Midterm Review

Week 7, Wed 16 Oct 2003

- midterm review
- project 1 demos, hall of fame
News

• homework 1 due now
  – one day late if in handin box 18 by 9am Thu
  – two days late if in at class beginning Fri
  – no homeworks accepted after Fri 9am!
    • solutions out then
Midterm Exam

• Monday Oct 20 9am-9:50am
  – you may use one handwritten 8.5”x11” sheet
    • OK to use both sides of page
  – no other notes, no books
  – nonprogrammable calculators OK
  – arrive on time!!
What’s Covered

• transformations
• viewing and projections
• coordinate systems of rendering pipeline
• picking
• lighting and shading
• scan conversion

• not sampling
Reading

• Angel book
  – Chap 1, 2, 3, 4, 5, 6, 8.9-8.11, 9.1-9.6
  – you can be tested on material in book but not covered in lecture
  – you can be tested on material covered in lecture but not covered in book
Old Exams Posted

• see course web page
The Rendering Pipeline

- pros and cons of pipeline approach
Transformations

\[
\text{translate}(a,b,c)
\]
\[
\begin{bmatrix}
    x' \\
    y' \\
    z' \\
    1
\end{bmatrix} =
\begin{bmatrix}
    1 & a & x \\
    1 & b & y \\
    1 & c & z \\
    1 & 1 & 1
\end{bmatrix}
\]

\[
\text{scale}(a,b,c)
\]
\[
\begin{bmatrix}
    x' \\
    y' \\
    z' \\
    1
\end{bmatrix} =
\begin{bmatrix}
    a & x \\
    b & y \\
    c & z \\
    1 & 1
\end{bmatrix}
\]

\[
\text{Rotate}(x, \theta)
\]
\[
\begin{bmatrix}
    x' \\
    y' \\
    z' \\
    1
\end{bmatrix} =
\begin{bmatrix}
    1 & \cos \theta & -\sin \theta & x \\
    \cos \theta & 1 & -\sin \theta & y \\
    \sin \theta & \cos \theta & 1 & z \\
    1 & 1 & 1 & 1
\end{bmatrix}
\]

\[
\text{Rotate}(y, \theta)
\]
\[
\begin{bmatrix}
    x' \\
    y' \\
    z' \\
    1
\end{bmatrix} =
\begin{bmatrix}
    \cos \theta & \sin \theta & \cos \theta & \cos \theta & -\sin \theta & 1 \\
    -\sin \theta & \cos \theta & 0 & \sin \theta & \cos \theta & 1
\end{bmatrix}
\]

\[
\text{Rotate}(z, \theta)
\]
\[
\begin{bmatrix}
    x' \\
    y' \\
    z' \\
    1
\end{bmatrix} =
\begin{bmatrix}
    \cos \theta & -\sin \theta & \cos \theta & 1 \\
    \sin \theta & \cos \theta & 0 & 1
\end{bmatrix}
\]
Homogeneous Coordinates

\[
\begin{bmatrix}
  x \\
  y \\
  w
\end{bmatrix} = \begin{bmatrix}
  1 \\
  1 \\
  w
\end{bmatrix}
\]
Composing Transformations

\[ \text{ORDER MATTERS!} \]

\[
\begin{align*}
T(1,1) & \quad \text{and} \quad R(45) T(1,1) \\
R(45) T(1,1) & \quad \text{and} \quad T(1,1) R(45)
\end{align*}
\]

\[ T_a T_b = T_b T_a, \text{ but } R_a R_b \neq R_b R_a \text{ and } T_a R_b 
eq R_b T_a \]
Composing Transformations

• example: rotation around arbitrary center
Composing Transformations

- example: rotation around arbitrary center
  - step 1: translate coordinate system to rotation center
Composing Transformations

• example: rotation around arbitrary center
  – step 2: perform rotation
Composing Transformations

- example: rotation around arbitrary center
  - step 3: back to original coordinate system
Composing Transformations

- rotation about a fixed point
  \[ p' = TRT^{-1}p \]
- rotation around an arbitrary axis
- considering frame vs. object

OpenGL:
- D
- C
- B
- A
  draw p

\[ p' = DCBAp \]
Transformation Hierarchies

• hierarchies don’t fall apart when changed
• transforms apply to graph nodes beneath
Matrix Stacks

- push and pop matrix stack
  - avoid computing inverses or incremental xforms
  - avoid numerical error
Matrix Stacks

- `glPushMatrix()`
- `glPopMatrix()`
- `glScale3f(2,2,2)`
- `D = C scale(2,2,2) trans(1,0,0)`
- `glTranslate3f(1,0,0)`
- `DrawSquare()`
- `glPushMatrix()`
- `glScale3f(2,2,2)`
- `glTranslate3f(1,0,0)`
- `DrawSquare()`
- `glPopMatrix()`
Transformation Hierarchies

- example

\[
\begin{align*}
\theta_1 & \quad \theta_2 \\
\theta_3 & \quad \theta_4 \\
\theta_5 & \quad \theta_6
\end{align*}
\]

```cpp
glTranslate3f(x,y,0);
glRotatef(\theta_1,0,0,1);
DrawBody();
glPushMatrix();
glTranslate3f(0,7,0);
DrawHead();
glPopMatrix();
```
Display Lists

• reuse block of OpenGL code
• more efficient than immediate mode
  – code reuse, driver optimization
• good for static objects redrawn often
  – can’t change contents
  – not just for multiple instances
    • interactive graphics: objects redrawn every frame
• nest when possible for efficiency
Double Buffering

• two buffers, front and back
  – while front is on display, draw into back
  – when drawing finished, swap the two

• avoid flicker
Projective Rendering Pipeline

OCS - object coordinate system
WCS - world coordinate system
VCS - viewing coordinate system
CCS - clipping coordinate system
NDCS - normalized device coordinate system
DCS - device coordinate system

modeling transformation

viewing transformation

projection transformation

viewport transformation

---

glVertex3f(x,y,z)

glTranslatef(x,y,z)

glRotatef(th,x,y,z)

gluLookAt(...)

glFrustum(...)

glutInitWindowSize(w,h)

glViewport(x,y,a,b)

---

alter w

/ w

perspective division

OCS - object coordinate system
WCS - world coordinate system
VCS - viewing coordinate system
CCS - clipping coordinate system
NDCS - normalized device coordinate system
DCS - device coordinate system
Projection

- theoretical pinhole camera

- image inverted, more convenient equivalent
Projection Taxonomy

planar projections

perspective: 1,2,3-point

parallel

oblique

cabinet

cavalier

orthographic

top, front, side

axonometric:

isometric
dimetric
trimetric
Projective Transformations

• transformation of space
  – center of projection moves to infinity
  – viewing frustum transformed into a parallelepiped
Normalized Device Coordinates

left/right \( x = +/- 1 \), top/bottom \( y = +/- 1 \), near/far \( z = +/- 1 \)

Camera coordinates

NDC

Frustum
Projection Normalization

• distort such that orthographic projection of distorted objects is desired persp projection
Transforming View Volumes

Perspective view volume

Orthographic view volume

NDCS

(-1,-1,-1)

(1,1,1)
Basic Perspective Projection

\[ \frac{y'}{d} = \frac{y}{z} \rightarrow y' = \frac{y \cdot d}{z} \]  
also \[ x' = \frac{x \cdot d}{z} \]  
but \[ z' = d \]

- nonuniform foreshortening
  - not affine
Basic Perspective Projection

- can express as homogenous 4x4 matrix!

\[
\begin{bmatrix}
  x \\
  y \\
  z \\
  z/d \\
\end{bmatrix}
= 
\begin{bmatrix}
  1 & 0 & 0 & 0 \\
  0 & 1 & 0 & 0 \\
  0 & 0 & 1 & 0 \\
  0 & 0 & 1/d & 0 \\
\end{bmatrix}
\begin{bmatrix}
  x \\
  y \\
  z \\
  1 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
  x \\
  y \\
  z \\
  z/d \\
\end{bmatrix}
\xrightarrow{\cdot w}
\begin{bmatrix}
  x \cdot d/z \\
  y \cdot d/z \\
  d \\
\end{bmatrix}
\]
Projective Transformations

• determining the matrix representation
  – need to observe 5 points in general position, e.g.
    • $[\text{left},0,0,1]^T \rightarrow [-1,0,0,1]^T$
    • $[0,\text{top},0,1]^T \rightarrow [0,1,0,1]^T$
    • $[0,0,-f,1]^T \rightarrow [0,0,1,1]^T$
    • $[0,0,-n,1]^T \rightarrow [0,0,-1,1]^T$
    • $[\text{left}*f/n,\text{top}*f/n,-f,1]^T \rightarrow [-1,1,1,1]^T$
  – solve resulting equation system to obtain matrix
OpenGL Orthographic Matrix

- scale, translate, reflect for new coord sys
  – understand derivation!

\[
\begin{bmatrix}
\frac{2}{right - left} & 0 & 0 & -\frac{right + left}{right - left} \\
0 & \frac{2}{top - bot} & 0 & -\frac{top + bot}{top - bot} \\
0 & 0 & -\frac{2}{far - near} & -\frac{far + near}{far - near} \\
0 & 0 & 0 & 1
\end{bmatrix}
\]
OpenGL Perspective Matrix

- shear, scale, reflect for new coord sys
  - understand derivation!

\[
\begin{bmatrix}
\frac{2 \cdot \text{near}}{\text{right} - \text{left}} & 0 & \frac{\text{right} + \text{left}}{\text{right} - \text{left}} & 0 \\
0 & \frac{2 \cdot \text{near}}{\text{top} - \text{bot}} & \frac{\text{top} + \text{bot}}{\text{top} - \text{bot}} & 0 \\
0 & 0 & \frac{-(\text{far} + \text{near})}{\text{far} - \text{near}} & \frac{-2 \cdot \text{far} \cdot \text{near}}{\text{far} - \text{near}} \\
0 & 0 & -1 & 0
\end{bmatrix}
\]
Viewport Transformation

onscreen pixels: map from $[-1,1]$ to $[0, \text{displaywidth}]$

$$x_{DCS} = w \frac{(x_{NDCS} + 1)}{2}$$

$$y_{DCS} = h \frac{(y_{NDCS} + 1)}{2}$$

$$z_{DCS} = \frac{(z_{NDCS} + 1)}{2}$$
3 Simple Picking Approaches

• manual ray intersection

• bounding extents

• backbuffer coloring
Picking Select/Hit

• assign (hierarchical) integer key/name(s)
• small region around cursor as new viewport
  


• redraw in selection mode
  – equivalent to casting pick “tube”
  – store keys, depth for drawn objects in hit list

• examine hit list
  – usually use frontmost, but up to application
Light Sources

- directional/parallel lights
  - point at infinity: \((x, y, z, 0)^T\)

- point lights
  - finite position: \((x, y, z, 1)^T\)

- spotlights
  - position, direction, angle

- ambient lights
Illumination as Radiative Transfer

– model light transport as packet flow
  • particles not waves
Reflectance

- *specular*: perfect mirror with no scattering
- *gloss*: mixed, partial specularity
- *diffuse*: all directions with equal energy

\[
\text{specular} + \text{glossy} + \text{diffuse} = \text{reflectance distribution}
\]
Reflection Equations

\[ I_{\text{diffuse}} = k_d I_{\text{light}} (n \cdot l) \]

\[ I_{\text{specular}} = k_s I_{\text{light}} (v \cdot r)^n_{\text{shiny}} \]

\[ 2 (N (N \cdot L)) - L = R \]
Reflection Equations

- Blinn improvement

\[ I_{\text{out}}(x) = k_s \cdot (h \cdot n)^{n_{\text{shiny}}} \cdot I_{\text{in}}(x); \]

\[ h = (l + v) / 2 \]

- full Phong lighting model
  - combine ambient, diffuse, specular components

\[ I_{\text{total}} = k_a \cdot I_{\text{ambient}} + \sum_{i=1}^{\text{#lights}} I_i \left( k_d \left( n \times l_i \right) + k_s \left( v \times r_i \right)^{n_{\text{shiny}}} \right) \]
Lighting vs. Shading

• lighting
  – simulating the interaction of light with surface

• shading
  – deciding pixel color
  – continuum of realism: when do we do lighting calculation?
Shading Models

• flat shading
  – compute Phong lighting once for entire polygon

• Gouraud shading
  – compute Phong lighting at the vertices and interpolate lighting values across polygon

• Phong shading
  – compute averaged vertex normals
  – interpolate normals across polygon and perform Phong lighting across polygon
Shutterbug: Flat Shading
Shutterbug: Gouraud Shading
Shutterbug: Phong Shading
Scanline Algorithms

• given vertices, fill in the pixels

**triangles**
- split into two regions
- fill in between edges

**arbitrary polygons**
- build edge table
- for each scanline
  - obtain list of intersections, i.e., AEL
  - use parity test to determine in/out and fill in the pixels
Edge Equations

• define triangle as intersection of three positive half-spaces:
Edge Equations

• So…simply turn on those pixels for which all edge equations evaluate to $> 0$: 
Parity for General Case

- use parity for interior test
  - draw pixel if edgecount odd
  - horizontal lines: count
  - vertical max: count
  - vertical min: don’t count
Edge Tables

- **edge table (ET)**
  - store edges sorted by y in linked list
    - at ymin, store ymax, xmin, slope
- **active edge table (AET)**
  - active: currently used for computation
  - store active edges sorted by x
    - update each scanline, store ET values + current_x
  - for each scanline (from bottom to top)
    - do EAT bookkeeping
    - traverse EAT (from leftmost x to rightmost x)
      - draw pixels if parity odd
Barycentric Coordinates

• weighted combination of vertices
  – understand derivation!

\[
P = \alpha \cdot P_1 + \beta \cdot P_2 + \gamma \cdot P_3
\]

\[
\alpha + \beta + \gamma = 1
\]

\[
0 \leq \alpha, \beta, \gamma \leq 1
\]

“convex combination of points”
Transforming Normals

• apply nonuniform scale: stretch along x by 2
  – can’t transform normal by modelling matrix

• solution:

\[
P \rightarrow P' = MP
\]
\[
N \rightarrow N' = QN
\]

\[
Q = (M^{-1})^T
\]

normal to any surface transformed by inverse transpose of modelling transformation