Topics

Perspective Projection
Coordinate Systems in OpenGL

Your starting coordinates

\( \{x, y, z\}\) object/model coordinates

Object units; could be meters, inches, etc.

Append \( w \) of 1.0

\( \{x, y, z, 1.0\}\) homogeneous model coordinates

Same units

User/shader transforms: scale, rotate, translate, project

\( \{x, y, z, w\}\) homogeneous clip coordinates

Units normalized such that divide by \( w \) leaves visible points between -1.0 to +1.0

OpenGL divide by \( w \)

\( \{x, y, z\}\) normalized device coordinates

Range of -1.0 to +1.0 for \( x \) and \( y \) and 0.0 to 1.0 for \( z \)

OpenGL clipping and viewport/depth-range transform

\( \{x, y\}\) are window coordinates
\( z \) is depth coordinate

\( \{x, y\}\) units are in pixels (with fractions)
\( z \) is in range of 0.0 to 1.0, or depth range

Rasterization
The default projection is orthogonal (orthographic) projection.

Most graphics systems use view normalization:
- All other views are converted to the orthographic view by distorting the objects -- normalization.
- Allows use of the same pipeline for all views.
Oblique Projections

The OpenGL projection functions cannot produce general parallel projections – the oblique projection

It seems the cube has been sheared

Oblique Projection = Shear + Orthogonal Projection
General Shear

- The far and near clipping planes are parallel to the view plane.
- The other four planes are parallel to the projection direction.

$$\tan \theta = \frac{z}{x_p - x}$$

$$x_p = x + z \cot \theta$$

$$y_p = y + z \cot \phi$$

E. Angel and D. Shreiner
Shear Matrix

xy shear (z values unchanged)

\[ H(\theta, \phi) = \begin{bmatrix} 1 & 0 & \cot \theta & 0 \\ 0 & 1 & \cot \phi & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \]

Projection matrix

\[ P = M_{\text{orth}} \ H(\theta, \phi) \]
Shear Matrix

General case:

\[ P = M_{\text{orth}} \cdot \text{STH}(\theta, \phi) \]

\[
\text{ST} = \begin{bmatrix}
2 & 0 & 0 & -\frac{\text{right} + \text{left}}{\text{right} - \text{left}} \\
\frac{2}{\text{top} - \text{bottom}} & 2 & 0 & -\frac{\text{top} + \text{bottom}}{\text{top} - \text{bottom}} \\
0 & 0 & 2 & -\frac{\text{far} + \text{near}}{\text{far} - \text{near}} \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

\[
\begin{align*}
\text{left} &= x_{\text{min}} - \text{near} \cdot \cot \theta \\
\text{right} &= x_{\text{max}} - \text{near} \cdot \cot \theta \\
\text{bottom} &= y_{\text{min}} - \text{near} \cdot \cot \phi \\
\text{top} &= y_{\text{max}} - \text{near} \cdot \cot \phi
\end{align*}
\]

\[ x_{\text{min}}, x_{\text{max}}, y_{\text{min}}, y_{\text{max}} \] are determined by intersections of the four side planes with the near plane
Effect on Clipping

The projection matrix $P = STH$ transforms the original clipping volume to the default clipping volume.
Equivalency

E. Angel and D. Shreiner
Simple Perspective

Center of projection at the origin, projection plane is orthogonal to the z-direction and is parallel to the lens

Projection plane $z = d, d < 0$

E. Angel and D. Shreiner
Perspective Equations

Consider top and side views

\[ \frac{x_p}{d} = \frac{x}{z} \quad \Rightarrow \quad x_p = \frac{x}{\frac{z}{d}} \]

\[ y_p = \frac{y}{\frac{z}{d}} \]

\[ z_p = d \]

E. Angel and D. Shreiner

Nonuniform foreshortening
Perspective Transformation

Perspective transformation is

• Not linear
• Not affine
• Not reversible
Homogeneous Coordinate Form

Consider $P_p = MP_c$ where

A point measured in the clipping frame

The corresponding point measured in the camera frame

$$P_c = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}, \quad M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \Rightarrow \quad P_p = \begin{bmatrix} x \\ y \\ z \\ z / d \end{bmatrix}$$
Perspective Division

Note that $w \neq 1$, so we must divide by $w$ to return from homogeneous coordinates.

This *perspective division* yields the desired perspective equations:

\[
\begin{align*}
    x_p &= \frac{x}{z/d} \\
    y_p &= \frac{y}{z/d} \\
    z_p &= d
\end{align*}
\]
Perspective with OpenGL

View volume is determined by the angle of view (field of view)
Consider a simple perspective with
- the COP at the origin,
- the near clipping plane at $z = -1$, and
- a 90 degree field of view determined by the planes $x = \pm z, y = \pm z$

• Perspective projection matrix is

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \text{ where } d = -1$$
Simple Perspective with OpenGL

A point \( P(x, y, z, 1) \) is projected to a new point \( Q \)

\[
Q = MP = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & -1 & 0 \\
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z \\
1 \\
\end{bmatrix}
= \begin{bmatrix}
x \\
y \\
z \\
-1 \\
\end{bmatrix}
= \begin{bmatrix}
-x/z \\
-y/z \\
-1 \\
1 \\
\end{bmatrix}
\]
Recall View Normalization

The default projection is orthogonal (orthographic) projection.

Most graphics systems use view normalization:
- All other views are converted to the orthographic view by distorting the objects -- normalization.
- Allows use of the same pipeline for all views.
Perspective Projection and Normalization

We will show the projection can be achieved by view normalization and an orthographic projection.

Consider a matrix

\[
\mathbf{N} = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & \alpha & \beta \\
0 & 0 & -1 & 0
\end{bmatrix}
\]

A point \( P = (x, y, z, 1) \) is transformed to a new point \( P' = (x', y', z', w') \) as

\[
P' = \mathbf{N}P
\]

\[
x' = x \\
y' = y \\
z' = \alpha z + \beta \\
w' = -z
\]
Perspective Projection and Normalization

After perspective division, we can have P' represented in 3D

\[ P' = (x'', y'', z'') \]

\[ x'' = -\frac{x}{z} \]
\[ y'' = -\frac{y}{z} \]
\[ z'' = -\left(\alpha + \frac{\beta}{z}\right) \]

Then, apply an orthographic projection along the z-axis, we have

\[ Q = \text{M}_{\text{orth}}P' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -\frac{x}{z} \\ -\frac{y}{z} \\ -(\alpha + \frac{\beta}{z}) \\ 1 \end{bmatrix} = \begin{bmatrix} -\frac{x}{z} \\ -\frac{y}{z} \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 0 \\ -z \end{bmatrix} \]

The result is exactly the same as performing perspective projection directly!
Picking $\alpha$ and $\beta$

What are $\alpha$ and $\beta$ for?

After applying view normalization, the new clipping volume should be transformed to the default clipping volume

- The near plane $z = -\text{near}$ needs to be mapped to $z'' = -1$
- The far plane $z = -\text{far}$ needs to be mapped to $z'' = 1$
- The sides $x = \pm z$ and $y = \pm z$ needs to be mapped to $x'' = \pm 1, y'' = \pm 1$
Normalization Transformation

Original clipping volume

- $z = -x$
- $z = -\text{near}$
- $z = -\text{far}$

Normalized clipping volume

- $z = x$
- $z = 1$

COP

original object

distorted object

projects correctly

$x = -1$

$x = 1$

$z = -1$

E. Angel and D. Shreiner
Picking $\alpha$ and $\beta$

$$z'' = -(\alpha + \beta / z)$$

$z = -\text{near}$ will transformed to

$$z'' = -(\alpha + \beta / z) = -(\alpha + \beta / (-\text{near})) = -1$$

$z = -\text{far}$ will transformed to

$$z'' = -(\alpha + \beta / z) = -(\alpha + \beta / (-\text{far})) = 1$$

$$\alpha = \frac{\text{near} + \text{far}}{\text{near} - \text{far}} \quad \text{and} \quad \beta = \frac{2\text{near} \cdot \text{far}}{\text{near} - \text{far}}$$
OpenGL Perspective

**Frustum**((left,right,bottom,top,near,far))

- Frustum can be either symmetric about the z-axis or asymmetric.
- All are measured in the *camera frame*.

For the symmetric case,

\[ \mathbf{M} = \mathbf{M}_{\text{orth}} \mathbf{N} \]
OpenGL Perspective

How do we handle the asymmetric frustum?

Convert the frustum to a symmetric one by performing a shear followed by a scaling to get the normalized perspective volume.

**Step 1 Shear:** Transform the point \((\frac{\text{left}+\text{right}}{2}, \frac{\text{top}+\text{bottom}}{2}, -\text{near})\) to \((0,0,-\text{near})\)

\[
H(\theta, \varphi) = \begin{bmatrix}
1 & 0 & \cot \theta & 0 \\
0 & 1 & \cot \varphi & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

where \(\cot \theta = \frac{\text{left}+\text{right}}{2\text{near}}\) and \(\cot \varphi = \frac{\text{top}+\text{bottom}}{2\text{near}}\)
OpenGL Perspective

After shearing, the resulting frustum is described by

\[ x = \pm \frac{\text{right - left}}{-2\text{near}} \quad y = \pm \frac{\text{top - bottom}}{-2\text{near}} \]

Near plane \[ z = -\text{near} \]

Far plane \[ z = -\text{far} \]
OpenGL Perspective

Step 2: Scaling

\[
S = \begin{bmatrix}
\frac{2\text{near}}{\text{right} - \text{left}} & 0 & 0 & 0 \\
0 & \frac{2\text{near}}{\text{top} - \text{bottom}} & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0
\end{bmatrix}
\]

Step 3: Perspective normalization \(N\)

The final perspective matrix

\[
M_p = NSH = \begin{bmatrix}
\frac{2\text{near}}{\text{right} - \text{left}} & 0 & \frac{\text{left} + \text{right}}{\text{right} - \text{left}} & 0 \\
0 & \frac{2\text{near}}{\text{top} - \text{bottom}} & \frac{\text{bottom} + \text{top}}{\text{top} - \text{bottom}} & 0 \\
0 & 0 & \frac{\text{near} + \text{far}}{\text{near} - \text{far}} & \frac{2\text{near} \times \text{far}}{\text{near} - \text{far}} \\
0 & 0 & 0 & -1
\end{bmatrix}
\]
Using Field of View: Perspective()

An alternative and more convenient way is to use the field of view

\textbf{Perspective(fovy, aspect, near, far)} often provides a better interface

- \textbf{Fovy} is the angle between the top and the bottom planes
- \textbf{aspect} = \textit{w/h} of projection plane
Using Field of View: Perspective()

Enforce a symmetric frustum

\[ left = -right \]
\[ bottom = -top \]

\[ Frustum() \iff Perspective() \]
\[ fovy = 2 \tan^{-1} \frac{top - bottom}{2\text{near}} \]
\[ left = \text{aspect} \times bottom \]
\[ top = \tan \left( \frac{fovy}{2} \right) \times \text{near} \]