Chapter 5

Viewing/Perspective Transformations

- Specify view point (change of coordinate system)
- Project from 3D to 2D (introduce perspective)
Computer Graphics

Transformations: Viewing & Perspective

Rendering Pipeline

- Scene graph
- Object geometry
- Modelling Transforms
- Viewing Transform
- Projection Transform

result
- all vertices of scene in shared 3D world coordinate system

Copyright 2011, A. Sheffer, UBC
Computer Graphics

Transformations:
Viewing & Perspective

Rendering Pipeline

- result
  - scene vertices in 3D view (camera) coordinate system

Rendering Pipeline

- result
  - 2D screen coordinates of clipped vertices
Projective Rendering Pipeline

OCS - object coordinate system
WCS - world coordinate system
VCS - viewing coordinate system
NDCS - normalized device coordinate system
DCS - device coordinate system

glVertex3f(x,y,z)

Modeling Transformation

glTranslatef(x,y,z)
glRotatef(th,x,y,z)

Viewing Transformation

gluLookAt(...)

Projection Transformation

glFrustum(...) Division

Viewport Transformation

Basic Viewing

- Starting spot - OpenGL
  - camera at world origin
    - probably inside an object
  - y axis is up
  - looking down negative z axis
    - why? RHS with x horizontal, y vertical, z out of screen
- To position - coordinate frame change
- Intuitive description
  - eye point, gaze/lookat direction, up vector
Camera Description/Motion

- arbitrary viewing position
  - eye point, gaze/lookat direction, up vector

From World to View Coordinates: W2V

- translate eye to origin
- rotate view vector (lookat - eye) to w axis
- rotate around w to bring up into vw-plane
Deriving W2V Transformation

- \( M = RT \)
- \( u = \frac{t \times w}{||t \times w||} \)
- \( v = w \times u \)
- \( w = \hat{g} = \frac{-g}{||g||} \)

\[
M_{\text{world} \rightarrow \text{view}} = \begin{bmatrix}
    u_x & u_y & u_z & 0 \\
    v_x & v_y & v_z & 0 \\
    w_x & w_y & w_z & 0 \\
    0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
    1 & 0 & 0 & -e_x \\
    0 & 1 & 0 & -e_y \\
    0 & 0 & 1 & -e_z \\
    0 & 0 & 0 & 1
\end{bmatrix}
\]

Notations/derivation from the board in class

OpenGL Viewing Transformation

- \( \text{gluLookAt}(ex, ey, ez, lx, ly, lz, ux, uy, uz) \)
- postmultiplies current matrix, so to be safe:

\[
\text{glMatrixMode}(\text{GL}_\text{MODELVIEW}); \\
\text{glLoadIdentity}(); \\
\text{gluLookAt}(ex, ey, ez, lx, ly, lz, ux, uy, uz) \\
// now ok to do model transformations
\]
**Computer Graphics**

**Transformations:**
**Viewing & Perspective**

---

**World vs. Camera Coordinates**

- \( a = (1,1)_w \)
- \( b = (1,1)_c^1 = (5,3)_w \)
- \( c = (1,1)_c^2 = (1,3)_c^1 = (5,5)_w \)

---

**Projective Rendering Pipeline**

- `glVertex3f(x,y,z)`
- `glTranslatef(x,y,z)`
- `glRotatef(th,x,y,z)`
- `gluLookAt(...)`
- `glFrustum(...)`

---

OCS - object coordinate system  
WCS - world coordinate system  
VCS - viewing coordinate system  
NDCS - normalized device coordinate system  
DCS - device coordinate system

---

Copyright 2011, A. Sheffer, UBC
Question: How to draw a 3D object on a 2D screen?

If we ignore perspective (viewer at infinity):

- Project transformed object along $Z$ axis onto $XY$ plane - and from there to screen (clipped)
- Canonical orthographic projection:
  \[
  \begin{bmatrix}
  1 & 0 & 0 & 0 \\
  0 & 1 & 0 & 0 \\
  0 & 0 & 0 & 0 \\
  0 & 0 & 0 & 1
  \end{bmatrix}
  \]

- In practice “ignore” $Z$ axis – use $X$ and $Y$ coordinates for screen coordinates

Clipping: View Volumes

- Specifies field-of-view, used for clipping
- Restricts domain of $Z$ stored for visibility test
Understanding Z

- z axis flip changes coord system handedness
- RHS before projection (eye/view coords)
- LHS after projection (clip, norm device coords)

VCS

$\begin{align*}
&\text{x=left} \\
&\text{y=bottom} \\
&\text{z=far} \\
&\text{y=top} \\
&\text{x=right} \\
&\text{z=near} \\
\end{align*}$

NDCS

$\begin{align*}
&(1,1,1) \\
&(-1,-1,-1) \\
&x \\
&y \\
&z \\
\end{align*}$

- why near and far plane?
  - near plane:
    - avoid singularity for perspective projection
      (division by zero, or very small numbers)
  - far plane:
    - store depth in fixed-point representation
      (integer), thus have to have fixed range of
      values (0...1)
    - avoid/reduce numerical precision artifacts for
      distant objects

Copyright 2011, A. Sheffer, UBC
Orthographic Derivation

- scale, translate, reflect for new coord sys

\[
\begin{align*}
y' &= a \cdot y + b \\
y &= \text{top} \implies y' = 1 \\
y &= \text{bot} \implies y' = -1
\end{align*}
\]

VCS

NDCS

\[
\begin{bmatrix}
2 \\
0 \\
0 \\
0 \\
0 \\
\end{bmatrix}
\begin{bmatrix}
\text{right} - \text{left} \\
\text{top} - \text{bot} \\
\text{far} - \text{near} \\
1
\end{bmatrix}
\]
Orthographic OpenGL

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(left, right, bot, top, near, far);
```

NDC to Viewport Transformation

- generate pixel coordinates
  - map x, y from range -1...1 (NDC) to pixel coordinates on the display
  - involves 2D scaling and translation

Copyright 2011, A. Sheffer, UBC
Yet more possibly confusing conventions
- OpenGL: lower left
- Most window systems: upper left
- Often have to flip your y coordinates
  - When interpreting mouse position

Viewing is from point at finite distance (origin)
- View volume is a frustum not a box
- Conversion to device coordinates
  - Warp view frustum to box
Perspective Derivation

VCS

NDCS

Projective Transformations

- OpenGL Convention

Copyright 2011, A. Sheffer, UBC
Perspective Derivation

Basic (derived in class)

\[
\begin{bmatrix}
  x' \\
  y' \\
  z' \\
  w'
\end{bmatrix} =
\begin{bmatrix}
  1 & 0 & 0 & 0 \\
  0 & 1 & 0 & 0 \\
  0 & 0 & 1 & 0 \\
  0 & 0 & 1/d & 1
\end{bmatrix}
\begin{bmatrix}
  x \\
  y \\
  z \\
  1
\end{bmatrix}
\quad (d = -1)
\]

complete: shear, scale, projection-normalization

\[
\begin{bmatrix}
  x' \\
  y' \\
  z' \\
  w'
\end{bmatrix} =
\begin{bmatrix}
  E & 0 & A & 0 \\
  0 & F & B & 0 \\
  0 & 0 & C & D \\
  0 & 0 & -1 & 0
\end{bmatrix}
\begin{bmatrix}
  x \\
  y \\
  z \\
  1
\end{bmatrix}
\]

Perspective Derivation

- Solve linear system to get A-F
- 6 planes, 6 unknowns

\[
\begin{bmatrix}
  x' \\
  y' \\
  z' \\
  w'
\end{bmatrix} =
\begin{bmatrix}
  1 & 0 & 0 & 0 \\
  0 & 1 & 0 & 0 \\
  0 & 0 & 1 & 0 \\
  0 & 0 & 1/d & 1
\end{bmatrix}
\begin{bmatrix}
  x \\
  y \\
  z \\
  1
\end{bmatrix}
\]

\[
\begin{bmatrix}
  2n/r-l & 0 & r+l/(r-l) & 0 \\
  0 & 2n/t-b & t+b/(t-b) & 0 \\
  0 & 0 & -(f+n)/(f-n) & -2fn/f-n \\
  0 & 0 & -1 & 0
\end{bmatrix}
\]
Projective Transformations

- Alternative specification of symmetric frusta
  - Field-of-view angles
    - In x-direction (fov) $\alpha$
    - In y-direction (fovy) given by aspect ratio

Perspective OpenGL

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();

glFrustum(left, right, bot, top, near, far);
```
or
```
glPerspective(fovy, aspect, near, far);
```
- symmetric version
## Another Transformations Quiz

**What does each transformation preserve?**

<table>
<thead>
<tr>
<th>Transformation</th>
<th>lines</th>
<th>parallel lines</th>
<th>distance</th>
<th>angles</th>
<th>normals</th>
<th>convexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>scaling</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rotation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>translation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>shear</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>perspective</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>