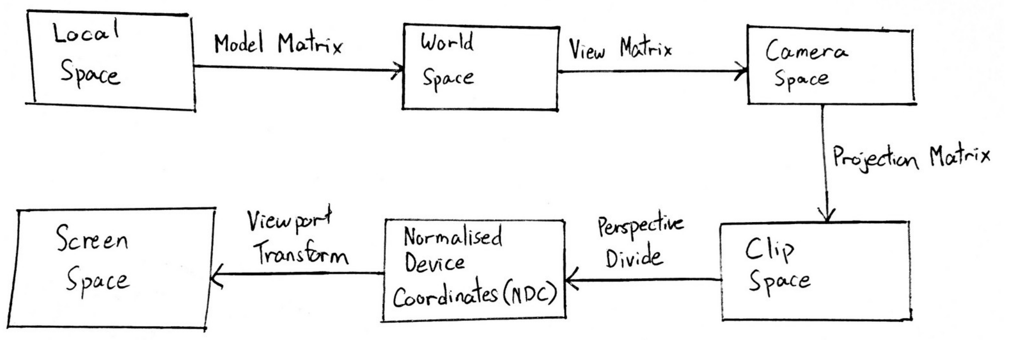
OpenGL's Transformation Pipeline



To give a brief overview of the OpenGL rendering pipeline, in order to render an object on the screen, the 3D model shall go through a pipeline with a few stages.

The coordinates of a point P is represented using homogeneous coordinates, where P = $\begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$.

Local Space

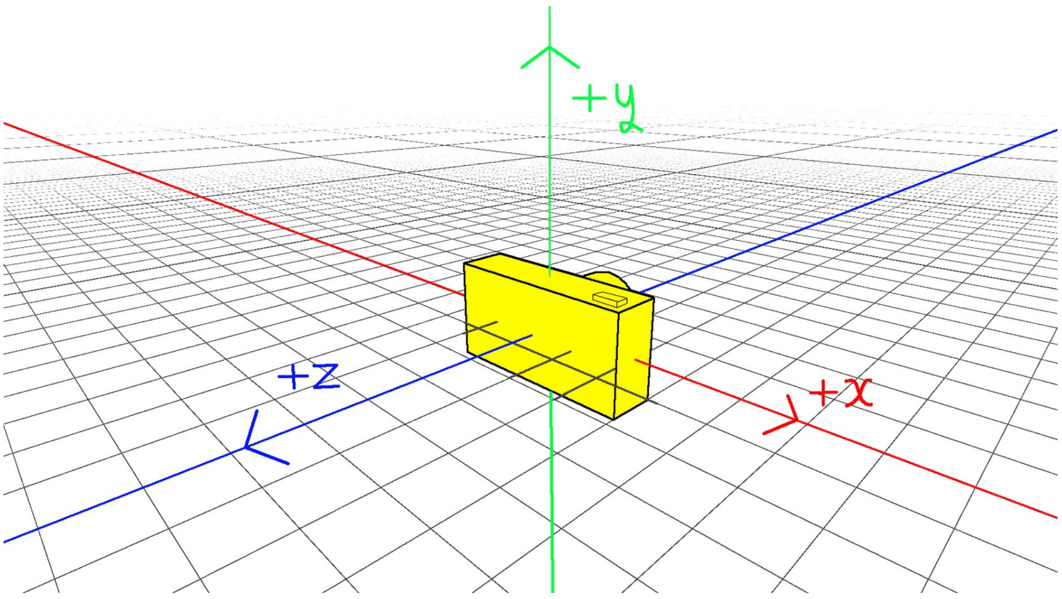
This represents P without any transformation applied. For example, a 3D model file such as Blender or Maya stores the vertices of the model in local space.

World Space

This represents P with its translation, rotation and scale applied. For example, a 1x1x1 cube that is rotated 45° and placed at (1, 5, -2) in the world.

Camera Space

This represents P relative to the camera. That is to say, we treat the camera's orientation to be the new origin, and find the transformation of P relative to this new origin. By convention, in OpenGL the camera looks towards the -z-axis, with the x-axis pointing to the right, and the y-axis pointing up (right-hand rule).



Clip Space

By applying the perspective matrix, we scale down objects which are further from the camera, and scale up objects closer to the camera, to simulate how we perceive distance in real life. Any points not within our view frustum is also “clipped” away here, as they cannot be seen.

Let Ω = aspect ratio, and Θ = field of view.
Perspective Matrix * P:

$$\begin{pmatrix} \frac{1}{\Omega \tan(0.5\Theta)} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan(0.5\Theta)} & 0 & 0 \\ 0 & 0 & \frac{near+far}{near-far} & \frac{2(near)(far)}{near-far} \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{x}{\Omega \tan(0.5\Theta)} \\ \frac{y}{\tan(0.5\Theta)} \\ \frac{z(near+far)}{near-far} + \frac{2(near)(far)}{near-far} \\ -z \end{pmatrix}$$

Take note that when multiplying the perspective matrix with P, the last row of the perspective matrix, $\begin{pmatrix} 0 & 0 & -1 & 0 \end{pmatrix}$, copies -z into w.

Normalised Device Coordinates (NDC)

Next, we normalise the points in our view frustum into the [-1, 1] range in all 3 axes, by dividing the x, y and z values of P with w. This is called *perspective divide*.

$$\begin{pmatrix} \frac{x}{\Omega \tan(0.5\Theta)} \\ \frac{y}{\tan(0.5\Theta)} \\ \frac{z(near+far)}{near-far} + \frac{2(near)(far)}{near-far} \\ -z \end{pmatrix} \left(\frac{-1}{z}\right) = \begin{pmatrix} \frac{-x}{z\Omega \tan(0.5\Theta)} \\ \frac{-y}{z\tan(0.5\Theta)} \\ \frac{(near+far)}{near-far} + \frac{2(near)(far)}{z(near-far)} \\ 1 \end{pmatrix}$$

While by convention OpenGL’s coordinate systems use the right-hand rule, the NDC uses the left-hand rule, with the z-axis pointing into the screen, with the x-axis pointing to the right, and the y-axis pointing up (right-hand rule). This is why in the previous step, we negate the z when copying into w.

Screen Space

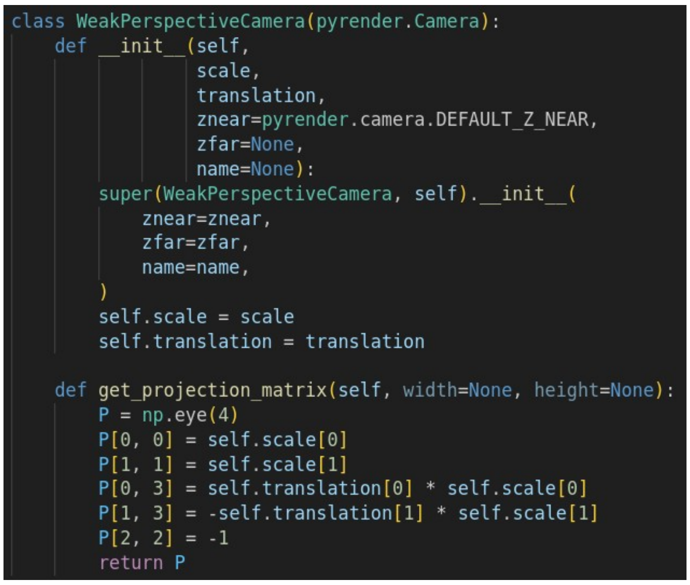
Finally, OpenGL maps the points from the [-1, 1] range, onto the actual pixels on our screen.

Weak Perspective Camera

For our purposes, we define camera to always be at the origin, the transformation of our scene to be in camera space. As such, we can ignore the model and view matrices.

In this project, we use VIBE to detect a human. But instead of using the usual perspective camera, VIBE uses a *WeakPerspectiveCamera* instead. Rather than scaling the human based on distance to the camera, the *WeakPerspectiveCamera* only translate and scale the human 3D models on the x and y axes, such that it best fits the screen position of the human in the video. This is probably because VIBE is unable to determine the distance of the human in a 2D video, from the camera in the 3D world.

For every human in every frame, VIBE provides a *WeakPerspectiveCamera*, with a x-scale S_x , y-scale S_y , x-translation T_x , and y-translation T_x . We do not need to derive these values ourselves as they are output by the VIBE AI.



```
class WeakPerspectiveCamera(pyrender.Camera):
    def __init__(self,
                 scale,
                 translation,
                 znear=pyrender.camera.DEFAULT_Z_NEAR,
                 zfar=None,
                 name=None):
        super(WeakPerspectiveCamera, self).__init__(
            znear=znear,
            zfar=zfar,
            name=name,
        )
        self.scale = scale
        self.translation = translation

    def get_projection_matrix(self, width=None, height=None):
        P = np.eye(4)
        P[0, 0] = self.scale[0]
        P[1, 1] = self.scale[1]
        P[0, 3] = self.translation[0] * self.scale[0]
        P[1, 3] = -self.translation[1] * self.scale[1]
        P[2, 2] = -1
        return P
```

Weak Perspective Matrix * P:

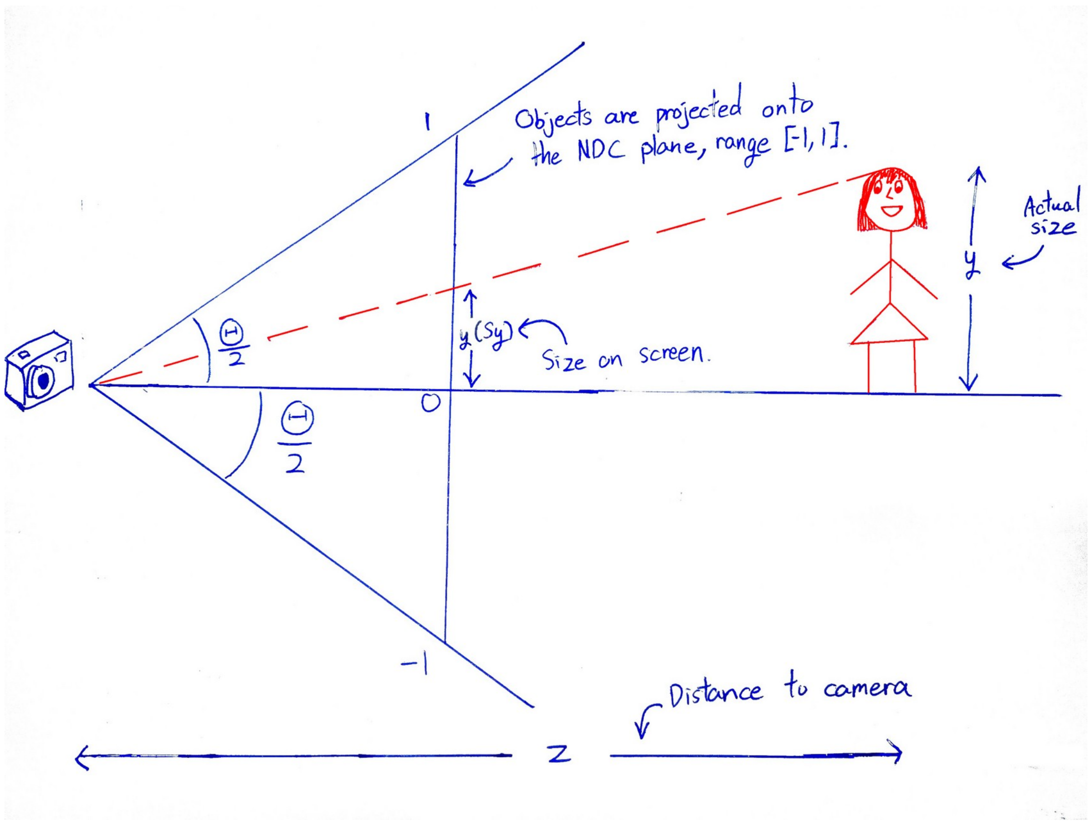
$$\begin{pmatrix} S_x & 0 & 0 & T_x \cdot S_x \\ 0 & S_y & 0 & -T_y \cdot S_y \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x(S_x) + (T_x \cdot S_x) \\ y(S_y) - (T_y \cdot S_y) \\ -z \\ 1 \end{pmatrix}$$

As can be seen, the weak perspective matrix does not translate the 3D model on the z-axis. It also does not scale the 3D model on the z-axis, other than negating it.

The last row $\begin{pmatrix} 0 & 0 & 0 & 1 \end{pmatrix}$ also does not copy z into w. That means that perspective divide does not occur as w = 1.

Estimating Depth

What we want to achieve, is to estimate a z translation for the human 3D model, without scaling it, such that when viewed using a regular perspective matrix, it appears the same size on the screen as if we were to simply to scale it on the x and y axis using the weak perspective matrix.



Using the model height, we can estimate $y(S_y) = \frac{-y}{z\tan(0.5\Theta)}$. Hence, $z = \frac{-1}{S_y \tan(0.5\Theta)}$.

Thus, by estimating z, we have a good estimate of how far the human is from the camera.

Estimating X & Y Translation

Weak Perspective Matrix * P:

$$\begin{pmatrix} S_x & 0 & 0 & T_x \cdot S_x \\ 0 & S_y & 0 & -T_y \cdot S_y \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x(S_x) + (T_x \cdot S_x) \\ y(S_y) - (T_y \cdot S_y) \\ -z \\ 1 \end{pmatrix}$$

Studying the weak perspective matrix, we can see that the final translation in clip space is $T_x \cdot S_x$ along the x-axis, and $-T_y \cdot S_y$ along the y-axis.

Working backwards, we can see that the matrix is derived via:

$$\begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & -T_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} S_x & 0 & 0 & T_x \cdot S_x \\ 0 & S_y & 0 & -T_y \cdot S_y \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

This means that when using the weak perspective matrix, the 3D model is *first translated* by T_x and $-T_y$, *then scaled* by S_x and S_y. This means that if we do not scale the models, such as when using our regular perspective matrix, we just need to translate it by T_x and $-T_y$.