3D Polygon Rendering Pipeline

Greg Humphreys
University of Virginia
CS 445, Fall 2003

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
(3D 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?

3D Rendering Pipeline (direct illumination)

This is a pipelined sequence of operations to draw a 3D primitive into a 2D image

Example: OpenGL

OpenGL executes steps of 3D rendering pipeline for each polygon

```c
glBegin(GL_POLYGON);
glVertex3f(0.0, 0.0, 0.0);
glVertex3f(1.0, 0.0, 0.0);
glVertex3f(1.0, 1.0, 1.0);
glVertex3f(0.0, 1.0, 1.0);
glEnd();
```
3D Rendering Pipeline (for direct illumination)

3D Geometric Primitives

- Transform into 3D world coordinate system
- Illuminate according to lighting and reflectance
- Transform into 3D camera coordinate system

3D Geometric Primitives

- Transform into 3D world coordinate system
- Illuminate according to lighting and reflectance
- Transform into 3D camera coordinate system
- Transform into 2D screen coordinate system
3D Rendering Pipeline (for direct illumination)

3D Geometric Primitives

Modeling Transformation
- Transform into 3D world coordinate system

Lighting
- Illuminate according to lighting and reflectance

Viewing Transformation
- Transform into 3D camera coordinate system

Projection Transformation
- Transform into 2D screen coordinate system

Clipping
- Clip primitives outside camera’s view

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

Image

Transformations

3D Geometric Primitives

Modeling Transformation
- Transform into 3D world coordinate system

Lighting
- Illuminate according to lighting and reflectance

Viewing Transformation
- Transform into 3D camera coordinate system

Projection Transformation
- Transform into 2D screen coordinate system

Clipping
- Clip primitives outside camera’s view

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

3D World Coordinates

Transformations map points from one coordinate system to another
Viewing Transformations

- 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

\[
p' = T \cdot p_w
\]

- Trick: find \( T^{-1} \) taking objects in camera to world

\[
p_w = T^{-1} \cdot p'
\]
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

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

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

Viewing Transformations

- General definition:
  - Transform points in $n$-space to $m$-space ($m<n$)

- In computer graphics:
  - Map 3D camera coordinates to 2D screen coordinates

Projection

- 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 View Volume](image)

H&B Figure 12.30

Parallel Projection Matrix

- General parallel projection transformation:

\[
\begin{bmatrix}
1 & 0 & L_x \cos \phi & 0 \\
0 & 1 & L_y \sin \phi & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

Parallel Projection View Volume

Taxonomy of Projections

![Taxonomy of Projections](image)

FVHP Figure 6.10

Perspective Projection

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

![Perspective Projection](image)

Angel Figure 5.9
Perspective Projection

- How many vanishing points?

3-Point Perspective 2-Point Perspective 1-Point Perspective

Angel Figure 5.10

Perspective Projection View Volume

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!

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

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

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

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

- What point is this in non-homogeneous coordinates?
  \((dx/z, dy/z, d)\)

Taxonomy of Projections

Perspective vs. Parallel

- 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

Classical Projections

- Planar geometric projections
  - Parallel
  - Orthographic
  - Cabinet
  - Isometric
  - Oblique
  - One-point
  - Perspective
  - Two-point
  - Three-point
Viewing in OpenGL

- 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

OpenGL 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();
    gluPerspective( 60, 1, 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 rows

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

What’s next?

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