Volume Rendering, Part 3
Announcements

• VisIt lecture: posted!
• Grading: will happen soon
• Projects:
  – ~30 hours
  – Aim for something worthy of your resume
• Pre-defined project: posted!
• Piazza:
Review
Volume rendering

• Important visualization technique for 3D data
• Use combination of color and transparency to see entire 3D data set at one time.

There are multiple ways to do volume rendering. I will describe one way today (raycasting). That will help explain the technique. I will describe alternate ways on Friday.
Ray casting game plan:
For every pixel on the screen,
Find ray for that pixel
Intersect volume with ray
Calculate color from intersection
Assign color to pixel
Outline

• Find Ray For That Pixel
• Intersect Volume With Ray
• Calculate Color From Intersection
• Assign Color To Pixel
Outline

• Find Ray For That Pixel
• Intersect Volume With Ray
• Calculate Color From Intersection
• Assign Color To Pixel
How do we specify a camera?

The “viewing pyramid” or “view frustum”.

Frustum: In geometry, a frustum (plural: frusta or frustums) is the portion of a solid (normally a cone or pyramid) that lies between two parallel planes cutting it.

class Camera
{
  public:
    double near, far;
    double angle;
    double position[3];
    double focus[3];
    double up[3];
};
This answers the “find ray for this pixel” question
Outline

• Find Ray For That Pixel
• Intersect Volume With Ray
• Calculate Color From Intersection
• Assign Color To Pixel
Intersect Volume With Ray

What is the result of the ray-volume intersection?
How would you calculate this intersection?
Ray-Volume Intersection: sampling

Sampling is the most common method for “ray-casting” volume rendering

Do we know how to do this sampling?
How to sample quickly

• Multiple strategies.
• For now, similar as before:
  – Find first cell intersected
    • Intersection is at a face
  – Find where ray exits that cell
    • Are there samples within the cell? Then sample them?
  – Go to next cell (which shares a face) and repeat
  – Keep going until you exit the volume, one cell at a time, and see what samples it covers

Approximately how many samples will we calculate?
Outline

• Find Ray For That Pixel
• Intersect Volume With Ray
• Calculate Color From Intersection
• Assign Color To Pixel
Transfer Function

Volume
Var: hardyglobal
Units: Joules

Max: 5.890
Min: 1.096
“Alpha Channel”

• Represents opacity
  – 1.0 or 255: fully opaque
  – 0: fully transparent

• Stored alongside RGB
  – Referred to as RGBA

• Floating point (1.0) vs byte (255):
  – Precision vs uniformity with RGB & performance
Applying a transfer function

<table>
<thead>
<tr>
<th>Sample</th>
<th>Scalar Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5.8</td>
</tr>
<tr>
<td>1</td>
<td>4.7</td>
</tr>
<tr>
<td>2</td>
<td>5.8</td>
</tr>
<tr>
<td>3</td>
<td>3.5</td>
</tr>
<tr>
<td>4</td>
<td>1.1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R</th>
<th>G</th>
<th>B</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>255</td>
<td>0</td>
<td>0</td>
<td>1.0</td>
</tr>
<tr>
<td>255</td>
<td>255</td>
<td>0</td>
<td>0.75</td>
</tr>
<tr>
<td>255</td>
<td>0</td>
<td>0</td>
<td>1.0</td>
</tr>
<tr>
<td>0</td>
<td>255</td>
<td>0</td>
<td>0.5</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>255</td>
<td>0</td>
</tr>
</tbody>
</table>

Quiz: calculate the results from transfer function for each sample
Transparency

• Quiz: If you have a red square that is 50% opaque in front of a black background, what color would you see?
  – Represent your answer in terms of (R, G, B)

Answer: (128, 0, 0)
Formula For Transparency

• Front = (Fr,Fg,Fb,Fa)
  – a = alpha, transparency factor
    • Sometimes percent
    • Typically 0-255, with 255 = 100%, 0 = 0%

• Back = (Br,Bg,Bb,Ba)

• Equation = (Fa*Fr+(1-Fa)*Ba*Br,
  Fa*Fg+(1-Fa)*Ba*Bg,
  Fa*Fb+(1-Fa)*Ba*Bb,
  Fa+(1-Fa)*Ba)

Alpha component is important! Any observations?
Outline

• Find Ray For That Pixel
• Intersect Volume With Ray
• Calculate Color From Intersection
• Assign Color To Pixel
Assign Color To Pixel

• Allocate a buffer for storing RGB values
  – Buffer should have one RGB for every pixel on the screen.

• As you calculate color for a ray, assign that color to its corresponding buffer entry

• When you have all of the colors, put that image up on the screen, as if you had rendered it using graphics cards.
Ray-Volume Intersection: sampling

What happens when we change the sampling rate?

Imagine if we had half or twice as many transparent squares...

Should the picture change if we change the sampling rate?
Opacity Correction

The assigned opacity also depends on the sampling rate. For example, when using fewer slices, the opacity has to be scaled up, so that the overall intensity of the image remains the same. Equation 3 is used for correcting the transfer function opacity whenever the user changes the sampling rate $s$ from the reference sampling rate $s_0$:

**Equation 3 Formula for Opacity Correction**

$$A = 1 - \left(1 - A_0\right)^{s_0/s}$$

New Material: Volume Rendering
Different Types of Volume Rendering

• Image Order: iterate over pixels
• Object Order: iterate over data
Object Order: Splatting

• Turn every point into a disk (aligned to the camera)
  – Color and transparency vary
• Render the splats from back to front using graphics hardware.
Object Order: Transparent Planes

• Strategy:
  – Slice the volume by many planes (200-1000)
  – Apply transfer function to each vertex on the plane
    • Result: plane with variation in color and transparency
  – Render the planes from back to front

These can be quickly rendered using “2D textures” or “3D textures”.

Image from VTK book
Volume Rendering: A Framework For Rendering

• Compositing: use combination of color and transparency to enable visualization of entire 3D data

• Alternate ideas:
  – maximum value along ray
  – average value along ray
  – distance to key value along ray
Ray functions: compositing
Ray functions: maximum
Ray functions: average value
Ray functions: distance to value
Volume rendering

• More on volume rendering
  – Shading
  – Multi-variate volume rendering
  – Optimizations
  – Combinations with surfaces
Shading surfaces

This is done by calculating surface normal and then calculating light reflection (or lack of light reflection from light source)
Shading volumes

Want to do all the same lighting equations, but we need a surface normal ... for a volume. What to do?

Answer: use gradient of field for “surface” normal
Volume rendering

- More on volume rendering
  - Shading
  - Multi-variate volume rendering
  - Optimizations
  - Combinations with surfaces
Multi-variate volume rendering

• Simplest form
Multi-variate transfer functions

From cs.utah.edu/~jmk
Multi-variate transfer functions
Volume rendering

• More on volume rendering
  – Shading
  – Multi-variate volume rendering
  – Optimizations
  – Combinations with surfaces
Optimizing Volume Rendering

• Big topic:
  – How to find samples quickly?
  – How to use advanced HW (GPUs) efficiently?

• Early ray termination
  – Just stop going when opacity gets greater than some threshold.
    • You do this for pre-defined project
Volume rendering

• More on volume rendering
  – Shading
  – Multi-variate volume rendering
  – Optimizations
  – Combinations with surfaces
Surface Rendering + Volume Rendering

How was this picture made?
Unstructured Meshes
An example mesh

Why do you think the triangles change size?
Types of Meshes

- Curvilinear
- Adaptive Mesh Refinement
- Unstructured
Curvilinear Mesh

• Logically rectilinear
  – Each cell has an \((i, j)\)
  – Always left, right, bottom, top neighbor (unless on boundary)

• Points can be anywhere ... as long as the cells don’t overlap

VTK calls this “vtkStructuredGrid”
Curvilinear Mesh

• A curvilinear mesh has 5x5 cells and a cell-centered variable stored.

• Quiz: how many bytes to store this data set if all data is in single precision floating point?

2 ints + 6x6 floats (points) + 5x5 floats (variable) = 2*4+36*4+25*4 = 63*4 = 272 bytes
Example unstructured mesh

- Meshes contain:
  - Cells
  - Points
- This mesh contains 3 cells and 13 vertices
- Pseudonyms:
  - Cell == Element == Zone
  - Point == Vertex == Node
If we stored each cell like this, how many bytes would it take? (assume single precision)

Let’s call this the “explicit” scheme.

A: 1 int (# cells), 3 ints (# pts per cell), 24+24+15 floats = 268 bytes

• Cell 0:
  - \{ (0, 0, 0), // 10
  - (1, 0, 0), // 11
  - (1, 1, 0), // 8
  - (0, 1, 0), // 7
  - (0, 0, 1), // 4
  - (1, 0, 1), // 5
  - (1, 1, 1), // 2
  - (0, 1, 1) \} // 1
Example unstructured mesh

- Pts:
  - \{ (0, 1, 1), (1, 1, 1), (2, 1, 1), (0, 0, 1), (1, 0, 1), (2, 0, 1), (0, 1, 0), (1, 1, 0), (2, 1, 0), (0, 0, 0), (1, 0, 0), (2, 0, 0), (2.5, 0.5, 0.5) \}

- Cells:
  - 3 cells
    - 1st cell: hexahedron – \( (10, 11, 8, 7, 4, 5, 2, 1) \)
    - 2nd cell: hexahedron – \( (11, 12, 9, 8, 5, 6, 3, 2) \)
    - 3rd cell: prism – \( (13, 3, 6, 12, 9) \)

If we stored each cell like this, how many bytes would it take? (assume single precision)

A: 1 int (# pts), 1 int (# cells), 3 ints (# cell type), 13*3 floats (pts), 8+8+5 ints = 260 bytes

Let’s call this the “connectivity” scheme
Comparing unstructured mesh storage schemes

• Hexahedral meshes: each internal point incident to 8 cells
  – Explicit scheme:
    • represent that point 8 times: 24 floats for storage
  – Connectivity scheme:
    • represent that point once in point list, 8 times in connectivity list: 3 floats + 8 ints

• (takeaway: connectivity wins!)

Further benefit to connectivity scheme is in finding exterior faces.
Finding external faces: motivation

• Interval volume, clip:
  – Take data set (rectilinear, unstructured, or other) and produce unstructured mesh
  – When rendering, only want to render the faces on the outside (the inside aren’t visible)

Question: what proportion of faces are exterior?

Question: how to find exterior faces?
Finding external faces: algorithm

• For each face, count how many cells are incident.
  – If “1”, then external
  – If “2”, then interior

Question: why does this work?
Finding exterior faces: algorithm

• Estimate # of faces (ncells * 6 / 2)
• Double that number
• Create a hash table of that size
• For each cell C
  – For each face F of C
    • Create hash index for F based on connectivity indices
    • Search hash table
      – If F already there, remove F from hash
      – If face not there, add F to hash

• All faces in hash are exterior
Interpolation for arbitrary cells: tetrahedrons

- Assume tetrahedron $T$, point $P$ in $T$
- Goal: calculate $F(P)$
Interpolation for arbitrary cells: tetrahedrons

• Assume tetrahedron $T$, point $P$ in $T$
• Goal: calculate $F(P)$

Set up parametric coordinate system
Interpolation for arbitrary cells: tetrahedrons

- Assume tetrahedron $T$, point $P$ in $T$
- Goal: calculate $F(P)$

Calculate parametric coordinates $(a, b, c)$

$$P = P0 + aR + bS + cT$$

This is a 3x3 matrix solve. This matrix is invertible since $R, S, T$ form a basis.
Interpolation for arbitrary cells: tetrahedrons

- Assume tetrahedron $T$, point $P$ in $T$
- Goal: calculate $F(P)$

$$P = P_0 + aR + bS + cT$$
$$F(P) = \text{sum}(W_i * F(P_i))$$

- $W_0 = 1 - a - b - c$
- $W_1 = a$
- $W_2 = b$
- $W_3 = c$

Calculate $F(P)$ as weighted sum of vertices
General idea

- Set up parametric coordinate system
- Calculate parametric coordinates for P
- Calculate $F(P)$ as $\text{Sum}(W_i*F(P_i))$
  - Weights $W_i$ can get pretty tricky.

VTK book has weights & good description in Ch. 8.2.
How to do contouring

• Basically the same:
  – Iterate over cells
  – Identify case
  – Lookup case in table
  – Create resulting geometry

• Difference:
  – New tables for each cell type
How to do ray casting

• Basically the same:
  – Cast rays for every pixel on the screen
  – Sample along rays
  – Apply transfer function
  – Composite front to back
  – Assign color to image

• Differences:
  – Sampling gets hard!
    • Which cell contains a sample point?
      – Need smart data structures ....
      – .... Or a way to transform data to make it easy
How to do particle advection

• Basically the same:
  – Start with a seed location
  – Evaluate velocity
  – Displace particle
  – Repeat until termination criteria reached

• Differences:
  – Evaluating velocity different x2:
    • Now a harder proposition: which cell contains particle?
    • Now more math: how to LERP velocity?
Adaptive Mesh Refinement

- Put resolution where you want it...
- ... but simpler than unstructured (indexing & interpolation) and cheaper (memory)
- Problems:
  - Everything great, except at the boundaries....
And now...

• You know everything I think you should know after having taken a sci-vis class.
  – Still need unstructured mesh lectures

• I am proud of this class.

• Next few lectures should be fun.

• Projects will hopefully be fun too.