projection matrix

#notes

Normalizing Screen Space

Screen should be centered at $(0, 0)$ with bounds $[-1, 1]$.

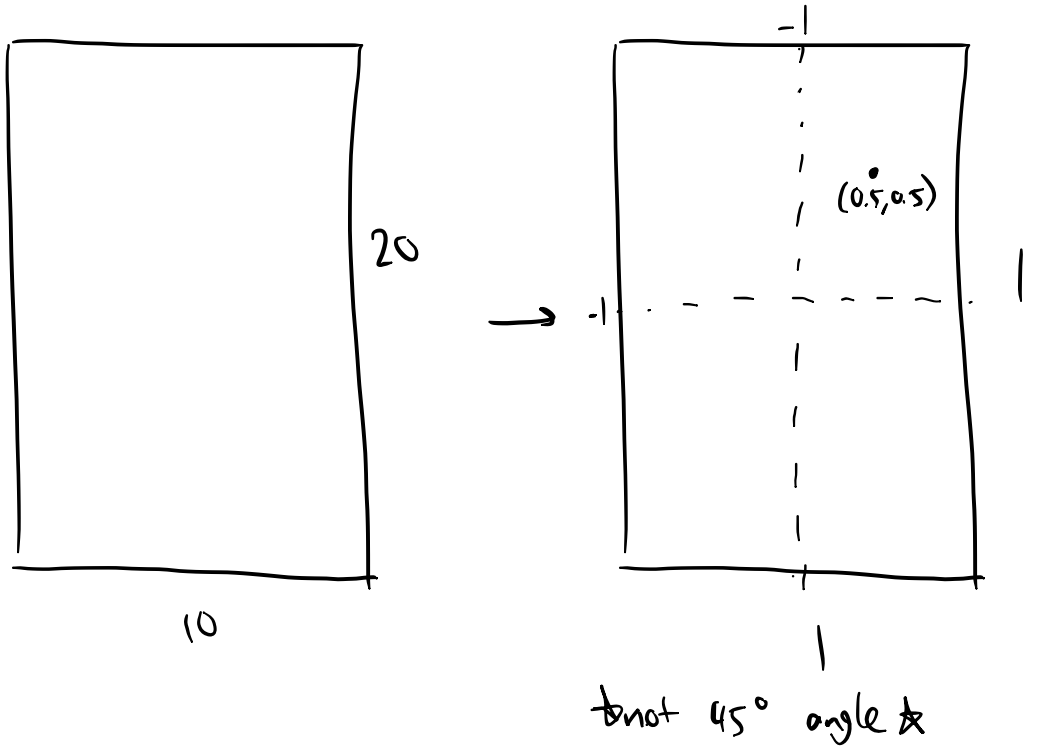
Scaling For Different Screen Sizes

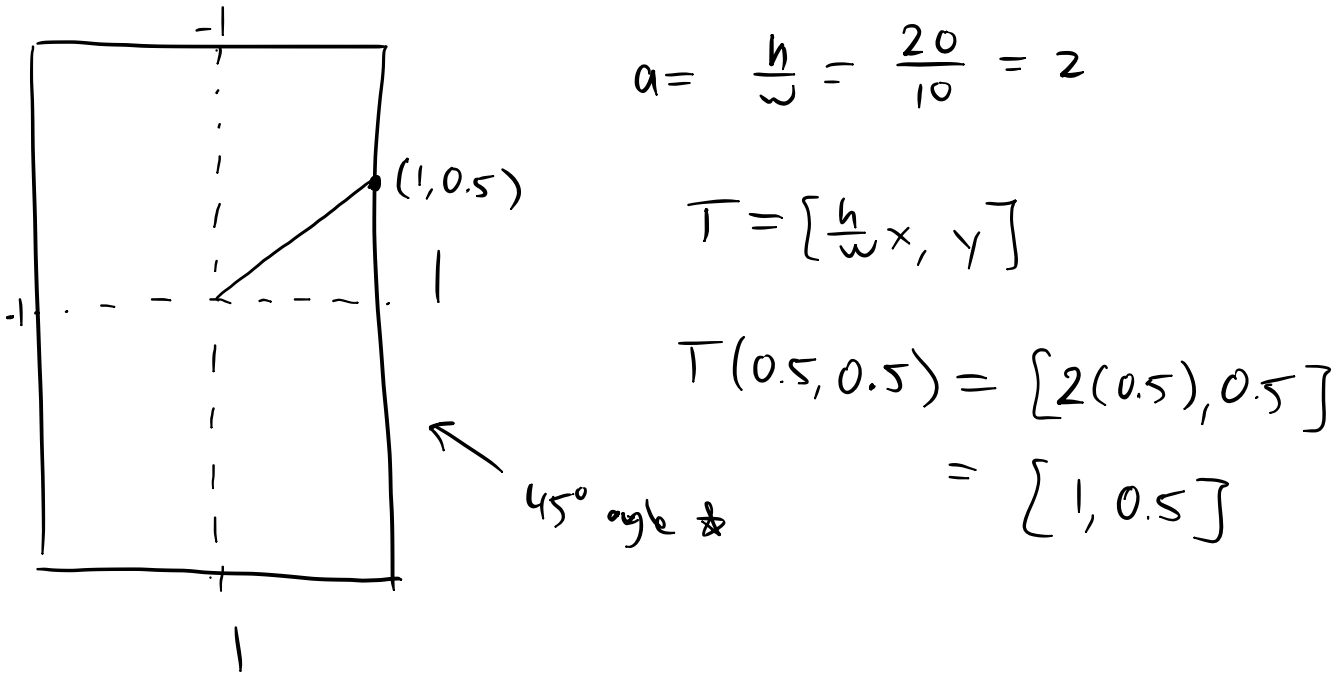
The screen can be all different sizes.

We'll define aspect ratio as $a = h/w$. This is because we want horizontal distances to equal vertical distances.

An example is if the width of our screen 10 and the height of our screen is 20. There is a lot more vertical distance on this screen so if we normalize each side to length 1, each unit of distance vertically is more than each unit of distance horizontally. This is almost as if we pixels that are double as tall as wide. We need to stretch out the horizontal distance so that a 1 unit change in the horizontal direction equals a 1 unit change in the vertical.

I'll illustrate the problem and solution by trying to get a point at a 45 degree angle to the origin.





So the transformation so far is:

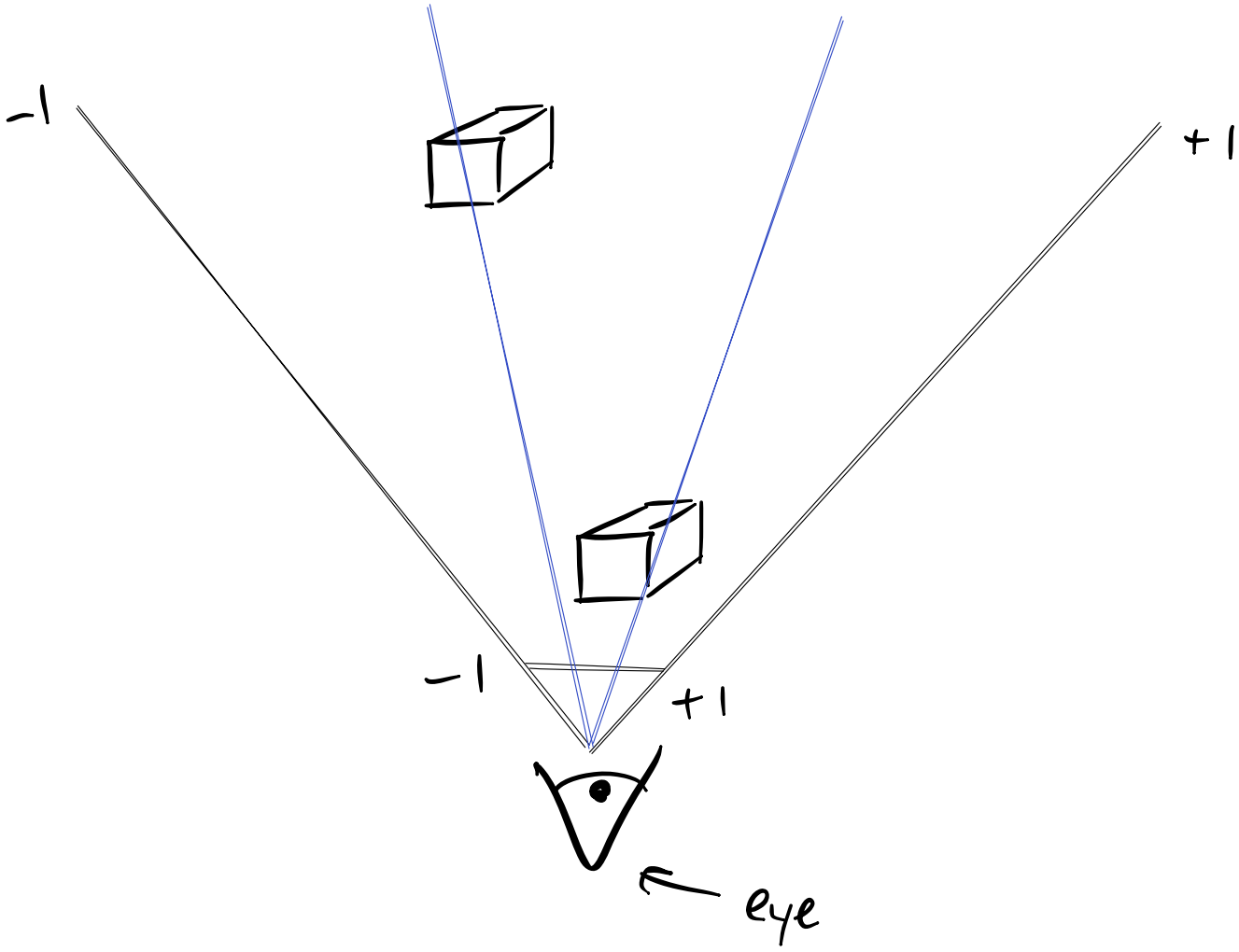
$$[x, y, z] \rightarrow \left[\frac{h}{w}x, y, z\right]$$

Field of View

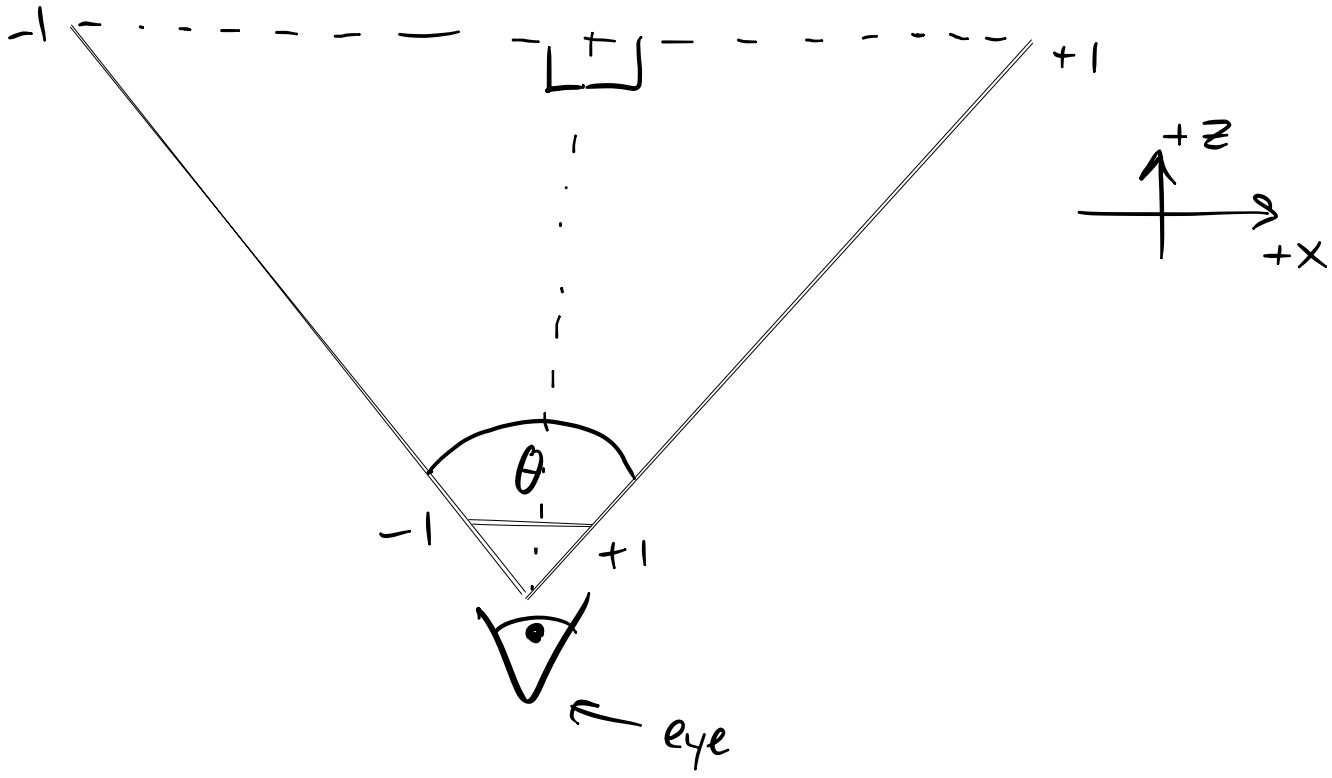
Our eyes don't just see straight ahead, but instead more of a "cone" of sight.

We can see more things when they're farther away than up close.

Additionally, if you increase your field of view, you can see more things, and that has the effect of zooming out. The opposite happens when you decrease your field of view -- it's like zooming in and you can see less. Zooming out seems to make things appear farther away and zooming in makes things appear closer.



As you can see in the diagram, with a greater field of view you can see more and with a smaller field (blue) of view you can see less.



The field of view can be described using a parameter θ, which is the angle between the two rays cast out by our eyes.

What we want is to be able to view more things at a distance and less things up close. Given objects of the same size, the one in the distance will take up less pixels on the screen than the one up close.

We can divide our viewing cone into 2 right triangles each with angle $\theta/2$ closest to the eye. As the angle $\theta/2$ increases, the opposite side of the triangle becomes larger, and this can be described using the tangent function such as $\tan \frac{\theta}{2}$.

Since we said before that objects should become narrower (think about width) when viewed at a distance or when zooming out, if we multiply the x values by $\tan \frac{\theta}{2}$, the objects become wider when the field of view increases. We want the opposite effect so instead we take the reciprocal to get our desired effect of condensing the pixels far away and spreading them out up close.

So our scaling factor for field of view is:

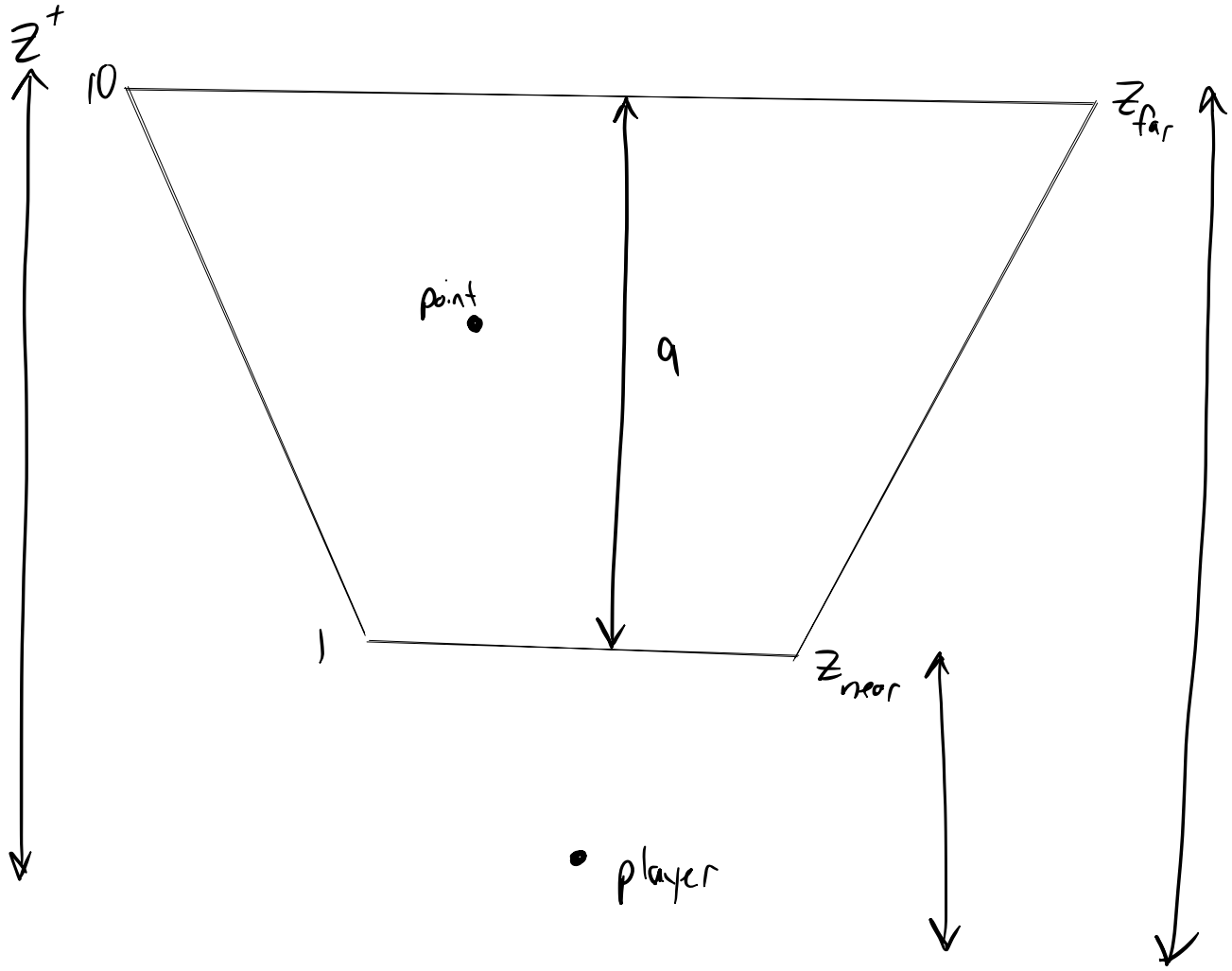
$$\frac{1}{\tan \frac{\theta}{2}}$$

The same thinking can be applied to the y-values or height of objects. The farther away they are or the more you zoom out, the less vertical space they take up

So the current transformation looks like:

$$[x, y, z] \rightarrow \left[\frac{h}{w} x \frac{1}{\tan\frac{\theta}{2}}, y \frac{1}{\tan\frac{\theta}{2}}, z\right]$$

Normalizing Z Axis



$z = 5$ $\qquad \frac{5}{9} \cdot 10 - \frac{1}{9} \cdot 10$

$\frac{50}{9} \cdot \frac{10}{9} = \frac{40}{9}$

With the viewing cone in mind, we have to realize that the player is not looking with their eyes directly on the screen but at some distance Z_{near} away from the screen. We also have a plane called Z_{far} which is the farthest something can be drawn in our engine.

The point is between Z_{far} and Z_{near} so we need to scale it between 0 and 1. The distance between Z_{far} and Z_{near} is $10 - 1 = 9$, so we divide the point's Z value by 9. This almost is all the work for scaling the Z coordinate of the point between 0 and 1, but we also need to take into account the offset by Z_{near}.

Think of Z_{near} as its own point that we need to normalize between 0 and 1, and we do the same operation. We do $Z_{\text{near}}/9$. Since the point on the screen is automatically moved away from the player by this distance, we subtract this distance so that the point has a normalized Z coordinate between 0 and 1.

So the overall formula for normalizing the Z coordinate of a point is then:

$$Z \to \frac{Z}{Z_{\text{far}} - Z_{\text{near}}} - \frac{Z_{\text{near}}}{Z_{\text{far}} - Z_{\text{near}}}$$

Combining this latest Z scaling factor with the ones before we get:

$$[x, y, z] \to \left[\frac{h}{w} x \frac{1}{\tan\frac{\theta}{2}}, y \frac{1}{\tan\frac{\theta}{2}}, z \frac{1}{Z_{\text{far}} - Z_{\text{near}}} - \frac{Z_{\text{near}}}{Z_{\text{far}} - Z_{\text{near}}}\right]$$

Scaling X and Y for Z depth

So we've already scaled X and Y for field of view given the aspect ratio. We also need to scale them once again in a similar fashion for the point's Z depth.

Imagine two objects moving at the same speed. One is a mile away and the other is 10 feet away. Even though they are moving at the same speed, there is more change in the horizontal location of the closer object than the farther away object from your perspective. Mathematically this means that the change in x is inversely proportional to z. The same can be said for y.

Our final scaling is then dividing the x and y components by z:

$$[x, y, z] \to \left[\frac{\frac{h}{w} x \frac{1}{\tan\frac{\theta}{2}}}{z}, \frac{y \frac{1}{\tan\frac{\theta}{2}}}{z}, z \frac{1}{Z_{\text{far}} - Z_{\text{near}}} - \frac{Z_{\text{near}}}{Z_{\text{far}} - Z_{\text{near}}}\right]$$

Transformation Using a Matrix

This transformation can be represented by a matrix multiplication.

I'm going to set some terms in the final scaling to variables so that the matrix doesn't get too messy.

$$a = \frac{h}{w}$$

$$f = \frac{1}{\tan\frac{\theta}{2}}$$

$$q = \frac{1}{Z_{\text{far}} - Z_{\text{near}}}$$

$$[x, y, z] \to \left[\frac{afx}{z}, \frac{fy}{z}, zq - Z_{\text{near}}q\right]$$

The matrix with the variables is:

$$T([x, y, z, 1]) = [x \quad y \quad z \quad 1] \begin{bmatrix} af & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & q & 1 \\ 0 & 0 & -Z_{\text{near}}q & 0 \end{bmatrix} = [afx, fy, zq - Z_{\text{near}}q, z]$$

The final operation to get cartesian coordinates for our 2D computer screen is to divide all entries by z.