CS 636
Advanced Rendering Techniques

Dr. David Breen
Wednesday 6PM → 8:50PM

Presentation 9
6/1/22
Logistics

• Deadline for last assignment is
  – Sunday night 6/5/22

• Deadline for extra credit assignment is
  – Friday night 6/10/22

• Online final exam 6/8/22 at 6PM

• Course evaluations
Questions from Last Time?

• 3D textures
• Procedural textures
• Perlin noise
Slide Credits

- Jian Huang - University of Tennessee
- Gordon Kindlmann & James Durkin - Cornell University
- Jonathan Cohen - Johns Hopkins University
- Pat Hanrahan - Stanford University
Direct Volume Rendering (DVR): Ray-casting

Jian Huang

This set of slides references slides used by Prof. Torsten Moeller (Simon Fraser), Prof. Han-Wei Shen (Ohio State).
Papers

• Tuy and Tuy, 1984, IEEE CG & A (one of the earliest volume rendering techniques)
• Levoy, 1988 IEEE CG&A, and later improvements
• Drebin, Carpenter, Hanrahan, 1988, SIGGRAPH

Direct: No conversion to surface geometry
Basic Idea

Based on the idea of ray tracing

• Trace a ray from each pixel into object space
• Compute color value along the ray
• Assign the value to the pixel
Data Representation

- 3D volume data are represented by a finite number of cross-sectional slices (hence a 3D raster).
- On each volume element (voxel), stores a data value (if it uses only a single bit, then it is a binary data set. Normally, we see a gray value of 8 to 16 bits on each voxel.)

\[ N \times 2D \text{ arrays} = 3D \text{ array} \]
What is a Voxel? – Two definitions

A voxel is a cubic cell, which has a single value cover the entire cubic region.

A voxel is a data point at a corner of the cubic cell. The value of a point inside the cell is determined by interpolation.
Viewing

Ray Casting

- Where to position the volume and image plane
- What is a ‘ray’
- How to march a ray
Early Methods

• Before 1988
• Did not consider transparency
• did not consider sophisticated light transportation theory
• were concerned with quick solutions
• hence more or less applied to binary data

non-binary data - require sophisticated classification/compositing methods!
Ray Tracing -> Ray Casting

- “another” typical method from traditional graphics
- Typically we only deal with primary rays - hence: ray-casting
- a natural image-order technique
- as opposed to surface graphics - how do we calculate the ray/surface intersection???
- Since we have no surfaces - we need to carefully step through the volume
Ray Casting

- Stepping through the volume: a ray is cast into the volume, sampling the volume at certain intervals.
  - The sampling intervals are usually equi-distant, but don’t have to be (e.g. importance sampling).
- At each sampling location, a sample is interpolated / reconstructed from the grid voxels.
  - Popular filters are: nearest neighbor (box), trilinear (tent), Gaussian, cubic spline.
- Along the ray - what are we looking for?
Example: Using the nearest neighbor kernel

In Tuys’ paper

\[ Q = P + K \times V \]

At each step \(k\), \(Q\) is rounded off to the nearest voxel (like the DDA algorithm)

Check if the voxel is on the boundary or not (compare against a threshold)

If yes, perform shading
Basic Idea of Ray-casting Pipeline

- Data are defined at the corners of each cell (voxel)

- The data value inside the voxel is determined using interpolation (e.g. tri-linear)

- Composite colors and opacities along the ray path

- Can use other ray-traversal schemes as well
Ray Traversal Schemes

Intensity
Max
Average
Accumulate
First

Depth
Ray Traversal - First

- **First**: extracts iso-surfaces (again!) done by Tuy&Tuy ’84
Ray Traversal - Average

- **Average**: produces basically an X-ray picture
Ray Traversal - MIP

- **Max**: Maximum Intensity Projection used for Magnetic Resonance Angiogram
Ray Traversal - Accumulate

- Accumulate opacity while compositing colors: make transparent layers visible!
  Levoy ‘88
Raycasting

volumetric compositing

object (color, opacity)

color

opacity

1.0
Raycasting

Interpolation kernel

volumetric compositing

object (color, opacity)
Raycasting

\[ \text{color } c = c_s \alpha_s (1 - \alpha) + c \]

\[ \text{opacity } \alpha = \alpha_s (1 - \alpha) + \alpha \]

object \((\text{color, opacity})\)

volumetric compositing

Interpolation kernel
Raycasting

volumetric compositing

object (color, opacity)
Raycasting

volumetric compositing

object (color, opacity)

1.0
Raycasting

volumetric compositing

Object (color, opacity)
Raycasting

volumetric compositing

object (color, opacity)
Raycasting

volumetric compositing

object(s) (color, opacity)
Volume Rendering Pipeline

- Acquired values
- Data preparation
- Prepared values
  - shading
    - Voxel colors
    - Ray-tracing / resampling
      - Sample colors
  - classification
    - Voxel opacities
    - Ray-tracing / resampling
      - Sample opacities
  - compositing
    - Image Pixels
DCH DVR Pipeline

original raw density data

segmentation
classification

gradient computation

shading

resampling

compositing
DCH - Pipeline

Original data

Material percentage volumes

Classification

Density volume

Opacity volume

Color volume

Normals

Gradient

Shading

Shaded volume

shears

Transformed volume

compositing

Final image
Common Components of General Pipeline

• Interpolation/reconstruction
• Classification or transfer function
• Gradient/normal estimation for shading
  – Question: are normals also interpolated?
Levoy - Interpolation

eye

image pixel

viewing ray

voxel

trilinear interpolation

sample point
Levoy - Gradient/Normals

- Central difference
- per voxel

\[ G_x = \frac{v_{i+1,j,k} - v_{i-1,j,k}}{2} \]

\[ G_y = \frac{v_{i,j+1,k} - v_{i,j-1,k}}{2} \]

\[ G_z = \frac{v_{i,j,k+1} - v_{i,j,k-1}}{2} \]
Levoy - Compositing

- Image order
- back-to-front
- using the over operator

\[
C_{out} = C_{in} \cdot (1 - \alpha) + C \cdot \alpha \\
\alpha_{out} = \alpha_{in} \cdot (1 - \alpha) + \alpha
\]
Levoy - Shading

- Phong Shading + Depth Cueing

\[ C(x) = C_p k_a + \frac{C_p}{k_1 + k_2 d(x)} (k_d (N(x) \cdot L) + k_s (N(x) \cdot H)^n) \]

- \( C_p \) = color of parallel light source
- \( k_a / k_d / k_s \) = ambient / diffuse / specular light coefficient
- \( k_1, k_2 \) = fall-off constants
- \( d(x) \) = distance to picture plane
- \( L \) = normalized vector to light
- \( H \) = normalized vector for maximum highlight
- \( N(x_i) \) = surface normal at voxel \( x_i \)
Classification

Classification: Mapping from data to opacities

Region of interest: high opacity (more opaque)
Rest: translucent or transparent

The opacity function is typically specified by the user

Levoy came up with two formula to compute opacity
1. Isosurface
2. Region boundary (e.g. between bone and fresh)
Classification/Transfer Function

• Maps raw voxel value into presentable entities: color, intensity, opacity, etc.
  Raw-data → material ($R, G, B, \alpha, K_a, K_d, K_s, ...$)

• May require probabilistic methods (Drebin). Derive material volume from input. Estimate % of each material in all voxels. Pre-computed. AKA segmentation.

• Often use look-up tables (LUT) to store the transfer function that are discovered
Opacity function (1)

Goal: visualize voxels that have a selected threshold value $fv$

- No intermediate geometry is extracted
- The idea is to assign voxels that have value $fv$ the maximum opacity (say $\alpha$)
- And then create a smooth transition for the surrounding area from 1 to 0
- Levoy wants to maintain a constant thickness for the transition area.
Opacity function (2)

Maintain a constant isosurface thickness

Can we assign opacity based on function value instead of distance? (local operation: we don’t know where the isosurface is)

Yes – we can base it on the value distance \( f - f_v \) but we need to take into account the local gradient

\[
\text{opacity} = 0 \quad \text{opacity} = \alpha
\]
DCH - Material Percentage V.

- Probabilistic classifier
- Probability that a voxel has intensity I:
  \[ P(I) = \sum_{i=1}^{n} p_i P_i(I) \]
- \( p_i \) - percentage of material
- \( P_i(I) \) - prob. that material i has value I
- \( P_i(I) \) given through statistics/physics
- \( p_i \) then given by:
  \[ p_i(I) = \frac{P_i(I)}{\sum_{j=1}^{n} P_j(I)} \]
DCH - Classification

- Like Levoy - assumes only two materials per voxel
- that will lead to material percentage volumes

- from them we conclude color/opacity:

\[ C = \sum_{i=1}^{n} p_i C_i \]

- where \( C_i = (\alpha_i R_i, \alpha_i G_i, \alpha_i B_i, \alpha_i) \)
DCH- Classification

histogram

Constituent’s Distributions

Material Assignment

CT Number
Levoy - Improvements

• Levoy 1990
• front-to-back with early ray termination $\alpha = 0.95$
• hierarchical octree data structure
  - skip empty cells efficiently
Semi-Automatic Generation of Transfer Functions

G. Kindlmann, J. Durkin
Cornell University

Presented by Jian Huang, CS594, Spring 2002
Direct Volume Rendering

- Render the volume by computing the volume integration – Direct Volume Rendering
- Iso-surfacing: extract the iso-surfaces from the dataset, and render as surface geometry primitives
- Pros and cons: ? depend on who you talk to
Example

(a) Isosurface Rendering  (b) Direct Volume Rendering
What are we looking for?

- look for boundary regions between relatively homogeneous material in the scalar volume
- The boundary might be associated with a range of values
- Use an opacity function to modulate the parameters corresponding to this range
Getting a good transfer function

- Transfer function: assign renderable optical properties to the numerical values
- This paper focuses on the opacity functions
- Getting a good transfer function is tricky
Example
The Boundary Model

• Assumption
  – There exist a sharp, discontinuous change in the physical property of the entity
  – The data/signal has been low-pass filtered, (band-limited, or, blurred)
  – The blur is isotropic
  – The blurring function (low-pass filter) is Gaussian
The Boundary Model (2)

(a) Step function
(b) Gaussian
(c) Measured boundary prior to sampling
Directional Derivatives

(a) $f(x)$  
(b) Isolevels of $f$  
(c) $\nabla f$
Directional Derivatives (2)

$\frac{d}{dx} f''(x)$  
$\frac{d}{dx} f'(x)$  
$\frac{d}{dx} f(x)$
Relations between \( f, f', f'' \)

Figure 5: \( f, f' \) and position \( x \).

Figure 6: \( f, f'' \) and position \( x \).
Histogram Volume

- Measure the relationship between the data value and its derivatives.

Figure 7: The underlying relationship of $f$, $f'$, and $f''$. 
Histogram volume
records $f - f' - f''$ relationship
Histogram Creation

- Measure $f$ and its directional derivatives exactly once per voxel, at the original sample points of the data set
Implementation

1. Initialize the histogram volume to all 0’s.

2. Make one pass through the volume looking for the highest values of $f'$ and $f''$, and the lowest value of $f''$; assume 0 for the lowest value of $f'$. Set ranges on the histogram volume axes accordingly.

3. On a second pass through the volume,

   3a. Measure $f$, $f'$, and $f''$ at each voxel,

   3b. Determine which bin in the histogram volume corresponds to the measured combination of $f$, $f'$, and $f''$, and

   3c. Increment the bin’s value.
Histogram Volume Inspection

(a) Cylinder

(b) Nested Cylinders
More examples

(a) Turbine Blade

(b) Head

(c) Engine Block
Review

- Found underlying $f - f' - f''$ relationship
- Measured with histogram volume
- Verified measurement with scatterplots
To Do

Map histogram volume to opacity function

- Not completely automatic
- “Position” as intermediate domain
- Two stages:
  - “position function” : $x = p(v)$
  - “boundary emphasis function” : $\alpha = b(x)$

- Opacity function generation: $\alpha(v) = b(p(v))$
Position function $p(v)$

- Maps data value to “position relative to boundary”

- Inherent characteristic of dataset
- Calculated from histogram volume

\[ p(v) \approx -\sigma^2 f''(f^{-1}(v)) \frac{1}{f'(f^{-1}(v))} \]
Where is the boundary?

- Average 1\textsuperscript{st} directional derivative of $f$ over all the positions $x$ at which $f(x)$ equals $v$, 
  \[ p(v) = \frac{-\sigma^2 h(v)}{g(v)} \]
  \[ -g(v) \]

- Average 2\textsuperscript{nd} directional derivative of $f$ over all the positions $x$ at which $f(x)$ equals $v$, 
  \[ \approx \frac{-\sigma^2 f''(f^{-1}(v))}{f'(f^{-1}(v))} = x \]
  \[ p(v) = \frac{-\sigma^2 h(v)}{\max(g(v) - g_{thresh}, 0)} \]
Boundary emphasis function $b(x)$

- Maps from “position” to opacity
- User controls character of rendered boundaries

\[ \alpha = b(x) \]

- Opacity function generation: $\alpha(v) = b(p(v))$
Synthetic dataset: $p(v)$ result

Double sphere slice  $f - f'$  $f - f''$

$p(v)$
CT head: $p(v)$ result

CT head slice  $f-f'$  $f-f''$

$p(v)$
$\alpha = b(x)$

$\alpha(v)$

$\alpha(v)$

b(x), $\alpha(v)$, rendering
neuron: $p(v)$ result

Neuron slice  $f-f'$  $f-f''$

$0.57$  $p(v)$
$b(x)$, $\alpha(v)$, rendering
Two-dimensional opacity functions in use
Conclusions

• Divided opacity function task into two stages

• Position function, Boundary emphasis function

• Underlying $f - f' - f''$ relationship

• User has more intuitive interface
Splatting

Jian Huang, CS 594, Spring 2002

This set of slides reference slides made by Ohio State University alumni over the past several years.
Volumetric Ray Integration

object (color, opacity)
Splatting

• Lee Westover - Vis 1989; SIGGRAPH 1990
• Object order method
• Front-To-Back or Back-To-Front
• Original method - fast, poor quality
• Many many improvements since then!
  – Crawfis’ 93: textured splats
  – Swan’ 96, Mueller’ 97: anti-aliasing
  – Mueller’ 98: image-aligned sheet-based splatting
  – Mueller’ 99: post-classified splatting
  – Huang’ 00: new splat primitive: FastSplats
Splatting

- Volume = field of 3D interpolation kernel
  - One kernel at each grid voxel
- Each kernel leaves a 2D footprint on screen
  - Voxel contribution = footprint \( \cdot (C, \text{opacity}) \)
- Weighted footprints accumulate into image

voxel kernels

screen footprints = splats

screen
Splatting

- Volume = field of 3D interpolation kernel
  - One kernel at each grid voxel
- Each kernel leaves a 2D **footprint** on screen
  - Voxel contribution = footprint · (C, opacity)
- Weighted footprints accumulate into image
Splatting

• Volume = field of 3D interpolation kernel
  – One kernel at each grid voxel

• Each kernel leaves a 2D *footprint* on screen
  – Voxel contribution = footprint · (C, opacity)

• Weighted footprints accumulate into image

![Diagram of voxel kernels mapping to screen splats](image)
Splatting

• Volume = field of 3D interpolation kernel
  – One kernel at each grid voxel
• Each kernel leaves a 2D footprint on screen
  – Voxel contribution = footprint \cdot (C, \text{opacity})
• Weighted footprints accumulate into image

voxel kernels

screen footprints = splats
Footprint Extent

Approximate the 3D kernel \( h(x,y,z) \) extent by a sphere
Footprint Table

A popular kernel is a three-dimensional Gaussian (radially symmetric)

As 1D integration of 3D Gaussian is still a 2D Gaussian – we can just skip the Z integration and evaluate the Gaussian function on 2D image space after voxel projection

Generic footprint table

preprocessing
View-dependent footprint

It is possible to transform a sphere kernel into an ellipsoid

- The projection of an ellipsoid is an ellipse
- We need to transform the generic footprint table to the ellipse
View-dependent footprint (2)

\[ T^{-1}(x) = x' \]
Example Footprint at Different Resolutions
Footprint - principal idea

- Draw each voxel as a cloud of points (footprint) that spreads the voxel contribution across multiple pixels.
- Larger footprint -> larger spatial kernel extent -> lower frequency components -> more blurring
  - Large pixel/voxel ratio
Rendering a Splat

- Use texture mapping hardware to resample footprint table (either single density channel or separate classified r,g,b,a channels)

- Or, use FastSplats to render each splat as a graphics primitive of itself
Splatting - efficiency

• “footprint” - splatted (integrated) kernel
• if interpolation kernel is isotropic (spherical) then its footprint is independent of the view point (for orthographic viewing)
• for perspective - footprint can be approximated with an ellipse
• Hence, for common cases, we can pre-integrate it (efficient!)
• for perspective projection, to approximate, we have to compute the orientation of the ellipse
Splatting - Highlights

• Footprints can be pre-integrated
  – fast voxel projection

• Advantages over ray-casting:
  – Fast: voxel interpolation is in 2D on screen
  – More accurate integration (analytic for X-ray)
  – More accurate reconstruction (afford better kernels)
  – Only relevant voxels must be projected
Early Implementation – Axis Aligned Splatting

- Voxel kernels are added within axis-aligned sheets
- Sheets are composited front-to-back
- Sheets = volume slices most perpendicular to the image plane

volume slices

image plane at 30°

volume slices

image plane at 70°
Early Implementation – Axis Aligned Splatting

• Volume

volume slices

image plane

sheet buffer

compositing buffer
Early Implementation – Axis Aligned Splatting

- Add voxel kernels within first sheet
Early Implementation – Axis Aligned Splatting

• Transfer to compositing buffer
Early Implementation – Axis Aligned Splatting

- Add voxel kernels within second sheet
Early Implementation – Axis Aligned Splatting

• Composite sheet with compositing buffer
Early Implementation – Axis Aligned Splatting

• Add voxel kernels within third sheet
Early Implementation – Axis Aligned Splatting

- Composite sheet with compositing buffer
What Doesn’t Work?

• Mathematically, the early splatting methods only work for X-ray type of rendering, where voxel ordering is not important
  – Bad approximation for other types of optical models
• Object ordering is important in volume rendering, front objects hide back objects
  – need to composite splats in proper order, else we get bleeding of background objects into the image (color bleeding!)
• Axis-aligned approach add all splats that fall within a volume slice most parallel to the image plane, composite these sheets in front-to-back order
  – Incorrect accumulating on axis-aligned face cause popping
• A better approximation with Riemann sum is to use the image-aligned sheet-based approach
Problems Early Implementation – Axis Aligned Splatting

- In-accurate compositing, result in color bleeding and popping artifacts (Demo)!

Part of this voxel gets composited **before** part of this voxel.

Problem: “popping” of brightness when the image plane becomes more parallel to a different volume face.
Image-Aligned Sheet-Buffer

- Slicing slab cuts kernels into sections
- Kernel sections are added into sheet-buffer
- Sheet-buffers are composited
Image-Aligned Sheet-Buffer

- Slicing slab cuts kernels into sections
- Kernel sections are added into sheet-buffer
- Sheet-buffers are composited
Image-Aligned Sheet-Buffer

- Slicing slab cuts kernels into sections
- Kernel sections are added into sheet-buffer
- Sheet-buffers are composited

image plane \rightarrow sheet buffer

compositing buffer
Image-Aligned Sheet-Buffer

- Slicing slab cuts kernels into sections
- Kernel sections are added into sheet-buffer
- Sheet-buffers are composited

image plane

sheet buffer

compositing buffer
Image-Aligned Sheet-Buffer

- Slicing slab cuts kernels into sections
- Kernel sections are added into sheet-buffer
- Sheet-buffers are composited
Image-Aligned Sheet-Buffer

- Slicing slab cuts kernels into sections
- Kernel sections are added into sheet-buffer
- Sheet-buffers are composited
Image-Aligned Sheet-Buffer

- Slicing slab cuts kernels into sections
- Kernel sections are added into sheet-buffer
- Sheet-buffers are composited
IASB Splatting

- No popping or color bleeding
- Sharp, noise-free images
Occlusion Culling

• A voxel is only visible if the volume material in front is not opaque

occluded voxel: does not pass visibility test

wall of occluding voxels

occlusion map = opacity image
Anti-aliasing

- Needed to preserve small features
- Needed for the diverging rays in perspective
- In splatting, resize the footprint according to depth

Aliased  
anti-aliased
Motion Blur

- Stretch the reconstruction kernel in the direction of movement.
- Stretch the splat footprint in the direction of the projected movement (2D).
Camera Depth-of-Field

- Two possible approaches:
  - Low-pass filter the splats
  - Low-pass filter the sheets
Procedural Textures

• Easily done with voxel coloring
Bump-Mapping

- If calculating the normal per-pixel, we can modulate it to achieve bump mapping.
Examples

7.25 sec  9.41 sec (procedural)  7.99 sec
Other DVR Algorithms and A Comparison

Jian Huang, CS 594, Spring 2002
Shear Warp Algorithm

- Goal – speed!!
- Approach: make viewing rays parallel to each other and perpendicular to the image
  - Achieved by a simple shear

Original method only works with orthogonal projection, later extended to perspective projection but less efficient and less known.
Shear Warp Algorithm

• Shoot rays in every sheared slice and use bilinear interpolation to estimate a sample value
  – only interpolate in-slice, not between slices as in usual raycasting
  – cheap bilinear interpolation instead of trilinear interpolation in raycasting
• Volume is projected onto a baseplane parallel to the volume slices
  – NOT the image plane: the resulting image is called _intermediate image_
• Undo the shearing by warping the intermediate image onto the true image plane (use $M^{-1}_{\text{shear}}$, the inverse of $M$-shear)
• The mapping of the pixels in the intermediate image to the true image is done via resampling, using another bilinear interpolation
Shear Warp Algorithm
Shear Warp Algorithm

• General algorithm:

\[ M_{\text{view}} = P \cdot S \cdot M_{\text{warp}} \]

• Where
  – \( M_{\text{view}} \) = general viewing matrix
  – \( P \) = permutation matrix, which transposes the coordinate system in order to make the z-axis the principal viewing axis
  – \( S \) = transforms volume into sheared object space
  – \( M_{\text{warp}} \) = sheared object coordinates into image coordinates
Shear Warp Algorithm

45°

splatting

shear-warp
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
Texture based volume rendering

Algorithm: (using 2D texture mapping hardware)

Turn off the z-buffer; 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 composite the polygon  (use OpenGL alpha blending)
}

Some Considerations… (2)

What if we change the viewing position?

That is okay, we just change the eye position (or rotate the polygons and re-render),

Until …
Some Considerations… (3)

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?
What do we do?

• Change the orientation of slicing planes

• Now the slice polygons are parallel to XZ plane in the object space

• We need to reorganize the input textures

• Reorganize the textures on the fly is too time consuming. We might want to prepare this texture sets beforehand
Some Considerations… (5)

When do we need to change the slicing planes?

When the major component of view vector changes from z to y
Some Considerations… (6)

Major component of view vector?

Normalized view vector \((x, y, z)\) \(\rightarrow\) get the maximum one

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

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

Therefore, we need to keep three copies of data around all the time!!
Object-space axis-aligned method can create artifact: *Popping Effect*

There is a sudden change of viewing slicing (and thus the sampling rate) then the view vector transits from one major direction to another. The change in the result image intensity can be quite visible.
Solution

Use image-aligned slicing plane:

the slicing planes are always parallel to the image plane
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
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: \((r,s,t)\)

\((0,0,0)\) \hspace{1cm} \((1,0,0)\)

\((0,1,0)\) \hspace{1cm} \((1,1,0)\)

\((0,1,1)\) \hspace{1cm} \((1,1,1)\)

\((r_0,s_0,t_0)\) \hspace{1cm} \((r_1,s_1,t_1)\)

\((r_2,s_2,t_2)\) \hspace{1cm} \((r_3,s_3,t_3)\)
Pros and Cons

Advantages:
- Fast with volume sizes that the hardware can take e.g. 2 fps for 256 cube volumes
- No popping effect

Disadvantages:
- Need to compute the slicing planes for every view angle
- Only supported on high end hardware
- Low quality without per-pixel classification shading and classification (i.e. post-classification and shading)

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.
A Practical Comparison

• Quality is directly correlated with
  – how closely the volume ray integration can be approximated
  – and the quality of volume reconstruction
  – post-reconstruction shading and classification are key to visual sharpness of renderings.

• Ray-casting and image-aligned splatting are of high quality, each is more efficient for different types of data sets
• Not considering limitations of hardware precision, 3D texture mapping can achieve high quality as well
• Shear-warp is fast but poor quality