CS 174 Week 4

Aidan Jan

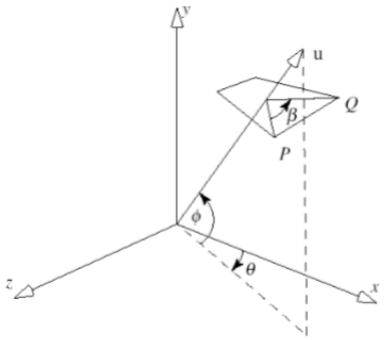
April 28, 2023

Rotation Around an Arbitrary Axis

- Vector (axis) = $u = [u_X, u_Y, u_Z]^T$
- Rotation angle: β
- Point: P

Method:

1. Two rotations to align u with the x-axis
2. Do x-roll by β
3. Undo the alignment



Similar to translating the shape to the origin then rotating it, this moves the axis (and the shape) to the x-axis and rotates it around the x-axis.

Change of Basis

Changing a basis is essentially creating a new coordinate grid. For now, we look at changing the basis with a **translation** - the new basis has axes parallel to the previous ones, just the origin is at a different place. In this case, if the new basis has an origin at (x_p, y_p) on the old basis, then some point (x, y) on the old basis would have coordinates (x', y') in the new basis where $x' = x - x_p$ and $y- = y - y_p$. This is basically the same as doing a translation.

Basis: $T(x_p, y_p)$

$$T^{-1} = T(-x_p, -y_p)P$$

T is a translation and T^{-1} is its inverse.

What if this time we want to **rotate** the basis? Consider a point in the original basis is (x, y), and the new basis was rotated relative to the original axis by an angle θ. In this case, if the basis is $R(\theta)$, then $R^{-1} = R(-\theta)P$.

$$R(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} i_x & j_x & 0 \\ i_y & j_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

i_x, j_x, i_y, and j_y are the x and y components compared to the original basis of the new basis defined by vectors i and j.
The rightmost matrix above is also known as **General Rotation Matrix** or GRM. GRM = $R(\theta)$.

If now we want to both translate **and** rotate, we simply orient the basis with the original basis, then translate it. Basis: $T(x_p, y_p)R(\theta)$. To move a point, $[TR]^{-1} = R^{-1}T^{-1}$.

Extending to 3D Space

If we have a coordinate system in three dimensions instead of two, then we have to make some changes to the change in basis. For the translation part, we simply add an extra vector to represent the z-direction. For the rotation part, instead of using $R(\theta)$, we use the General Rotation Matrix in three dimensions. Thus, the transformation becomes:

$$[TR]^{-1} = \text{GRM}^{-1}T^{-1}P$$

Introduction to Eye Space

Eye Space is essentially what a camera would see in a 3-D world. This is essentially a change of basis from the **World Space**, the 3-D space all the objects are located in.

Flipping the X-Axis

Note that in the World Space, the coordinate system used follows the right hand rule. Moving right increases the x-value, moving up increases the y-value, and moving deeper increases z-value. However, if we try to do this on a screen, moving down the y-axis increases y-value instead of up, which means we are using the left-hand coordinate system instead of the right-hand coordinate system. To correct this, during the transformation to the Eye Space, we flip the x-axis, or we are viewing a "mirrored" image.

Implementation of Eye Space

We define the **Eye Vector** using the position of the Eye, or camera, and the Point of Interest - the point that is center of the camera view.

E = COI - Eye

The Eye Vector is the vector that passes through the eye position and the center of interest. However, having one vector like this is not enough to define the camera, because with only one vector, the camera can still freely spin around it. Thus, we use another vector, called the **Top Vector** that defines which direction "up" is.

$$\text{T} = \text{Top - Eye}$$

The Top is a vector that passes through the top of the frame in view and parallel to the eye vector.

The **Eye Matrix** is the matrix that is multiplied to the objects in the World Space to convert them to the Eye Space.

$$\text{EM} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot [\text{GRM}]^{-1} \cdot T \cdot (-\text{Eye})$$

The first matrix is to flip the x-axis.

Composite 3D Rotation About the Origin

$$R(\theta_1, \theta_2, \theta_3) = R_z(\theta_3) R_y(\theta_2) R_x(\theta_1)$$

- This is know as the "Euler angle" representation of 3D rotations
- The order of the rotation matrices is important!!
- Note: The Euler angle representation suffers from singularities

Gimbal Lock

$$\begin{aligned} \mathbf{R}(\theta_1, \theta_2, \theta_3) &= \mathbf{R}_z(\theta_3)\mathbf{R}_y(\theta_2)\mathbf{R}_x(\theta_1) \\ &= \begin{bmatrix} \cos\theta_3 & -\sin\theta_3 & 0 & 0 \\ \sin\theta_3 & \cos\theta_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta_2 & 0 & \sin\theta_2 & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta_2 & 0 & \cos\theta_2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta_1 & -\sin\theta_1 & 0 \\ 0 & \sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

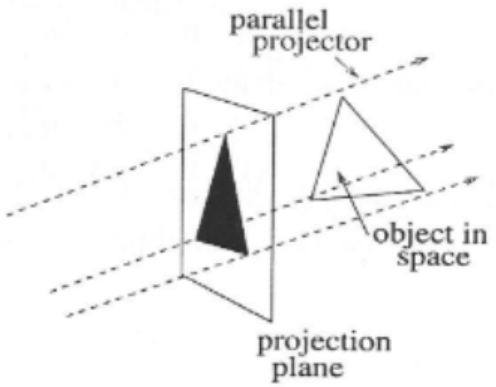
What happens when the middle angle is 90°?

$$\begin{aligned} \mathbf{R}(\theta_1, 90^\circ, \theta_3) &= \mathbf{R}_z(\theta_3)\mathbf{R}_y(90^\circ)\mathbf{R}_x(\theta_1) \\ &= \begin{bmatrix} \cos\theta_3 & -\sin\theta_3 & 0 & 0 \\ \sin\theta_3 & \cos\theta_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta_1 & -\sin\theta_1 & 0 \\ 0 & \sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} \cos\theta_3 & -\sin\theta_3 & 0 & 0 \\ \sin\theta_3 & \cos\theta_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & \sin\theta_1 & \cos\theta_1 & 0 \\ 0 & \cos\theta_1 & -\sin\theta_1 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 0 & \cos\theta_3\sin\theta_1 - \sin\theta_3\cos\theta_1 & \cos\theta_3\cos\theta_1 + \sin\theta_3\sin\theta_1 & 0 \\ 0 & \cos\theta_3\cos\theta_1 + \sin\theta_3\sin\theta_1 & -\cos\theta_3\sin\theta_1 + \sin\theta_3\cos\theta_1 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

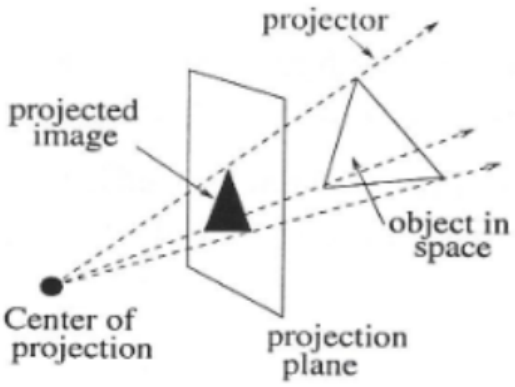
When one of the angles is 90°, we lose a degree of freedom.

Projections

There are two types of orthographic projections.

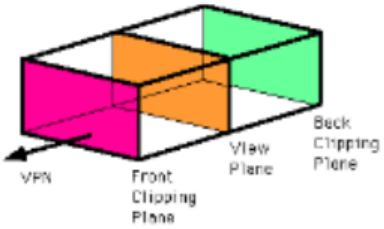


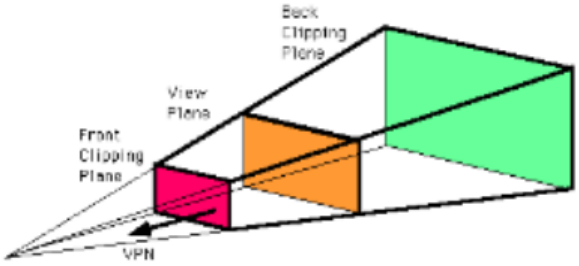
Parallel projection



Perspective projection

The two projections create different **view volumes**.





The parallel projections create a view volume shaped as a prism; the depth does not change the size of the plane. However, with the perspective projection, the frame grows larger the further they are away from the eye.

Parallel Projections

The parallel PM (projection matrix) is simply the 4x4 identity matrix, since shapes are not distorted in a parallel projection. The **canonical view volume** is essentially a box in $-1 < x, y < 1, 0 < z < 1$. The depth maps to the z-axis, and the height and width maps to the y and x-axes, respectively. In a canonical view volume, the eye is located at the origin and facing the positive z-axis. Thus, in a typical view volume,

the bounds for X, Y, and Z are as follows:

$$\begin{aligned} -\frac{W}{2} \leq & X \leq \frac{W}{2} \\ -\frac{H}{2} \leq & Y \leq \frac{H}{2} \\ N \leq & Z \leq F \end{aligned}$$

N and F refer to the near and far bounds of Z, respectively.

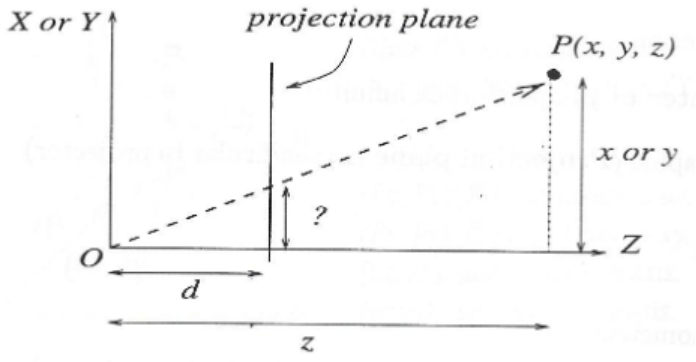
To match the canonical view volume, where X and Y must range from -1 to 1, and Z must range from 0 to 1, we scale the parallel PM matrix.

$$\text{Normalized Parallel PM} = \begin{bmatrix} \frac{2}{W} & 0 & 0 & 0 \\ 0 & \frac{2}{H} & 0 & 0 \\ 0 & 0 & \frac{1}{F-N} & -\frac{N}{F-N} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Canonical View Volume:

$$\begin{aligned} -1 \leq & X' \leq 1 \\ -1 \leq & Y' \leq 1 \\ 0 \leq & Z' \leq 1 \end{aligned}$$

Perspective Projection



In perspective projection, the camera is defined by not only the camera position and center of focus, but also an angle referred to as the **Angle of View**.

Angle of view represents the angle from the center of the pyramid to one of its triangular faces. Thus, the full angle of view that represents the angle between the two edges is actually $2 \cdot \text{AOV}$.

If the view frame (eye space) is distance z away from the camera and has dimensions x, y, then the projection plane with angle of view θ would be distance d from the camera. θ always represents the angle on the x-axis.

The angle on the y-axis is instead calculated by multiplying x by A_r, the aspect ratio. $A_r = \frac{x}{y}$ To solve for the new height and width, we can use trigonometry. It turns out that the new width would be $\frac{x}{z\tan(\theta)}$ and the new height would be $\frac{y \cdot A_r}{z\tan(\theta)}$.

To convert the eye space to the projection (window), we multiply by the following perspective projection matrix (PPM):

$$\text{Normalized PPM} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & A_r & 0 & 0 \\ 0 & 0 & A\tan(\theta) & B\tan(\theta) \\ 0 & 0 & \tan(\theta) & 0 \end{bmatrix}$$

In this matrix, $A = \frac{F}{F-N}$ and $B = -\frac{N \cdot F}{F-N}$.

What if our aspect ratio is 1? This would represent a square frame.

In this case, $A \cdot \tan(\theta)$ and $B \cdot \tan(\theta)$ simplify to 1 and 0, respectively. Thus, the Normalized Parallel PM, assuming the viewport is square, would be:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{d} & 0 \end{bmatrix}$$

Note that the window is always a 2x2 unit area, where $-1 \leq x, y \leq 1$. Thus, objects may be distorted.

Screen Space

The final step of the graphics pipeline is to render it on the screen. This is easy; it is a simple scale from the normalized projection window.

In general, the window uses coordinates $x_{min}, y_{min}, x_{max}, y_{max}$ to define the bounds. The viewport uses $u_{min}, v_{min}, u_{max}, v_{max}$ to define the bounds. Thus, the window to viewport matrix is as follows:

$$\text{WTV Matrix} = T(u_{min}, v_{min}) \cdot S\left(\frac{u_{max} - u_{min}}{x_{max} - x_{min}}, \frac{v_{max} - v_{min}}{y_{max} - y_{min}}\right) \cdot T(-x_{min}, -y_{min})$$

where T represents translations and S represents a scaling. Since the scaling is different in the x and y directions and defined by the dimensions, this fixes the distortion in normalizing the window (assuming the projection space has the same aspect ratio as the screen space). Most programs show different bounds (the black bars at the sides of the screen) to correct for possible distortion caused by screen dimensions.