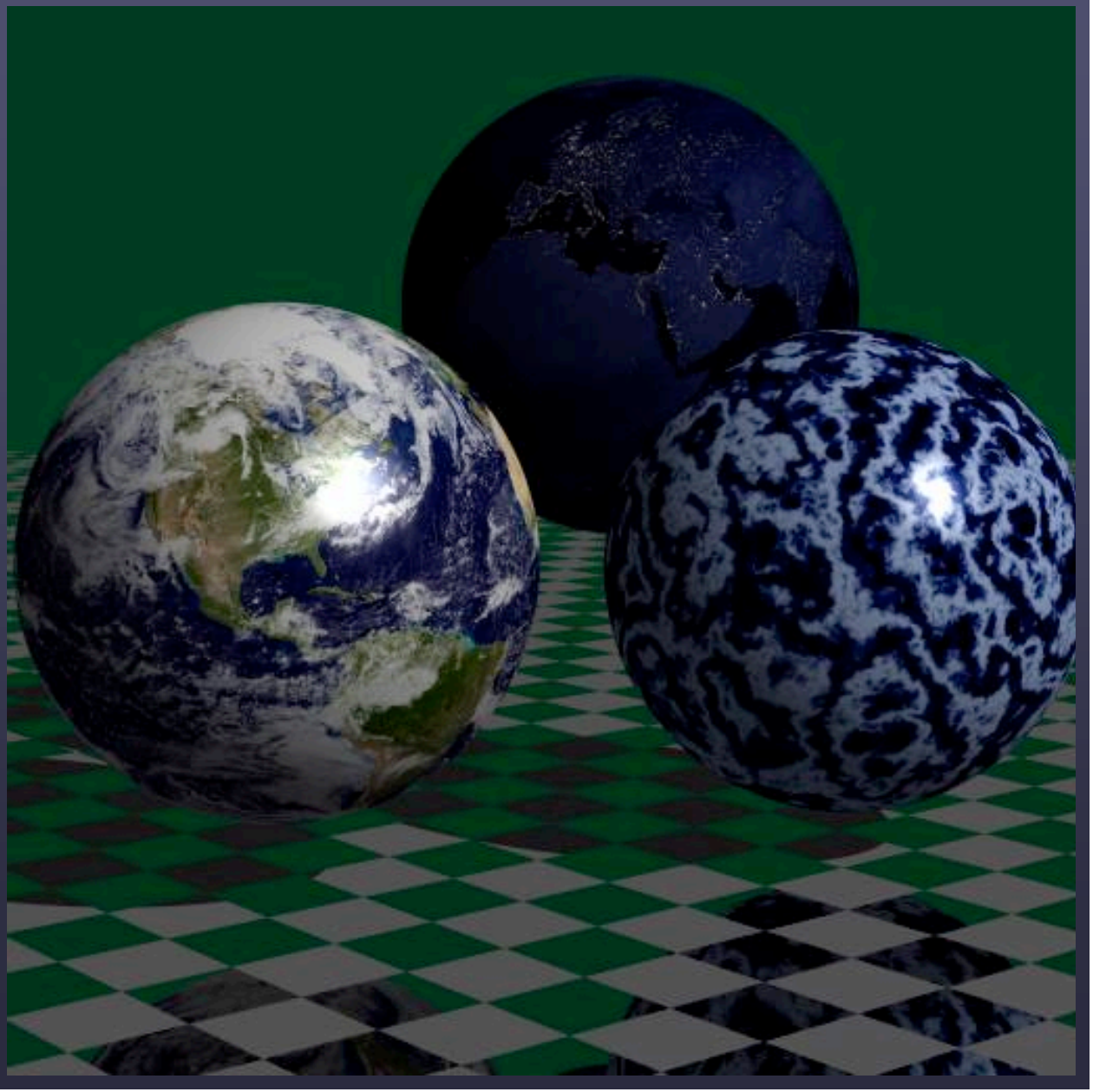
Volume Rendering

March 21, 2005

- Program 7 questions?

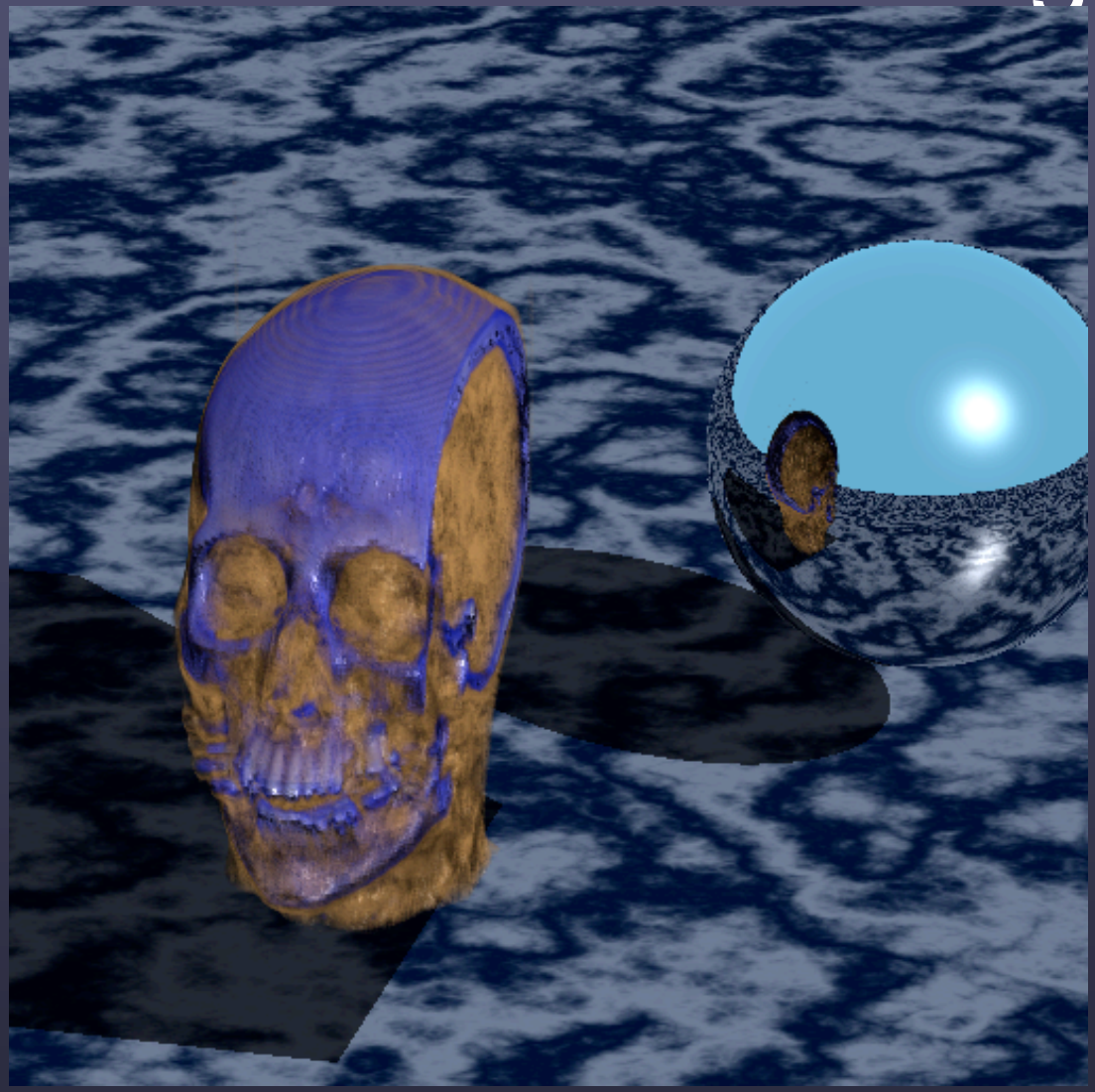


Review

- Hypertextures
- Volume rendering

Program 8 - Volume Rendering

- Due March 28th
- Last required image!



Program 8

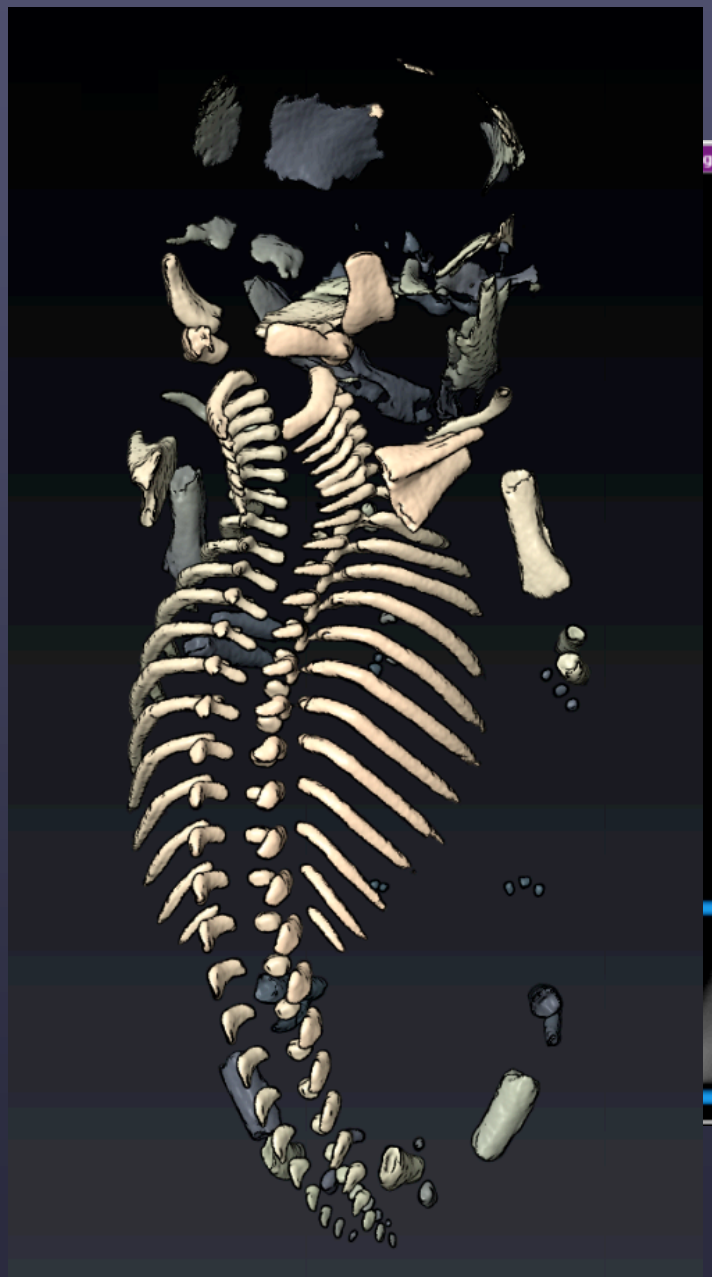
---------- Assignment #8----------

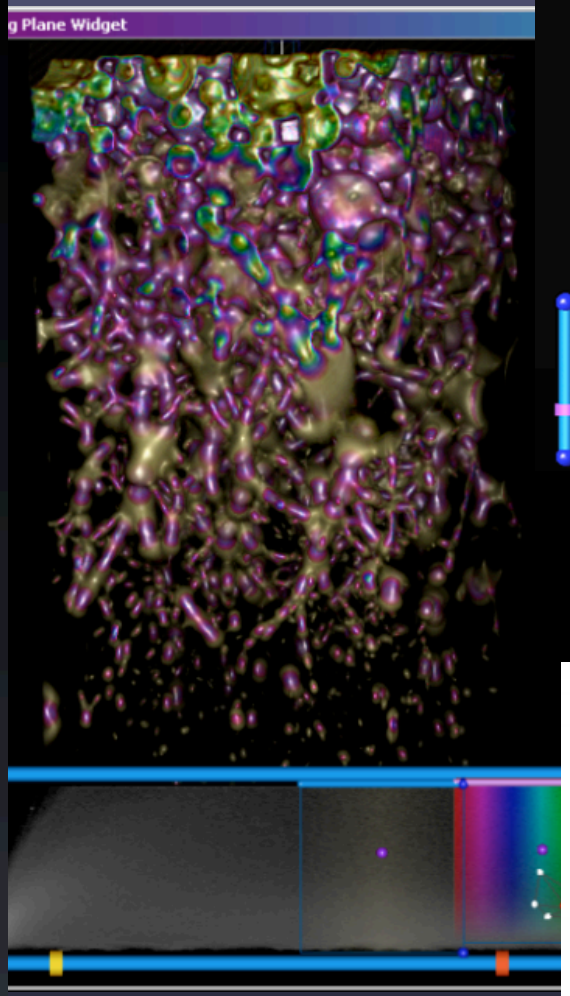
Programming language: C++
CPU(s): PowerPC 970
Clock rate: 2.5 Ghz
Threads count: 1
Thirdparty libraries used: none
Compiler: IBM Xlc++ version 6.0
Compiler flags: -O5 -qarch=g5 -qtune=g5
Setup/load time: 0.177 seconds
Render time: 3.05 seconds
Post-process time: 0.00593 seconds

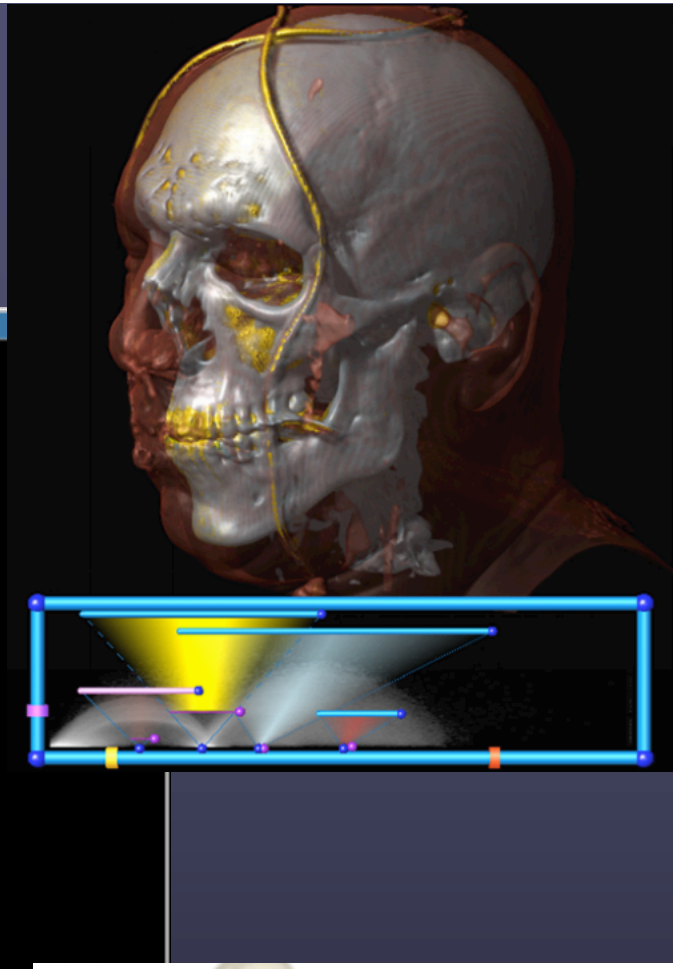
Volume Rendering

- Disclaimer
  - Volume rendering is a complex subject
  - We will just scrape the surface
- We will not cover
  - Filtering
  - Self-shadowing
  - Adaptive sampling
  - Complex acceleration algorithms

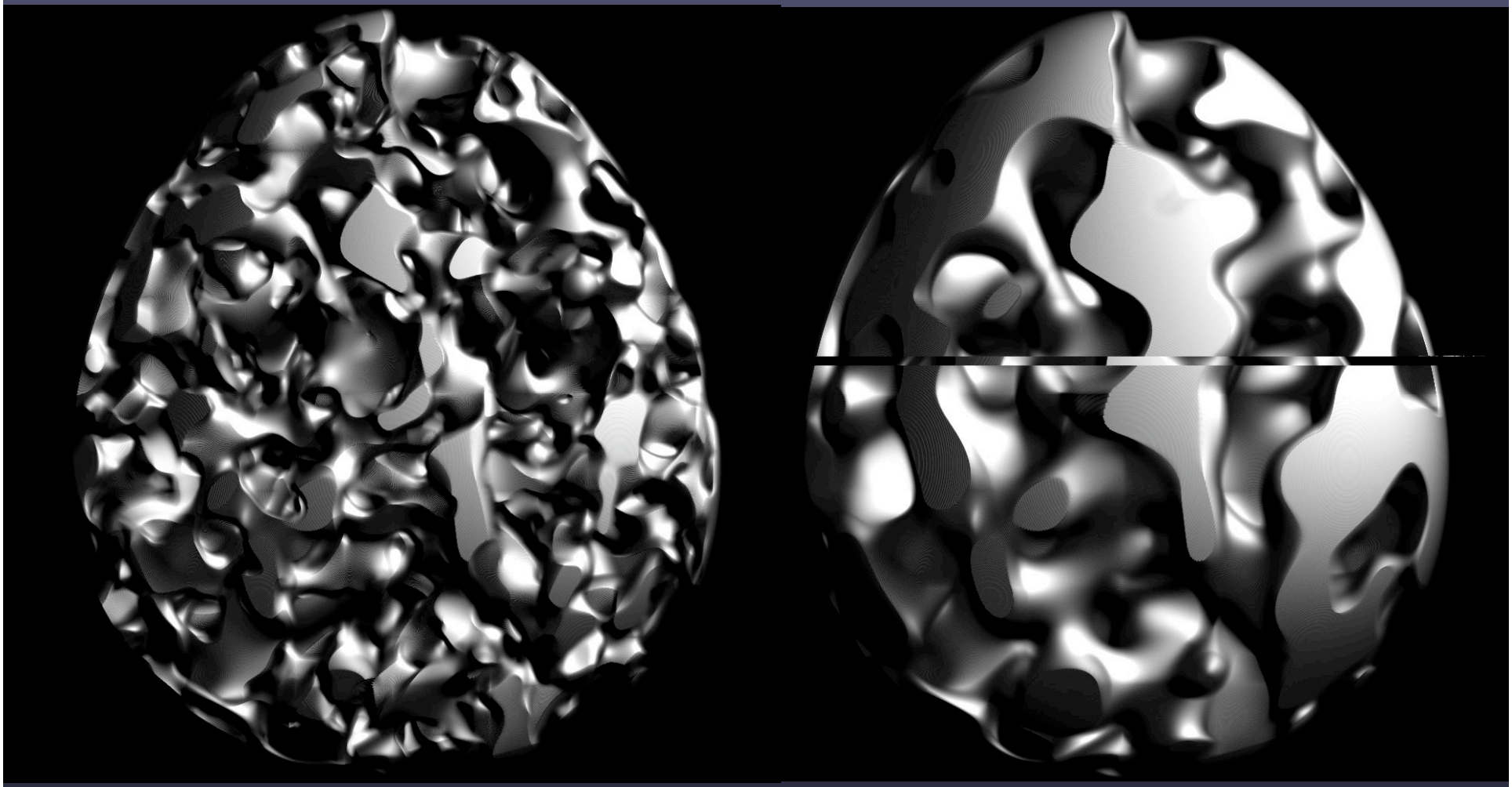
Volume Rendering





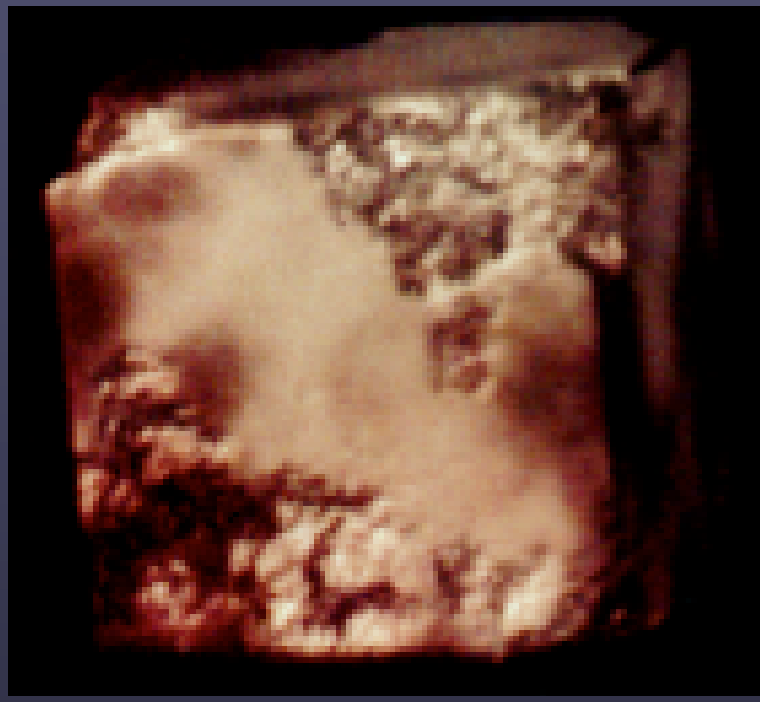


More hypertextures



Hypertextures

- Multiplication can be used to combine textures (coarse/fine)
  - Eroded cube:
    - Low frequency cutouts
    - High frequency rough inside
    - Subtle changes in color



Beer's Law

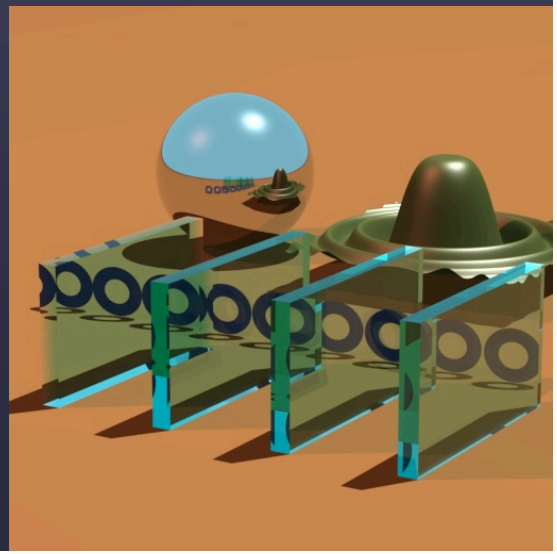
Relates the amount of absorption to the distance the light has traveled through the medium

As a ray travels through a medium, it loses intensity according to:

$$dI = -CIdx \quad \Rightarrow \quad \frac{dI}{dx} = -CI$$

This equation is solved by the exponential:

$$I = ke^{-Cx}$$

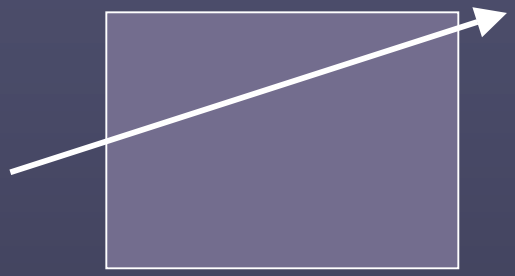


Absorption

- C is called the absorption coeffecient
  - It describes how quickly different wavelengths are absorbed
  - It is a color (components not necessarily 0-1)
- What if C varies spatially?
  - One form of volume rendering

Volume Rendering Integral

$$C_{ray} = \int_0^{t_{exit}} C(t)\mu(t)e^{\int_0^t \mu(s)ds}\,dt$$



where :

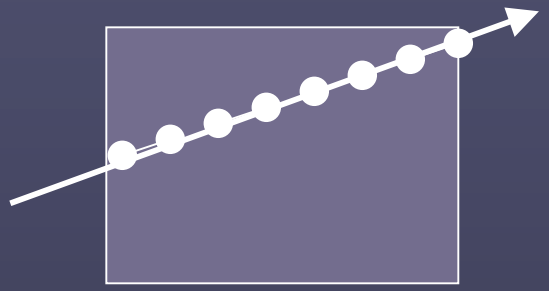
$$C(t) = C(\vec{P}) = C\left(data(\vec{P})\right)$$

$$\mu(t) = \mu(\vec{P}) = \mu\left(data(\vec{P})\right)$$

- Inside integral attenuates contribution
- Outside integral totals all contributions
- In general, cannot analytically compute

Discrete Approximation

$$C_{ray} = \sum_{i=0}^{\frac{t_{exit}}{\Delta t}} C(i\Delta t)\mu(i\Delta t)\Delta t \cdot \prod_{j=0}^{i-1} e^{(-\mu(j\Delta t)\Delta t)}$$



$$C_{ray} = \sum_{i=0}^{\frac{t_{exit}}{\Delta t}} C(i\Delta t)\alpha(i\Delta t) \cdot \prod_{j=0}^{i-1} (1-\alpha(j\Delta t))$$

$where:$

$$\alpha(t) = 1 - e^{-\mu(t)\Delta t}$$

- Physical interpretation: absorb light at each sample

Volume rendering algorithm (ray casting)

Compute enter/exit points (typically a box)
t=tenter
cumulative_opacity=0
cumulative_color=black
while(t<texit){
  P=ray.origin+t*ray.direction
  o = opacity(P); c = color(P)
  c = shade c
  cumulative_color += c*o*(1-cumulative_opacity)
  cumulative_opacity += o*(1-cumulative_opacity)
}
Trace ray from texit and add to final color

Opacity/Color functions

- Hypertexture:
  - Noise/turbulence to determine color and opacity
  - Simple mathematical expressions to control sharpness of edges
  - Functions can be combined with multiplication
- Sampled data:
  - Scientific visualization
  - Transfer functions convert data values to color/opacity

Options for shading

- Use color directly (models emission of light in a volume)
- Use lambertian or phong shading (models surfaces)
- Use lambertian or phong with attenuated shadows
- Full scattering models (clouds)

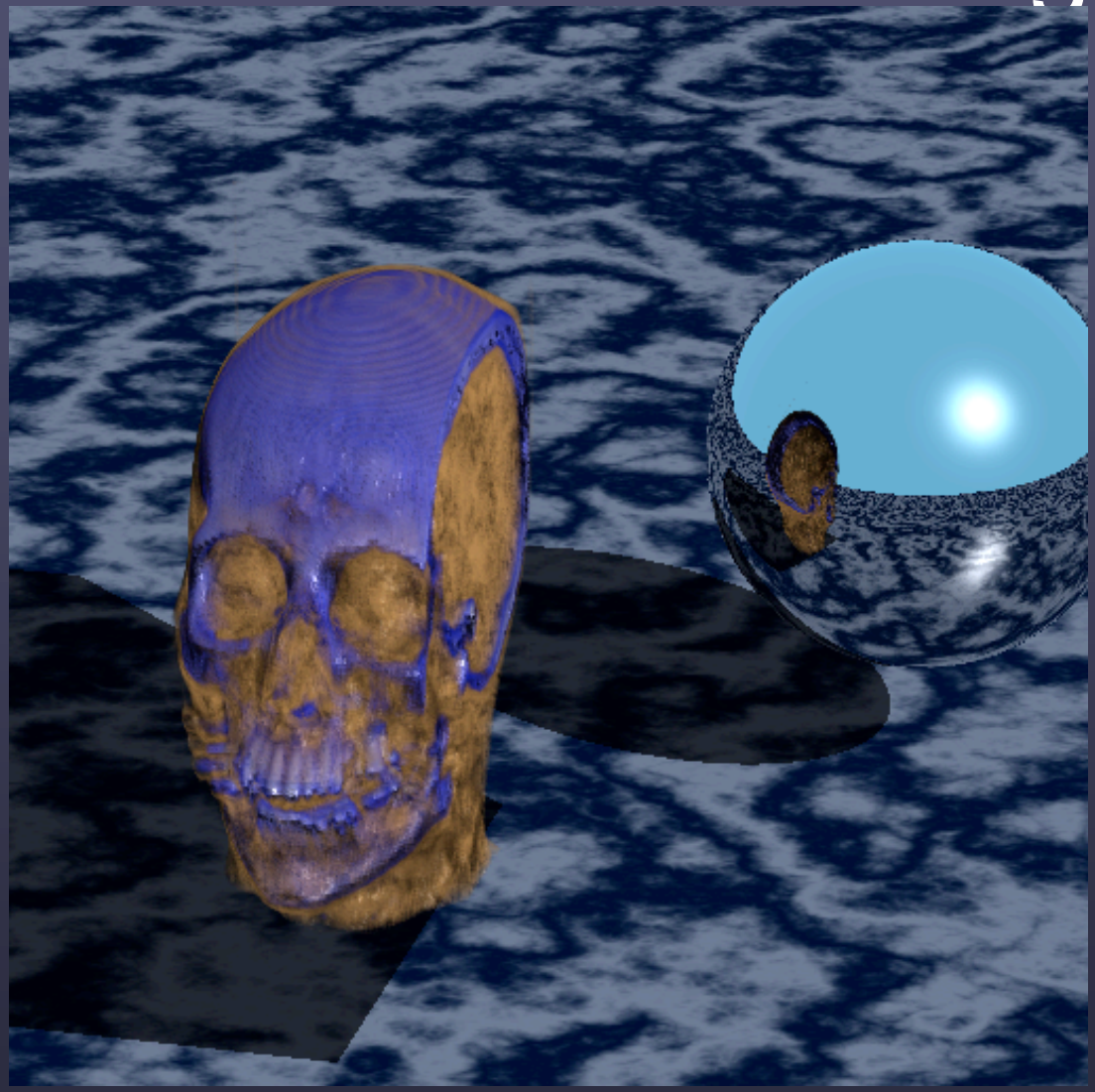
- Normal vector: gradient of opacity(P)

Software architecture

- Where do we put this code?
  - Material?
  - Object?
- Surface shader vs. volume shader

Program 8 - Volume Rendering

- Note incorrect shadows
- Note that head is “cut-away”



Material Pseudocode

Constructor:

Read in volume, colormap (code provided)

Possibly byteswap data (code provided)

diag = upper-lower

cellsize = diag * Vector(1./(nx-1), 1./(ny-1), 1./(nz-1));

world_stepsize = cell_stepsize*cellsize.length()/cbrt(3)

Call colormap.rescale to get world space attenuation values

Material pseudocode

In shade method (setup):

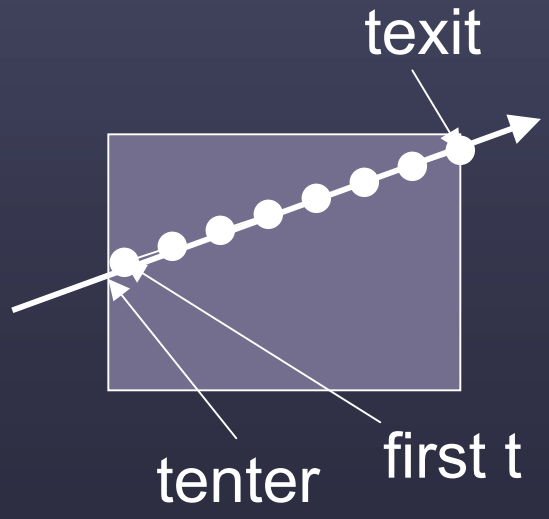
Intersect hit primitive to find exit point
(texit = tenter + thit)

Round to nearest sample:

```
it = Ceil(tenter/stepsize);
t = it*stepsize;
```

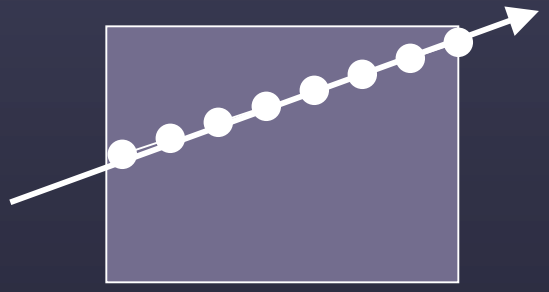
accum_color = 0,0,0

accum_opacity = 0



Material Pseudocode

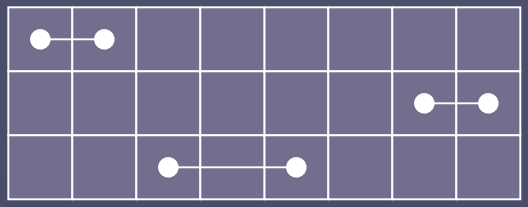
```
In shade method (loop):
while(t<texit){
  P = ray.origin + t* ray.direction;
  transform P to lattice coordinates
  value = tri-linearly interpolate data value at P
  opacity,color = lookup color/interpolate from
    colormap (code provided)
  if(opacity > 0)
   shade (next slide)
  t += world_stepsize
}
```



Material Pseudocode

```
shade points:
   get normal (next slide)
   compute phong lighting (no shadows)
   color = difflight * color
         + speclight * phong_color;
   accum_color +=
         color * opacity * (1-accum_opacity)
   accum_opacity +=
         opacity * (1-accum_opacity)
```

Material Pseudocode



Compute normal:

Use data gradient:

if i==0:

Nx = (data(i+1,j,k) - data(i,j,k))*cellsize.x

else if i == nx-1

Nx = (data(i,j,k) - data(i-1,j,k))*cellsize.x

else

Nx = (data(i+1,j,k)-data(i-1,j,k))*2*cellsize.x

Ny, Nz similar

normalize

what if length == 0? punt (use normal=ray.direction)

Material Pseudocode

Material should be “see-through”

Cast a final ray

origin = ray.origin + ray.direction * texit

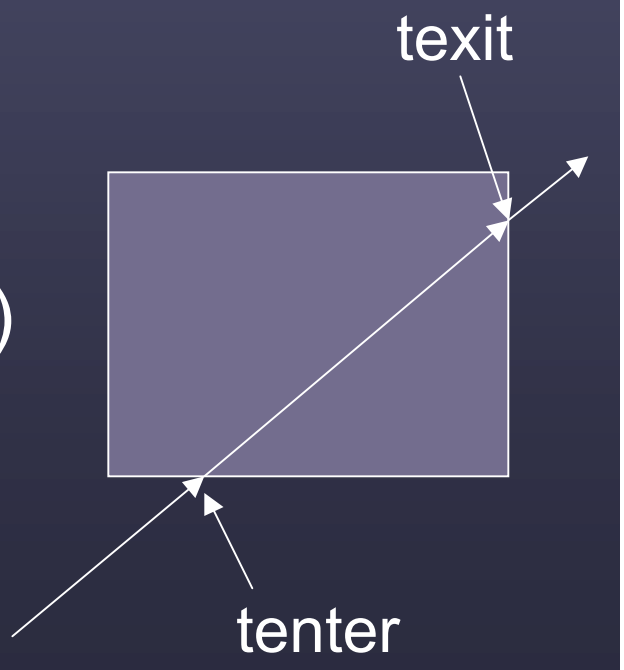
direction = ray.direction

exit_color = trace ray (…)

accum_color +=

exit_color * (1-accum_opacity)

Final result is accum_color



Colormap class

- Constructor: read colors, attenuation from simple text file
- rescale method (world_stepsize): convert attenuations into step-wise opacity
- lookup method: binary search + linear interpolation to get color/opacity for data value
- Code on the web site

Simple optimizations

- Stop accumulating if accumulated opacity gets close to 1 (I used 0.999)
- Don’t trace transmitted ray if accumulated opacity gets close to 1
- Use tiled/bricked memory layouts (I did not)
- Incrementally compute lattice coordinates (I did not, may not be a win)

Complex optimizations

- Use adaptive stepping
- Use adaptive grid
- Make a faster way to skip over transparent space (maybe adaptive grid)

Possible improvements

- Use a better transfer function
- Change shadow infrastructure to allow attenuated shadows
- Allow self-shadowing by tracing shadow rays at each point
- Combine above two effects
- Full scattering model (nice for clouds, but slow)

Extra credit

- Implement attenuated shadows to get correct shadows

OR

- Use volume rendering infrastructure to implement hypertextures

Course roadmap

- Almost 2/3 done
- Hopefully been fun so far
- Fun part is just beginning
- Remaining assignments:
  - Instancing (lecture on Friday)
  - Distributed ray tracing (your choice of glossy reflections, motion blur, depth-of-field, etc.)
  - Final project (your choice, possible contest)