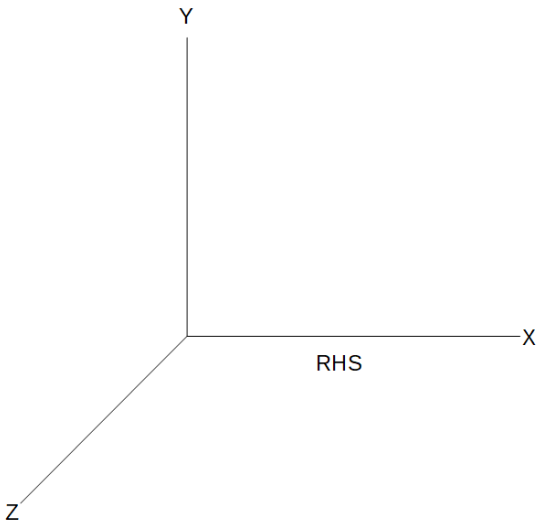
Introduction to 3D

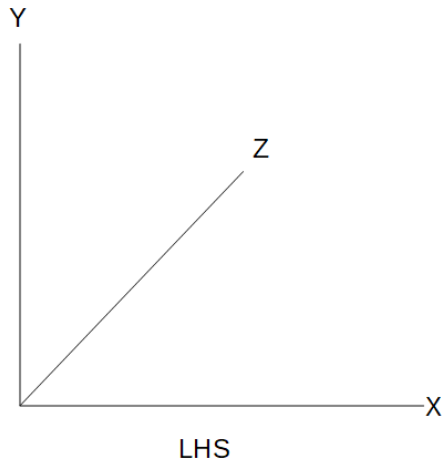
Coordinate Systems

Two primary coordinate systems used by 3D systems; and yes, both are used (unfortunately).

Right Handed System (RHS)



Left Handed System (LHS)



Definition of a Point

$$p=\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Just as we learned with 2D, going to use homogeneous coordinates

Operations in 3D

Translation

$$T=\begin{bmatrix}1 & 0 & 0 & dx\\ 0 & 1 & 0 & dy\\ 0 & 0 & 1 & dz\\ 0 & 0 & 0 & 1\end{bmatrix} \qquad P'=T\cdot P \qquad \begin{bmatrix}x_p\\ y_p\\ z_p\\ 1\end{bmatrix}=\begin{bmatrix}1 & 0 & 0 & dx\\ 0 & 1 & 0 & dy\\ 0 & 0 & 1 & dz\\ 0 & 0 & 0 & 1\end{bmatrix}.\begin{bmatrix}x\\ y\\ z\\ 1\end{bmatrix}$$

Scaling

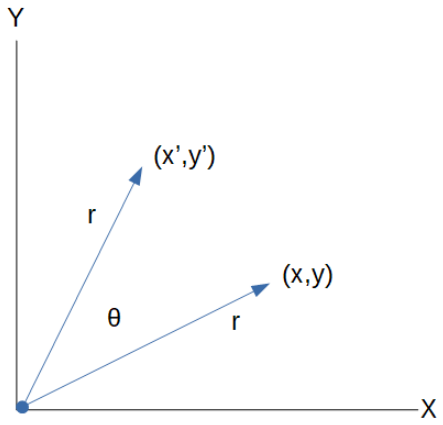
$$T=\begin{bmatrix}S_x & 0 & 0 & 0\\ 0 & S_y & 0 & 0\\ 0 & 0 & S_z & 0\\ 0 & 0 & 0 & 1\end{bmatrix} \qquad P'=S\cdot P \qquad \begin{bmatrix}x_p\\ y_p\\ z_p\\ 1\end{bmatrix}=\begin{bmatrix}S_x & 0 & 0 & 0\\ 0 & S_y & 0 & 0\\ 0 & 0 & S_z & 0\\ 0 & 0 & 0 & 1\end{bmatrix}.\begin{bmatrix}x\\ y\\ z\\ 1\end{bmatrix}$$

Rotation

We’ll start with rotation in the plane, which gives us three rotations:

- About z-axis : x-y plane
- About x-axis : y-z plane
- About y-axis : x-z plane

Z-Axis Rotation (x-y plane)



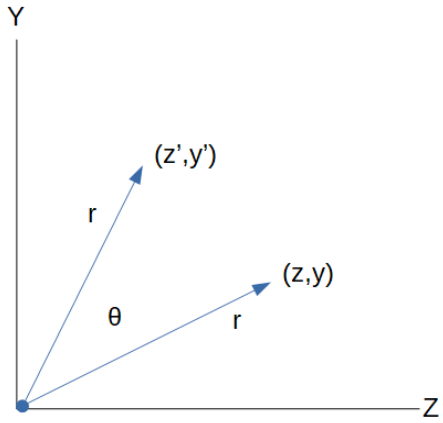
$$x' = x\cos(\Theta) - y\sin(\Theta)$$

$$y' = x\sin(\Theta) + y\cos(\Theta)$$

$$z' = z$$

$$R_z = \begin{bmatrix} \cos(\Theta) & -\sin(\Theta) & 0 & 0 \\ \sin(\Theta) & \cos(\Theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

X-Axis Rotation (y-z plane)



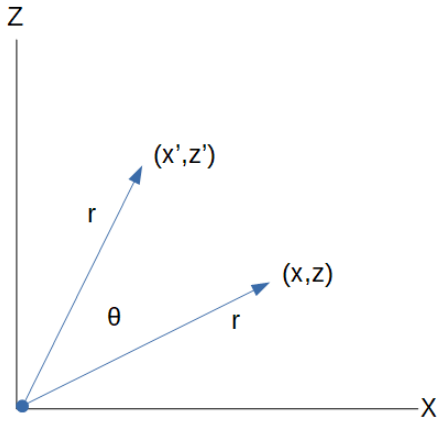
$$x' = x$$

$$y' = y\cos(\Theta) - z\sin(\Theta)$$

$$z' = y\sin(\Theta) + z\cos(\Theta)$$

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\Theta) & -\sin(\Theta) & 0 \\ 0 & \sin(\Theta) & \cos(\Theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Y-Axis Rotation (x-z plane)



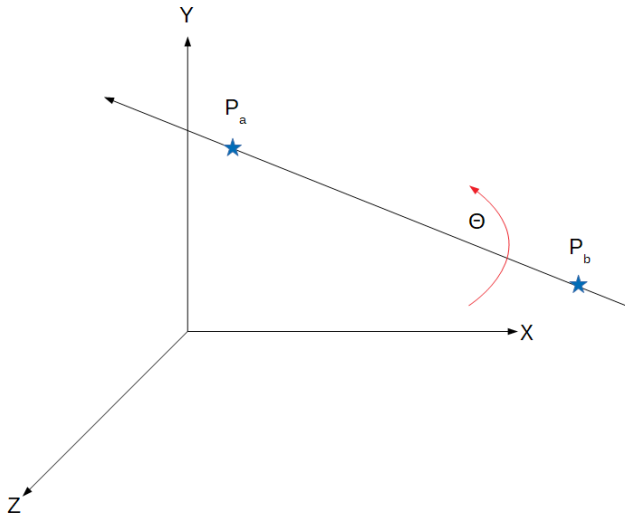
$$x' = x\cos(\Theta) + z\sin(\Theta)$$

$$y' = y$$

$$z' = -x\sin(\Theta) + z\cos(\Theta)$$

$$R_y = \begin{bmatrix} \cos(\Theta) & 0 & \sin(\Theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\Theta) & 0 & \cos(\Theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Arbitrary 3D Rotation

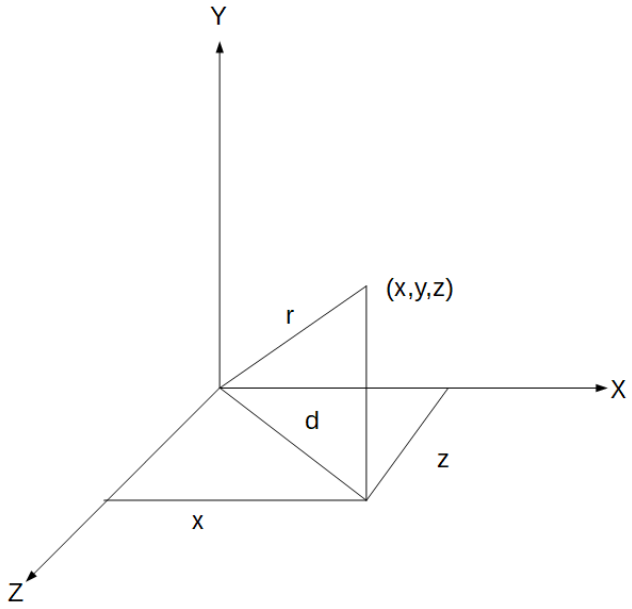


Line defined by: $P_a \rightarrow P_b$

1. Translate P_a to the origin, T_1
2. Rotate line into a coordinate axis (two rotations), R_α, R_β
3. Rotate by the specified Θ about the coordinate axis
4. Rotate back by negative R_α, R_β
5. Translate back by $-T_1$

$$P' = -T_1 \cdot R_{-\alpha} \cdot R_{-\beta} \cdot R_\Theta \cdot R_\beta \cdot R_\alpha \cdot T_1 \cdot P$$

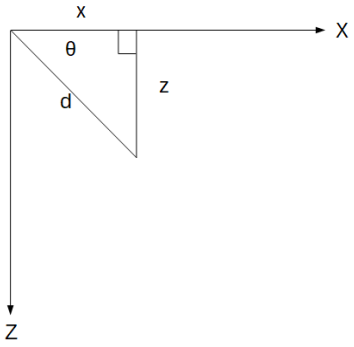
Beginning with Step 2, let's look at the setup:



$$r=\sqrt{(x^2+y^2+z^2)}$$

$$d=\sqrt{(x^2+z^2)}$$

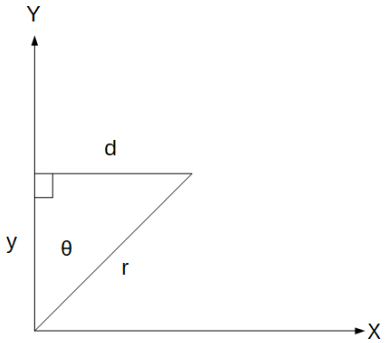
R_α is rotation into the x-y plane, about the y-axis.



$$\sin(\Theta)=\frac{z}{d} \qquad \cos(\Theta)=\frac{x}{d}$$

$$R_\alpha=\begin{bmatrix} x/d & 0 & z/d & 0 \\ 0 & 1 & 0 & 0 \\ -z/d & 0 & x/d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

R_β is rotation into the y-axis, about the z-axis.



$$\sin(\Theta)=\frac{d}{r} \qquad \cos(\Theta)=\frac{y}{r}$$

$$R_\beta=\begin{bmatrix} y/r & -d/r & 0 & 0 \\ d/r & y/r & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

R_Θ is in the y-z plane, about the x-axis.

$$R_\Theta=\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\Theta) & -\sin(\Theta) & 0 \\ 0 & \sin(\Theta) & \cos(\Theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$R_{-\beta}$ is rotation away the y-axis, about the z-axis.

Note that: $\sin(\Theta) = -\sin(-\Theta)$ and $\cos(\Theta) = \cos(-\Theta)$

Therefore

$$R_{-\beta} = \begin{bmatrix} y/r & d/r & 0 & 0 \\ -d/r & y/r & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$R_{-\alpha}$ is rotation away from the x-y plane, about the y-axis.

$$R_{-\alpha} = \begin{bmatrix} x/d & 0 & -z/d & 0 \\ 0 & 1 & 0 & 0 \\ z/d & 0 & x/d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Projection

Projection is the process of taking a snapshot of a scene and “projecting” it onto a viewing surface. There are two kinds of 3D projection we’ll examine:

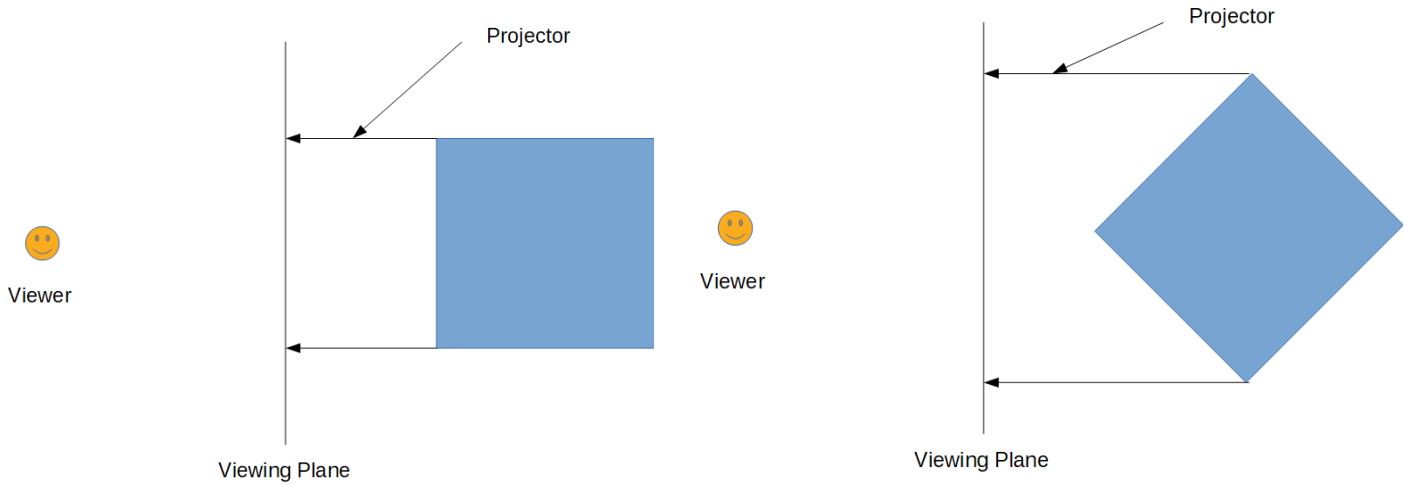
- Parallel
- Perspective

There are three components to projection, whether parallel or perspective.

1. Specify a 3D viewing volume (application); also knowing as a ***frustum***.
2. Clip objects to the viewing volume (usually graphics API); frustum culling.
3. Project onto a viewing plane (application)

Parallel Projection

Also known as Orthographic (orthogonal) projection. The projection (or projectors) are perpendicular to the view plane.

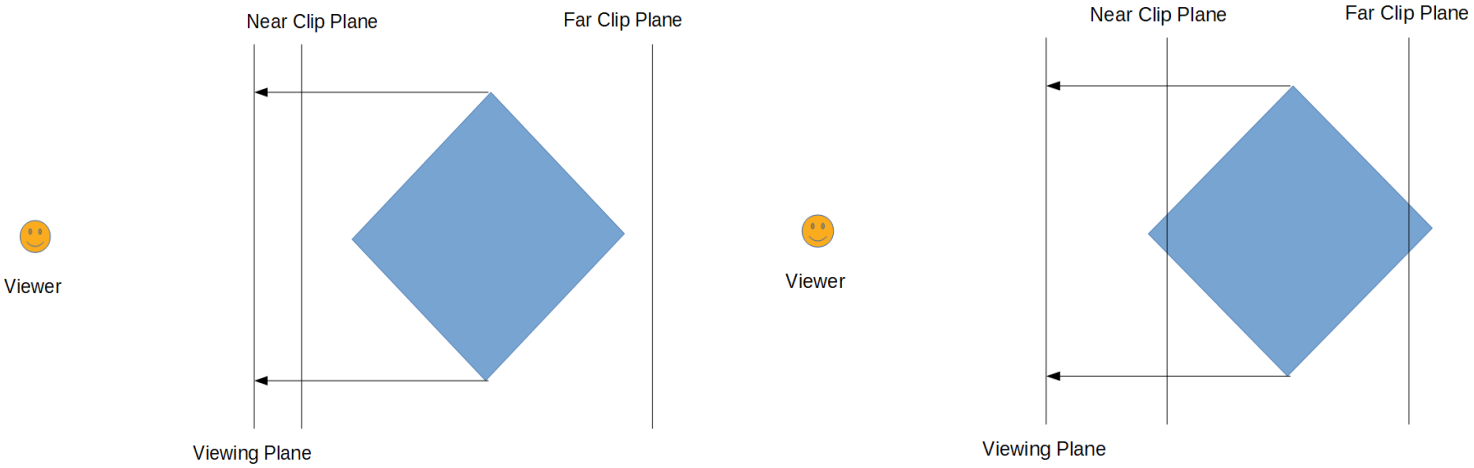


In simple terms, the z value is set to 0. In matrix form, looks like this.

$$M_{par} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

That is a good start, but it actually isn’t enough. We need to think about a whole viewing volume and transform from world coordinates into the viewing volume.

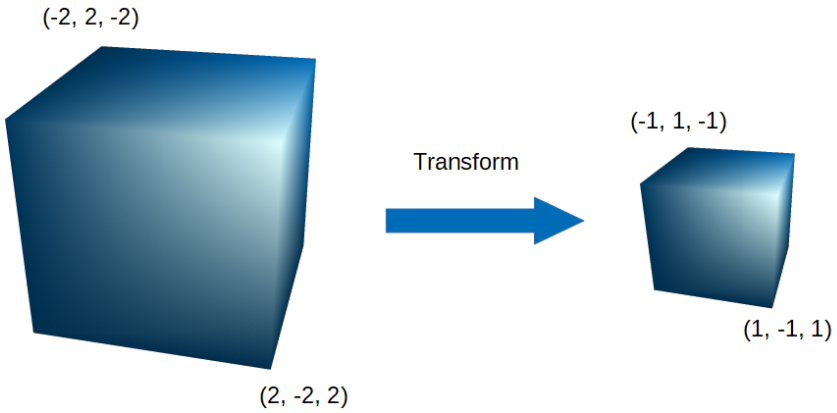
Let’s start with what are known as clipping planes.



Why in the world would we want to do this?

- A near clipping plane is used to remove objects that are close or in front of the camera. In addition to performance gained by this, it helps ensure vertices have valid 2D coordinates.
- A far clipping plane is used to remove objects that are too distant to make an impact on the scene.
- The combination of a near and clipping plane allows the resolution of the depth buffer to be controlled. For example, if the depth buffer is a 16 bit float, the precision must be evenly divided between the values of the near and far plane. A smaller numerical distance between the near and far clipping plane, means higher resolution depth granularity.

The next thing we need to do is to convert from a viewing volume (in world coordinates) to a canonical viewing volume (in our case, WebGL defined a unit viewing volume). It looks like this:



Combined, there are two transformations we must perform. The first is to move the center of the viewing volume to the center of the canonical viewing volume. The second is to scale the viewing volume to the canonical viewing volume. Remember, the center of the viewing volume might not be (0, 0, 0), and the shape of the viewing volume might not be a cube.

Moving the center of the viewing volume to the center of the canonical viewing volume is this:

$$T=\begin{bmatrix} 1 & 0 & 0 & -\left(\frac{\text{left}+\text{right}}{\text{right}-\text{left}}\right) \\ 0 & 1 & 0 & -\left(\frac{\text{top}+\text{bottom}}{\text{top}-\text{bottom}}\right) \\ 0 & 0 & 1 & -\left(\frac{\text{far}+\text{near}}{\text{far}-\text{near}}\right) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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

Multiplying these two together gives us our parallel projection matrix.

$$Projection=S\cdot T$$

$$Projection=\begin{bmatrix} \frac{2}{\text{right}-\text{left}} & 0 & 0 & -\left(\frac{\text{left}+\text{right}}{\text{right}-\text{left}}\right) \\ 0 & \frac{2}{\text{top}-\text{bottom}} & 0 & -\left(\frac{\text{top}+\text{bottom}}{\text{top}-\text{bottom}}\right) \\ 0 & 0 & -\frac{2}{\text{far}-\text{near}} & -\left(\frac{\text{far}+\text{near}}{\text{far}-\text{near}}\right) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Important note about this matrix: We could set the Z scaling value to 0, because we are supposed to convert Z to 0 for visualization. The problem with this is that the graphics API we are using (WebGL) uses a depth buffer and without a Z value, we effectively lose the depth buffer and the rendering won't be correct; instead, depending on the order the polygons are rendered.

Perspective Projection

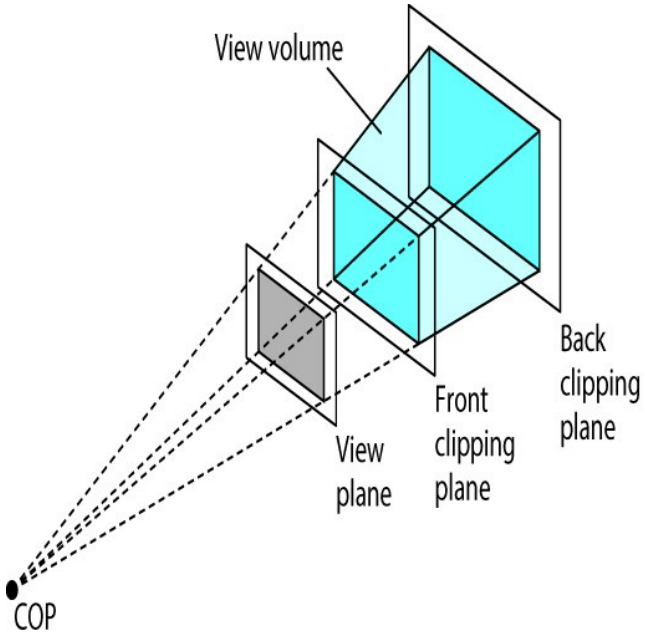
Notes for me:

- http://www.songho.ca/opengl/gl_projectionmatrix.html

Perspective projection is a representation of what the eye sees, but represented on a 2D (flat) surface. In simple terms, things that are closer appear larger than things that are further in the distance.

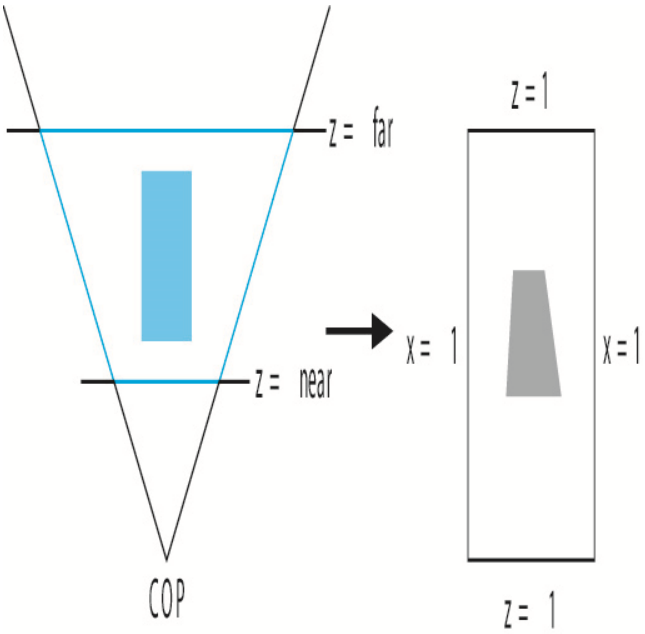
The following diagram illustrates perspective projection…

- COP : Center of Projection (the eye/viewer)
- View plane : surface onto which the scene is rendered. View plan and near plane are often the same plane.
- Front/Near clipping plane : Objects closer to the eye than this are clipped.
- Back/Far clipping plane : Objects further from the eye than this are clipped.
- There are also four polygons formed between the near and far clipping planes where objects are also clipped. Top, bottom, left, and right clipping planes.



The following diagram illustrates how we are projecting from world space to device space.

(when drawing this, don't write out the z=1, x=1, etc)



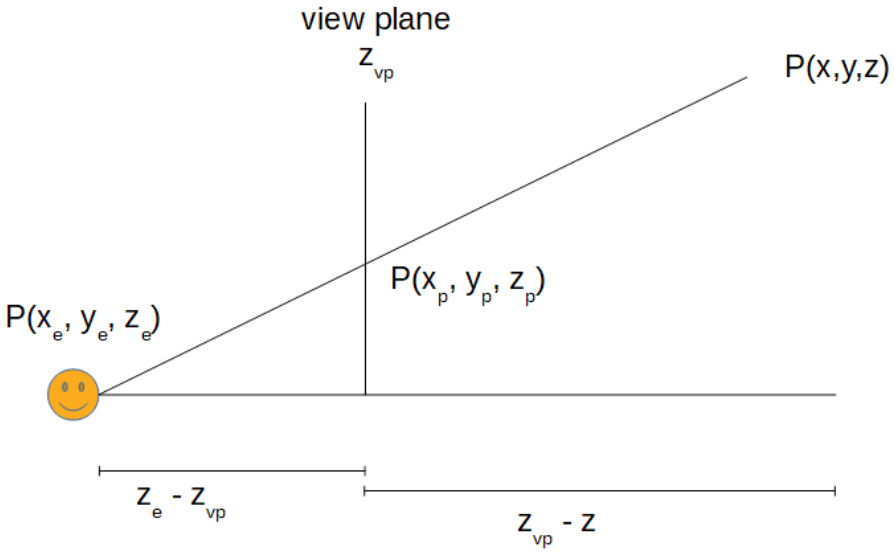
Simple Perspective Projection

$$d = z_e - z_{vp}$$

$$x_p = \frac{(z_e - z_{vp})}{(z_e - z)} x \qquad x_p = \frac{d}{(z_e - z)} x$$

$$y_p = \frac{(z_e - z_{vp})}{(z_e - z)} y \qquad y_p = \frac{d}{(z_e - z)} y$$

$$z_p = z_{vp}$$



Putting this into matrix form:

$$\begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{d}{(z_e - z)} & 0 & 0 & 0 \\ 0 & \frac{d}{(z_e - z)} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

In simple homogeneous form

$$\begin{bmatrix} q_p \\ q_p \\ q_p \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$ which gives $$q = \begin{bmatrix} x \\ y \\ z \\ \frac{z}{d} \end{bmatrix}$$ but then divide by h $$p' = \begin{bmatrix} \frac{x}{z/d} \\ \frac{y}{z/d} \\ d \\ 1 \end{bmatrix}$$

In full homogeneous form

$$\begin{bmatrix} x_h \\ y_h \\ z_h \\ h \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{-z_{vp}}{d} & z_{vp}\frac{z_e}{d} \\ 0 & 0 & \frac{-1}{d} & \frac{z_e}{d} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$ where $$h = \frac{(z_e - z)}{d}$$

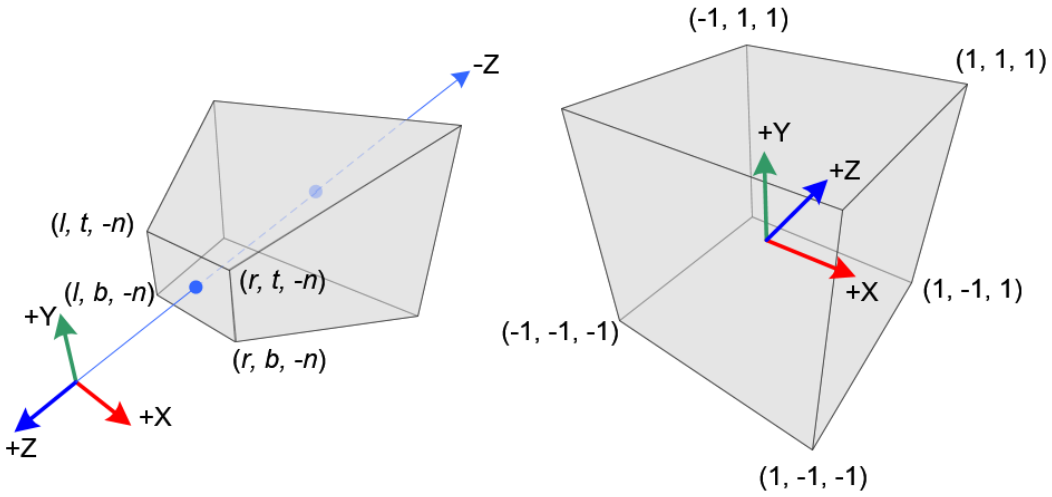
After multiplication, divide x_h, etc by *h* and you get the same result as above.

Full Perspective Projection

When working with a rendering API, such as OpenGL, you are responsible for defining the full viewing frustum, which is done by providing the appropriate perspective projection matrix. But it is actually more than that. OpenGL uses something called Normalized Device Coordinates (NDC), which is oriented in left-handed coordinates…

- x-coordinates [left, right] in the range of [-1, 1]
- y-coordinates [bottom, top] in the range of [-1, 1]
- z-coordinates [near, far] in the range of [-1, 1]

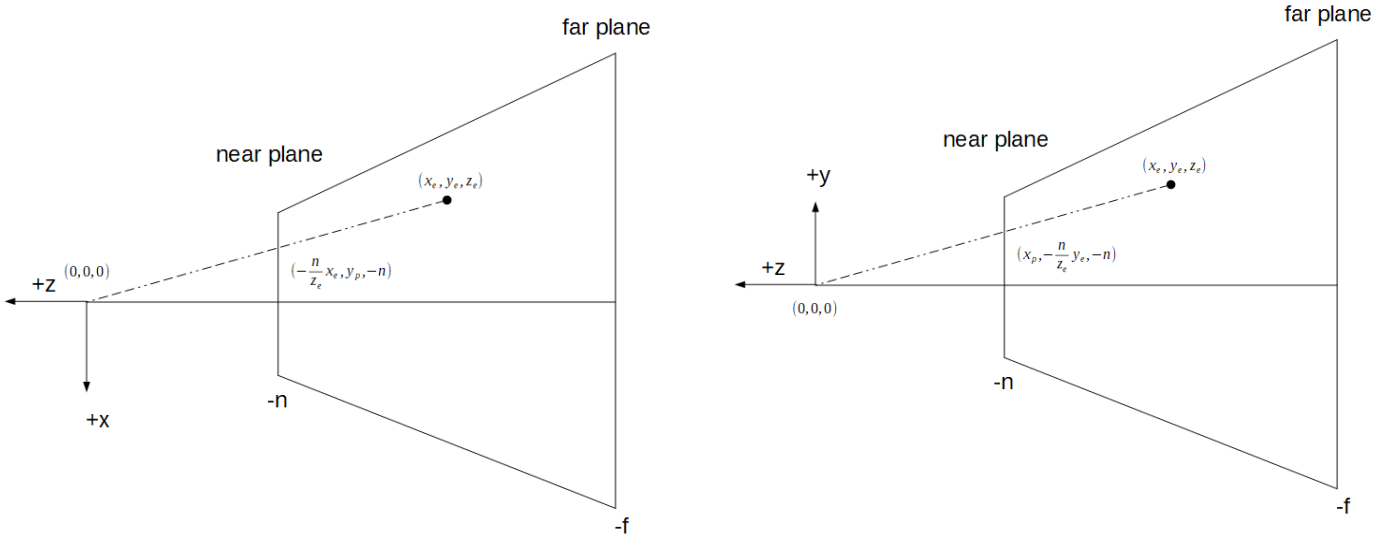
Remember, we are doing all of our other rendering using a right handed coordinate system.



A reminder of the difference "spaces" involved in our rendering:

- **Local space**. The coordinate system that is local to the object. e.g., all the vertices are defined about (0,0,0) and then a world space center is provided.
- **World space**. The coordinate system used to define all objects in the model.
- **View space** (or eye space, or camera space). This is the viewing frustum, it defines the space from which objects are rendered.
- **Clip space**. OpenGL only renders objects in the ranges as defined above, all polygons are clipped to fit within this space.

Consider the following two diagrams. Diagram on the left is the top-view of the frustum, while the diagram on the right is the side-view of the frustum.



From the top view, we can see:

$$\frac{x_p}{x_e}=\frac{-n}{z_e} \qquad x_p=\frac{-n\cdot x_e}{z_e}=\frac{n\cdot x_e}{-z_e}$$

From the side view, we can see:

$$\frac{y_p}{y_e}=\frac{-n}{z_e} \qquad y_p=\frac{-n\cdot y_e}{z_e}=\frac{n\cdot y_e}{-z_e}$$

...a whole bunch of hand-waving (refer them to the songho web reference) to create a matrix that does the perspective transformation and puts things into NDC…

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

This is for a general frustum.

If the viewing volume is symmetric, where r = -l and t = -b, then it can be simplified by:

$$\begin{Bmatrix} r+l=0 \\ r-l=2r(width) \end{Bmatrix}, \begin{Bmatrix} t+b=0 \\ t-b=2t(height) \end{Bmatrix}$$

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