Projection

- **Projection** - the transformation of points from a coordinate system in $n$ dimensions to a coordinate system in $m$ dimensions where $m < n$.

- We will be talking about projections from 3D to 2D, where the 3D space represents a world coordinate system and the 2D space is a window which is mapped to a screen viewport.

Specifying a Projection

- Two things must be specified:
  - **Projection plane** and a **center of projection**.

  **Projection plane**
  - A 2D coordinate system onto which the 3D image is to be projected. We’ll call this the VRP for view reference plane.

  **Center of projection**
  - A point in space which serves as an end point for projectors. We’ll refer to this point as the COP. It is also called a PRP for a projection reference point.

Projectors

- **Projectors** - a ray originating at the center of projection and passing through a point to be projected. Here is an example of a projection:

  ![Projectors Diagram](image)

  - Projectors
  - Object in 3 space
  - Projected Image
  - COP
  - VRP

Parallel Projection

- A simple case of a projection is if the projectors are all in parallel.

  ![Parallel Projection Diagram](image)

  - Projectors
  - Object in 3 space
  - Projected Image
  - COP
  - VRP
Direction of Projection

- We can’t specify the COP for parallel projection
  - We’ll use Direction of Project (DOP) instead

Some Trivia

- **Planar geometric projection**
  - A projection onto a planar surface (planar) using straight lines (geometric).

- **Foreshortening**
  - Varying lengths of lines due to angle of presentation and/or distance from center of projection. Applies to both parallel and perspective projections.

Orthographic Projections

- **Orthographic projection**
  - parallel projection with the direction of projection and the projection plane normal parallel.

- **Elevation**
  - an orthographic projection in which the view plane normal is parallel to an axis.

- The three elevations
  - front-elevation
  - top-elevation (plan-elevation)
  - side-elevation.

Axonometric orthographic projections

- **Axonometric orthographic projections**
  - Use projection planes which are not normal to an axis. They show more than one face of an object at a time. They induce uniform foreshortening unrelated to depth.

- AOP preserves parallelism of lines. It does not preserve angles.
**Isometric projection**

- Axonometric orthographic projection where the projection plane normal (and the direction of projection) makes identical angles with each principle axis. How many of these are there?

![Isometric Projection Diagram](image)

**Oblique Projection**

- The projection plane normal and the direction of projection are at angles to each other.

![Oblique Projection Diagram](image)

**Cavalier Projection**

- An Oblique projection
  - DOP is at 45 degree angle to VPN
  - Lines parallel to any axis are foreshortened equally. Lines parallel to the z axis appear at an angle $\alpha$, which is dependent upon the direction of projection.
  - Two common projections have $\alpha$ as 45° and 30°.

![Cavalier Projection Diagram](image)

**Cavalier Projection Angles**

- Why?

![Cavalier Projection Angles Diagram](image)
Cabinet projection

- Oblique projection
  - projection plane normal is at an arctan(2) = 63.4° degree angle to the projection plane. (typically projecting onto the x, y plane)
  - Lines parallel to the axis defining the projection plane are foreshortened equally. *Lines parallel to the projection plane normal are halved!*

Parallel Projection

- After transforming the object to the eye space, parallel projection is relatively easy – we could just drop the Z
  - \( X_p = x \)
  - \( Y_p = y \)
  - \( Z_p = -d \)

- We actually want to keep \( Z \) – why?

Parallel Projection (2)

- OpenGL maps (projects) everything in the visible volume into a **canonical view volume**
  - \( g1Ortho(xmin, xmax, ymin, ymax, -near) \)
  - (\( xmin, ymin, -near \))
  - (\( xmax, ymax, -far \))
  - (\( -1, -1, 1 \))
  - (\( 1, 1, -1 \))
  - **Canonical View Volume**
Parallel Projection (3)

Transformation sequence:
1. Translation (M1): (-near = zmax, -far = zmin)
   - (xmax+xmin)/2, -(ymax+ymin)/2, -(zmax+zmin)/2
2. Scaling (M2):
   2/(xmax-xmin), 2/(ymax-ymin), 2/(zmax-zmin)

\[
M_2 \times M_1 = \begin{bmatrix}
\frac{2}{(x_{\text{max}}-x_{\text{min}})} & 0 & 0 & \frac{-(x_{\text{max}}+x_{\text{min}})}{(x_{\text{max}}-x_{\text{min}})} \\
0 & \frac{2}{(y_{\text{max}}-y_{\text{min}})} & 0 & \frac{-(y_{\text{max}}+y_{\text{min}})}{(y_{\text{max}}-y_{\text{min}})} \\
0 & 0 & \frac{2}{(z_{\text{max}}-z_{\text{min}})} & \frac{-(z_{\text{max}}+z_{\text{min}})}{(z_{\text{max}}-z_{\text{min}})} \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

Perspective Projection

Perspective projections have projectors at angles to each other radiating from a center of projection.
- Parallel lines not parallel to the projection plane will not appear parallel in the projection.

Vanishing Points

If not parallel?
- If the lines are not parallel anymore, they must meet somewhere. In 3D space that point will be at infinity and is referred to as a vanishing point. There are an infinite number of vanishing points.
- **Axis vanishing points**
  - Lines parallel to one of the major axis come to a vanishing point, these are called axis vanishing points. Only three axis vanishing points in 3D space.

Center of Projection in OpenGL

- OpenGL always puts the center of projection at 0,0,0
  - The projection plane is at z = -d
  - This is sometimes called the “focal length” or “f”
Frustums

- The region we can see is called the frustum

\[ \text{glFrustum}(\text{left}, \text{right}, \text{bottom}, \text{top}, \text{znear}, \text{zfar}) \]

- \( \text{znear} \) and \( \text{zfar} \) are positive

 gluPerspective

- How do we get from:

\[ \text{gluPerspective}(\text{fovy}, \text{aspect}, \text{znear}, \text{zfar}) \]

- To

\[ \text{glFrustum}(\text{left}, \text{right}, \text{bottom}, \text{top}, \text{znear}, \text{zfar}) \]

fov to near frustum

\[ y = \text{znear} \times \tan\left(\frac{\text{fovy}}{2}\right) \]

\[ x = ? \]

Projection Structure

- Pinhole Camera Model of Projection

- Proportional!

\[ \frac{x'}{-d} = \frac{x}{z}, \quad \frac{y'}{-d} = \frac{y}{z} \]

\[ x' = \frac{-dx}{z}, \quad y' = \frac{-dy}{z} \]
Matrix for Perspective Projection?

- We need division to do projection!
- But, matrix multiplication only does multiplication and addition
- What about:

\[
M_{\text{per}} = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & -1/d & 0
\end{bmatrix}
\]

\[
M_{\text{per}}P = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & -1/d & 0
\end{bmatrix} \begin{bmatrix}
x \\
y \\
z \\
d
\end{bmatrix} = \begin{bmatrix}
x/d \\
y/d \\
z/d \\
-1/d
\end{bmatrix}
\]

Homogenous Coordinates (again)

- A 3D homogeneous coordinate: \((x, y, z, w)\)
- We had been saying that \(w\) is 1
- But –
  - \((x, y, z, w)\) corresponds to \((x/w, y/w, z/w)\)
  - Dividing by \(w\) is called **homogenizing**
  - If \(w=1\), \(x,y,z\) are unchanged.
  - But, if \(w=-z/d\)?
    - \((x/(-z/d), y/(-z/d), z/(-z/d)) = (-dx/z, -dy/z, -d)\)

\[
x' = -dx/z, \quad y' = -dy/z, \quad -d
\]

The Entire Viewing Process

- Rotate world so that the COP is at 0,0,0 and DOP is parallel to the Z axis
- Apply perspective projection
- Homogenize
- Viewport transformation

Viewport Transformation (Window to Viewport)

- **Window**
  - Area of the projection plane
  - Typically some normalized area with 0,0 in the center
- **Viewport**
  - Area of the computer display window
  - Example:
    - \((0, 0)\) to \((640, 480)\)
Window to Viewport Example

- Assume Window (-1,-1) to (1,1)
  - OpenGL calls these normalized device coordinates
- Viewport (0, 0) to (640, 480)
  - OpenGL calls these window coordinates

\[
x_w = (x_{nd} + 1) \left( \frac{\text{width}}{2} \right) + x_{\text{lowerleft}}
\]

\[
y_w = (y_{nd} + 1) \left( \frac{\text{height}}{2} \right) + y_{\text{lowerleft}}
\]

Perspective Projection (6)

- Final Projection Matrix:

\[
x' = \frac{2N}{(xmax-xmin)} x - \frac{2N}{(xmax-xmin)} (xmax+xmin) / (xmax-xmin)
\]

\[
y' = \frac{2N}{(ymax-ymin)} y - \frac{2N}{(ymax-ymin)} (ymax+ymin) / (ymax-ymin)
\]

\[
z' = \frac{-F + N}{F-N} z - \frac{2F+N}{F-N}
\]

\[
w' = 1
\]

\[
glFrustum(xmin, xmax, ymin, ymax, N, F) \quad N = \text{near plane, } F = \text{far plane}
\]

Within OpenGL

```
glBegin(GL_POLYGON);
glVertex3dv(a);
glVertex3dv(b);
glVertex3dv(c);
glEnd;
```