3D to 2D Projection

Prof. Aaron Lanterman
(Based on slides by Prof. Hsien-Hsin Sean Lee)
School of Electrical and Computer Engineering
Georgia Institute of Technology
Specifying the view transformation

• Most commonly parameterized by:
  – Position of camera
  – Position of point to look at
  – Vector indicating “up” direction of camera

• In Direct3D: D3DXMatrixLookAtLH
  – D3D uses a LHS, but also have D3DXMatrixLookAtRH

• In XNA: Matrix.CreateLookAt (RHS)

• In OpenGL: gluLookAt (RHS)

• Can also build a rotation+translation matrix as if the camera was an object in scene, then take the inverse of that matrix

Projection from 3D space

• Projection transforms 3D geometry into a form that can be rendered as a 2D image

Much discussion adapted from Joe Farrell’s article (http://www.codeguru.com/cpp/misc/misc/math/article.php/c10123_1/)
Canonical view volume

- Projection transforms your geometry into a *canonical view volume* in *normalized device coordinates*
- Only X- and Y-coordinates will be mapped onto the screen
- Z will be almost useless, but used for depth test

![Canonical view volume](image)
Strange “conventions”

Canonical view volume (LHS) (-1, -1, 0) to (1,1,1) used by D3D/XNA
(remember eye-space coordinates in XNA are in a RHS!!!)

Canonical view volume (LHS) (-1, -1, 1) to (1,1,1) used by OpenGL
(remember eye-space coordinates in OpenGL are in a RHS!!!)
Orthographic (or parallel) projection

- Project from 3D space to the viewer’s 2D space
Style of orthographic projection

- Same size in 2D and 3D
- No distance feel
- Parallel lines remain parallel
- Good for tile-based games where camera is in fixed location (e.g., Mahjong or 3D Tetris)

Orthographic projection

Canonical view volume (D3D & XNA)

View Volume
(an axis-aligned box)

Orthographic projection

View Volume
(an axis-aligned box)
Orthographic projection math (1)

• Derive $x'$ and $y'$

$$x \in [l, r] \quad x' \in [-1, 1]$$

$$-1 \leq \frac{2(x-l)}{r-l} \leq 1$$

$$l \leq x \leq r$$

$$-1 \leq \frac{2x-2l-r+l}{r-l} \leq 1$$

$$0 \leq x-l \leq r-l$$

$$-1 \leq \frac{2x}{r-l} - \frac{r+l}{r-l} \leq 1$$

$$0 \leq \frac{x-l}{r-l} \leq 1$$

$$0 \leq \frac{2(x-l)}{r-l} \leq 2$$

$$\therefore x' = \frac{2x}{r-l} - \frac{r+l}{r-l}$$

$$y' = \frac{2y}{t-b} - \frac{t+b}{t-b}$$

Orthographic projection math (2)

- Derive $z'$ (slightly different for the range in D3D)

\[ z \in [n, f] \quad z' \in [0, 1] \]

\[ 0 \leq \frac{z}{f-n} - \frac{n}{f-n} \leq 1 \]

\[ n \leq z \leq f \]

\[ 0 \leq z - n \leq f - n \]

\[ 0 \leq \frac{z - n}{f - n} \leq 1 \]

- OpenGL transform for $z$ looks more like $x$ & $y$ transforms

Ortho projection matrix (LHS)

• Put all together

\[ [x', y', z', 1] = [x, y, z, 1] \cdot P \text{ where } P = \]

\[
\begin{bmatrix}
\frac{2}{r - l} & 0 & 0 & 0 \\
0 & \frac{2}{t - b} & 0 & 0 \\
0 & 0 & \frac{1}{f - n} & 0 \\
-\frac{r + l}{r - l} & -\frac{t + b}{t - b} & -\frac{n}{f - n} & 1
\end{bmatrix}
\]
Ortho proj (LHS) Microsoft style

• Rearranging to look like Microsoft documentation

\[
\begin{bmatrix}
\frac{2}{r-l} & 0 & 0 & 0 \\
0 & \frac{2}{t-b} & 0 & 0 \\
0 & 0 & 1 & 0 \\
\frac{l+r}{l-r} & \frac{t+b}{b-t} & \frac{n}{n-f} & 1
\end{bmatrix}
\]

\[\begin{bmatrix}x', y', z', 1\end{bmatrix} = \begin{bmatrix}x, y, z, 1\end{bmatrix} \cdot P\] where \(P = \) \[
\begin{bmatrix}
\frac{2}{r-l} & 0 & 0 & 0 \\
0 & \frac{2}{t-b} & 0 & 0 \\
0 & 0 & 1 & 0 \\
\frac{l+r}{l-r} & \frac{t+b}{b-t} & \frac{n}{n-f} & 1
\end{bmatrix}
\]

• In Direct3D: D3DXMatrixOrthoOffCenterLH(*o,l,r,b,t,n,f)
• LHS is default system in Direct3D

Orthographic projection (RHS)

• Math the same, but z clipping plane inputs in most API calls are negated so z input parameters are positive

\[ [x',y',z',1] = [x,y,z,1] \cdot P \] where

\[
P = \begin{bmatrix}
\frac{2}{r-l} & 0 & 0 & 0 \\
0 & \frac{2}{t-b} & 0 & 0 \\
0 & 0 & \frac{1}{n-f} & 0 \\
\frac{l+r}{l-r} & \frac{t+b}{b-t} & \frac{n}{n-f} & 1
\end{bmatrix}
\]

• In Direct3D: D3DXMatrixOrthoOffCenterRH(*o,l,r,b,t,n,f)
• In XNA: Matrix.CreateOrthographicOffCenter(l,r,b,t,n,f)
• In OpenGL: glOrtho(l,r,b,t,n,f) (matrix is different)
  • OpenGL maps z to [-1,1] & uses column vectors

http://www.cs.utk.edu/~vose/c-stuff/opengl/glotho.html
Simpler ortho projection (LHS)

- In most orthographic projection setups
  - Z-axis passes through the center of your view volume
  - Field of view (FOV) extends equally far
    - To the left as to the right (i.e., \( r = -l \))
    - To the top as to the below (i.e., \( t = -b \))

\[
[x', y', z', 1] = [x, y, z, 1] \cdot P \quad \text{where} \quad P = \begin{bmatrix}
\frac{2}{w} & 0 & 0 & 0 \\
0 & \frac{2}{h} & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & \frac{f - n}{n - f} & 1
\end{bmatrix}
\]

- In Direct3D: D3DXMatrixOrthoLH(*o,w,h,n,f)

Simpler ortho projection (RHS)

• Math the same, but z clipping plane inputs in most API calls are negated so z input parameters are positive

\[
[x', y', z', 1] = [x, y, z, 1] \cdot P \quad \text{where} \quad P =
\begin{bmatrix}
\frac{2}{w} & 0 & 0 & 0 \\
0 & \frac{2}{h} & 0 & 0 \\
0 & 0 & \frac{1}{n-f} & 0 \\
0 & 0 & \frac{n}{n-f} & 1
\end{bmatrix}
\]

• In Direct3D: D3DXMatrixOrthoRH(*o,w,h,n,f)
• In XNA: Matrix.CreateOrthographic(w,h,n,f)

Perspective projection

- The farther the object is, the smaller it appears
- Some photo editing software allows you to perform "Perspective Correction"
Viewing frustum

Think about looking through a window in a dark room
Viewing frustum with furniture
Perspective projection

Canonical view volume (D3D & XNA)

View Frustum (a truncated pyramid)
Perspective projection mapping

• Given a point \((x,y,z)\) within the view frustum, project it onto the near plane \(z=n\)
  - \(x \in [l, r]\) and \(y \in [b, t]\)

• We will map \(x\) from \([l,r]\) to \([-1,1]\) and \(y\) from \([b,t]\) to \([-1,1]\)

Perspective projection math (1)

To calculate new coordinates of $x'$ and $y'$

$$\frac{x'}{x} = \frac{n}{z} \Rightarrow x' = \frac{nx}{z}$$

$$\frac{y'}{x'} = \frac{y}{x} \Rightarrow y' = \frac{yx'}{x} = \frac{y \cdot nx}{z} = \frac{ny}{z}$$

Next apply our orthographic projection formulas

Perspective projection math (2)

Now let's tackle the $z'$ component

Perspective projection math (3)

\[
x' \cdot z = \frac{2n}{r-l} \cdot x - \frac{r+l}{r-l} \cdot z \\
y' \cdot z = \frac{2n}{t-b} \cdot y - \frac{t+b}{t-b} \cdot z \\
z' \cdot z = p \cdot z + q \quad \text{where } p \text{ and } q \text{ are constants}
\]

- We know \( z \) (depth) transformation has nothing to do with \( x \) and \( y \)

Perspective projection math (4)

\[ z' \cdot z = p \cdot z + q \quad \text{where } p \text{ and } q \text{ are constants} \]

\[
\begin{align*}
0 &= p \cdot n + q \\
f &= p \cdot f + q
\end{align*}
\]

\[
\Rightarrow \quad :. \quad p = \frac{f}{f-n} \quad \text{and} \quad q = -\frac{fn}{f-n}
\]

\[ z' \cdot z = \frac{f}{f-n} \cdot z - \frac{fn}{f-n} \]

- We know (boxed equations above)
  - \( z' = 0 \) when \( z = n \) (near plane)
  - \( z' = 1 \) when \( z = f \) (far plane)

Perspective projection math (5)

\[ x' \cdot z = \frac{2n}{r - l} \cdot x - \frac{r + l}{r - l} \cdot z \]

\[ y' \cdot z = \frac{2n}{t - b} \cdot y - \frac{t + b}{t - b} \cdot z \]

\[ z' \cdot z = \frac{f}{f - n} \cdot z - \frac{fn}{f - n} \]

\[ w' \cdot z = z \]

\[ [x'z, y'z, z'z, w'z] = [x, y, z, 1] \cdot P \quad \text{where} \quad P = \begin{bmatrix}
\frac{2n}{r - l} & 0 & 0 & 0 \\
0 & \frac{2n}{t - b} & 0 & 0 \\
-\frac{r + l}{r - l} & -\frac{t + b}{t - b} & \frac{f}{f - n} & 1 \\
0 & 0 & -\frac{fn}{f - n} & 0
\end{bmatrix} \]

Simpler perspective projection

• Similar to orthographic projection, if \( l=-r \) and \( t=-b \), we can simplify to

\[
[x'z, y'z, z'z, w'z] = [x, y, z, 1] \cdot P \text{ where } P =
\begin{bmatrix}
\frac{2n}{w} & 0 & 0 & 0 \\
0 & \frac{2n}{h} & 0 & 0 \\
0 & 0 & \frac{f}{f-n} & 1 \\
0 & 0 & -\frac{f}{f-n} & 0
\end{bmatrix}
\]

• In any case, we will have to divide by \( z \) to obtain \([x', y', z', w']\)
  
  – Implemented by dividing by the fourth \((w'z)\) coordinate

Define viewing frustum

Parameters:
FOV: Field of View
Aspect ratio = Width/Height
Near z
Far z

XNA: Matrix.CreatePerspectiveFieldOfView

Reparameterized matrix

\[
P = \begin{bmatrix}
\frac{2n}{w} & 0 & 0 & 0 \\
0 & \frac{2n}{h} & 0 & 0 \\
0 & 0 & \frac{f}{f-n} & 1 \\
0 & 0 & \frac{fn}{f-n} & 0
\end{bmatrix}
\]

\[
\cot\left(\frac{a}{2}\right) = \frac{2n}{w}
\]

\[
r = \frac{w}{h}
\]

\[
\frac{2n}{w} = \frac{2n}{rh} = \frac{2n}{2n} = \frac{1}{r} \cdot \cot\left(\frac{a}{2}\right)
\]

Need to replace \( w \) and \( h \) with FOV and aspect ratio

Final matrix of perspective proj (LHS)

\[
[x',z',z',w'] = [x,y,z,1] \cdot P \quad \text{where } P = \\
\begin{bmatrix}
\frac{1}{r} \cdot \cot\left(\frac{a}{2}\right) & 0 & 0 & 0 \\
0 & \cot\left(\frac{a}{2}\right) & 0 & 0 \\
0 & 0 & \frac{f}{f-n} & 1 \\
0 & 0 & -\frac{fn}{f-n} & 0 \\
\end{bmatrix}
\]

- \(a\) : Field of View (FOV)
- \(r\) : aspect ratio = \(\frac{\text{width}}{\text{height}}\)
- \(n\) : near plan
- \(f\) : far plane

- In Direct3D: D3DXMatrixPerspectiveFovLH(*o,a,r,n,f)
- LHS is default system in Direct3D
Final matrix of perspective proj (RHS)

\[
\begin{bmatrix}
\frac{1}{r} \cdot \cot\left(\frac{a}{2}\right) & 0 & 0 & 0 \\
0 & \cot\left(\frac{a}{2}\right) & 0 & 0 \\
0 & 0 & \frac{f}{n-f} & -1 \\
0 & 0 & \frac{fn}{n-f} & 0
\end{bmatrix}
\]

\[[x', y', z', w' \cdot z] = [x, y, z, 1] \cdot P \quad \text{where} \quad P = \]

\[
\begin{bmatrix}
\frac{1}{r} \cdot \cot\left(\frac{a}{2}\right) & 0 & 0 & 0 \\
0 & \cot\left(\frac{a}{2}\right) & 0 & 0 \\
0 & 0 & \frac{f}{n-f} & -1 \\
0 & 0 & \frac{fn}{n-f} & 0
\end{bmatrix}
\]

a : Field of View (FOV) \quad r : \text{aspect ratio} = \frac{\text{width}}{\text{height}} \quad n : \text{near plan} \quad f : \text{far plane}

- In Direct3D: D3DXMatrixPerspectiveFovRH(*o,a,r,n,f)
- In XNA: Matrix.CreatePerspectiveFieldOfView(a,r,n,f)
- In OpenGL: gluPerspective(a,r,n,f)

Viewport transformation

- The actual 2D projection to the viewer
- Copy to your back buffer (frame buffer)
- Can be programmed, scaled, ...
Backface culling

• Determine “facing direction”
• Triangle order matters
• How to compute a normal vector for 2 given vectors?
  – Using Cross product of 2 given vectors

2 Vectors

\[ \vec{V}_1 = (b_1 - a_1)i + (b_2 - a_2)j + (b_3 - a_3)k \]
\[ \vec{V}_2 = (c_1 - a_1)i + (c_2 - a_2)j + (c_3 - a_3)k \]

Cross product

\[ \vec{V}_1 = x_1i + x_2j + x_3k \]
\[ \vec{V}_2 = y_1i + y_2j + y_3k \]
\[ \vec{V}_1 \times \vec{V}_2 = (x_2y_3 - x_3y_2)i + (x_3y_1 - x_1y_3)j + (x_1y_2 - x_2y_1)k \]
Compute the surface normal for a triangle

- Clockwise normals, LHS

\[
\vec{v}_1 = 3\hat{i} + 3\hat{j} + 0\hat{k} \\
\vec{v}_2 = 4\hat{i} + 0\hat{j} + 0\hat{k}
\]

\[
\vec{v}_1 = 4\hat{i} + 0\hat{j} + 0\hat{k} \\
\vec{v}_2 = 3\hat{i} + 3\hat{j} + 0\hat{k}
\]
Backface culling method

- Check if the normal is facing the camera
- How to determine that?
  - Use Dot Product

Surface vectors

Eye vector
Dot product method (1)

\[ \vec{A} \cdot \vec{B} = |A||B| \cos \theta \]

\[ \vec{A} \cdot \vec{B} > 0 \implies -\frac{\pi}{2} < \theta < \frac{\pi}{2} \]
Dot product method (2)

\[ \vec{A} \cdot \vec{B} > 0 \implies -\frac{\pi}{2} < \theta < \frac{\pi}{2} \]
When to perform backface culling?

Transform your camera to the world space first!
3D clipping

- Test 6 planes if a triangle is inside, outside, or partially inside the view frustum
- Clipping creates new triangles (triangulation)
  - Interpolate new vertices info
Clipping against a plane

• Test each vertex of a triangle
  – Outside
  – Inside
  – Partially inside

• Incurred computation overhead

• Save unnecessary computation (and bandwidth) later

• Need to know how to determine a plane

• Need to know how to determine a vertex is inside or outside a plane
Specifying a plane

- You need two things to specify a plane
  - A point on the plane \((p_0, p_1, p_2)\)
  - A vector (normal) perpendicular to the plane \((a, b, c)\)
  - Plane \(a(x - p_0) + b(y - p_1) + c(z - p_2) = 0\)
Distance calculation from a plane (1)

- Given a point R, calculate the distance
  - Distance > 0 inside the plane
  - Distance = 0 on the plane
  - Distance < 0 outside the plane

\[ d = | R - P | \cdot \cos \theta = | R - P | \cdot \frac{\vec{v} \cdot (R - P)}{|\vec{v}| \cdot |R - P|} = \frac{\vec{v} \cdot (R - P)}{|\vec{v}|} \]
Distance calculation from a plane (2)

\[ d = |R - P| \cdot \cos(180 - \theta) \]
Triangulation using interpolation

\[ s = \frac{d_1}{d_1 + d_2} \]

\[ x = a_1 + s \cdot (a_2 - a_1) \]

\[ y = b_1 + s \cdot (b_2 - b_1) \]

\[ z = c_1 + s \cdot (c_2 - c_1) \]