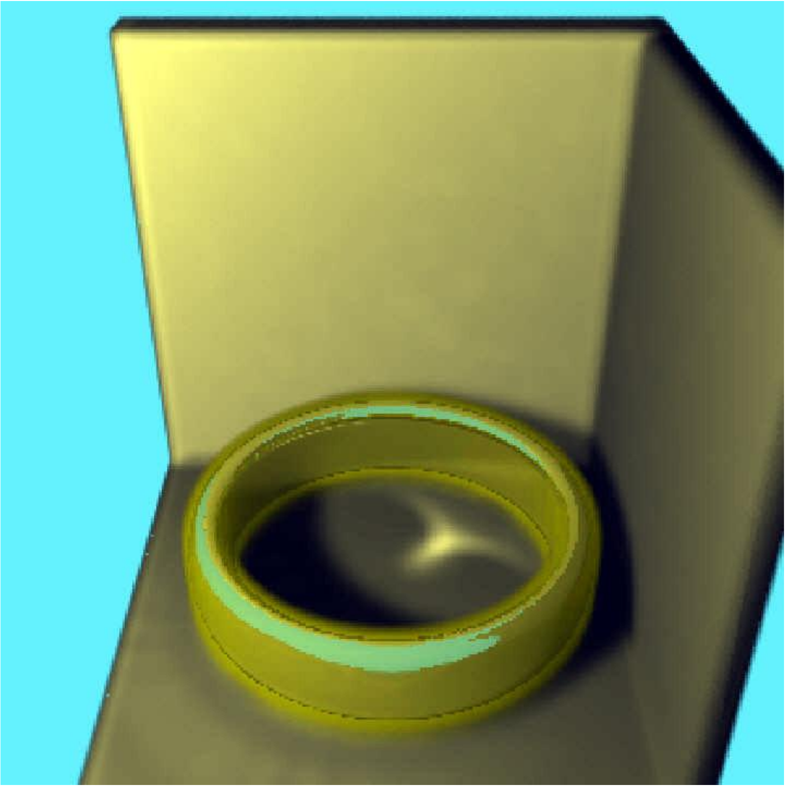




STONY BROOK STATE UNIVERSITY OF NEW YORK

Global Illumination for Volume Visualization



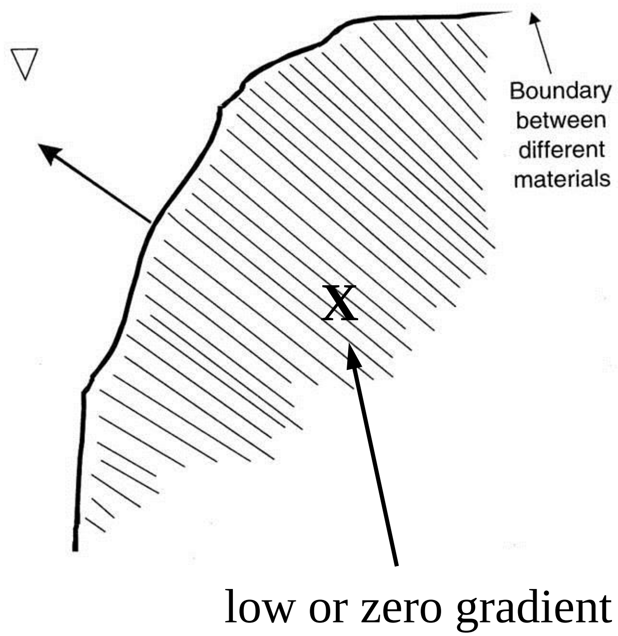


Integrating Shading Model with Volume Visualization

- All of these shading models can be adapted to generate very effective volume renderings
- In polygon rendering, we compute the shading information for each vertex or pixel
- In volume rendering, we perform the shading calculations for each voxel
- But what about the normal vectors?
- We can assign a direction at each voxel using the voxel's *gradient*, which is the indication of greatest change of the dataset
- Let's look at what the gradient is and how to compute it

What is the Gradient

- Gradient is a vector that measures how quickly voxel intensities in a data set change
- Evaluated at some 3D point in space
- Useful for revealing certain characteristics about the data set
- Consider the engine with the two different types of metal of differing densities
- The gradient will be high at those voxels at the boundary where the two metals meet
- In regions where the material is constant, the gradient is zero because there is no change
- Hence, gradient is the vector of first derivatives in the *x*, *y* and *z* directions



Gradient Definition

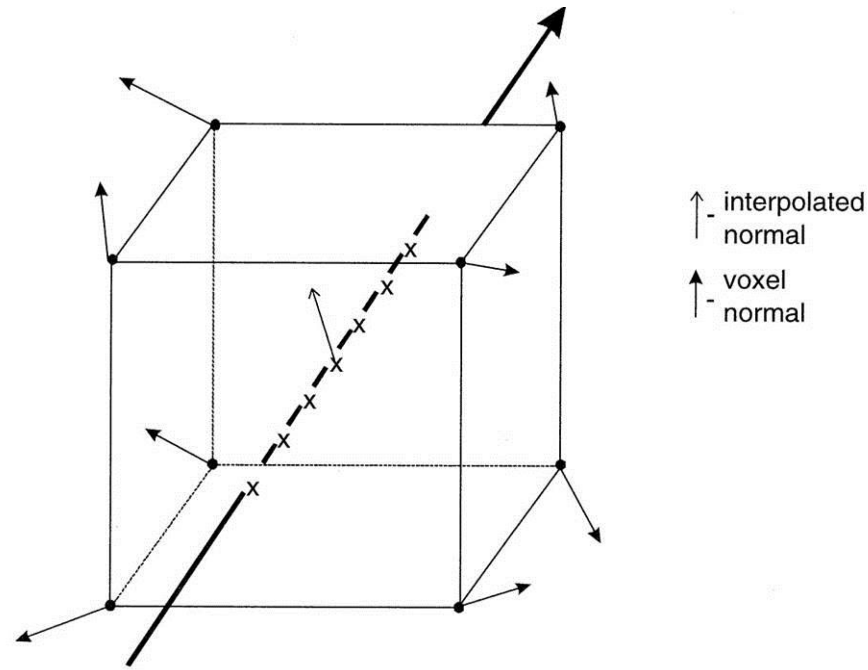
- Let's define a function $f\,(x,y,z)$ that returns the value of the data-set at the given position
- If (x,y,z) lies on the grid, then we just return its value
- Otherwise, interpolate the value
- The gradient is therefore:

$$\nabla f(x,y,z) = \begin{bmatrix} \frac{df}{dx} \\ \frac{df}{dy} \\ \frac{df}{dz} \end{bmatrix}$$

STONY BROOK
STATE UNIVERSITY OF NEW YORK

Gradient Interpolation

- As we march along the ray, we interpolate the densities in order to assign colors to samples
- We can also interpolate the gradients
- At the start of the algorithm, we compute the gradient at each voxel
- During ray traversal, we employ trilinear interpolation to estimate the gradient at the sample position
- Note that gradient is invariant of the illumination model, light positions, etc.
- Like the densities, the gradients are intrinsic attributes of the models
- In contrast, colors and opacities can be changed at run-time since they are actually not part of the data



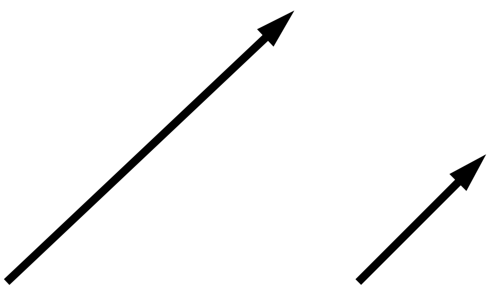
- Later we will look at *compositing* more closely to see exactly how we mathematically accumulate colors and opacities, and how shading is incorporated

STONY BROOK
STATE UNIVERSITY OF NEW YORK

Gradient Magnitude

- The gradient gives both the direction and magnitude of the rate of change in the intensities
- Sometimes we need only the magnitude, sometimes just the direction, sometimes we need both
- It depends on the volume rendering algorithm we are using and which stage of the algorithm we are currently executing
- Gradient magnitude:

$$\left|\nabla f(x,y,z)\right| = \sqrt{\left(\frac{df}{dx}\right)^2 + \left(\frac{df}{dy}\right)^2 + \left(\frac{df}{dz}\right)^2}$$



Same direction, different magnitude

Same magnitude, different direction

Gradient Computation

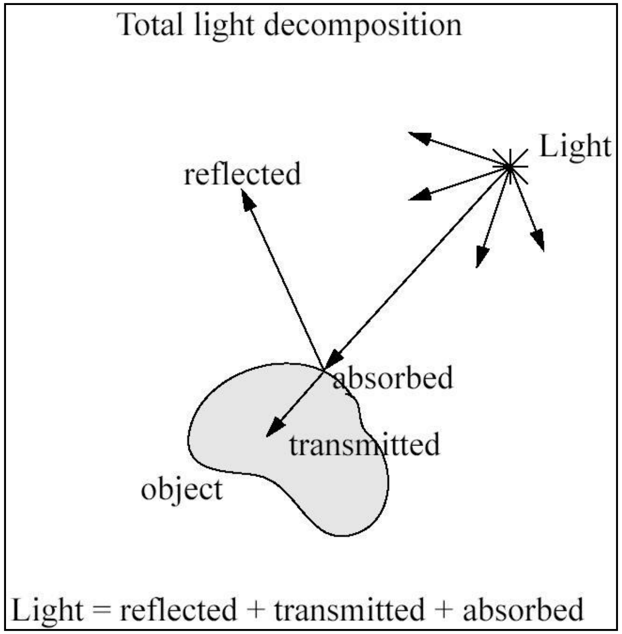
- In calculus, often we can calculate the gradient analytically because we are given a continuous function
- 99% of the time in volume visualization we are dealing with discrete data, so we need to estimate the gradient somehow
- The most popular technique is called *central differences*
- The central difference gradient operator at point (x,y,z) is defined as

$$D_x = f(x-1, y, z) - f(x+1, y, z)$$
$$D_y = f(x, y-1, z) - f(x, y+1, z)$$
$$D_z = f(x, y, z-1) - f(x, y, z+1)$$

- The gradient at (x,y,z) is therefore: $D(x,y,z) = [D_x\ D_y\ D_z]^T$
- Note that sometimes we normalize the gradient by dividing it by its length to generate a unit vector: $D/|D|$
- Our decision to normalize the gradient depends on our reason for computing the gradient in the first place
- Throughout future discussion it will be clear when we are using the normalized or un-normalized gradient

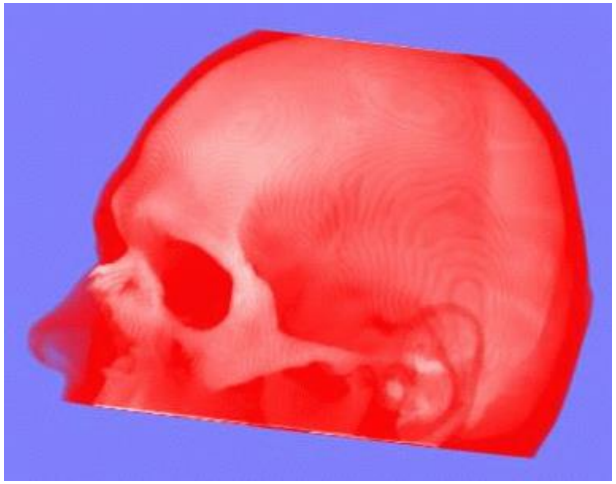
Why Volumetric Shading (What's the Point)?

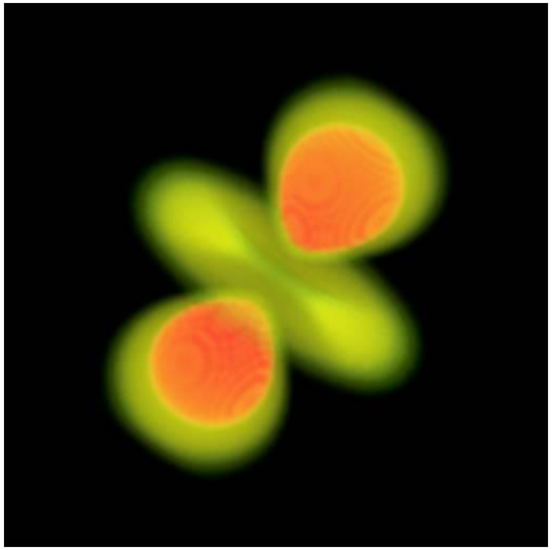
- The gradient allows us to assign a direction vector to each voxel
- This (normalized) vector is used just like the normal vector in surface graphics
- It will modulate the color we assign to samples and thereby allow us to create 3D effects
- Look at the skull on the right
- It looks 3D because we have incorporated diffuse reflection into the illumination model
- The light source is in front of the volume, which causes the top and sides of the skull to look rounded, which they are!
- Voxels whose gradient vectors make a large angle with the light ray appear darker

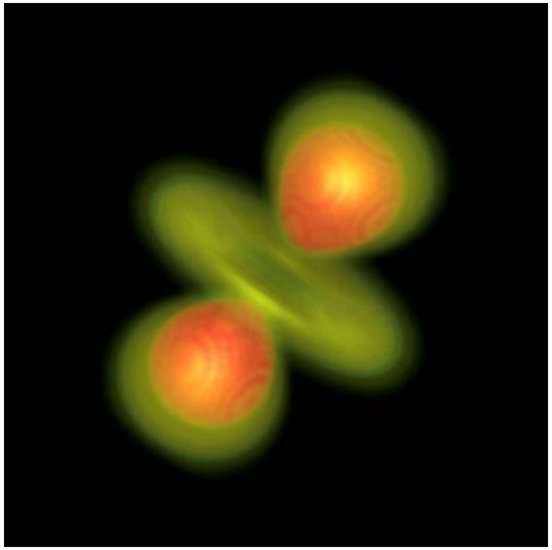


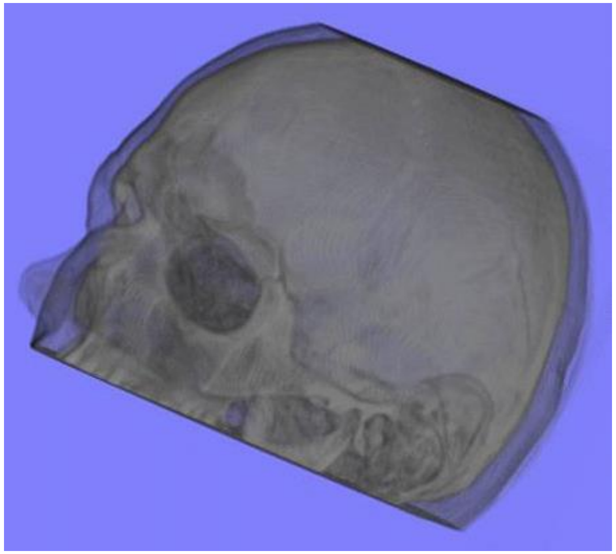
Volumetric Shading

- Volumetric shading is useful not only for viewing surfaces implied by the data (e.g., skull/skin boundary), but also for semitransparent renderings, especially when combined with good R, G, B, A transfer functions
- Compare:



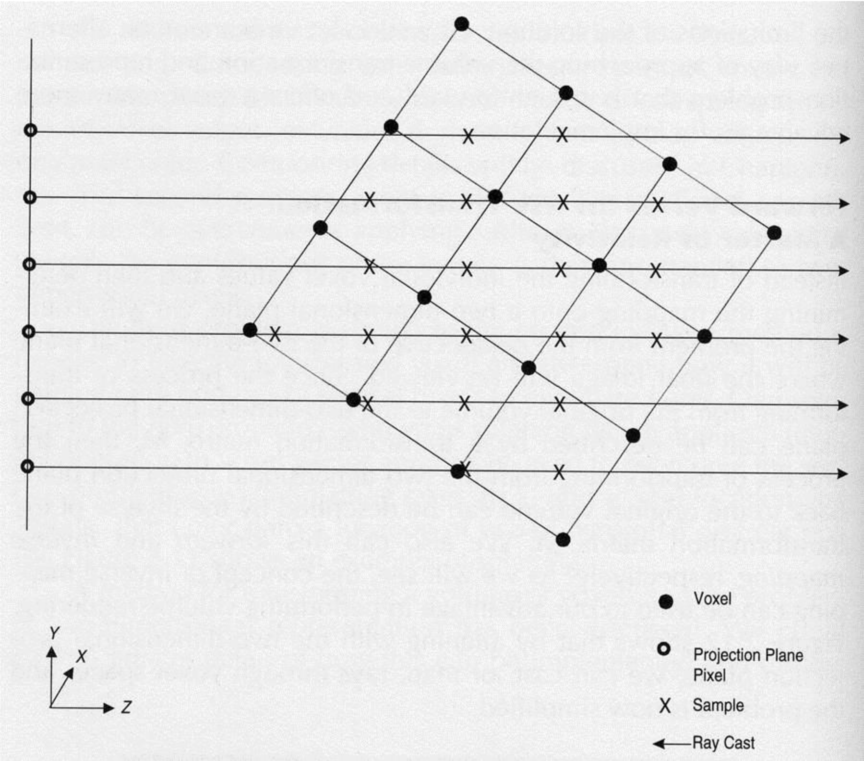






Volumetric Shading

- When we looked at X-ray and maximum intensity projection, we learned about compositing, which is the process for accumulating data along the viewing rays
- But how do we incorporate color, opacity and shading information?
- First we interpolate the density at a given sample position
- Then we assign a color and opacity to each the sample, and shade using the interpolated gradient
- When we shoot the rays through the volume, we have to blend all these samples together, …



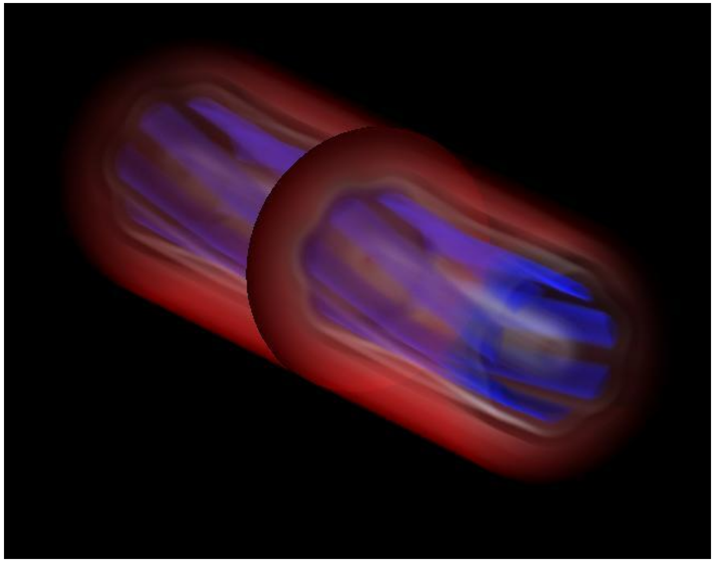
- We'll see how this is done in the near future

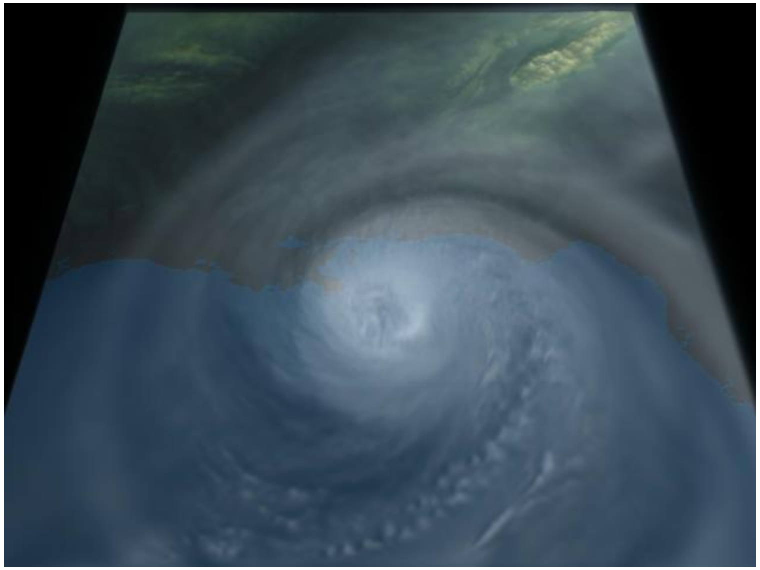
Handling Ambient and Diffuse Shading

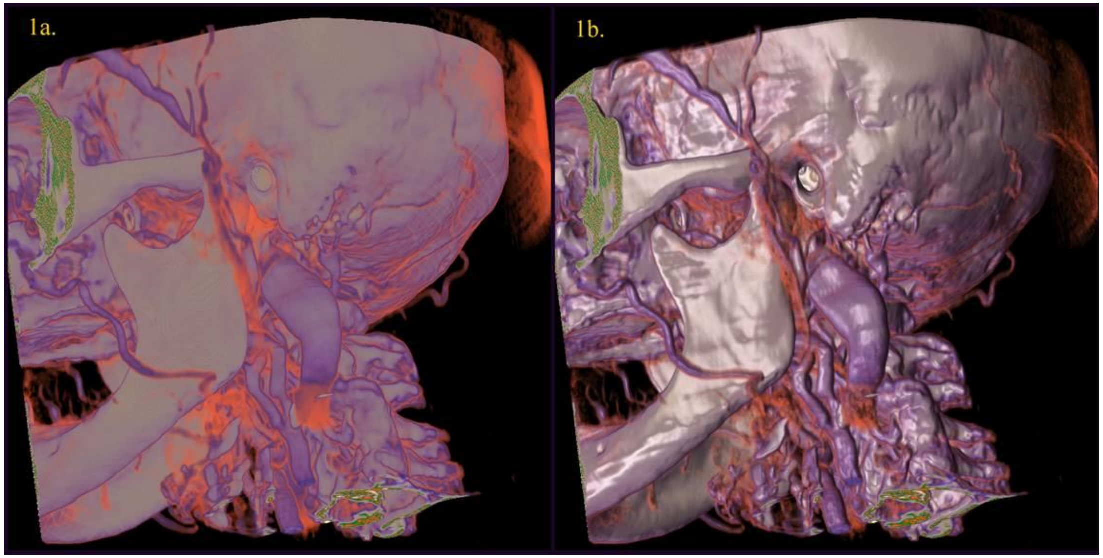
$$C = C_{obj}(k_a I_A + k_d I_L N \cdot L)$$

- You should be capable of implementing ambient and diffuse shading for a single light source
- Extra credit to implement Phong shading terms
- The normal vector N is replaced by the estimated gradient D
- Make sure you remember to normalize the gradients for the shading computations or else the volumes will come out either much darker or much brighter than you expect
- The dot product ($N \cdot L$) should be between [-1, +1]
- What if it's 0? What does that mean?
- How about if it's negative? What should you do in this situation?

More Volume Shading Examples







STONY BROOK
STATE UNIVERSITY OF NEW YORK

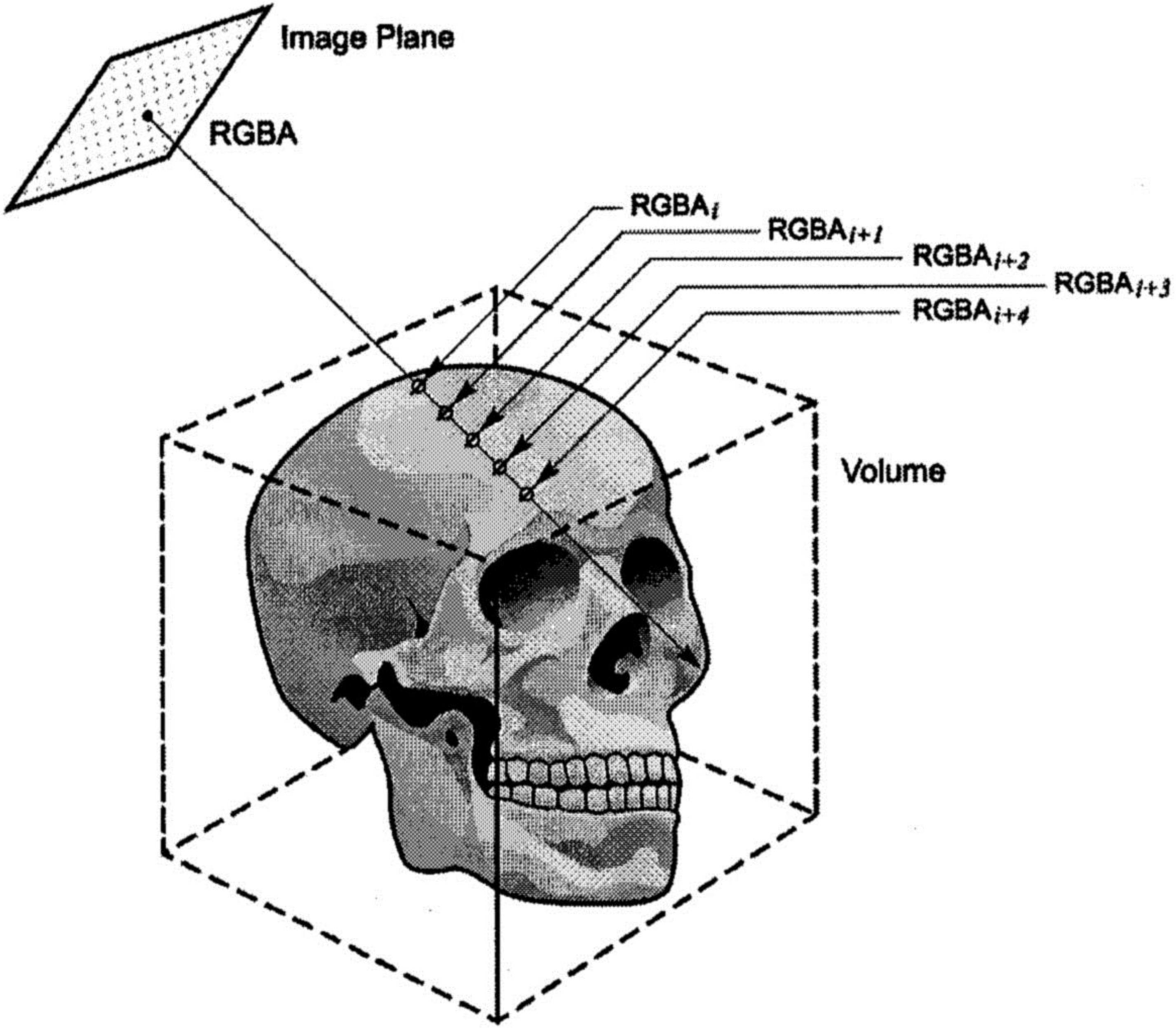
Summary So Far

- Density and gradient interpolation
- Color specification via transfer functions
- Shading and illumination models

- Q: How do we put this all together?
- A: Compositing.
- Compositing is what will enable us to shoot the viewing rays through the volume, accumulating color and opacity as we go, in order to generate the final rendered image

STONY BROOK
STATE UNIVERSITY OF NEW YORK

Compositing Overview



Compositing

- Recall the continuous and discretized forms of the X-ray integral:

$$I_{i,j} = \int_0^L f(P_{i,j} + t \cdot r_{i,j})dt$$

$$I_{i,j} = \sum_{k=0}^{L/\Delta t} f(P_{i,j} + k \cdot \Delta t \cdot r_{i,j}) \cdot \Delta t$$

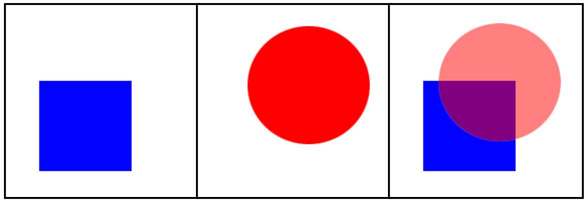
- Compositing (i.e., accumulation of densities) was simply the addition of interpolated densities along the ray
- In essence, each sample was assigned the same opacity
- We know now that in general volume rendering, we can use transfer functions to assign opacities and colors as desired
- Compositing in general is a more sophisticated process, which we need to investigate carefully
- Without a proper formulation for accumulating colors and opacities, our rendered images won’t turn out as we expect

Compositing

- More specifically, compositing is the accumulation of colors weighted by opacities; we weight by opacity in order to support semitransparent rendering
- Suppose we had two images we wanted to composite (blend) together
- Colors and opacities of back pixels are attenuated by opacities of front pixels:

$$rgb_{new} = RGB_{back} \cdot \alpha_{back} (1 - \alpha_{front}) + RGB_{front} \cdot \alpha_{back}$$

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



- By combining terms for efficiency we get:

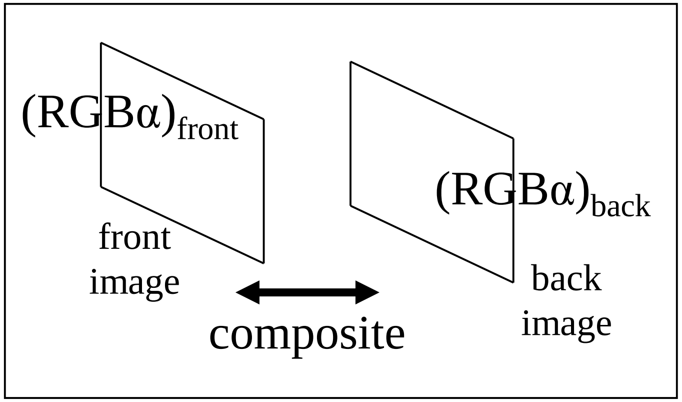
$$rgb_{back} = RGB_{back} \cdot \alpha_{back}$$

$$rgb_{front} = RGB_{front} \cdot \alpha_{front}$$

two recursive equations that can be used to composite any number of objects front-to-back:

$$rgb_{new\_front} = rgb_{back} \cdot (1 - \alpha_{front}) + rgb_{front}$$

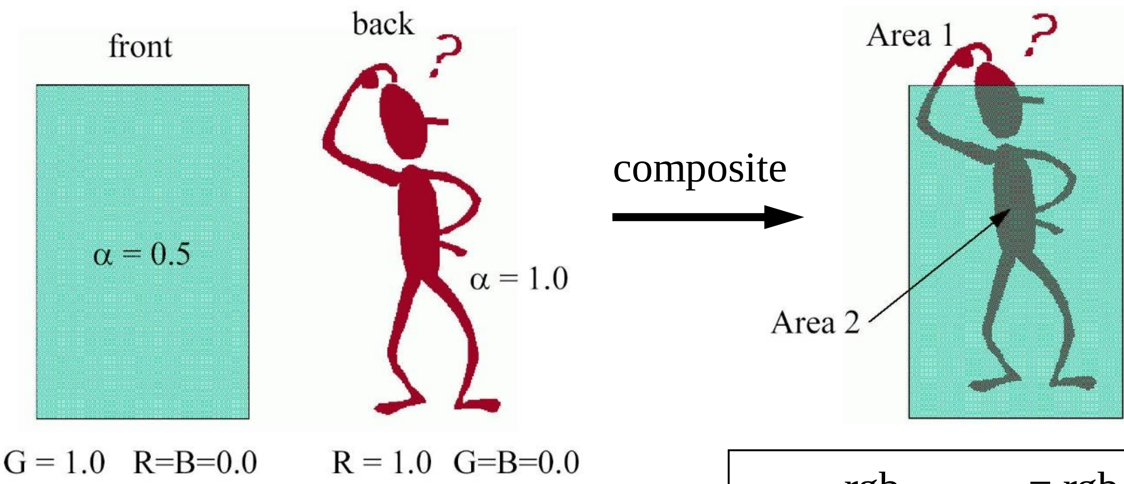
$$\alpha_{new\_front} = \alpha_{back} \cdot (1 - \alpha_{front}) + \alpha_{front}$$



- Volume rendering uses this recursive expression to combine (=composite) the samples taken along the ray
- You can see why opacity plays such an important role in this process

STONY BROOK
STATE UNIVERSITY OF NEW YORK

Compositing Example



$$rgb_{new\_front} = rgb_{back}\,(1 - \alpha_{front}) + rgb_{front}$$
$$\alpha_{new\_front} = \alpha_{back}\,(1 - \alpha_{front}) + \alpha_{front}$$

Area 1:

$\{r, g, b\}_{front} = \{0.0, 0.0, 0.0\}$

$\{r, g, b\}_{back} = \{1.0, 0.0, 0.0\}$

$\{r, g, b\}_{new\_front} =$

$\{1.0\,(1 - 0.0) + 0.0,$
$0.0\,(1 - 0.0) + 1.0,$
$0.0\,(1 - 0.0) + 0.0\}$
$= \{1.0, 0.0, 0.0\}$

$\alpha_{new\_front} = 1.0\,(1 - 0.0) + 0.0 = 1.0 = \alpha_{back}$

Area 2:

$\{r, g, b\}_{front} = \{0.0, 0.5, 0.0\}$

$\{r, g, b\}_{back} = \{1.0, 0.0, 0.0\}$

$\{r, g, b\}_{new\_front} =$

$\{1.0\,(1 - 0.5) + 0.0,$
$0.0\,(1 - 0.5) + 1.0,$
$0.0\,(1 - 0.5) + 0.0\}$
$= \{0.5, 0.5, 0.0\}$

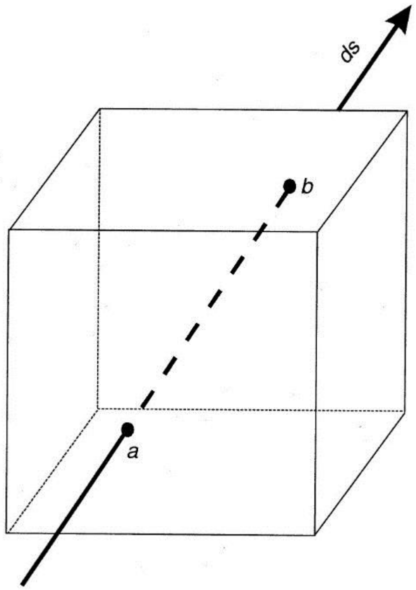
$\alpha_{new\_front} = 1.0\,(1 - 0.5) + 0.5 = 1.0$

STONY BROOK STATE UNIVERSITY OF NEW YORK

Ray Casting Integral

- We've looked at particular ray casting integrals: for X-ray rendering, and MIP rendering
- These are special cases of the general ray casting integral, which incorporates user-defined color, opacity and shading information
- The general ray casting integral is:

$$I(a,b) = \int_a^b g(s) e^{-\int_a^s \tau(s)\,dx} ds$$



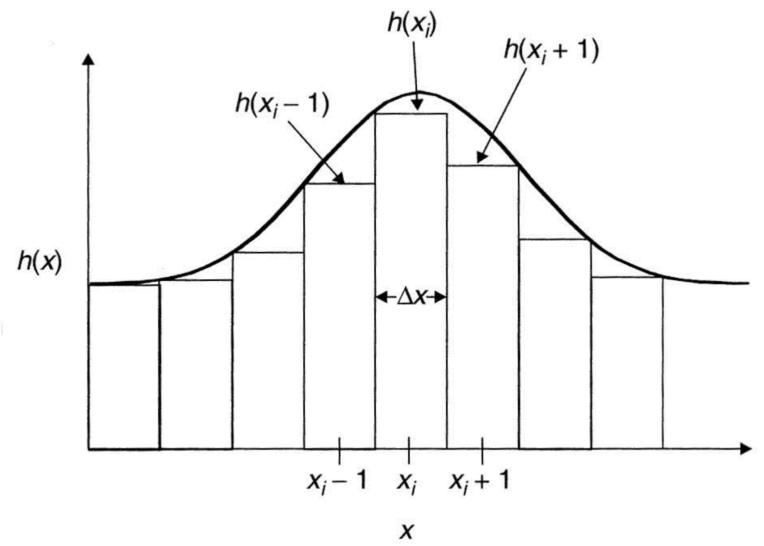
- *I(a,b)* is the intensity (not exactly color) of one pixel
- *ds* is the direction of the ray
- The ray runs from *a* to *b*
- *g(s)* is the source term (describes the illumination model)
- *τ(x)* is the extinction coefficient that describes the rate that light is occluded per unit length due to scattering or extinction of light (i.e., voxel transparency)
- *g(s)* incorporates the color transfer functions, *τ(x)* incorporates the opacity transfer function

STONY BROOK
STATE UNIVERSITY OF NEW YORK

Integral Discretization

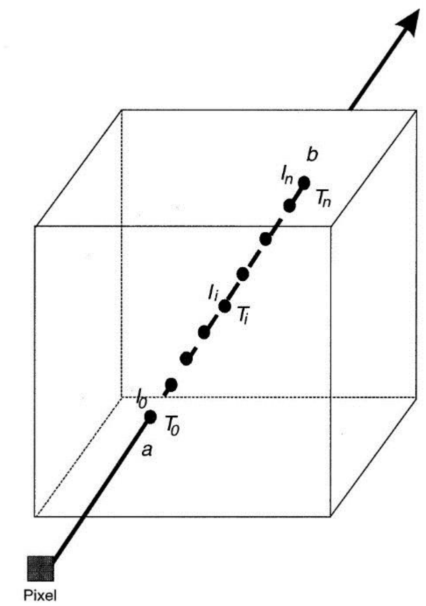
- In order to evaluate any definite integral in a computer, we need to discretize it
- One popular technique is to use a *Riemann sum*.

$$\int_0^d h(x)dx \approx \sum_{i=0}^{n} h(x_i)\Delta x$$



- Step size is Δx, and we assume the value inside an interval is constant
- Suppose we have computed the color and opacity on one ray at discrete sample points
- The discrete ray integral becomes:

$$I(a,b) = \sum_{i=0}^{n} I_j \prod_{j=0}^{i-1} T_j$$

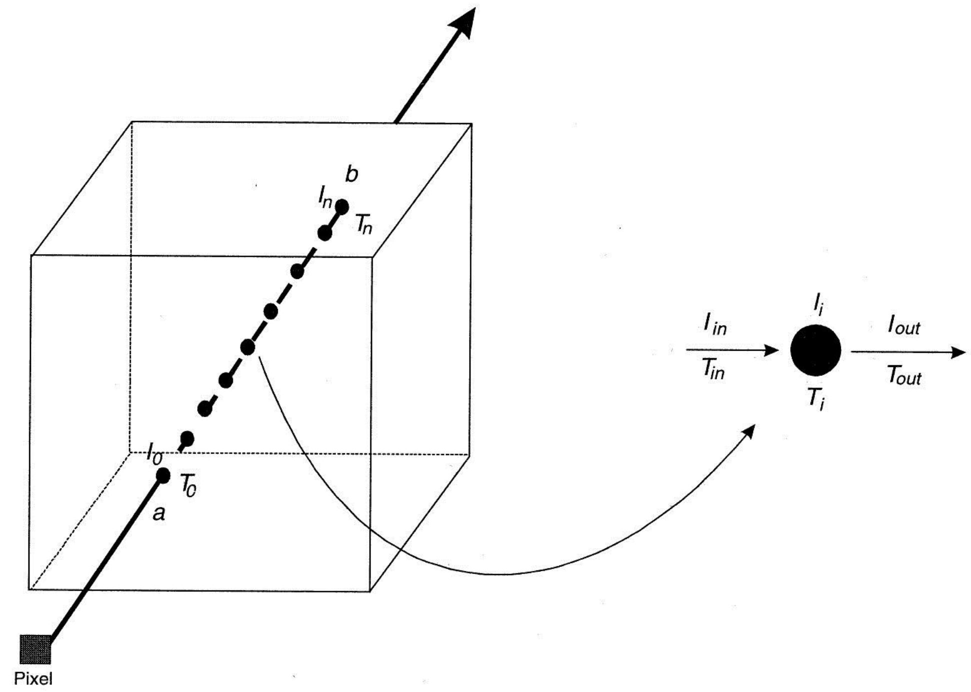


- I_i is the total light emitted (intensity) of a point at position *i* on the ray
- T_j is the transparency (1 – opacity or 1 – α)

Compositing

$$I(a,b) = \sum_{i=0}^{n} I_j \prod_{j=0}^{i-1} T_j$$

- Intuitively, the equation tells us that the total intensity I accumulated on one ray at the current sample point is the intensity I_i multiplied with all the transparencies $(1 - \alpha_j)$ encountered so far on the ray
- Thus, I_i is weighted by all preceding sample points
- The intensity is not the color, but rather the color times opacity at the sample point: $I_i = C_i \bullet \alpha_i$
- This is not the only way to define intensity, but it is the most common
- The colors are determined using the transfer functions and modulated using the illumination model, if any
- So we composite samples that have been both classified and shaded



STONY BROOK
STATE UNIVERSITY OF NEW YORK

Compositing Implementation

$$I(a,b) = \sum_{i=0}^{n} I_j \prod_{j=0}^{i-1} T_j$$

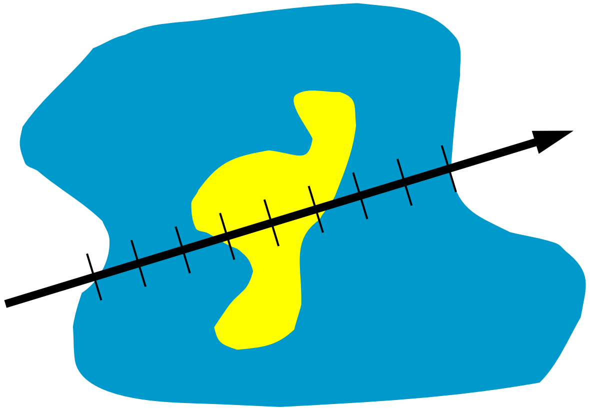
```
Trans = 1.0;
Inten = I[0];  // I[0..n] stores the intensities of the sample points

for (i = 1; i <= n; i++)
{
        Trans = Trans * T[i-1];  // T[0..n] stores sample point transparencies
        Inten = Inten + Trans * I[i];
}
```

- We should break the loop when *Trans* equals zero or gets very close. This is the same as saying we should stop when opacity hits almost 1.0
- This optimization is called *early ray termination*
- Remember that intensity is color times opacity

STONY BROOK
STATE UNIVERSITY OF NEW YORK

Full Volume Rendering

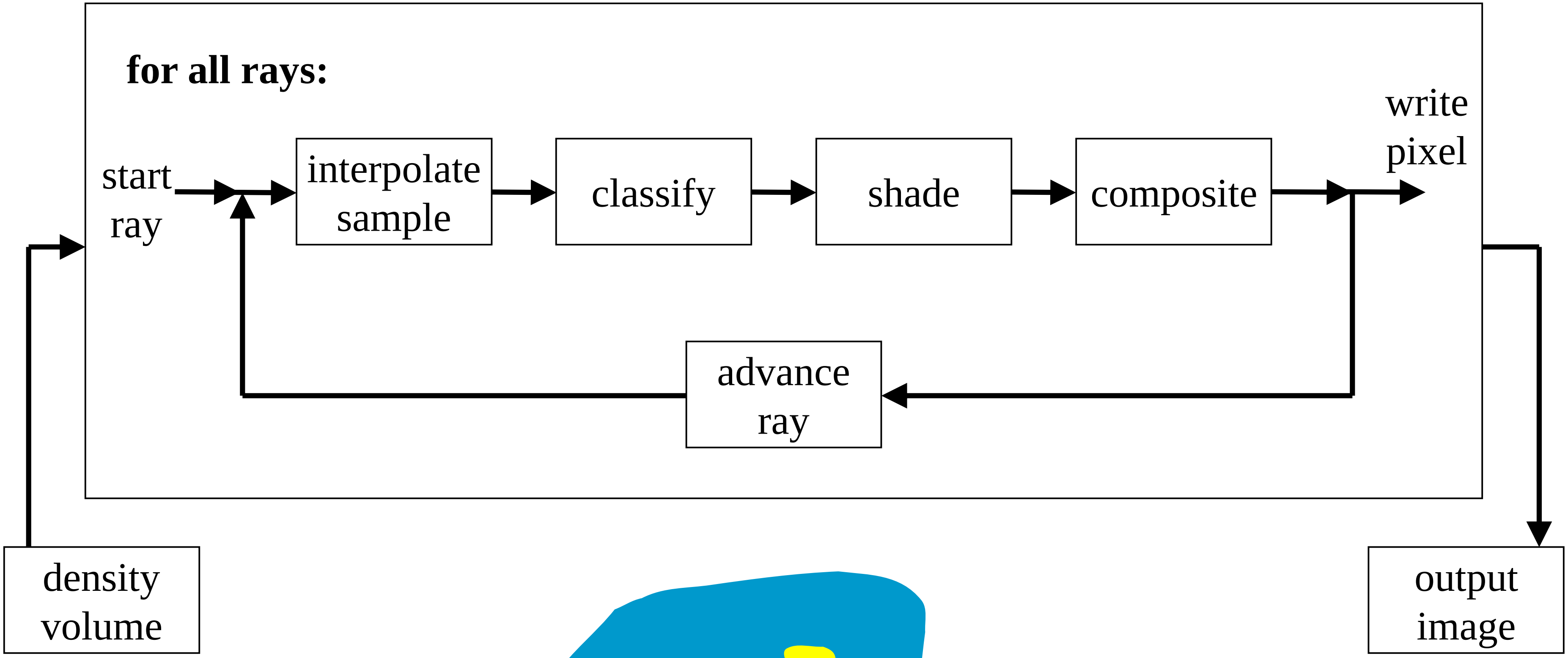


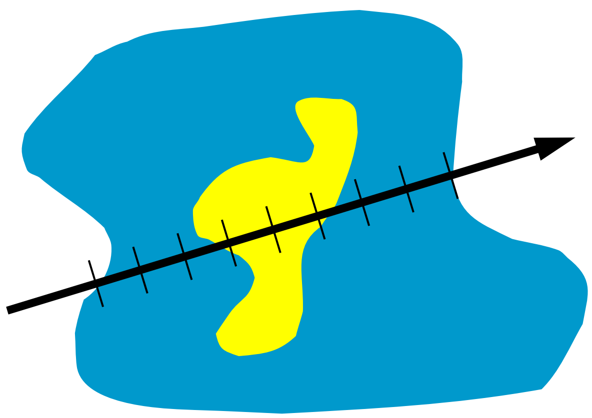
- We have looked a number of tasks:
- Interpolation
- Classification
- Shading
- Compositing
- In volume rendering, we will use them in the following algorithm:
- Along each viewing ray:

1. Interpolate density at current ray position
2. Classify interpolated density to assign color and opacity
3. Shade interpolated density to modulate color, depending on the shading and illumination parameters/model
4. Composite the classified and shaded sample
5. Advance ray to the next interpolated position and go to step 1

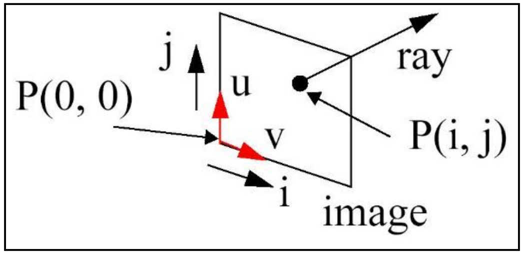
- We stop when the accumulated opacity reaches 1.0 on the ray

Full Volume Rendering





Full Volume Rendering: Algorithm for Orthographic Viewing

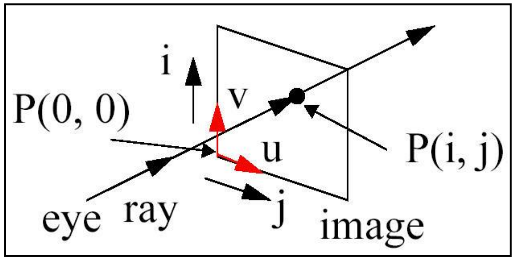


FullVolRenOrtho(Volume V, int stepSize, Image I)

```
ray = u x v / | u x v| // vector perpendicular to camera plane
for each image pixel i, j
    P(i, j) = P(0, 0) + i · v + j · u; // the location of image pixel (i, j) in world (volume) space
    {r, g, b} = 0, α = 0; // initialize red, green, blue color and opacity α to 0
    for (t = t_front; t <= t_back; t += stepSize) // traverse the volume front to back
        sampleLoc = P(i, j) + t · stepSize · ray; // step along the ray
        intVal = Interpolate(V, sampleLoc);
        if (AlphaTransFunc(intVal) > 0.05) // only do work for non-transparent samples
            gradVec = ComputeGradientVector(V, sampleLoc);
            {R,G,B}=Shade(gradVec, lightSource, eye, sampleLoc, {R,G,B}TransFunc(intVal));
            {r, g, b}=AlphaTransFunc(intVal) · {R,G,B} · (1 - α) + {r, g, b}; // composite color
            α = AlphaTransFunc(intVal) · (1 - α) + α; // composite opacity
            if (α > 0.95) // everything further is hidden and can't be seen, so stop the ray
                I(i, j) = {r, g, b}; break; // write color to image pixel and go to next pixel
```

STONY BROOK
STATE UNIVERSITY OF NEW YORK

Full Volume Rendering: Algorithm for Perspective Viewing

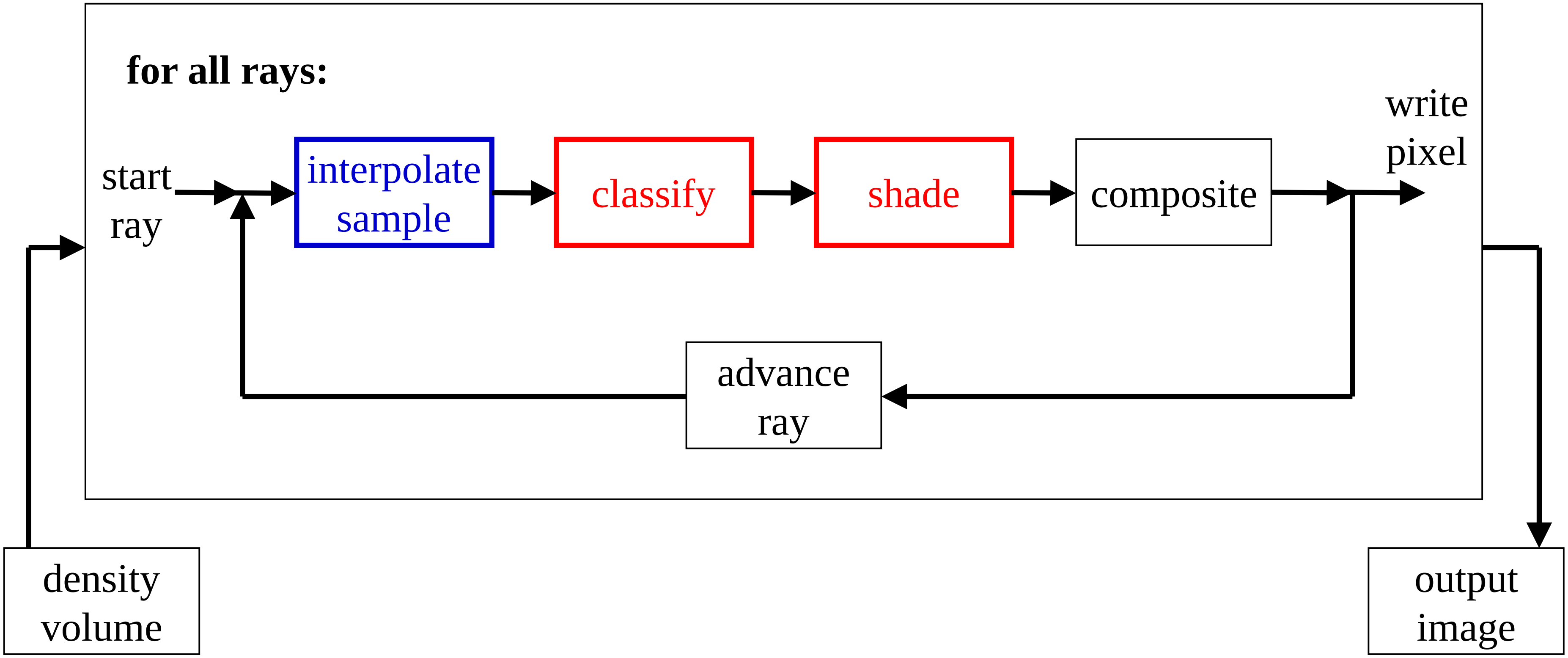


FullVolRenPersp(Volume V, int stepSize, Image I)

```
for each image pixel i, j
    ray = (P(i, j) - eye) / | (P(i, j) - eye) | // the ray direction vector, normalized
    P(i, j) = P(0, 0) + i · v + j · u; // the location of image pixel (i, j) in world (volume) space
    {r, g, b} = 0, α = 0; // initialize red, green, blue color and opacity α to 0
    for (t = t_front; t <= t_back; t += stepSize) // traverse the volume front to back
        sampleLoc = P(i, j) + t · stepSize · ray; // step along the ray
        intVal = Interpolate(V, sampleLoc);
        if (AlphaTransFunc(intVal) > 0.05) // only do work for non-transparent samples
            gradVec = ComputeGradientVector(V, sampleLoc)
            {R,G,B}=Shade(gradVec, lightSource, eye, sampleLoc, {R,G,B}TransFunc(intVal));
            {r, g, b}=AlphaTransFunc(intVal) · {R,G,B} · (1 - α) + {r, g, b}; // composite color
            α = AlphaTransFunc(intVal) · (1 - α) + α; // composite opacity
            if (α > 0.95) // everything further is hidden and can't be seen, so stop the ray
                I(i, j) = {r, g, b}; break; // write color to image pixel and go to next pixel
```

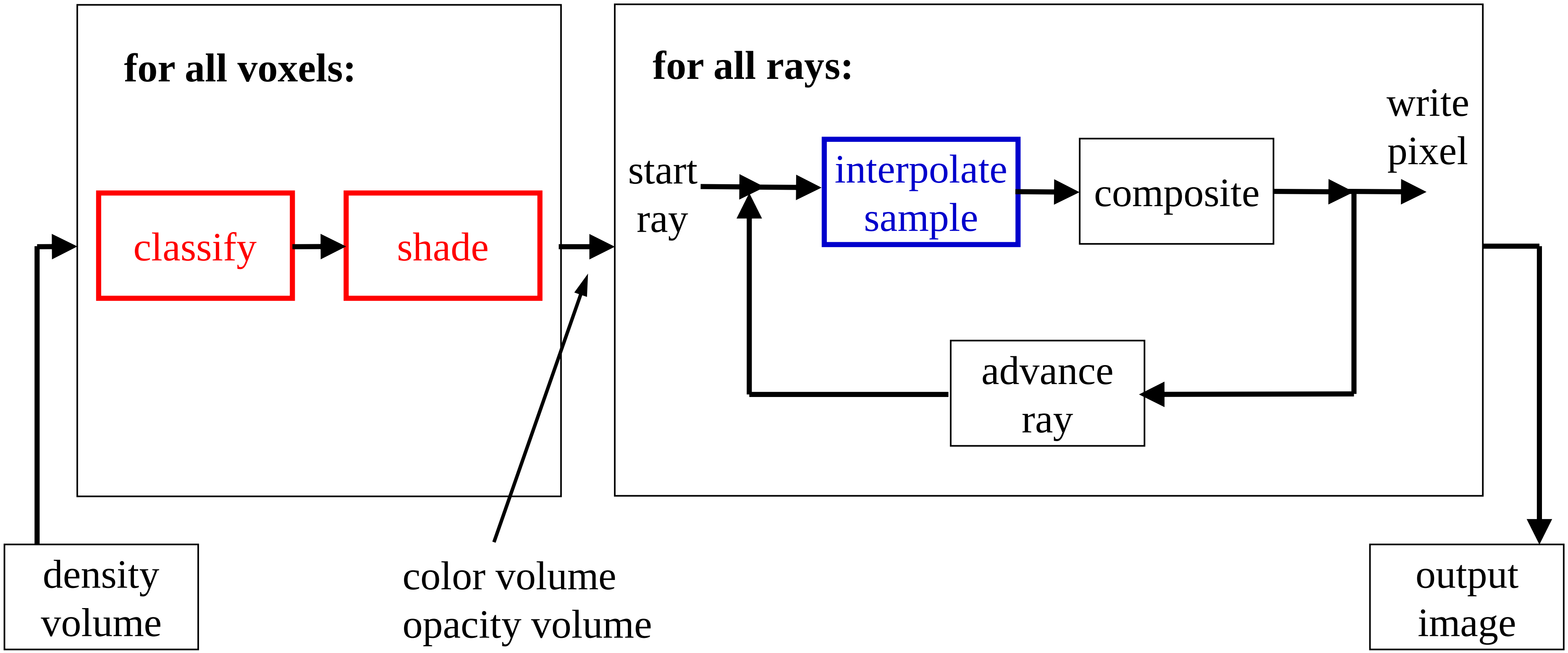
Full Volume Rendering

- This volume rendering framework is known as the *post-shaded pipeline*
- Classification and shading are performed after interpolation and sampling



Pre-shaded Pipeline

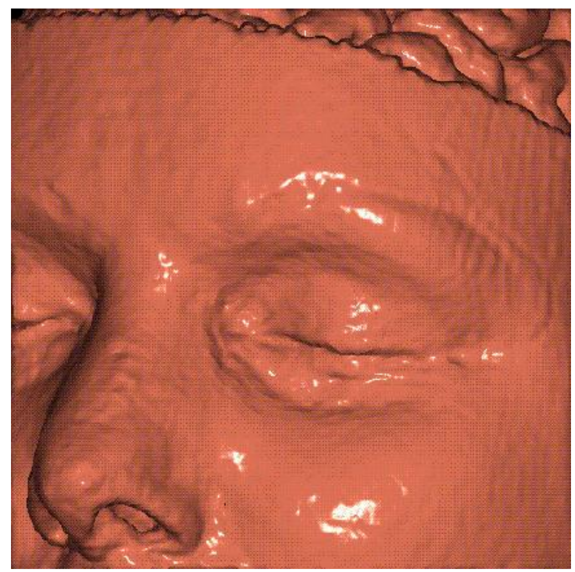
- The other possibility is to perform interpolation after classification and shading
- This is known as the *pre-shaded pipeline*



Which one is Better: Pre vs Post-Shading?

- Pre-shading is potentially faster because we can classify and shade all voxels before ray casting starts and then just interpolate the colors along the rays at sample positions
- With post-shading, we perform the classification during ray-traversal, which is more expensive
- However, pre-shading introduces blurring artifacts

pre-shaded



post-shaded

STONY BROOK STATE UNIVERSITY OF NEW YORK

What Causes the Blurring?