Volume Rendering, Part 1

November 13th, 2013

Hank Childs, University of Oregon
Announcements

• Final project
  – Come in & talk with me

• OH:
  – Weds 10-11:30
  – Thurs 10-11

• Who wants to take 610 class?
Project #7A

• You write VTK program.
  – I expect that your answers will be partially derived from previous VTK programs
  – Python OK

• 4 renderers in 1 window

<table>
<thead>
<tr>
<th>three slices</th>
<th>streamlines</th>
</tr>
</thead>
<tbody>
<tr>
<td>isosurface</td>
<td>hedgehogs</td>
</tr>
</tbody>
</table>
Proj #6A

```c
int pt1 = j*dims[0]+i;
int pt2 = j*dims[0]+i+1;
int pt3 = (j+1)*dims[0]+i;
int pt4 = (j+1)*dims[0]+i+1;
int idx = 0;
if (F[pt1] < isoval)
    idx += 1;
if (F[pt2] < isoval)
    idx += 2;
if (F[pt3] < isoval)
    idx += 4;
if (F[pt4] < isoval)
    idx += 8;
```

Can do even better with bitmasks.
for (int s = 0; s < nsegments[idx]; s++)
{
    float pt[2][2];
    for (int e = 0; e < 2; e++)
    {
        int edgeIdx = lup[idx][2*s+e];
        if (edgeIdx == 0)
        {
            float t = (isoval-F[pt1])/(F[pt2]-F[pt1]);
            pt[e][0] = X[i]+t*(X[i+1]-X[i]);
            pt[e][1] = Y[j];
        }
        else if (edgeIdx == 1)
        {
            float t = (isoval-F[pt2])/(F[pt4]-F[pt2]);
            pt[e][0] = X[i+1];
            pt[e][1] = Y[j]+t*(Y[j+1]-Y[j]);
        }
        else if (edgeIdx == 2)
        {
            float t = (isoval-F[pt3])/(F[pt4]-F[pt3]);
            pt[e][0] = X[i]+t*(X[i+1]-X[i]);
            pt[e][1] = Y[j+1];
        }
        else if (edgeIdx == 3)
        {
            float t = (isoval-F[pt1])/(F[pt3]-F[pt1]);
            pt[e][0] = X[i];
            pt[e][1] = Y[j]+t*(Y[j+1]-Y[j]);
        }
    }
    sl.AddSegment(pt[0][0], pt[0][1], pt[1][0], pt[1][1]);
}
Proj. 6B

Great job!!

Put these on your resumes
int caseIdx = 0;
int ptIdx[8];
int case_mask[8] = { 1, 2, 4, 8, 16, 32, 64, 128 };  
for (int v = 0; v < 8; v++)
{
    int k2 = k + (v/4);
    int j2 = j + (v/2)%2;
    int i2 = i + (v%2);
    int idx = k2*(dims[0]*dims[1]) + j2*dims[0] + i2;
    ptIdx[v] = idx;
    if (F[idx] < isoval)
        caseIdx |= case_mask[v];
}
int *verts = triCase[hexCase];
while (*verts >= 0)
{
    int e[3];
e[0] = *verts;
e[1] = *(verts+1);
e[2] = *(verts+2);
static int edges[12][2] = {{0,1}, {1,3}, {2,3}, {0,2},
                            {4,5}, {5,7}, {6,7}, {4,6},
                            {0,4}, {1,5}, {2,6}, {3,7}};
float pts[9];
for (int eIdx = 0; eIdx < 3; eIdx++)
{
    int vert1 = edges[e[eIdx]][0];
    int vert2 = edges[e[eIdx]][1];
    float t = (isoval-F[ptIdx[vert1]])/
               (F[ptIdx[vert2]]-F[ptIdx[vert1]]);
    int leftX = vert1%2 + i;
    int rightX = vert2%2 + i;
    int bottomY = (vert1/2)%2 + j;
    int topY = (vert2/2)%2 + j;
    int frontZ = vert1/4 + k;
    int backZ = vert2/4 + k;
    pts[3*eIdx+0] = t*(X[rightX]-X[leftX]) + X[leftX];
    pts[3*eIdx+1] = t*(Y[topY]-Y[bottomY]) + Y[bottomY];
    pts[3*eIdx+2] = t*(Z[backZ]-Z[frontZ]) + Z[frontZ];
}
    tl.AddTriangle(pts[0], pts[1], pts[2], pts[3], pts[4],
    pts[5], pts[6], pts[7], pts[8], hexCase);
    verts += 3;
}
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.
Volume rendering overview

Camera

Pixels on the screen

3D data
Volume rendering overview

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

Which of these 4 steps will be easy? Which will be hard?
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];
};
NOT doing it the graphics way (yet)

World space:
- Triangles in native Cartesian coordinates
- Camera located anywhere

Camera space:
- Camera located at origin, looking down -Z
- Triangle coordinates relative to camera frame

Image space:
- All viewable objects within -1 <= x,y,z <= +1

Screen space:
- All viewable objects within -1 <= x, y <= +1

Device space:
- All viewable objects within 0 <= x <= width, 0 <= y <= height

We don’t need to do it this way for volume rendering (although we could)
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?
Ray-Volume Intersection

How would you calculate this intersection?
Ray-Volume Intersection: How do you do it?

• Find first cell intersected
  – Intersection is at a face

• Find where ray exits that cell
  – That face is shared with another cell

• Keep going until you exit the volume, one cell at a time.

What would this technique give you?
What data would we be working with?
Can we calculate the intersections at the faces? Can we calculate the data at the faces?

Is this a good idea? Why or why not?

(it is not a good idea if we interpolate between the face values)

How do we fix this?
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
Refresher: Color Maps

Pseudocolor
Var: hardyglobal
Units: Joules

1.096  2.294  3.493  4.691  5.890
Transfer function

- Assigns a color and an opacity to each scalar value.

<table>
<thead>
<tr>
<th>Technique</th>
<th>Abstraction for assigning color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pseudocoloring</td>
<td>Color Map</td>
</tr>
<tr>
<td>Volume Rendering</td>
<td>Transfer Function</td>
</tr>
</tbody>
</table>
Transfer Function

Volume
Var: hardyglobal
Units: Joules

Max: 5.890
Min: 1.096
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

• If you have an opaque red square in front of a blue square, what color would you see?
  – Red

• If you have a 50% transparent red square in front of a blue square, what color would you see?
  – Purple

• If you have a 100% transparent red square in front of a blue square, what color would you see?
  – Blue
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)*Br,
  Fa*Fg+(1-Fa)*Bg,
  Fa*Fb+(1-Fa)*Bb,
  Fa+(1-Fa)*Ba)

Alpha component is important! Any observations?
Transparency

• If you have an 25% transparent red square (255,0,0) in front of a blue square (0,0,255), what color would you see (in RGB)?
  – (192,0,64)

• If you have an 25% transparent blue square (0,0,255) in front of a red square (255,0,0), what color would you see (in RGB)?
  – (64,0,192)
Calculating Color

<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></td>
</tr>
<tr>
<td>3</td>
<td></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></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>255</td>
<td>0</td>
</tr>
</tbody>
</table>

Note: we will talk more about alpha

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

Apply this equation over and over until you run out of samples (then use background color)
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 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

Which of these 4 steps were easy? Which were hard?
Next Time

• A lot more about volume rendering
  – More techniques
  – Multi-variate volume rendering
  – Lighting calculations
  – Optimizations
  – Combinations with surface rendering