# Linear Algebra in 3D Graphics Programming

**Zhou Yihua, Bai Yuyang & Wang Yuhao**

Shanghai Jiao Tong University, UM-SJTU Joint Institute

**Abstract**

In this project, we will study how to apply OpenGL to 3D graphics programming. The project will mainly include three parts:

1. The transformation of 3D graphics.
2. Producing 2D projection using 3D graphics.
3. Realizing the functions using OpenGL.

## Introduction

In class, we've studied linear transformation and its application in 2D graphics processing. However, nowadays 3D graphics is the mainstream in computer graphics, and is widely used in many industries such as game developing, industrial designing and medical industry. Therefore, it is very important to extend 2D graphics to 3D graphics.

OpenGL is a fundamental and basic Application Programming Interface in both 2D and 3D graphics programming. So, for beginner programmers like us, learning OpenGL is useful to understand the principle and learn basic skills. Therefore, we decide to study the using of linear algebra in OpenGL.

## Rotation

- **Rotation Around the Z Axis**

  *We will lay more emphasis on rotation transformation than the former parts. Most basically, we consider rotation around the Z axis.*

  For this case, the X-Y plane stay in the X-Y plane. The rotation factor for z is just 1.

$$R_z = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

  *Note: $\theta$ is in radians, and the positive angle results in a counterclockwise rotation.*

- **Rotation Around Any Axis**

  This is much more complex than the above one. We consider the arbitrary axis is defined by two points $P_1 = (x_1, y_1, z_1)$ and $P_2 = (x_2, y_2, z_2)$. The transformation process can be divided into five steps below:

1. *Translate the space so that let the axis pass through the origin O.* Translate space by $P_1 = (x_1, y_1, z_1)$.

$$\mathbf{T} = \begin{pmatrix} 1 & 0 & 0 & -\mathrm{x}_1 \\ 0 & 1 & 0 & -\mathrm{y}_1 \\ 0 & 0 & 1 & -\mathrm{z}_1 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad \mathbf{T}^{-1} = \begin{pmatrix} 1 & 0 & 0 & \mathrm{x}_1 \\ 0 & 1 & 0 & \mathrm{y}_1 \\ 0 & 0 & 1 & \mathrm{z}_1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

2. *Rotate the space about the X axis so that the rotation axis lies in the X-Z plane.* Let $u = (a, b, c)$ be the unit vector along the rotation axis. If $b = c = 0$, then it can be interpreted as a rotation about the $x$ axis.

$$\cos(t) = c/d \ \ \& \ \ \sin(t) = b/d$$

   Get the rotation matrix and its inverse:

$$\mathbf{R}_\mathrm{X} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \mathrm{c/d} & -\mathrm{b/d} & 0 \\ 0 & \mathrm{b/d} & \mathrm{c/d} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad \mathbf{R}_\mathbf{x}^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \mathrm{c/d} & \mathrm{b/d} & 0 \\ 0 & -\mathrm{b/d} & \mathrm{c/d} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

3. *Rotate the space about the Y axis so that the rotation axis lies along the Z axis.* Similar to Step 2, get the rotation matrix and its inverse

$$\mathbf{R}_y = \begin{pmatrix} d & 0 & -a & 0 \\ 0 & 1 & 0 & 0 \\ a & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad \mathbf{R}_y^{-1} = \begin{pmatrix} d & 0 & a & 0 \\ 0 & 1 & 0 & 0 \\ -a & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

4. *Rotation about the z axis*

$$\mathbf{R}_\mathrm{z} = \begin{pmatrix} \cos(t) & -\sin(t) & 0 & 0 \\ \sin(t) & \cos(t) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

5. *Finally, we can get the rotation transformation*

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \mathrm{T}^{-1}\mathrm{R}_\mathrm{X}^{-1}\mathrm{R}_\mathrm{y}^{-1}\mathrm{R}_\mathrm{z}\mathrm{R}_\mathrm{y}\mathrm{R}_\mathrm{X}\mathrm{T} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \tag{1}$$

### 3D Perspective Projection

Our aim is to program 3D graphics to produce 2D pictures. And for this, we can think of the portion of the world that is visible through a virtual camera as a pyramid with the top chopped off.

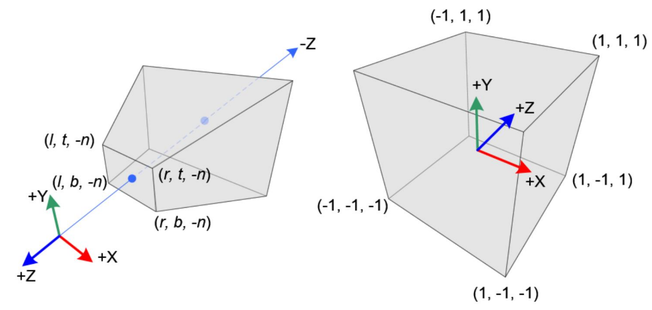


**Figure 1:** Perspective Frustum and Normalized Device Coordinates (NDC)

We first consider a 3D point in eye space is projected onto the near plane (projection plane).

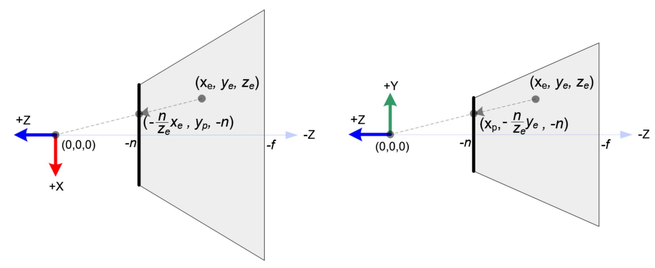


**Figure 2:** Top & side views of frustum.

$$x_p = \frac{-n \cdot x_e}{z_e} = \frac{n \cdot x_e}{-z_e} \ \ \& \ \ y_p = \frac{-n \cdot y_e}{z_e} = \frac{n \cdot y_e}{-z_e}$$

Then we need the two coordinates, clip coordinates and normalized device coordinates. Here the clip coordinates is a homogeneouus coordinates.

$$\begin{pmatrix} x_{clip} \\ y_{clip} \\ z_{clip} \\ w_{clip} \end{pmatrix} = M_{projection} \cdot \begin{pmatrix} x_{eye} \\ y_{eye} \\ z_{eye} \\ w_{eye} \end{pmatrix}$$

Next, we map $x_p$ and $y_p$ to $x_n$ and $y_n$ of NDC with linear relationship; $[l, r]$ to $[1, 1]$ and $[b, t]$ to $[-1, 1]$.

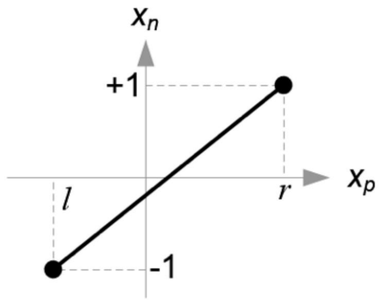


**Figure 3:** Mapping from $x_p$ to $x_n$.

$$x_n = \frac{1-(-1)}{r-l} \cdot x_p + \beta$$

Substitute $(r, 1)$ for $(x_p, x_n)$, and $x_p = \frac{nx_e}{-z_e}$, we get:

$$\beta = -\frac{r+l}{r-l}$$

$$x_n = \frac{2x_p}{r-l} - \frac{r+l}{r-l} = \left(\frac{2n}{r-l} \cdot x_e + \frac{r+l}{r-l} \cdot z_e\right) / -z_e$$

$$y_n = \frac{2y_p}{t-b} - \frac{t+b}{t-b} = \left(\frac{2n}{t-b} \cdot y_e + \frac{t+b}{t-b} \cdot z_e\right) / -z_e$$

With these equations, we can find the 1st and 2nd rows for the projection matrix because of its homogeneity.

$$\begin{pmatrix} x_c \\ y_c \\ z_c \\ w_c \end{pmatrix} = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & A & B \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} x_e \\ y_e \\ z_e \\ w_e \end{pmatrix}$$

In eye space, $w_e$ equals to 1. Therefore, the equation becomes: $z_n = \frac{Az_e+B}{-z_e}$

Here $w_e = 1$. For the farther plane shown in OpenGL is $z_n = 1$ and the near plane is $z_n = 1$. So, we apply $(n, 1)$ and $(f, 1)$ for the $(z_e, z_n)$. We can get the projection matrix:

$$\begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix} \tag{2}$$

## OpenGL Implimentation

In this part of OpenGL implimentation, we first plot one 3D graph, then use the transformation **(1)** to achieve the roatation transformation and use matrix **(2)** to achieve the projection.

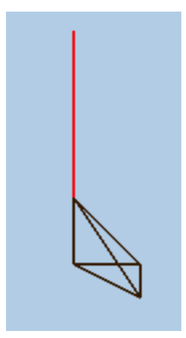

**Figure 4:** Original image.

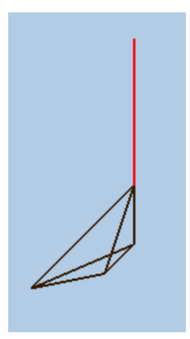

**Figure 5:** Rotation ransformation image.

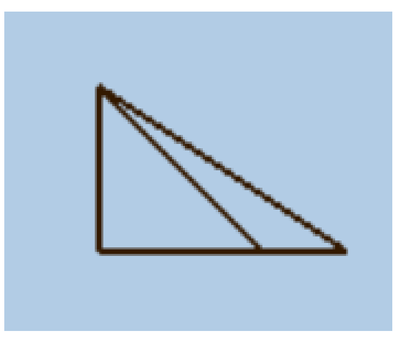

**Figure 6:** Projection image.

In the first two images, the red line symbolizes the ratation axis of this tetrahedral, and the last image shows the projection of this tetrahedral onto the reference plane.

## References

1. *"OpenGL Projection Matrix"*, Song Ho Ahn. `http://www.songho.ca/opengl/gl_matrix.html`
2. *Linear Algebra for Graphics Programming*. `http://metalbyexample.com/linear-algebra/`