Models and The Viewing Pipeline

Jian Huang

CS456
Polygon Mesh

- Vertex coordinates list, polygon table and (maybe) edge table
- Auxiliary:
  - Per vertex normal
  - Neighborhood information, arranged with regard to vertices and edges

---

**VERTEX TABLE**

<table>
<thead>
<tr>
<th>$V_1$</th>
<th>$x_1$, $y_1$, $z_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_2$</td>
<td>$x_2$, $y_2$, $z_2$</td>
</tr>
<tr>
<td>$V_3$</td>
<td>$x_3$, $y_3$, $z_3$</td>
</tr>
<tr>
<td>$V_4$</td>
<td>$x_4$, $y_4$, $z_4$</td>
</tr>
<tr>
<td>$V_5$</td>
<td>$x_5$, $y_5$, $z_5$</td>
</tr>
</tbody>
</table>

**EDGE TABLE**

<table>
<thead>
<tr>
<th>$E_1$</th>
<th>$V_1$, $V_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E_2$</td>
<td>$V_2$, $V_3$</td>
</tr>
<tr>
<td>$E_3$</td>
<td>$V_3$, $V_4$</td>
</tr>
<tr>
<td>$E_4$</td>
<td>$V_4$, $V_5$</td>
</tr>
<tr>
<td>$E_5$</td>
<td>$V_5$, $V_1$</td>
</tr>
</tbody>
</table>

**POLYGON-SURFACE TABLE**

<table>
<thead>
<tr>
<th>$S_1$</th>
<th>$E_1$, $E_2$, $E_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_2$</td>
<td>$E_3$, $E_4$, $E_5$, $E_6$</td>
</tr>
</tbody>
</table>
Transformations – Need?

- Modeling transformations
  - build complex models by positioning simple components
- Viewing transformations
  - placing virtual camera in the world
  - transformation from world coordinates to eye coordinates
- Animation: vary transformations over time to create motion
Viewing Pipeline

<table>
<thead>
<tr>
<th>Object Space</th>
<th>World Space</th>
<th>Eye Space</th>
<th>Clipping Space</th>
<th>Canonical view volume</th>
<th>Screen Space</th>
</tr>
</thead>
</table>

- Object space: coordinate space where each component is defined
- World space: all components put together into the same 3D scene via affine transformation. (camera, lighting defined in this space)
- Eye space: camera at the origin, view direction coincides with the z axis. Hither and Yon planes perpendicular to the z axis
- Clipping space: do clipping here. All point is in homogeneous coordinate, i.e., each point is represented by (x,y,z,w)
- 3D image space (Canonical view volume): a parallelepiped shape defined by (-1:1,-1:1,0,1). Objects in this space is distorted
- Screen space: x and y coordinates are screen pixel coordinates
Spaces

Object Space and World Space:

Eye-Space:
Spaces

Clip Space:

Image Space:
2D Transformation

- Translation
  \[
  \begin{aligned}
  x' &= x + t_x \\
  y' &= y + t_y 
  \end{aligned}
  \]

- Rotation
  \[
  \begin{aligned}
  x' &= x \cdot \cos \theta - y \cdot \sin \theta \\
  y' &= x \cdot \sin \theta + y \cdot \cos \theta 
  \end{aligned}
  \]

Matrix and Vector format:

\[
\begin{bmatrix}
  x' \\
  y' 
\end{bmatrix} =
M \begin{bmatrix}
  x \\
  y 
\end{bmatrix} =
\begin{bmatrix}
  \cos \theta & -\sin \theta \\
  \sin \theta & \cos \theta 
\end{bmatrix} \begin{bmatrix}
  x \\
  y 
\end{bmatrix}
\]
Homogeneous Coordinates

- Matrix/Vector format for translation:

\[
\begin{bmatrix}
  x' \\
  y'
\end{bmatrix} = M \begin{bmatrix}
  x \\
  y
\end{bmatrix} = \begin{bmatrix}
  m_{00} & m_{01} & m_{02} \\
  m_{10} & m_{11} & m_{12} \\
  m_{20} & m_{21} & m_{22}
\end{bmatrix} \begin{bmatrix}
  x \\
  y \\
  1
\end{bmatrix}
\]

\[
\begin{aligned}
x' &= x + t_x \\
y' &= y + t_y
\end{aligned}
\]

\[
M = \begin{bmatrix}
  1 & 0 & t_x \\
  0 & 1 & t_y \\
  0 & 0 & 1
\end{bmatrix}
\]
Translation in Homogenous Coordinates

- There exists an inverse mapping for each function
- There exists an identity mapping

\[
M^{-1} = \begin{bmatrix}
1 & 0 & -t_x \\
0 & 1 & -t_y \\
0 & 0 & 1
\end{bmatrix}
\]

\[
M \bigg|_{t_x=0} = \begin{bmatrix}
1 & 0 & -t_x \\
0 & 1 & -t_y \\
0 & 0 & 1
\end{bmatrix} = \begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix} = \text{Identity}(I)
\]
Why these properties are important

• when these conditions are shown for any class of functions it can be proven that such a class is closed under composition

• i.e. any series of translations can be composed to a single translation.

\[ x' = \underbrace{T_1 T_2 \cdots T_n}_T x \]
Rotation in Homogeneous Space

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

\[
M_R =
\begin{bmatrix}
\cos \theta & -\sin \theta & 0 \\
\sin \theta & \cos \theta & 0 \\
0 & 0 & 1
\end{bmatrix}
\]

\[
M_R^{-1} =
\begin{bmatrix}
\cos \theta & \sin \theta & 0 \\
-\sin \theta & \cos \theta & 0 \\
0 & 0 & 1
\end{bmatrix}
\]

The two properties still apply.

\[M_R \big|_{\theta=0} = \text{Identity}\]
Putting Translation and Rotation Together

• Order matters !!
Affine Transformation

• Property: preserving parallel lines
• The coordinates of three corresponding points uniquely determine any Affine Transform!!
Affine Transformations

- Translation
- Rotation
- Scaling
- Shearing

\[ M = \begin{bmatrix} m_{00} & m_{01} & 0 \\ m_{10} & m_{11} & 0 \\ m_{20} & m_{21} & 1 \end{bmatrix}^T \]
How to determine an Affine 2D Transformation?

• We set up 6 linear equations in terms of our 6 unknowns. In this case, we know the 2D coordinates before and after the mapping, and we wish to solve for the 6 entries in the affine transform matrix.

\[
\begin{bmatrix}
    x_1' \\
    y_1' \\
    x_2' \\
    y_2' \\
    x_3' \\
    y_3'
\end{bmatrix}
= \begin{bmatrix}
    x_1 & y_1 & 1 & 0 & 0 & 0 \\
    0 & 0 & 0 & x_1 & y_1 & 1 \\
    x_2 & y_2 & 1 & 0 & 0 & 0 \\
    0 & 0 & 0 & x_2 & y_2 & 1 \\
    x_3 & y_3 & 1 & 0 & 0 & 0 \\
    0 & 0 & 0 & x_3 & y_3 & 1
\end{bmatrix}
\begin{bmatrix}
m_{00} \\
m_{01} \\
m_{10} \\
m_{11} \\
m_{20} \\
m_{21}
\end{bmatrix}
\]
Affine Transformation in 3D

- Translation

\[
\begin{pmatrix}
1 & 0 & 0 & T_x \\
0 & 1 & 0 & T_y \\
0 & 0 & 1 & T_z \\
0 & 0 & 0 & 1
\end{pmatrix}
\]

- Rotate

\[
\begin{pmatrix}
\cos \theta & -\sin \theta & 0 & 0 \\
\sin \theta & \cos \theta & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\]

- Scale

\[
\begin{pmatrix}
S_x & 0 & 0 & 0 \\
0 & S_y & 0 & 0 \\
0 & 0 & S_z & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\]

- Shear

\[
\begin{pmatrix}
1 & 0 & S H_x & 0 \\
0 & 1 & S H_y & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\]
More Rotation

• Which axis of rotation is this?

$$R_x = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & \cos\theta & -\sin\theta & 0 & 0 \\
0 & \sin\theta & \cos\theta & 0 & 0 \\
0 & 0 & 0 & 1 & 0
\end{bmatrix}$$

$$R_y = \begin{bmatrix}
\cos\theta & 0 & \sin\theta & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
-\sin\theta & 0 & \cos\theta & 0 & 0 \\
0 & 0 & 0 & 1 & 0
\end{bmatrix}$$

$$R_z = \begin{bmatrix}
\cos\theta & -\sin\theta & 0 & 0 & 0 \\
\sin\theta & \cos\theta & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1
\end{bmatrix}$$
Viewing

- Object space to World space: affine transformation
- World space to Eye space: how?
- Eye space to Clipping space involves projection and viewing frustum
Perspective Projection

- Projection point sees anything on ray through pinhole $F$
- Point $W$ projects along the ray through $F$ to appear at $I$ (intersection of $WF$ with image plane)
Image Formation

Projecting shapes

- project points onto image plane
- lines are projected by projecting its end points only
Orthographic Projection

- focal point at infinity
- rays are parallel and orthogonal to the image plane
Comparison
Simple Perspective Camera

- camera looks along $z$-axis
- focal point is the origin
- image plane is parallel to $xy$-plane at distance $d$
- $d$ is call focal length for historical reason
Similar Triangles

- Similar situation with $x$-coordinate
- Similar Triangles: point $[x, y, z]$ projects to $[(d/z)x, (d/z)y, d]$
Projection Matrix

Projection using homogeneous coordinates:

- transform \([x, y, z]\) to \([(d/z)x, (d/z)y, d]\)

\[
\begin{bmatrix}
d & 0 & 0 & 0 \\
0 & d & 0 & 0 \\
0 & 0 & d & 0 \\
0 & 0 & 1 & 0
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix}
= \begin{bmatrix}
dx & dy & dz & z
\end{bmatrix} \Rightarrow \begin{bmatrix}
d & d & d & d \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}

\text{Divide by 4th coordinate (the “w” coordinate)}

- 2-D image point:
  - discard third coordinate
  - apply viewport transformation to obtain physical pixel coordinates
View Volume

- Defines visible region of space, pyramid edges are clipping planes
- Frustum: truncated pyramid with near and far clipping planes
  - Near (Hither) plane? Don’t care about behind the camera
  - Far (Yon) plane, define field of interest, allows $z$ to be scaled to a limited fixed-point value for $z$-buffering.
Difficulty

• It is difficult to do clipping directly in the viewing frustum
Canonical View Volume

- Normalize the viewing frustum to a cube, canonical view volume
- Converts perspective frustum to orthographic frustum – perspective transformation
Perspective Transform

• The equations

\[
\begin{align*}
\alpha &= \frac{yon}{yon-hither} \\
\beta &= \frac{yon \times hither}{hither - yon}
\end{align*}
\]

s: size of window on the image plane

\[
\begin{align*}
x &= \frac{x \times d}{z \times s} \\
y &= \frac{y \times d}{z \times s} \\
z &= \alpha + \frac{\beta}{z}
\end{align*}
\]
About Perspective Transform

• Some properties
About Perspective Transform

• Clipping can be performed against the rectilinear box
• Planarity and linearity are preserved
• Angles and distances are not preserved
• Side effects: objects behind the observer are mapped to the front. Do we care?
Perspective + Projection Matrix

- AR: aspect ratio correction, ResX/ResY
- s = ResX,
- Theta: half view angle, \( \tan(\theta) = \frac{s}{d} \)

\[
P = \begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & AR & 0 & 0 \\
0 & 0 & \alpha \tan \theta & \tan \theta \\
0 & 0 & \beta \tan \theta & 0 \\
\end{pmatrix}.
\]
Camera Control and Viewing

Focal length (d), image size/shape and clipping planes included in perspective transformation

- $\rho$  
  Angle or Field of view (FOV)
- $AR$  
  Aspect Ratio of view-port
- $Hither, Yon$  
  Nearest and farthest vision limits (WS).

Lookat - coi
Lookfrom - eye
View angle - FOV
Complete Perspective

• Specify near and far clipping planes - transform $z$ between $z_{near}$ and $z_{far}$ on to a fixed range
• Specify field-of-view (fov) angle
• OpenGL’s `glFrustum` and `gluPerspective` do these
More Viewing Parameters

Camera, Eye or Observer:
- *lookfrom*: location of focal point or camera
- *lookat*: point to be centered in image

Camera orientation about the *lookat-lookfrom* axis

* vup: a vector that is pointing straight up in the image. This is like an orientation.
Implementation … Full Blown

• Translate by -lookfrom, bring focal point to origin
• Rotate lookat-lookfrom to the z-axis with matrix R:
  • \( v = (\text{lookat-lookfrom}) \) (normalized) and \( z = [0,0,1] \)
  • rotation axis: \( a = (v \times z)/|v \times z| \)
  • rotation angle: \( \cos \theta = a \cdot z \) and \( \sin \theta = |r \times z| \)

• OpenGL: `glRotate(\theta, a_x, a_y, a_z)`
• Rotate about z-axis to get vup parallel to the y-axis
Viewport mapping

• Change from the image coordinate system \((x, y, z)\) to the screen coordinate system \((X, Y)\).
• Screen coordinates are always non-negative integers.
• Let \((v_r, v_t)\) be the upper-right corner and \((v_l, v_b)\) be the lower-left corner.
• \[ X = x \times \frac{(v_r - v_l)}{2} + \frac{(v_r + v_l)}{2} \]
• \[ Y = y \times \frac{(v_t - v_b)}{2} + \frac{(v_t + v_b)}{2} \]
In perspective transformation parallelism is not preserved.
- Parallel lines converge
- Object size is reduced by increasing distance from center of projection
- Non-uniform foreshortening of lines in the object as a function of orientation and distance from center of projection
- Aid the depth perception of human vision, but shape is not preserved
True Or False

• Affine transformation is a combination of linear transformations
• The last column/row in the general 4x4 affine transformation matrix is $[0 \ 0 \ 0 \ 1]^T$.
• After affine transform, the homogeneous coordinate $w$ maintains unity.