Texture-based Volume Rendering

- Volume rendering by ray casting is time-consuming
  - one ray per pixel
  - each ray involves tracking through volume calculating samples, and then compositing
    - different for each viewpoint
- Alternative approach - using texture maps - can exploit graphics hardware

Texture Mapping

- Modern graphics hardware includes facility to draw a textured polygon
- The texture is an image with red, green, blue and alpha components...
- ...so several overlapping polygons can be composited

Texture-based Volume Rendering

- Draw from back-to-front a set of rectangles
  - first rectangle drawn as an area of coloured pixels, with associated opacity, as determined by transfer function and interpolation - and merged with background in a compositing operation (supported by hardware)
  - successive rectangles drawn on top

Texture-based Volume Rendering

- For a given viewing direction, we would need to select slices perpendicular to this direction
- This requires interpolation to get the values on the slices
- Only expensive 3D texture hardware can do this fast enough...

Texture-based Volume Rendering

- Simpler solution - 2D texture mapping:
  - view volume as set of slices parallel to coordinate planes
  - choose the orientation best suited to viewing direction

Texture Mapping

- 2D image + 2D polygon → Textured-mapped polygon
Texture Mapping (2)

Each texel has 2D coordinates assigned to it. assign the texture coordinates to each polygon to establish the mapping.

Tex. Mapping for Volume Rendering

Remember ray casting …

Texture based volume rendering

• Render each xz slice in the volume as a texture-mapped polygon
• The texture contains RGBA (color and opacity)
• The polygons are drawn from back to front

Changing Viewing Direction

What if we change the viewing position?

Changing View Direction (2)

Until …

You are not going to see anything this way …

This is because the view direction now is Parallel to the slice planes

What do we do?
Switch Slicing Planes

What do we do?
- Change the orientation of slicing planes
- Now the slice polygons are parallel to YZ plane in the object space

Some Considerations... (5)

When do we need to change the slicing orientation?

Some Considerations... (6)

Major component of view vector?

Given the view vector \((x, y, z)\) -> get the maximal component

- If \(x\): then the slicing planes are parallel to \(yz\) plane
- If \(y\): then the slicing planes are parallel to \(xz\) plane
- If \(z\): then the slicing planes are parallel to \(xy\) plane

- This is called (object-space) **axis-aligned** method.

Three copies of data needed

- We need to reorganize the input textures for different view directions.
- Reorganize the textures on the fly is too time consuming. We want to prepare the texture sets beforehand

Texture based volume rendering

Algorithm: (using 2D texture mapping hardware)

- Turn off the depth test; Enable blending
- For (each slice from back to front) {
  - Load the 2D slice of data into texture memory
  - Create a polygon corresponding to the slice
  - Assign texture coordinates to four corners of the polygon
  - Render and blend the polygon (use OpenGL alpha blending) to the frame buffer
}

Problem (1)

**Non-even sampling rate**

\(d'' > d' > d\)

Sampling artifact will become visible
Problem (2)

Object-space axis-aligned method can create artifacts:

*Popping Effect*

There is a sudden change of slicing direction when the view vector transitions from one major direction to another. The change in the image intensity can be quite visible.

Solution (1)

*Insert intermediate slides to maintain the sampling rate*

Solution (2)

Use Image-space axis-aligned slicing plane:

*the slicing planes are always parallel to the view plane*

3D Texture Based Volume Rendering

3D Texture Mapping

Arbitrary slicing through the volume and texture mapping capabilities are needed

- Arbitrary slicing polygon: this can be computed using software in real time

This is basically polygon-volume clipping

Slice-Interpolated Volume Rendering

Similar to raycasting with simultaneous rays
3D Texture Mapping

Texture mapping to the arbitrary slices

This requires 3D texture mapping hardware

Input texture: volume (pre-classified and shaded) essentially an (R,G,B,α) volume

Depending on the position of the polygon, appropriate textures are resampled, constructed and mapped to the polygon.

Solid (3D) Texture Mapping

Now the input texture space is 3D

Texture coordinates: (r,s,t)

Slice Based Rendering

Slices

Graphics Hardware

• Polygons – Proxy geometry
• Textures – Data & interpolation
• Blending operations – Numerical integration

Slice Based Problems?

※ Does not perform correct
  ◆ Illumination
  ◆ Accumulation - but can get close
※ Can not easily add correct illumination and shadowing
  ◆ See the Van Gelder paper for their addition for illumination
  ◆ Stored in LUT quantized normal vector directions
  ◆ See Kniss papers (Utah) for use of vertex shaders and new hardware to solve many of these problems.

Lighting and Shading

※ 3D texture mapping with hardware tricks to achieve lighting is becoming feasible.
Pros and Cons

<table>
<thead>
<tr>
<th>Advantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Fast with volume sizes that the hardware can take e.g. 2 fps for 256 cube volumes</td>
</tr>
<tr>
<td>- No popping effect</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Need to compute the slicing planes for every view angle</td>
</tr>
<tr>
<td>- only supported on high end hardware</td>
</tr>
<tr>
<td>- low quality without per-pixel classification shading and classification (i.e. post-classification and shading)</td>
</tr>
</tbody>
</table>

Both 2D or 3D hardware texture mapping methods can not compute shading on the fly. The input textures have to be pre-shaded.

With multi-texturing functions, per-pixel shading and classification are becoming possible.