Midterm Review
Week 7, Wed 16 Oct 2003
• midterm review
• project 1 demos, hall of fame

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

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

Old Exams Posted
• see course web page
The Rendering Pipeline

• pros and cons of pipeline approach

- Geometry Database
- Model/View Transform,
- Lighting
- Perspective Transform,
- Clipping
- Scan Conversion
- Texturing
- Depth Test
- Blending
- Frame-buffer

Transformations

• translate(a,b,c)

\[
\begin{bmatrix}
  x' \\
  y' \\
  z' \\
  1
\end{bmatrix} =
\begin{bmatrix}
  1 & a & 0 & 0 \\
  1 & b & 0 & 0 \\
  1 & c & 0 & 0 \\
  1 & 1 & 1 & 1
\end{bmatrix}
\begin{bmatrix}
  x \\
  y \\
  z \\
  1
\end{bmatrix}
\]

• scale(a,b,c)

\[
\begin{bmatrix}
  x' \\
  y' \\
  z' \\
  1
\end{bmatrix} =
\begin{bmatrix}
  a & 0 & 0 & 0 \\
  0 & b & 0 & 0 \\
  0 & 0 & c & 0 \\
  0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
  x \\
  y \\
  z \\
  1
\end{bmatrix}
\]

• Rotate(\(x, \theta\))

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

• Rotate(\(y, \theta\))

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

• Rotate(\(z, \theta\))

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

Homogeneous Coordinates

- \(x' \cdot w, y' \cdot w, z' \cdot w, w=1\)

Composing Transformations

- Ta Tb = Tb Ta, but Ra Rb \neq Rb Ra and Ta Rb \neq Rb Ta

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

Week 7, Wed 15 Oct 03 © Tamara Munzner

Composing Transformations

• example: rotation around arbitrary center
  – step 3: back to original coordinate system

Week 7, Wed 15 Oct 03 © Tamara Munzner

Composing Transformations

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

OpenGL:

\[
\begin{align*}
&D &\quad C &\quad B &\quad A \\
\text{frame} &\quad &\quad &\quad &\quad \\
p' &\quad &\quad &\quad &\quad p
\end{align*}
\]

Week 7, Wed 15 Oct 03 © Tamara Munzner

Transformation Hierarchies

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

Week 7, Wed 15 Oct 03 © Tamara Munzner

Matrix Stacks

• push and pop matrix stack
  – avoid computing inverses or incremental xforms
  – avoid numerical error

Week 7, Wed 15 Oct 03 © Tamara Munzner

Matrix Stacks

\[
\begin{align*}
&\text{glPushMatrix()} &\quad \text{glPushMatrix()} \\
&C &\quad C &\quad D &\quad C \\
&B &\quad B &\quad B &\quad B \\
&A &\quad A &\quad A &\quad A \\
\text{D} = \text{C scale}(2,2,2) \text{ trans}(1,0,0) \\
\text{DrawSquare()} &\quad \text{glPushMatrix()} \\
\end{align*}
\]
Transformation Hierarchies

- example

```
glTranslate3f(x,y,0);
glRotatef(θ,0,0,1);
DrawBody();
glPushMatrix();
glTranslate3f(0,7,0);
DrawHead();
glPopMatrix();
glPushMatrix();
glTranslate(2.5,5.5,0);
glRotatef(θ,0,0,1);
DrawUArm();
glTranslate(-3.5,0);
glRotatef(θ,0,0,1);
DrawLArm();
glPopMatrix();
... (draw other arm)
```

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

```
glVertex3f(x,y,z)
glTranslatef(x,y,z)
glRotatef(th,x,y,z)
....
gluLookAt(...)
glFrustum(...)
```

Projective Rendering Pipeline

```
object modeling transformation viewing/ transformation to camera
OCS - object coordinate system WCS - world coordinate system VCS - viewing coordinate system
Modeling/ transformation:
glTranslate(x,y,z)
glRotate(x,y,z)
Clipping/ transformation:
glFrustum(x,y,z)
```

Projection

- theoretical pinhole camera
  - image inverted, more convenient equivalent

```
glVertex3f(x,y,z)
glTranslatef(x,y,z)
glRotatef(th,x,y,z)
....
gluLookAt(...)
glFrustum(...)
```

Projection Taxonomy

```
planar projections
perspective: 1,2,3-point parallel
oblique
cavalier
top, front, side
axonometrie: isometric, dimetric, trimetric
```

Week 7, Wed 15 Oct 03 © Tamara Munzner
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 \( z = +/-1 \)

Projection Normalization
• distort such that orthographic projection of distorted objects is desired perp projection

Transforming View Volumes
perspective view volume
orthographic view volume

Basic Perspective Projection
• can express as homogenous 4x4 matrix!

Basic Perspective Projection
similarity triangles
\[
\frac{y'}{d} = \frac{y}{z} \quad \text{also} \quad \frac{x'}{d} = \frac{x}{z}
\]
• nonuniform foreshortening
  – not affine
Projective Transformations
• determining the matrix representation
  – need to observe 5 points in general position, e.g.
  \[
  \begin{pmatrix}
  \text{left} & 0 & 0 & 1 \\
  0 & \text{top} & 0 & 1 \\
  \text{left} \cdot f/n & \text{top} \cdot f/n & -f & 1 \\
  0 & 0 & 0 & 1 \\
  \end{pmatrix} = \begin{pmatrix}
  \text{right} - \text{left} & 0 & 0 & \text{right} - \text{left} \\
  0 & \text{top} + \text{bot} & 0 & \text{top} + \text{bot} \\
  -(\text{far} + \text{near}) & -2 \cdot \text{far} \cdot \text{near} & \text{far} \cdot \text{near} & 1 \\
  0 & 0 & 0 & 1 \\
  \end{pmatrix}
  \]
  – solve resulting equation system to obtain matrix

OpenGL Orthographic Matrix
• scale, translate, reflect for new coord sys
  – understand derivation!

OpenGL Perspective Matrix
• shear, scale, reflect for new coord sys
  – understand derivation!

Viewport Transformation
onscreen pixels: map from \([-1,1]\) to \([0, \text{displaywidth}]\)

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
- glossy: mixed, partial specularity
- diffuse: all directions with equal energy

Reflection Equations

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

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

$$I_{\text{total}} = I_{\text{ambient}} + \sum_{i=1}^{\text{lights}} I_i \left( k_d \cdot (n \cdot l_i) + k_s \cdot (v \cdot r_i)^{n_{\text{shiny}}} \right)$$

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

Scanline Algorithms

• given vertices, fill in the pixels
  triangles
    • split into two regions
    • fill in between edges
  arbitrary polygons (non-simple, non-convex)
    • 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 AET bookkeeping
      • traverse AET (from leftmost x to rightmost x)
        – draw pixels if parity odd

Barycentric Coordinates

• weighted combination of vertices
  – understand derivation!

Transforming Normals

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

• solution:
  \[ P' = MP \]
  \[ N' = QN \]
  \[ Q = (M^{-1})^T \]
  normal to any surface transformed by inverse transpose of modelling transformation