Assignment 1 Review

32 Students, 29 submissions

Grade Distribution

<table>
<thead>
<tr>
<th>Grade</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 to 5</td>
<td></td>
</tr>
<tr>
<td>5 to 10</td>
<td></td>
</tr>
<tr>
<td>10 to 15</td>
<td></td>
</tr>
<tr>
<td>15 to 20</td>
<td></td>
</tr>
<tr>
<td>20 to 25</td>
<td></td>
</tr>
<tr>
<td>25 to 30</td>
<td></td>
</tr>
<tr>
<td>&gt;30</td>
<td>16</td>
</tr>
</tbody>
</table>

20 Students in the >30 range.
Quick Aside: Texture Mapping

We know how to go from this… to this

J. Birn
Quick Aside: Texture Mapping

But what about this…

to this?

J. Birn
Textures

- How can we go about drawing surfaces with complex detail?
Textures

• How can we go about drawing surfaces with complex detail?

• We could tessellate the sphere in a complex fashion and then associate the appropriate material properties to each vertex.

Target Model

Complex Surface
Textures

- How can we go about drawing surfaces with complex detail?

- We could use a simple tessellation and use the location of surface points to look up the appropriate color values.
Textures

• Advantages:
  ° The 3D model remains simple
  ° It is easier to design/modify a texture image than it is to design/modify a surface in 3D.
Another Example: Brick Wall
Another Example: Brick Wall
2D Texture

- Coordinates described by variables $s$ and $t$ and range over interval (0,1)
- Texture elements are called *texels*
- Often 4 bytes (rgba) per texel
Texture Mapping a Sphere

• How do you generate texture coordinates at each intersection point?
3D Polygon Rendering Pipeline

Jason Lawrence

CS445: Graphics

Acknowledgment: slides by Misha Kazhdan, Allison Klein, Tom Funkhouser, Adam Finkelstein and David Dobkin
Road Map for Next Lectures

- Leaving ray-tracing
- Moving on to polygon-based rendering
  - Rendering pipeline (today)
  - Clipping
  - Scan conversion & shading
  - Texture-mapping
  - Hidden-surface removal
- This polygon-based rendering is what happens in your PC
3D Polygon Rendering

- Many applications use rendering of 3D polygons with direct illumination
3D Polygon Rendering

• Many applications use rendering of 3D polygons with direct illumination

Half-Life 2
(Valve Software)
3D Polygon Rendering

• Many applications use rendering of 3D polygons with direct illumination

Star Wars: Knights of the Old Republic
(BioWare)
Ray Casting Revisited

• For each sample ...
  • Construct ray from eye position through view plane
  • Find first surface intersected by ray through pixel
  • Compute color of sample based on surface radiance

More efficient algorithms utilize spatial coherence!
3D Polygon Rendering

- Logical inverse of ray casting
- Idea: Instead of sending rays from the camera into the scene, send rays from the scene into the camera.
3D Polygon Rendering

- Ray casting: pick pixel and figure out what color it should be based on what object its ray hits.
- Polygon rendering: pick polygon and figure out what pixels it should affect.
3D Rendering Pipeline (for direct illumination)

This is a pipelined sequence of operations to draw a 3D primitive into a 2D image.
3D Rendering Pipeline (for direct illumination)

3D Geometric Primitives

- **Modeling Transformation**
  - Transform from current (local) coordinate system into 3D world coordinate system

- **Camera Transformation**

- **Lighting**

- **Projection Transformation**

- **Clipping**

- **Scan Conversion**

- **Image**
3D Rendering Pipeline (for direct illumination)

3D Geometric Primitives

- Modeling Transformation
  - Transform into 3D world coordinate system

- Camera Transformation
  - Transform into 3D camera coordinate system

- Lighting
  - 

- Projection Transformation

- Clipping

- Scan Conversion

- Image
3D Rendering Pipeline (for direct illumination)

3D Geometric Primitives

- **Modeling Transformation**
  - Transform into 3D world coordinate system

- **Camera Transformation**
  - Transform into 3D camera coordinate system

- **Lighting**
  - Illuminate according to lighting and reflectance

- **Projection Transformation**

- **Clipping**

- **Scan Conversion**

- **Image**
3D Rendering Pipeline (for direct illumination)

3D Geometric Primitives

Modeling Transformation

Camera Transformation

Lighting

Illuminate according to lighting and reflectance

Projection Transformation

Transform into 3D world coordinate system

Transform into 3D camera coordinate system

Clipping

Transform into 2D camera coordinate system

Scan Conversion

Image
3D Rendering Pipeline (for direct illumination)

3D Geometric Primitives

- **Modeling Transformation**
  - Transform into 3D world coordinate system

- **Camera Transformation**
  - Transform into 3D camera coordinate system

- **Lighting**
  - Illuminate according to lighting and reflectance

- **Projection Transformation**
  - Transform into 2D camera coordinate system

- **Clipping**
  - Clip (parts of) primitives outside camera’s view

- **Scan Conversion**

- **Image**
3D Rendering Pipeline (for direct illumination)

3D Geometric Primitives

1. **Modeling Transformation**
   - Transform into 3D world coordinate system

2. **Camera Transformation**
   - Transform into 3D camera coordinate system

3. **Lighting**
   - Illuminate according to lighting and reflectance

4. **Projection Transformation**
   - Transform into 2D camera coordinate system

5. **Clipping**
   - Clip (parts of) primitives outside camera’s view

6. **Scan Conversion**
   - Draw pixels (includes texturing, hidden surface, ...)

7. **Image**
Transformations

3D Geometric Primitives

Modeling Transformation

Camera Transformation

Lighting

Projection Transformation

Clipping

Scan Conversion

Image

**Transform** into 3D world coordinate system

**Transform** into 3D camera coordinate system

Illuminate according to lighting and reflectance

**Transform** into 2D camera coordinate system

Clip primitives outside camera’s view

Draw pixels (includes texturing, hidden surface, etc.)
Transformations map points from one coordinate system to another.

\[ p(x,y,z) \]

1. **Modeling Transformation**
   - Result: 3D Object Coordinates

2. **Camera Transformation**
   - Result: 3D World Coordinates

3. **Projection Transformation**
   - Result: 3D Camera Coordinates

4. **Window-to-Viewport Transformation**
   - Result: 2D Screen Coordinates

\[ p'(x',y') \]
Viewing Transformations

\[ p(x, y, z) \]

- 3D Object Coordinates
- Modeling Transformation
- 3D World Coordinates
- Camera Transformation
- 3D Camera Coordinates
- Projection Transformation
- 2D Screen Coordinates
- Window-to-Viewport Transformation
- 2D Image Coordinates

\[ p'(x', y') \]
Viewing Transformation

- Mapping from world to camera coordinates
  - Eye position maps to origin
  - Right vector maps to X axis
  - Up vector maps to Y axis
  - Back vector maps to Z axis
Camera Coordinates

- Canonical coordinate system
  - Convention is right-handed (looking down -z axis)
  - Convenient for projection, clipping, etc.

![Camera Coordinates Diagram]

- Camera up vector maps to Y axis
- Camera back vector maps to Z axis (pointing out of page)
- Camera right vector maps to X axis
Finding the viewing transformation

• We have the camera (in world coordinates)

• We want $T$ taking objects from world to camera

\[ p^c = T \ p^w \]

• Trick: find $T^{-1}$ taking objects in camera to world

\[ p^w = T^{-1} p^c \]

\[
\begin{bmatrix}
  x' \\
  y' \\
  z' \\
  w'
\end{bmatrix} =
\begin{bmatrix}
  a & b & c & d \\
  e & f & g & h \\
  i & j & k & l \\
  m & n & o & p
\end{bmatrix}
\begin{bmatrix}
  x \\
  y \\
  z \\
  w
\end{bmatrix}
\]
Finding the Viewing Transformation

• Trick: map from camera coordinates to world
  ◦ Origin maps to eye position
  ◦ Z axis maps to Back vector
  ◦ Y axis maps to Up vector
  ◦ X axis maps to Right vector

\[
\begin{bmatrix}
 x' \\
 y' \\
 z' \\
 w'
\end{bmatrix} = \begin{bmatrix}
 R_x & U_x & B_x & E_x \\
 R_y & U_y & B_y & E_y \\
 R_z & U_z & B_z & E_z \\
 0 & 0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
 x \\
 y \\
 z \\
 w
\end{bmatrix}
\]

\[ p^w = T^{-1} p^c \]

• This matrix is \( T^{-1} \) so we invert it to get \( T \) … easy!
Finding the Viewing Transformation

• Trick: map from camera coordinates to world
  ◦ Remember, with homogeneous coordinates, we divide through by $w$ values…
  ◦ So if we know actual point in 3D, $w = 1$
  ◦ Easy to find code to invert a matrix

$$p^w = T^{-1}p^c$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} R_x & U_x & B_x & E_x \\ R_y & U_y & B_y & E_y \\ R_z & U_z & B_z & E_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

• This matrix is $T^{-1}$ so we invert it to get $T$ … easy!
Viewing Transformations

\( p(x, y, z) \)

- **Modeling Transformation**
  - 3D Object Coordinates
- **Camera Transformation**
  - 3D World Coordinates
- **Projection Transformation**
  - 3D Camera Coordinates
- **Window-to-Viewport Transformation**
  - 2D Screen Coordinates
  - 2D Image Coordinates

\( p'(x', y') \)
Projection

• General definition:
  ° A linear transformation of points in \( n \)-space to \( m \)-space \( (m<n) \)

• In computer graphics:
  ° Map 3D camera coordinates to 2D screen coordinates
Projection

- Two general classes of projections, both of which shoot rays from the scene, through the view plane:
  - **Parallel Projection:**
    - Rays converge at a point at infinity and are **parallel**
  - **Perspective “Projection”:**
    - Rays converge at a finite point, giving rise to **perspective distortion**
Taxonomy of Projections

Planar geometric projections

Parallel

Orthographic
- Top (plan)
- Front elevation
- Side elevation

Axonometric

Oblique
- Cabinet
- Cavalier

One-point

Two-point

Three-point

Perspective

Other
Parallel Projection

- Center of projection is at infinity
  - Direction of projection (DOP) same for all points
Parallel Projection

• Parallel lines remain parallel
• Relative proportions of objects preserved
• Angles are not preserved
• Less realistic looking
  ◦ Far away objects don’t get smaller
Taxonomy of Projections

Planar geometric projections

Parallel

Orthographic
- Top (plan)
- Front elevation
- Side elevation

Axonometric

Oblique
- Cabinet
- Cavalier

One-point
- Two-point
- Three-point

Perspective

Other
Orthographic Projections

- DOP perpendicular to view plane

Angel Figure 5.5
Orthographic Projections

- DOP perpendicular to view plane

- Lines perpendicular to the view plane vanish
- Faces parallel to the view plane are un-distorted.
Taxonomy of Projections

Planar geometric projections

Parallel
Orthographic
- Top (plan)
- Front elevation
- Side elevation
Axonometric
- Isometric

Oblique
Cabinet
- Cavalier
One-point
Two-point
Three-point
Perspective

Other

FvDFH Figure 6.13
Oblique Projections

- DOP not perpendicular to view plane

Cavalier
(DOP $\alpha = 45^\circ$)

Cabinet
(DOP $\alpha = 63.4^\circ$)

- $\phi$ describes the angle of the projection of the view plane’s normal
- $L$ represents the scale factor applied to the view plane’s normal

H&B Figure 12.21
Parallel Projection Matrix

- General parallel projection transformation:

\[
\begin{bmatrix}
    x_s \\
    y_s
\end{bmatrix} =
\begin{bmatrix}
    1 & 0 & L \cos \phi \\
    0 & 1 & L \sin \phi
\end{bmatrix}
\begin{bmatrix}
    x_c \\
    y_c \\
    z_c
\end{bmatrix}
\]

Cavalier
(DOP \( \alpha = 45^\circ \))

\[
\begin{bmatrix}
    1 \\
    1
\end{bmatrix}
\]

Cabinet
(DOP \( \alpha = 63.4^\circ \))

\[
\begin{bmatrix}
    1/2 \\
    1
\end{bmatrix}
\]

H&B Figure 12.21
Parallel Projection View Volume

H&B Figure 12.30
Taxonomy of Projections

Planar geometric projections

Parallel

Perspective

Orthographic

Top (plan)

Front elevation

Axonometric

Side elevation

Isometric

Oblique

Cabinet

Cavalier

One-point

Two-point

Three-point

Other
Perspective “Projection”

- Map points onto “view plane” along “projectors” emanating from “center of projection” (COP)
Perspective Projection

• How many vanishing points?

Number of vanishing points determined by number of axes parallel to the view plane

Angel Figure 5.10
Perspective Projection

• As with translation, perspective “projection” is not really a projection because it is not a linear map from 3D to 2D.
  ◦ Parallel lines do not remain parallel!
Perspective Projection View Volume

H&B Figure 12.30
Perspective Projection

- What are the coordinates of the point resulting from projection of \((x_0, y_0, z_0)\) onto the view plane at a distance of \(D\) along the \(z\)-axis?
Perspective Projection

- Use the fact that for any point \((x_0, y_0, z_0)\) and any scalar \(\alpha\), the points \((x_0, y_0, z_0)\) and \((\alpha x_0, \alpha y_0, \alpha z_0)\) map to the same location:
Perspective Projection

- Use the fact that for any point \((x_0, y_0, z_0)\) and any scalar \(\alpha\), the points \((x_0, y_0, z_0)\) and \((\alpha x_0, \alpha y_0, \alpha z_0)\) map to the same location.

- Since we want the position of the point on the line that intersect the image plane at a distance of \(D\) along the z-axis:

\[
(x_0, y_0, z_0) \overset{\text{®}}{\rightarrow} \left( x_0 \frac{D}{z_0}, y_0 \frac{D}{z_0}, D \right)
\]
Perspective Projection Matrix

- 4x4 matrix representation?

\[
x_s = x_c \frac{D}{z_c}
\]
\[
y_s = y_c \frac{D}{z_c}
\]
\[
z_s = D
\]
\[
w_s = 1
\]
Perspective Projection Matrix

• 4x4 matrix representation?

\[
\begin{align*}
    x_s &= x_c \frac{D}{z_c} \\
    y_s &= y_c \frac{D}{z_c} \\
    z_s &= D \\
    w_s &= 1
\end{align*}
\]

We want to divide by the \( z \) coordinate. How do we do that with a 4x4 matrix?
Perspective Projection Matrix

- 4x4 matrix representation?

\[
x_s = x_c \frac{D}{z_c}
\]
\[
y_s = y_c \frac{D}{z_c}
\]
\[
z_s = D
\]
\[
w_s = 1
\]

We want to divide by the \(z\) coordinate. How do we do that with a 4x4 matrix?

Recall that in homogenous coordinates:
\((x, y, z, w) = (x/w, y/w, z/w, 1)\)
Perspective Projection Matrix

- 4x4 matrix representation?

\[
\begin{align*}
x_s &= x_c \frac{D}{z_c} \\
y_s &= y_c \frac{D}{z_c} \\
z_s &= D \\
w_s &= 1
\end{align*}
\]

We want to divide by the \(z\) coordinate. How do we do that with a 4x4 matrix?

Recall that in homogenous coordinates:
\[(x, y, z, w) = (x/w, y/w, z/w, 1)\]
Perspective Projection Matrix

• 4x4 matrix representation?

\[
\begin{align*}
    x_s &= x_c D / z_c \\
    y_s &= y_c D / z_c \\
    z_s &= D \\
    w_s &= 1
\end{align*}
\]

We want to divide by the \( z \) coordinate. How do we do that with a 4x4 matrix?

Recall that in homogenous coordinates:

\[
(x, y, z, w) = (x/w, y/w, z/w, 1)
\]

\[
\begin{pmatrix}
    x_c D \\
    y_c D \\
    z_c \\
    x_c, y_c, z_c, \frac{z_c}{D}
\end{pmatrix}
\]

\[
\begin{pmatrix}
    1 & 0 & 0 & 0 \\
    0 & 1 & 0 & 0 \\
    0 & 0 & 1 & 0 \\
    0 & 0 & 1/D & 0
\end{pmatrix}
\begin{pmatrix}
    x_c \\
    y_c \\
    z_c \\
    1
\end{pmatrix}
\]
Perspective vs. Parallel

• Perspective projection
  + Size varies inversely with distance - looks realistic
    – Distance and angles are not preserved
    – Only parallel lines that are parallel to the view plane remain parallel

• Parallel projection
  + Good for exact measurements
  + Parallel lines remain parallel
  + Angles are preserved on faces parallel to the view plane
  – Less realistic looking