Three-Dimensional Viewing

Jehee Lee
Seoul National University
Viewing Pipeline
Virtual Camera Model

• Viewing Transformation
  – The camera position and orientation is determined

• Projection Transformation
  – The selected view of a 3D scene is projected onto a view plane
General 3D Viewing Pipeline

- Modeling coordinates (MC)
- World coordinates (WC)
- Viewing coordinates (VC)
- Projection coordinates (PC)
- Normalized coordinates (NC)
- Device coordinates (DC)
Viewing-Coordinate Parameters

• View point (eye point or viewing position)

\[ P_0 = (x_0, y_0, z_0) \]

• View-plane normal vector \( \mathbf{N} \)
Viewing-Coordinate Parameters

- Look-at point $P_{\text{ref}}$

\[ N = P_0 - P_{\text{ref}} \]

- View-up vector $V$
  - $N$ and $V$ are specified in the world coordinates
Viewing-Coordinate Reference Frame

• The camera orientation is determined by the $uvn$ reference frame

\[
\mathbf{n} = \frac{\mathbf{N}}{\|\mathbf{N}\|} = (n_x, n_y, n_z)
\]

\[
\mathbf{u} = \frac{\mathbf{V} \times \mathbf{n}}{\|\mathbf{V}\|} = (u_x, u_y, u_z)
\]

\[
\mathbf{v} = \mathbf{n} \times \mathbf{u} = (v_x, v_y, v_z)
\]
World-to-Viewing Transformation

- Transformation from world to viewing coordinates
  - Translate the viewing-coordinate origin to the world-coordinate origin
  - Apply rotations to align the $u$, $v$, $n$ axes with the world $x_w$, $y_w$, $z_w$ axes, respectively
World-to-Viewing Transformation

\[
R = \begin{bmatrix}
    u_x & u_y & u_z & 0 \\
    v_x & v_y & v_z & 0 \\
    n_x & n_y & n_z & 0 \\
    0  & 0 & 0 & 1
\end{bmatrix}
\quad T = \begin{bmatrix}
    1 & 0 & 0 & -x_0 \\
    0 & 1 & 0 & -y_0 \\
    0 & 0 & 1 & -z_0 \\
    0 & 0 & 0 & 1
\end{bmatrix}
\]

\[
M_{wc,vc} = R \cdot T = \begin{bmatrix}
    u_x & u_y & u_z & -u \cdot P_0 \\
    v_x & v_y & v_z & -v \cdot P_0 \\
    n_x & n_y & n_z & -n \cdot P_0 \\
    0  & 0 & 0 & 1
\end{bmatrix}
\]
Perspective Projection

• Pin-hole camera model
  – Put the optical center (Center Of Projection) at the origin
  – Put the image plane (Projection Plane) in front of the COP
  – The camera looks down the negative z axis
    • we need this if we want right-handed-coordinates
Perspective Projection

- **Projection equations**
  - Compute intersection with PP of ray from \((x,y,z)\) to COP
  - Derived using similar triangles (on board)
    
    \[
    (x, y, z) \rightarrow (-d \frac{x}{z}, -d \frac{y}{z}, -d)
    \]
    
    - We get the projection by throwing out the last coordinate:
      
      \[
      (x, y, z) \rightarrow (-d \frac{x}{z}, -d \frac{y}{z})
      \]
Homogeneous coordinates

- Is this a linear transformation?

\[(x, y, z) \rightarrow (-d\frac{x}{z}, -d\frac{y}{z})\]
Homogeneous coordinates

- Trick: add one more coordinate:

\[(x, y) \Rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}\]

homogeneous projection coordinates

\[(x, y, z) \Rightarrow \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}\]

homogeneous viewing coordinates

- Converting from homogeneous coordinates

\[
\begin{bmatrix} x \\ y \\ w \end{bmatrix} \Rightarrow (x/w, y/w) \quad \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \Rightarrow (x/w, y/w, z/w)
\]
Perspective Projection

• Projection is a matrix multiply using homogeneous coordinates:

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & -1/d & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix}
= 
\begin{bmatrix}
x \\
y \\
z/d \\
1
\end{bmatrix} 
\Rightarrow (-d \frac{x}{z}, -d \frac{y}{z})
\]

divide by third coordinate

• This is known as **perspective projection**
  – The matrix is the **projection matrix**
  – Can also formulate as a 4x4

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & -1/d & 0
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix}
= 
\begin{bmatrix}
x \\
y \\
z \\
-z/d
\end{bmatrix} 
\Rightarrow (-d \frac{x}{z}, -d \frac{y}{z})
\]

divide by fourth coordinate
Perspective Projection

• The projection matrix can be much involved, if the COP is different from the origin of the uvn coordinates
  – See the textbook for the detailed matrix
Traditional Classification of Projections

- Three principle axes of the object is assumed
  - The front, top, and side face of the scene is apparent

Figure 7-44
Principal vanishing points for perspective-projection views of a cube. When the cube in (a) is projected to a view plane that intersects only the z axis, a single vanishing point in the z direction (b) is generated. When the cube is projected to a view plane that intersects both the z and x axes, two vanishing points (c) are produced.
Traditional Classification of Projections

3-Point Perspective

2-Point Perspective

1-Point Perspective
Perspective-Projection View Volume

• Viewing frustum
  – Why do we need near and far clipping plane?
Normalizing Transformation

• Transform an arbitrary perspective-projection view volume into the canonical view volume

• Step 1: from frustum to parallelepiped
Normalizing Transformation

- Transform an arbitrary perspective-projection view volume into the canonical view volume

- Step 2: from parallelepiped to normalized
Parallel Projection

• Special case of perspective projection
  – Distance from the COP to the PP is infinite
  – Also called “parallel projection”
  – What’s the projection matrix?

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix}
= 
\begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix} \Rightarrow (x, y)
\]
Taxonomy of Geometric Projections

- geometric projections
  - parallel
    - orthographic
    - axonometric
      - trimetric
      - dimetric
        - isometric
  - perspective
    - oblique
      - cavalier
      - cabinet
    - two-point
    - three-point
Orthographic Transformation

- Preserves relative dimension
- The center of projection at infinity
- The direction of projection is parallel to a principle axis
- Architectural and engineering drawings
Axonometric Transformation

• Orthogonal projection that displays more than one face of an object
  – Projection plane is not normal to a principal axis, but DOP is perpendicular to the projection plane
  – Isometric, dimetric, trimetric
Oblique Parallel Projections

- Projection plane is not normal to a principal axis, but DOP is perpendicular to the projection plane
- Only faces of the object parallel to the projection plane are shown true size and shape
Oblique Parallel Projections
Oblique Parallel Projections

Figure 7-39

Top view of an oblique parallel-projection transformation. The oblique view volume is converted into a rectangular parallelepiped, and objects in the view volume, such as the green block, are mapped to orthogonal-projection coordinates.
Oblique Parallel Projections

- Typically, $\phi$ is either 30° or 45°
- $L_1$ is the length of the projected side edge
  - Cavalier projections
    - $L_1$ is the same as the original length
  - Cabinet projections
    - $L_1$ is the half of the original length
Oblique Parallel Projections

- Cavalier projections

- Cabinet projections
OpenGL 3D Viewing Functions

• Viewing-transformation function
  – `glMatrixMode(GL_MODELVIEW);`
  – `gluLookAt(x0,y0,z0,xref,yref,zref,vx,vy,vz);`
  – Default: `gluLookAt(0,0,0, 0,0,-1, 0,1,0);`

• OpenGL orthogonal-projection function
  – `glMatrixMode(GL_PROJECTION);`
  – `gluOrtho(xwmin,xwmax, ywmin,ywmax, dnear,dfar);`
  – Default: `gluOrtho(-1,1, -1,1, -1,1);`
  – Note that
    • `dnear` and `dfar` must be assigned positive values
    • `znear=-dnear` and `zfar=-dfar`
    • The near clipping plane is the view plane
OpenGL 3D Viewing Functions

- OpenGL perspective-projection function
  - The projection reference point is the viewing-coordinate origin
  - The near clipping plane is the view plane
  - Symmetric: `gluPerspective(theta, aspect, dnear, dfar)`
  - General: `glFrustum(xwmin, xwmax, ywmin, ywmax, dnear, dfar)`
Line Clipping

• Basic calculations:
  – Is an endpoint inside or outside the clipping window?
  – Find the point of intersection, if any, between a line segment and an edge of the clipping window.

✓ Both endpoints inside: trivial accept
✓ One inside: find intersection and clip
✓ Both outside: either clip or reject
Cohen-Sutherland Line Clipping

- One of the earliest algorithms for fast line clipping
- Identify trivial accepts and rejects by bit operations
Cohen-Sutherland Line Clipping

- Compute region codes for two endpoints
- If (both codes = 0000) trivially accepted
- If (bitwise AND of both codes ≠ 0000) trivially rejected
- Otherwise, divide line into two segments
  - test intersection edges in a fixed order.
    (e.g., top-to-bottom, right-to-left)
3D Clipping Algorithms

- Three-dimensional region coding
Cyrus-Beck Line Clipping

• Use a parametric line equation

\[ P(t) = P_0 + t(P_1 - P_0), \quad 0 \leq t \leq 1 \]

• Reduce the number of calculating intersections by exploiting the parametric form

• Notations
  – \( E_i \): edge of the clipping window
  – \( N_i \): outward normal of \( E_i \)
  – An arbitrary point \( P_{E_i} \) on edge \( E_i \)
Cyrus-Beck Line Clipping

\[ N_i \cdot (P(t) - P_{E_i}) < 0 \quad \Leftrightarrow \quad \text{a point in the inside halfplane} \]
\[ N_i \cdot (P(t) - P_{E_i}) = 0 \quad \Leftrightarrow \quad \text{a point on the line containing the edge} \]
\[ N_i \cdot (P(t) - P_{E_i}) > 0 \quad \Leftrightarrow \quad \text{a point in the outside halfplane} \]
Cyrus-Beck Line Clipping

- Solve for the value of $t$ at the intersection of $P_0P_1$ with the edge
  - $N_i \cdot [P(t) - P_{Ei}] = 0$ and $P(t) = P_0 + t(P_1 - P_0)$
  - letting $D = (P_1 - P_0)$,

$$t = \frac{N_i \cdot [P_0 - P_{Ei}]}{-N_i \cdot D}$$

- Where
  - $N_i \neq 0$
  - $D \neq 0$ (that is, $P_0 \neq P_1$)
  - $N_i \cdot D \neq 0$ (if not, no intersection)
Cyrus-Beck Line Clipping

- Given a line segment $P_0P_1$, find intersection points against four edges
  - Discard an intersection point if $t \notin [0, 1]$
  - Label each intersection point either PE (potentially entering) or PL (potentially leaving)
  - Choose the smallest (PE, PL) pair that defines the clipped line

\[
N_i \cdot P_0P_1 < 0 \implies \text{PE}
\]
\[
N_i \cdot P_0P_1 > 0 \implies \text{PL}
\]
3D Clipping Algorithms

- Parametric line clipping

\[ P(t) = P_0 + t(P_1 - P_0), \quad 0 \leq t \leq 1 \]

\[ N_i \cdot (P(t) - P_{F_i}) < 0 \iff \text{inside halfplane} \]

\[ N_i \cdot (P(t) - P_{F_i}) = 0 \iff \text{on the face} \]

\[ N_i \cdot (P(t) - P_{F_i}) > 0 \iff \text{outside halfplane} \]
Polygon Fill-Area Clipping

- Polyline vs polygon fill-area

- Early rejection is useful

![Clipping Window](image)

Bounding box of polygon fill area

Clipping Window
Sutherland-Hodgman Polygon Clipping

- Clip against 4 infinite clip edges in succession
Sutherland-Hodgman Polygon Clipping

• Accept a series of vertices (polygon) and outputs another series of vertices

• Four possible outputs

1. out → in
   Output: $V_1', V_2$

2. in → in
   Output: $V_2$

3. in → out
   Output: $V_1'$

4. out → out
   Output: none
Sutherland-Hodgman Polygon Clipping

- The algorithm correctly clips convex polygons, but may display extraneous lines for concave polygons.
Weiler-Atherton Polygon Clipping

- For an outside-to-inside pair of vertices, follow the polygon boundary.
- For an inside-to-outside pair of vertices, follow the window boundary in a clockwise direction.
Weiler-Atherton Polygon Clipping

- Polygon clipping using nonrectangular polygon clip windows

![Diagram of polygon clipping](image)

**Figure 6-30**

Clipping a polygon fill area against a concave-polygon clipping window using the Weiler-Atherton algorithm.
3D Clipping Algorithms

• Three-dimensional polygon clipping
  – Bounding box or sphere test for early rejection
  – Sutherland-Hodgman and Weiler-Atherton algorithms can be generalized
Programming Assignment #2 (3D Viewer)

- You are required to implement a 3D OpenGL scene viewer
- The viewer should use a virtual trackball to rotate the view
  - The point of rotation is by default the center of the world coordinate system, but can be placed anywhere in the scene
- The viewer should allow you to
  - translate in the screen plane
  - dolly in and out (forward/backward movement)
  - zoom in/out (change the camera field of view)
- Your are required to submit a report of at most 3 pages
  - Describe how to use your program
  - Describe what you implemented, and what you haven’t
Programming Assignment #2 (3D Viewer)

• Virtual trackball
  – A trackball translates 2D mouse movements into 3D rotations
  – This is done by projecting the position of the mouse on to an imaginary sphere behind the viewport
  – As the mouse is moved the camera (or scene) is rotated to keep the same point on the sphere underneath the mouse pointer
Programming Assignment #2 (3D Viewer)

• Virtual trackball

\[
\text{Axis} = \mathbf{v}_1 \times \mathbf{v}_2
\]

\[
\theta = \arccos \left( \frac{\mathbf{v}_1 \cdot \mathbf{v}_2}{|\mathbf{v}_1| |\mathbf{v}_2|} \right)
\]
Programming Assignment #2 (3D Viewer)

• (Extra credits)
  – Show all: Decide the camera position and orientation such that the entire scene is viewed in a single screen
    • Pull the camera backward
    • change the viewing direction to aim the center of the scene
  – Seek: Pick a 3D point in the scene by the mouse pointer and place the center of rotation at that point.