lecture 21

volume rendering

- blending N layers
- OpenGL fog (not on final exam)
- transfer functions
- rendering level surfaces

- 3D objects

  Clouds, fire, smoke, fog, and dust are difficult to model with vertices and polygons.

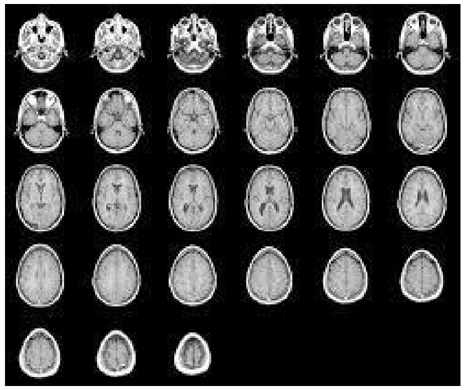
  Volumetric models assume that light is emitted, absorbed, and scattered by a large number of particles.

- Visualization of 3D data

  - medical imaging
  - seismic data for oil and gas exporation
  - distribution of temperature or density over a space
  - ...

Visualization of 3D data:

Is there an alternative to N x 2D slices ?

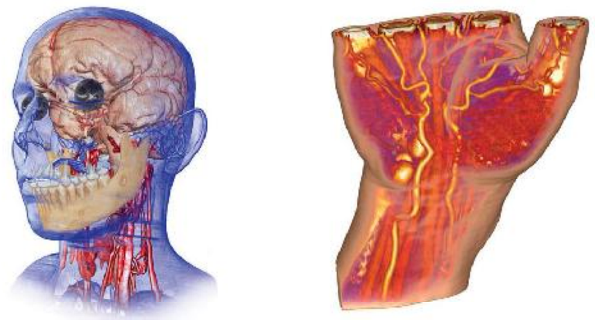


Volume rendering -- how to display "scalar field" ?

Two general approaches:

- integrating density along rays to the camera
- displaying "level surfaces"   ("iso-surfaces")

Hybrid approaches possible too...



http://www.cg.tuwien.ac.at/research/publications/2008/bruckner-2008-IIV/image-orig.jpg

---

Computer Science Questions:

How to select surfaces from 3D data and render them ?

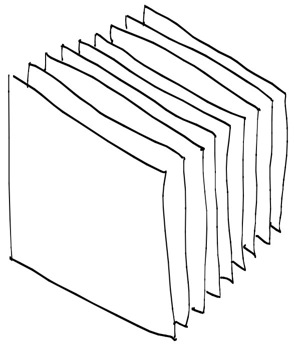
Costs ? (Tradeoffs: computation time and space, user effort)

Non- Computer Science Questions:

Which surfaces to show and which to hide ?
(class and instance specific)

What properties to give the surfaces... (perception issues)
- -- to illustrate their shape ?
- to illustrate their spatial arrangement ?
- to make a pretty picture ?

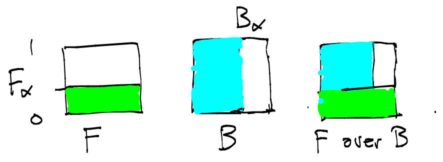
3D data = N x 2D images



Assume each layer is an RGBA image.

Recall "F over B" model from last lecture

Let $(r,g,b,\alpha) = (\alpha R, \alpha G, \alpha B, \alpha)$.   Blend pixels as follows:



If F_{rgb} , B_{rgb} are pre-multiplied by α, then

$$(F \text{ over } B)_{rgb\alpha} = F_{rgb\alpha} + (1 - F_\alpha)\, B_{rgb\alpha}$$

Suppose we are given N layers, L(i) for i = 1 to N.

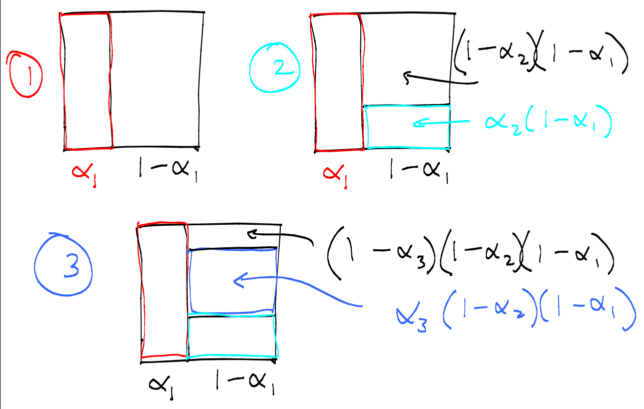
We want to compute:

L(1) over L(2) over L(3) over L(4) over .... L(N)

We will derive a formula for computing it from front to back:

(...   (L(1) over L(2)) over L(3))   ... over L(N)

Let's first examine the opacity (α) channel.



1
α_1
$1-\alpha_1$
2
$(1-\alpha_2)(1-\alpha_1)$
$\alpha_2(1-\alpha_1)$
α_1
$1-\alpha_1$
3
$(1-\alpha_3)(1-\alpha_2)(1-\alpha_1)$
$\alpha_3(1-\alpha_2)(1-\alpha_1)$
α_1
$1-\alpha_1$

Opacity of k layers

Start with an empty pixel. Define $\alpha_0 = 0$.

Fill a fraction α_1, leaving $1-\alpha_1$ empty.

Fill a fraction α_2 of the empty part, leaving $(1-\alpha_1)(1-\alpha_2)$ empty.

:

Fill a fraction α_k of the empty part, leaving $(1-\alpha_1)(1-\alpha_2) \ldots (1-\alpha_k)$ empty.

Q: How much of the original pixel gets (incrementally) filled in step k ?

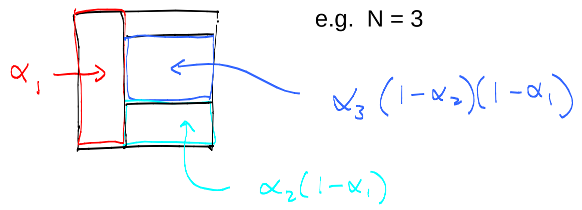
A:

$$\alpha_k (1-\alpha_1)(1-\alpha_2) \ldots (1-\alpha_{k-1}) = \alpha_k \prod_{j=1}^{k-1} (1 - \alpha_j)$$

Q: What is the accumulated opacity of N layers ?

A: $$\sum_{k=1}^{N} \alpha_k \prod_{j=0}^{k-1}(1 - \alpha_j)$$ recall $\alpha_0 = 0$.

e.g. N = 3



(Pre-multiplied) rgb

The *pre-multiplied* rgb image of the N layers is a weighted sum of the N *non-premultiplied* RGB images.

The k-th weight is the incremental opacity of the k-th layer.

Thus,

(L(1) over L(2) over L(3) over L(4) over .... L(N))rgb

$$= \sum_{k=1}^{N} (R_k, G_k, B_k) \underbrace{\alpha_k \prod_{j=0}^{k-1} (1 - \alpha_j)}_{\text{weight}}$$

[ASIDE: Analogy: conditional probability]

Suppose we can play a game where we flip some unfair coin up to N times.

For the k-flip, the probability of it coming up heads is α_k.

If a flip comes up heads, we get a payoff of R_k and the game ends.

If a flip comes up tails, we get to flip again (until a max of N flips).

Q: What is the expected value (average over many games) of the payoff ?

A: $$\sum_{k=1}^{N} R_k \, \alpha_k \prod_{j=0}^{k-1} (1 - \alpha_j)$$

i.e. Each term in the sum is the payoff on the k-th flip, weighted by the conditional probability that you get a payoff on the k-th flip, given that you didn't get a head in the first k-1 flips.

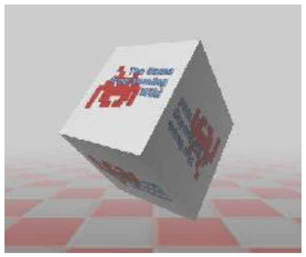
lecture 21

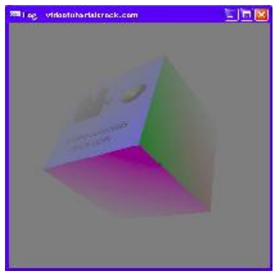
volume rendering

- blending N layers
- OpenGL fog (not on final exam)
- transfer functions
- rendering level surfaces

OpenGL Fog

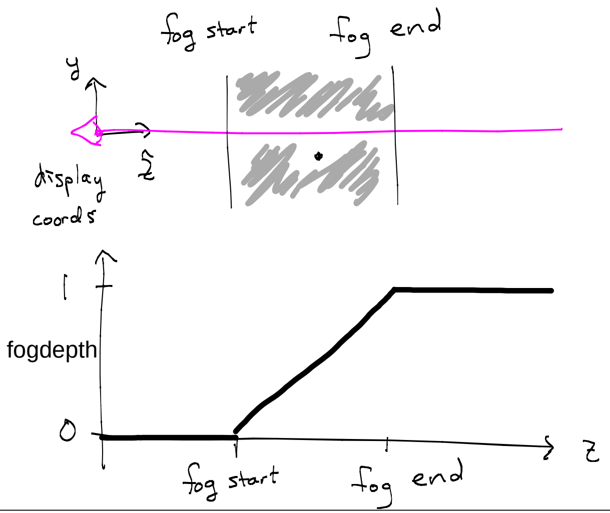
OpenGL 1.x provides a special case that we have uniform fog + opaque rendered surface. Here are a few simple examples





http://content.gpwiki.org/index.php/OpenGL:Tutorials:Tutorial_Framework:Light_and_Fog

http://www.videotutorialsrock.com/opengl_tutorial/fog/video.ph



OpenGL Fog blending formula

$$I_{rgb} = I^{fog}_{RGB} \, \alpha^{fog} + I_{RGB} (1 - \alpha^{fog})$$

where

- α^{fog} depends on fogdepth (next two slides)
- I_{RGB} is the rendered value for the visible surface e.g. Blinn-Phong.

lecture 21

volume rendering

- blending N layers
- OpenGL fog
- transfer functions
- rendering level surfaces

Recall the general model of N layers with RGBA in each layer. Where do the RGBA values come from ?

There are several possibilities.

- emission/absorption
- texture mapping
- rendering with light and material

Emission/absorption model

When I discussed the Blinn-Phong model in OpenGL, I said that RGB color of surfaces was the sum of three components: DIFFUSE, SPECULAR, AMBIENT.

There is a fourth component called GL_EMISSION. This component is independent of any lighting. It is added to the other three components.

Normally if one uses Blinn-Phong then one doesn't include an emission component and similarly if one uses an emission component then one doesn't include the other three components.

For blending N layers, one could be to use an emission component for the RGB colors. The alpha would account for "absorption"

The contribution of N layers of fog is:

(L(1) over L(2) over L(3) over L(4) over .... L(N))rgb

$$= \sum_{k=1}^{N} (R_k, G_k, B_k)^{fog}\ \alpha_k \prod_{j=0}^{k-1} (1 - \alpha_j)$$

Assume α_j is small for all j and recall $\alpha_0 = 0$.
Then, the fraction that passes through layer j is:

$$1-\alpha_j \approx e^{-\alpha_j}$$

and so

$$\prod_{j=0}^{k-1}(1-\alpha_j) \approx \prod_{j=0}^{k-1} e^{-\alpha_j} = e^{-\sum_{j=1}^{k-1}\alpha_j}$$

Assume the fog density is uniform so α_j is constant α for all j > 0. Then,

$$\prod_{j=1}^{k-1}(1-\alpha_j) \approx e^{-(k-1)\alpha}$$

If fog color I^{fog}_{RGB} i.e. RGB^{fog} is also constant, then

the contribution of the fog alone is:

(L(1) over L(2) over L(3) over L(4) over .... L(N))rgb

$$= \alpha\, I^{fog}_{RGB} \sum_{k=1}^{N} e^{-(k-1)\alpha}$$

To simplify the sum, use the fact that

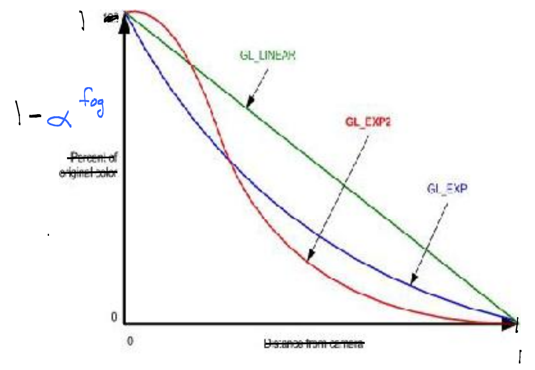
$$\sum_{k=1}^{N} e^{-(k-1)\alpha} = \frac{1-e^{-N\alpha}}{1-e^{-\alpha}} \qquad e^{-\text{fog depth}\ \text{GL\_FOG\_DENSITY}}$$

and plug into the previous slide.

Finally, since $1-e^{-\alpha}\approx\alpha$, we get:

$$I_{rgb} = I_{RGB}\,\alpha^{fog} + I^{fog}_{RGB}\,(1-\alpha^{fog})$$

$$I_{rgb} = I^{fog}_{RGB}\,\alpha^{fog} + I_{RGB}\,(1-\alpha^{fog})$$



OpenGL Fog (details)

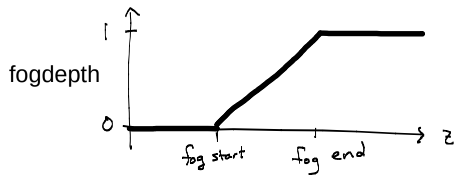
```
glFogfv(GL_FOG_COLOR, fogColor);            // I^fog_RGB
glFogf(GL_FOG_DENSITY,  0.35);              // How Dense Will The Fog Be ?
glFogf(GL_FOG_START,     1.0)               // Fog Start Depth
glFogf(GL_FOG_END,       5.0)               // Fog End Depth
glEnable(GL_FOG);

glFogi(GL_FOG_MODE,  fogMode);      // Fog Mode
```

$$\alpha^{fog} = \begin{cases} 1 - \text{fogdepth} & \text{if fogMode = GL\_LINEAR} \\ e^{-\text{fogdepth} * \text{GL\_FOG\_DENSITY}} & \text{if fogMode = GL\_EXP} \\ e^{-(\text{fogdepth} * \text{GL\_FOG\_DENSITY})^2} & \text{if fogMode = GL\_EXP2} \end{cases}$$

Derivation of fog formula

$$\alpha^{fog} = e^{-\text{fogdepth} * \text{GL\_FOG\_DENSITY}} \qquad \text{if fogMode = GL\_EXP}$$



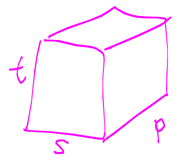
The other two cases are just to give the user more flexibility

3D Texture Mapping

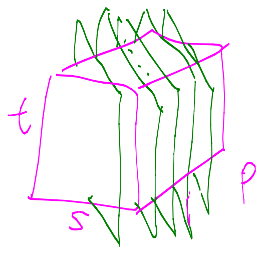
Consider a 3D scalar texture or a 3D RGBA texture.

The texture coordinates for indexing into the RGBA values are (s, t, p, q). This allows us to perform perspective mappings -- i.e. a class of deformations -- on the textures. Similar idea as homographies but now 4D rather than 3D, so more general.

You can think of the 3D texture coordinates as (s, t, p, 1).

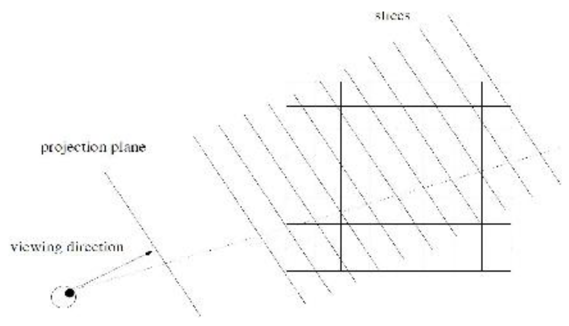


You can define texture coordinates on the corners of a cube.
If you define a planar slice through the cube, OpenGL will interpolate the texture coordinates for you.



Plane slices are sometimes referred to as "proxy geometry".

"Tri-linear" interpolation (see Exercises)



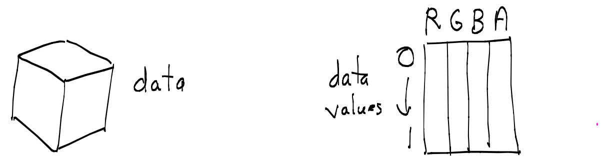
The intersection of a ray with the plane slices typically will not occur exactly at the grid points where the data is defined.

Transfer Function

In many applications such as in medical imaging, we have **scalar** data values (not RGBA) defined over a 3D volume e.g. cube.
Usually the data values are normalized to [0, 1].

We need to define a "transfer function" which maps data values to RGBA values. This is typically implemented using a "lookup table".

A transfer function can be represented as a 1D texture, i.e. it maps data values in [0, 1] to RGBA values.

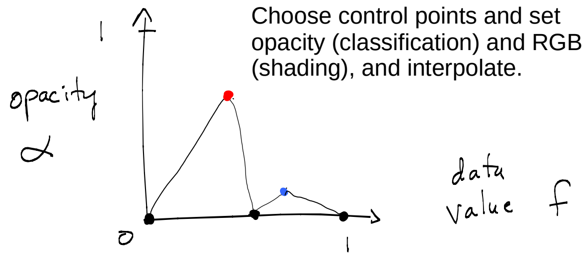


Note transfer function domain says nothing about position.

Transfer function Editing

There is no 'right way' to define a transfer function for a given 3D data set. It is an interactive process.

https://www.youtube.com/watch?v=dmh-8nKSzTc
See ~1:30-1:40 where they add skin.



Choose control points and set opacity (classification) and RGB (shading), and interpolate.

lecture 21

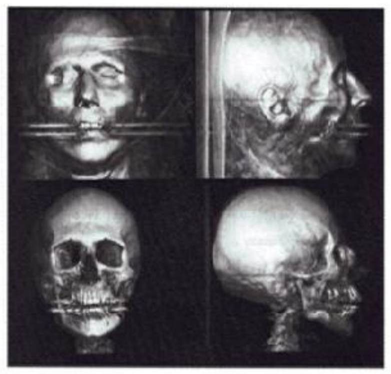
volume rendering

- blending N layers
- OpenGL fog
- transfer functions
- rendering level surfaces (iso-surfaces)

e.g. Levoy 1988 and others in same year

[This particular paper has been cited over 3000 times.]

Artifacts are scattering from dental fillings



air-skin interface

skin-bone interface

https://graphics.stanford.edu/papers/volume-cga88/

Rendering Level Surfaces (sketch only)

Volume rendering methods do not compute polygonal representations of these surfaces ("geometric primitives").

Rather, they assign "surface normals" to all the points within the 3D volume and then compute the RGB color using Blinn-Phong or some other model.

Thus, we need need to define a surface normal and a material at each point in the volume (plus a lighting model).

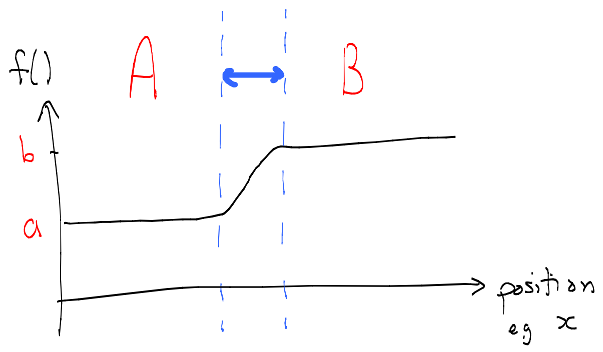
We can define the material using a transfer function. But what about the normal?

Define a surface normal at (x,y,z) by the 3D gradient of data value f(x,y,z).

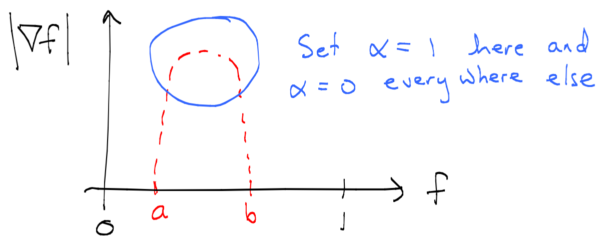
$$\nabla f(x,y,z) = \Big(\big(f(x+1,y,z) - f(x-1,y,z)\big)/2,\ \big(f(x,y+1,z) - f(x,y-1,z)\big)/2,\ \big(f(x,y,z+1) - f(x,y,z-1)\big)/2 \Big)$$

$$n(x,y,z) = \frac{\nabla f(x,y,z)}{\| \nabla f(x,y,z) \|}$$

The boundary between two regions A and B with data values a and b, respectively, is characterized by a large gradient of f, and by data values f in some range.



To render the boundary region between A and B, define the opacity component of the transfer function using both the data f(x,y,z) and its gradient.



For details, see multidimensional transfer functions[Kniss et al 2003]
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.92.502&rep=rep1&type=pdf

Many transfer functions were invented in decade between ~1995 and ~2005.

There is still no consensus on what is a good transfer function.

It depends on:

- what is the user's goal ?
- how class/instance specific should it be ?
- how much work is required by user to define it ?

Like many problems, there are many good solutions but no best solution ....

Announcements

- A2 grading is near done (except Yi-Z). Class ave 85%.
- Thurs. class will be Exercises / Review (shading-shadows)
- A4 (6%) is under construction. It will be posted before Easter weekend (and hopefully sooner).