3D Polygon Rendering Pipeline

Greg Humphreys
CS445: Intro Graphics
University of Virginia, Fall 2004

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

Quake II
(Id Software)

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

• What steps are necessary to utilize spatial coherence while drawing these polygons into a 2D image?
This is a pipelined sequence of operations to draw a 3D primitive into a 2D image.

Example: OpenGL

```
glBegin(GL_POLYGON);
gVertex3f(0.0, 0.0, 0.0);
gVertex3f(1.0, 0.0, 0.0);
gVertex3f(1.0, 1.0, 1.0);
gVertex3f(0.0, 1.0, 1.0);
gEnd();
```

OpenGL executes steps of 3D rendering pipeline for each polygon.
3D Rendering Pipeline (for direct illumination)

Transform into 3D world coordinate system
Illuminate according to lighting and reflectance
Transform into 3D camera coordinate system
Transform into 2D screen coordinate system
Clip primitives outside camera’s view

Transformations

Transform into 3D world coordinate system
Illuminate according to lighting and reflectance
Transform into 3D camera coordinate system
Transform into 2D screen coordinate system
Clip primitives outside camera’s view
Draw pixels (includes texturing, hidden surface, etc.)

Viewing Transformations

p(x,y,z)
3D Object Coordinates
Modeling Transformation

Viewing Transformations

p’(x’,y’)
3D Camera Coordinates

Camera Coordinates

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

Finding the viewing transformation

• We have the camera (in world coordinates)
• We want $T$ taking objects from world to camera
  
  $p^w = T p^c$

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

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

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

Projection

• General definition:
  - Transform points in $n$-space to $m$-space ($m<n$)
• In computer graphics:
  - Map 3D camera coordinates to 2D screen coordinates

Taxonomy of Projections

---

FVFHP Figure 6.10
Taxonomy of Projections

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

Orthographic Projections
- DOP perpendicular to view plane

Oblique Projections
- DOP not perpendicular to view plane

Parallel Projection View Volume

Parallel Projection Matrix
- General parallel projection transformation:

\[
\begin{pmatrix}
    x' \\
y' \\
z' \\
w'
\end{pmatrix} =
\begin{pmatrix}
    1 & 0 & 0 & 0 \\
    0 & 1 & 0 & 0 \\
    0 & 0 & 0 & 0 \\
    0 & 0 & 0 & 1
\end{pmatrix}\begin{pmatrix}
    x \\
y \\
z \\
w
\end{pmatrix}
\]
**Taxonomy of Projections**

- Planar geometric projections
  - Parallel
  - Orthographic
  - Axonometric
  - Oblique
  - Cabinet
  - Cavalier
  - One-point
  - Two-point
  - Three-point

**Perspective Projection**

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

**Perspective Projection View Volume**

- How many vanishing points?

- 3-Point Perspective

- 2-Point Perspective

- 1-Point Perspective

**Camera to Screen**

- Remember: Object → Camera → Screen

- Just like raytracer
  - “screen” is the z=d plane for some constant d

- Origin of screen coordinates is (0,0,d)

- Its x and y axes are parallel to the x and y axes of the eye coordinate system

- All these coordinates are in camera space now

**Overhead View of Our Screen**

- Yeah, similar triangles!

\[
\frac{x}{z} = \frac{x'}{d} \Rightarrow x' = \frac{dx}{z} \quad \frac{y}{z} = \frac{y'}{d} \Rightarrow y' = \frac{dy}{z}
\]
The Perspective Matrix

- This “division by z” can be accomplished by a 4x4 matrix too!

\[ P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1/d & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \]

- What happens to the point \((x, y, z, 1)\)?

\((x, y, z/z, d)\)

- What point is this in non-homogeneous coordinates?

\((dx/z, dy/z, d)\)

Taxonomy of Projections

- Perspective projection
  + Size varies inversely with distance - looks realistic
  - Distance and angles are not (in general) preserved
  - Parallel lines do not (in general) remain parallel

- Parallel projection
  + Good for exact measurements
  + Parallel lines remain parallel
  - Angles are not (in general) preserved
  - Less realistic looking

Perspective vs. Parallel

- OpenGL has multiple matrix stacks - transformation functions right-multiply the top of the stack

- Two most important stacks: GL_MODELVIEW and GL_PROJECTION

- Points get multiplied by the modelview matrix first, and then the projection matrix

- GL_MODELVIEW: Object->Camera

- GL_PROJECTION: Camera->Screen

- glViewport(0,0,w,h): Screen->Device

Open GL Example

```c
void SetUpViewing()
{
    // The viewport isn’t a matrix, it’s just state...
    glViewport( 0, 0, window_width, window_height );

    // Set up camera-screen transformation first
    glMatrixMode( GL_PROJECTION );
    glLoadIdentity();
    float asp = (float)window_width / (float)window_height;
    gluPerspective( 60, asp, 1, 1000 ); // fov, aspect, near, far

    // Set up the model-camera transformation
    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity();
    gluLookAt( 3, 3, 2, // eye point
                0, 0, 0, // look at point
                0, 0, 1 ); // up vector
    glRotatef( theta, 0, 0, 1 ); // rotate the model
    glScalef( zoom, zoom, zoom ); // scale the model
}
```
Summary

• Camera transformation
  ▫ Map 3D world coordinates to 3D camera coordinates
  ▫ Matrix has camera vectors as columns

• Projection transformation
  ▫ Map 3D camera coordinates to 2D screen coordinates
  ▫ Two types of projections:
    » Parallel
    » Perspective

What’s next?

<table>
<thead>
<tr>
<th>3D Geometric Primitives</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transform into 3D world coordinate system</td>
</tr>
<tr>
<td>Illuminate according to lighting and reflectance</td>
</tr>
<tr>
<td>Transform into 3D camera coordinate system</td>
</tr>
<tr>
<td>Transform into 2D camera coordinate system</td>
</tr>
<tr>
<td>Clip primitives outside camera’s view</td>
</tr>
<tr>
<td>Draw pixels (includes texturing, hidden surface, etc.)</td>
</tr>
</tbody>
</table>