Virtual Trackball,
Scientific Visualization

Week 12, Wed Mar 30

http://www.ugrad.cs.ubc.ca/~cs314/Vjan2005
News

- reminder: my office hours today 3:45
- proposals: email out to several of you
- midterm 2 solutions out
Review: Splines

- **spline** is parametric curve defined by *control points*
  - *knots*: control points that lie on curve
  - engineering drawing: spline was flexible wood, control points were physical weights

A Duck (weight)

Ducks trace out curve
Review: Hermite Spline

- user provides
  - endpoints
  - derivatives at endpoints

\[ x = \begin{bmatrix} x_1 & x_0 & x'_1 & x'_0 \end{bmatrix} \begin{bmatrix} -2 & 3 & 0 & 0 \\ 2 & -3 & 0 & 1 \\ 1 & -1 & 0 & 0 \\ 1 & -2 & 1 & 0 \end{bmatrix} \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix} \]
Review: Bézier Curves

- four control points, two of which are knots
  - more intuitive definition than derivatives
- curve will always remain within convex hull (bounding region) defined by control points
Review: Basis Functions

- point on curve obtained by multiplying each control point by some basis function and summing
Review: Comparing Hermite and Bézier

Hermite

Bézier

(a)  (b)  (c)  (d)  (e)
Review: Sub-Dividing Bézier Curves

- find the midpoint of the line joining $M_{012}$, $M_{123}$. call it $M_{0123}$
Review: de Casteljau’s Algorithm

- can find the point on Bézier curve for any parameter value $t$ with similar algorithm
  - for $t=0.25$, instead of taking midpoints take points 0.25 of the way

demo: [www.saltire.com/applets/advanced geometry/spline/spline.htm](http://www.saltire.com/applets/advanced_geometry/spline/spline.htm)
Review: Continuity

- piecewise Bézier: no continuity guarantees

- continuity definitions
  - $C^0$: share join point
  - $C^1$: share continuous derivatives
  - $C^2$: share continuous second derivatives
Review: B-Spline

- $C_0$, $C_1$, and $C_2$ continuous
- piecewise: locality of control point influence
Virtual Trackball
Virtual Trackball

- interface for spinning objects around
  - rolling ball model, glass ball model
- imagine a trackball embedded in screen
  - if I click on screen, what point on trackball am I touching?

![Diagram of virtual trackball with 3D coordinates and image plane]
Trackball Rotation

- translating mouse from \( \mathbf{p}_1 \) to \( \mathbf{p}_2 \) corresponds to rotating about the axis \( \mathbf{n} = \mathbf{p}_1 \times \mathbf{p}_2 \)
- fixed point: origin (unit sphere)
- angle of rotation: \( \mathbf{p}_1 \cdot \mathbf{p}_2 = |\mathbf{p}_1| \ |\mathbf{p}_2| \cos \theta \)
Virtual Trackball

- using mouse to control the 2-D rotation of viewing volume
- imagine a track ball
  - user moves point on ball from \((x, y, z)\) to \((a, b, c)\)
- imagine the points projected onto the ground
  - user moves point on ground from \((x, 0, z)\) to \((a, 0, c)\)
- movement of points on track ball can be inferred from mouse drags on screen
- inverse problem
  - where on trackball does \((a, 0, c)\) hit?
  - ball is unit sphere, so \(||x, y, z|| = 1.0\)
  - \(x = a, z = c, y = \text{solve for it}\)
Virtual Trackball

- user defines two points
  - place where first clicked $X = (x, y, z)$
  - place where released $A = (a, b, c)$
- ball rotates along axis perp to line defined by these two points
  - compute cross produce of lines to origin: $(X - O) \times (A - O)$
- ball rotates by amount proportional to distance between lines
  - magnitude of cross product tells us angle between lines
    - (dot product too)
    - $|\sin q| = ||\text{cross product}||$
- compute rotation matrix and use it to rotate world
Scientific Visualization
Surface Graphics

- objects explicitly defined by surface or boundary representation
  - mesh of polygons

200 polys 1000 polys 15000 polys
Volume Graphics
Surface Graphics

**pros**
- fast rendering algorithms available
- hardware acceleration cheap
- OpenGL API for programming
- use texture mapping for added realism

**cons**
- discards interior of object, maintaining only the shell
- operations such cutting, slicing & dissection not possible
- no artificial viewing modes such as semi-transparencies, X-ray
- surface-less phenomena such as clouds, fog & gas are hard to model and represent
Volume Graphics

- for some data, difficult to create polygonal mesh
- voxels: discrete representation of 3D object
  - volume rendering: create 2D image from 3D object
- translate raw densities into colors and transparencies
  - different aspects of the dataset can be emphasized via changes in transfer functions
Volume Graphics

- **pros**
  - formidable technique for data exploration

- **cons**
  - rendering algorithm has high complexity!
  - special purpose hardware costly (~$3K-$10K)

volumetric human head (CT scan)
Isosurfaces

- 2D scalar fields: isolines
  - contour plots, level sets
  - topographic maps
- 3D scalar fields: isosurfaces
Volume Graphics: Examples

- Anatomical atlas from visible human (CT & MRI) datasets
- Industrial CT - structural failure, security applications
- Flow around airplane wing
- Shockwave visualization: simulation with Navier-Stokes PDEs
Isosurface Extraction

- array of discrete point samples at grid points
  - 3D array: voxels
- find contours
  - closed, continuous
  - determined by iso-value
- several methods
  - marching cubes is most common
MC 1: Create a Cube

- consider a cube defined by eight data values

\[(i,j,k) \quad (i+1,j,k) \quad (i,j+1,k) \quad (i+1,j+1,k) \quad (i,j+1,k+1) \quad (i+1,j+1,k+1) \quad (i,j,k+1) \quad (i+1,j,k+1)\]
MC 2: Classify Each Voxel

- classify each voxel according to whether lies
  - outside the surface (value > iso-surface value)
  - inside the surface (value <= iso-surface value)

![Diagram showing classification of voxels based on iso-surface values.](image)

- Iso=9
- Iso=7

- green = inside
- blue = outside
MC 3: Build An Index

- binary labeling of each voxel to create index

<table>
<thead>
<tr>
<th>v1</th>
<th>v2</th>
<th>v3</th>
<th>v4</th>
<th>v5</th>
<th>v6</th>
<th>v7</th>
<th>v8</th>
</tr>
</thead>
</table>

Index:

1110100
00110000
MC 4: Lookup Edge List

- use index to access array storing list of edges
  - all 256 cases can be derived from 15 base cases
MC 4: Example

- index = 00000001
- triangle 1 = a, b, c
MC 5: Interpolate Triangle Vertex

- for each triangle edge
  - find vertex location along edge using linear interpolation of voxel values

\[ x = i + \left( \frac{T - v[i]}{v[i+1] - v[i]} \right) \]

\[ T=5 \]

\[ T=8 \]
MC 6: Compute Normals

- calculate the normal at each cube vertex
  - use linear interpolation to compute the polygon vertex normal

\[
\begin{align*}
G_x &= v_{i+1,j,k} - v_{i-1,j,k} \\
G_y &= v_{i,j+1,k} - v_{i,j-1,k} \\
G_z &= v_{i,j,k+1} - v_{i,j,k-1}
\end{align*}
\]
MC 7: Render!
Direct Volume Rendering

- do not compute surface
Rendering Pipeline
Classification

- data set has application-specific values
  - temperature, velocity, proton density, etc.
- assign these to color/opacity values to make sense of data
- achieved through transfer functions
Transfer Functions

- map data value to color and opacity
Transfer Functions

Human Tooth CT

Gordon Kindlmann
Setting Transfer Functions

- can be difficult, unintuitive, and slow
Rendering Pipeline

Classify

Shade
Light Effects

- usually only consider reflected part

\[
I = k_a I_a + k_d I_d + k_s I_s
\]
Rendering Pipeline

Classify → Shade → Interpolate
Interpolation

2D

- given:

- needed:

1D

- given:

- needed:

nearest neighbor

linear
Rendering Pipeline
Volume Rendering Algorithms

- ray casting
  - image order, forward viewing

- splatting
  - object order, backward viewing

- texture mapping
  - object order
  - back-to-front compositing
Ray Traversal Schemes

Intensity
Max

Average

Accumulate

First

Depth
Ray Traversal - First

- first: extracts iso-surfaces (again!)
Ray Traversal - Average

- average: looks like X-ray
Ray Traversal - MIP

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

- accumulate: make transparent layers visible
Splatting

- each voxel represented as fuzzy ball
  - 3D gaussian function
  - RGBA value depends on transfer function
- fuzzy balls projected on screen, leaving footprint called **splat**
  - composite front to back, in object order
Texture Mapping

- **2D**: axis aligned 2D textures
  - back to front compositing
  - commodity hardware support
  - must calculate texture coordinates, warp to image plane
- **3D**: image aligned 3D texture
  - simple to generate texture coordinates