# CAMERA SPACE TO CLIP SPACE PERSPECTIVE TRANSFORM

In world space, we assume our camera frame to be at some position in the world and looking at some direction ($\hat{c}$). $\hat{c}$ forms the $-Z$ axis of the camera frame and we move this camera frame to the origin and rotate the frame to align with standard XYZ basis vectors.

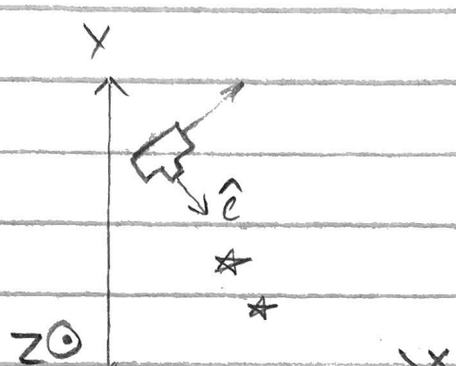



fig: World space

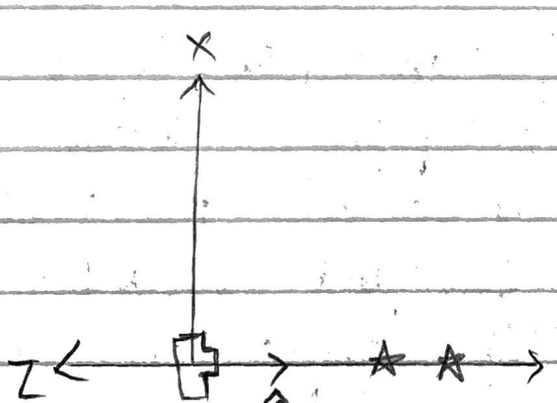



fig: Camera space.

Here we assume that we are already in camera space. for perspective transform, we can construct an imaginary line emanating from every point infront of camera, converging at origin. Camera only captures a portion of these points, and this is bounded by a "view pyramid" whose tip is at the origin. A field of view (FOV)

determines how "wide" the pyramid is. Greater the FOV, wider the pyramid, larger area captured unto a film and smaller the things appear in our film. Basically increasing FOV, is equivalent to zooming out.

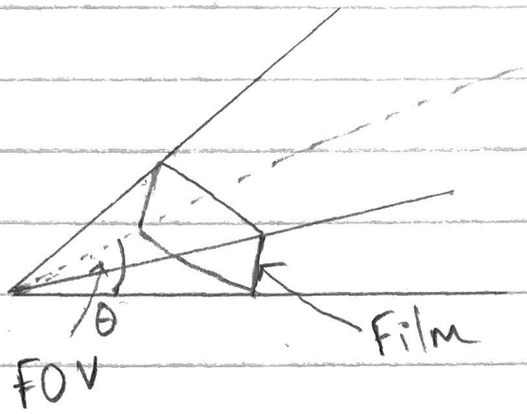



Fig: View Pyramid

The film, aka projection plane is a 2×2 plane, placed parallel to XY plane. Having this as 2×2 in size is convinient mathematically later as we will see and is also what OpenGL assumes. Points inside the pyramid project unto the film, thus forming image.

We further bound our view pyramid in Z direction, with near and far clipping plane such that points outside these clipping planes donot form an image in our film. Our view pyramid at this point looks like so:

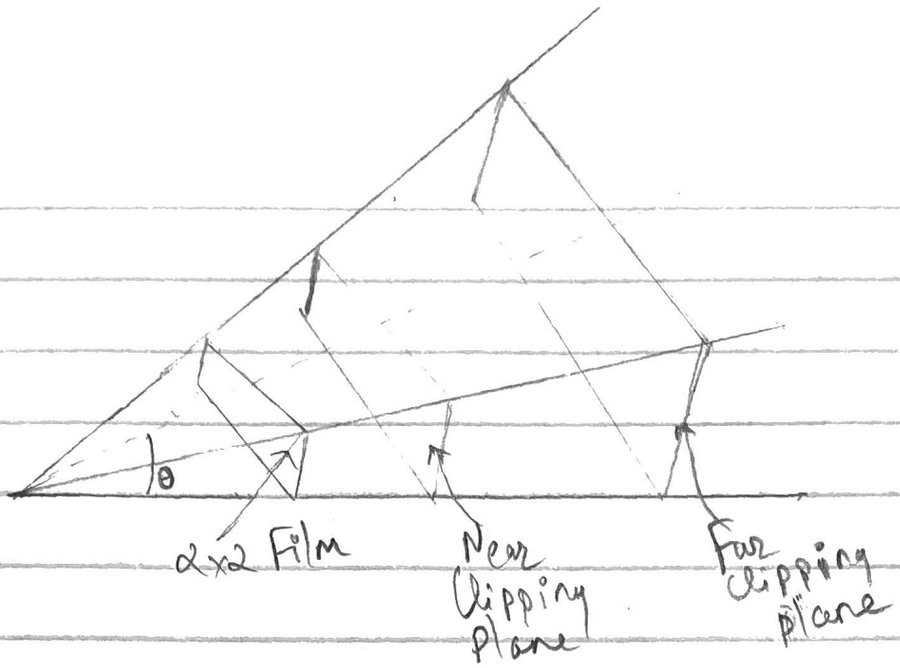
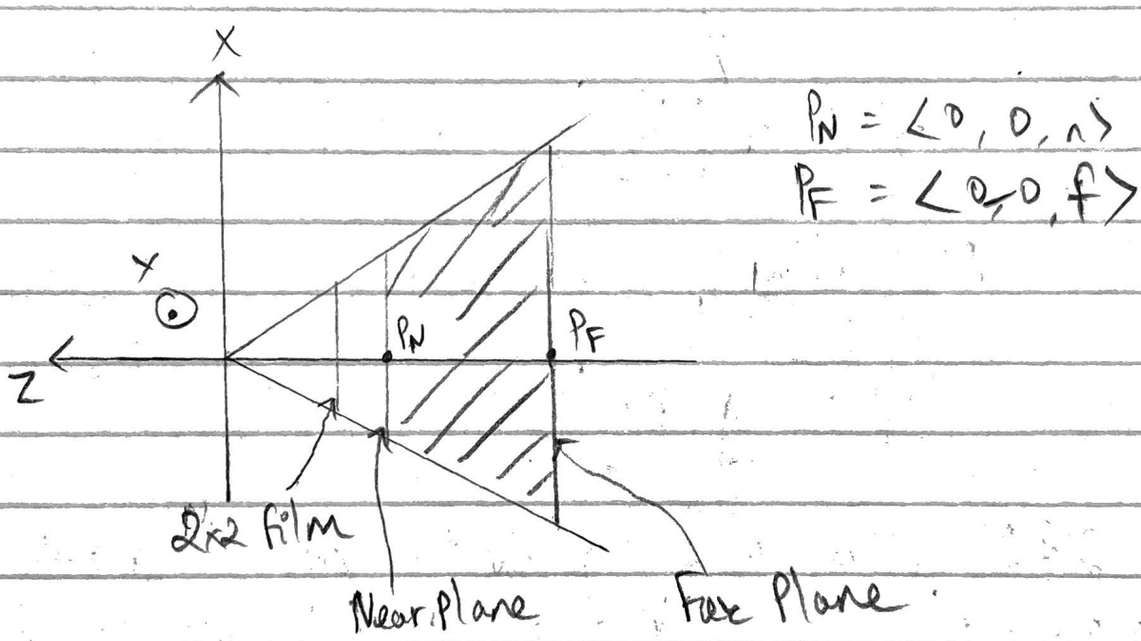



fig: View Pyramid with Clipping Planes.




$P_N = \langle 0, 0, n \rangle$

$P_F = \langle 0, 0, f \rangle$

fig: View Pyramid XZ plane

The shaded area, resembling a chopped off pyramid is also called a frustum. Also note in the image "n" in $P_N\langle 0, 0, n \rangle$ and "f" in $P_F\langle 0, 0, f \rangle$ are -ve however "N" and "F" are positive.

Now imagine a point inside frustum, the projection of X component of this point onto projection plane would look as follows

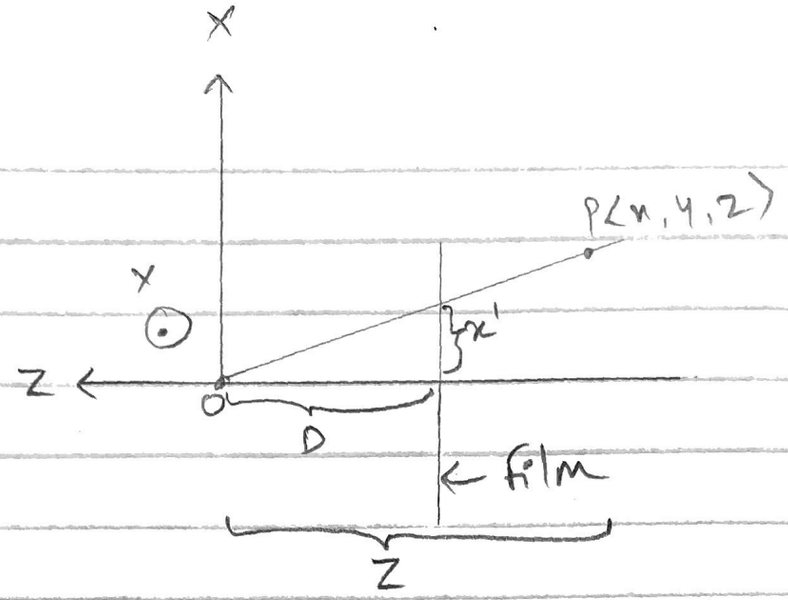



fig: Projection of X component.

$$\frac{x'}{D} = \frac{x}{Z}$$

$$x' = x\left(D/Z\right)$$

Similarly,

$$y' = y\left(D/Z\right)$$

The 2x2 Film and FOV creates a right angeled triangle through which we can get D:

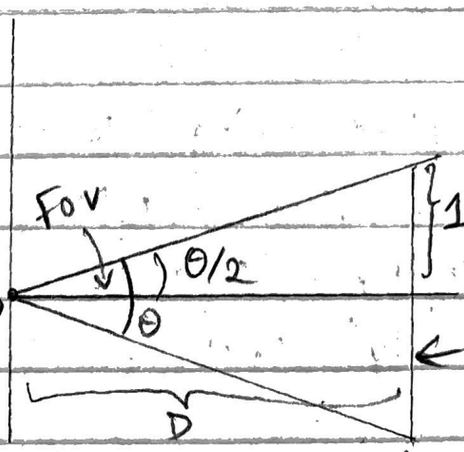



fig: Relation between 2x2 film and FOV

$$\tan\theta/2 = 1/D \;;\; D = 1/\tan\theta/2$$

Thus the $x'$ and $y'$ in above eqns become:

$$x' = x\left(\frac{1/\tan\theta/2}{z}\right) \quad \text{— (1)}$$

$$y' = y\left(\frac{1/\tan\theta/2}{z}\right) \quad \text{— (2)}$$

Encoding this division by $Z$ as 4th coordinate of output vector, so that our transformation would look as such:

$$T\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ k \\ -z \end{bmatrix} \quad \text{— (3)}$$

Notice how the "z" has -ve in eqn 3, which makes it a +ve number. The "z" in eqn 3 is always -ve for the points lying in the region where camera is facing. "z" in eqn 1 and 2 were absolute distance from the origin, so these two different "z"s can be confusing.

Thus in eqn 3, by sticking "-" in front of z, we achieve division by the z. Notice that I have placed "k" in the z coordinate of the output vector in eqn 3. Say if we had just put z there, then by virtue of that

-Z as 4th coordinate, that would mean every output vector would have -1 as Z coordinate. We need this Z coordinate to determine if the point lies within the clipping planes, and also to determine which image sits on top of another.

You may argue why not just leave that as Z and in our program have a special condition that takes into consideration that Z, before doing the division. That would certainly simplify things theoretically, but my guess is that having this special condition makes hardware complex and/or makes graphics pipeline slow.

Therefore, there is this wonderful hack, that graphics guys have come up with that preserves the relative order of Z of the points being transformed, although the values may be distorted. Here is the derivation of transform "T":

$$\begin{bmatrix} 1/\tan\theta/2 & 0 & 0 & 0 \\ 0 & 1/\tan\theta/2 & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} (1/\tan\theta/2)x \\ (1/\tan\theta/2)y \\ az+b \\ -z \end{bmatrix}$$

When z is "n"

$$\frac{an+b}{-n} = n$$

$$an + b = -n^2$$

$$a = \frac{-n^2 - b}{n} \quad \text{— (4)}$$

When z is "f"

$$\frac{af+b}{-f} = f$$

$$af + b = -f^2$$

$$\left(\frac{-n^2-b}{n}\right)f + b = -f^2 \quad [\text{from (4)}]$$

$$(-n^2 - b)f + bn = -nf^2$$

$$-n^2f - bf + bn = -nf^2$$

$$-b(n-f) = n^2f - nf^2$$

$$b = \frac{nf(n-f)}{(n-f)}$$

$$b = nf$$

$$b = -N \times -F \quad [\because n = -N \text{ and } f = -F]$$

$$b = NF \quad \text{— (5)}$$

So,

$$a = \frac{-n^2 - NF}{n} \quad [\text{From (4) and (5)}]$$

$$a = \frac{-(n^2 + NF)}{n}$$

$$a = \frac{N^2 + NF}{N} \; ; \; a = N + F$$

Thus our transformation becomes

$$\begin{bmatrix} 1/\tan\theta/2 & 0 & 0 & 0 \\ 0 & 1/\tan\theta/2 & 0 & 0 \\ 0 & 0 & N+F & NF \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} (1/\tan\theta_2)x \\ (1/\tan\theta/2)y \\ (N+F)z + NF \\ -z \end{bmatrix}$$

This transformation transforms our frustum to 2x2x(F-N) rectangular prism

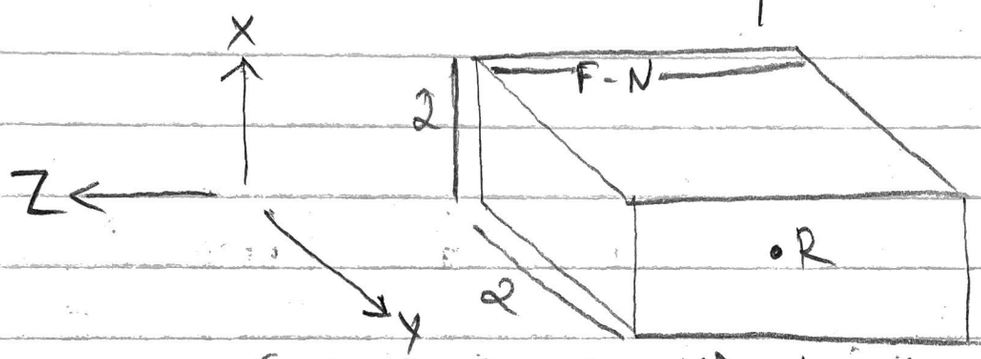



fig: 2x2x(F-N) transformed frustum

and the center R is located at $\langle 0, 0, n+f/2 \rangle$. So, to transform this box to clipspace which is 2x2x2 cube, with center at origin, we ought to translate it by $\langle 0, 0, -(n+f)/2 \rangle$ i.e. $\langle 0, 0, (N+F)/2 \rangle$ and then scale the Z by 2/F-N which is:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & (2/F-N) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & (N+F)/2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 2/F-N & (N+F)/(F-N) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

So, the final camera space to clip space transformation involves, first transforming to rectangular prism then applying the translate / scale operation. This is given as:

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 2/F-N & (F+N)/(F-N) \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1/\tan\theta/2 & 0 & 0 & 0 \\ 0 & 1/\tan\theta/2 & 0 & 0 \\ 0 & 0 & N+F & NF \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

$$T = \begin{bmatrix} 1/\tan\theta/2 & 0 & 0 & 0 \\ 0 & 1/\tan\theta/2 & 0 & 0 \\ 0 & 0 & c & d \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

where, $C = 2(N+F)/(F-N) - (F+N)/(F-N)$

$= (F+N)/(F-N)$

and, $d = 2NF/(F-N)$

Thus, $T = \begin{bmatrix} 1/\tan\theta/2 & 0 & 0 & 0 \\ 0 & 1/\tan\theta/2 & 0 & 0 \\ 0 & 0 & (F+N)/(F-N) & 2NF/(F-N) \\ 0 & 0 & -1 & 0 \end{bmatrix}$

is the final camera to clip space perspective transform.

An openGL gotcha to beware:

The above transformation transforms "n" to 1 and "f" to -1. OpenGL by default expects "n" to be transformed to -1 and "f" to 1 instead. So, to fix that, call glDepthRange(1, 0).

Alternately, you could also negate the $T_{3,3}$ and $T_{3,4}$ term in the matrix above, so that the matrix would look like

$$\begin{bmatrix} 1/\tan\theta/2 & 0 & 0 & 0 \\ 0 & 1/\tan\theta/2 & 0 & 0 \\ 0 & 0 & -(F+N)/(F-N) & -2NF/(F-N) \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

and this would also solve that issue