Mini 3D Engine

David Cao

Dec 10

Problem Statement (Computer Graphics)

Given the ability to draw point, lines, and access to time
(1) Can we create a computer program that mimics a 3D object?
(2) Can we have a camera and have free control to the camera, moving it left, right, top, down, forward, backward, rotating it upward, downward, leftward and rightward?
(3) Can we create a method to visualize a linear transformation of a 3D object using the computer program we just created?

Why Linear Algebra?

Compare to simple algebra, linear algebra and matrices gives us some unique advantages:

1. Better visualization and geometric representation
2. Allows composition (multiplication of matrices) to represent complex operations
3. Optimized for computer hardwares

Visualizing a 3D Object

In order to have a 3D object, it must have three properties: width, height, and depth, which we denote to be x, y, and z value.

Using the simplist object - a sqaure, we can use an 3x1 vector $\begin{bmatrix} x \\ y \\ z \end{bmatrix}$ to denote eight of its vertices. However, for the simplicity of matrix calculation (you will see what that means in

a second), we use a 4x1 vector $\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$ with "w" denoting the homogenous coordinate (you will see what that means in a second as well) Moreover, in order to get the shape of the square, we can use vectors to store the polygon faces:

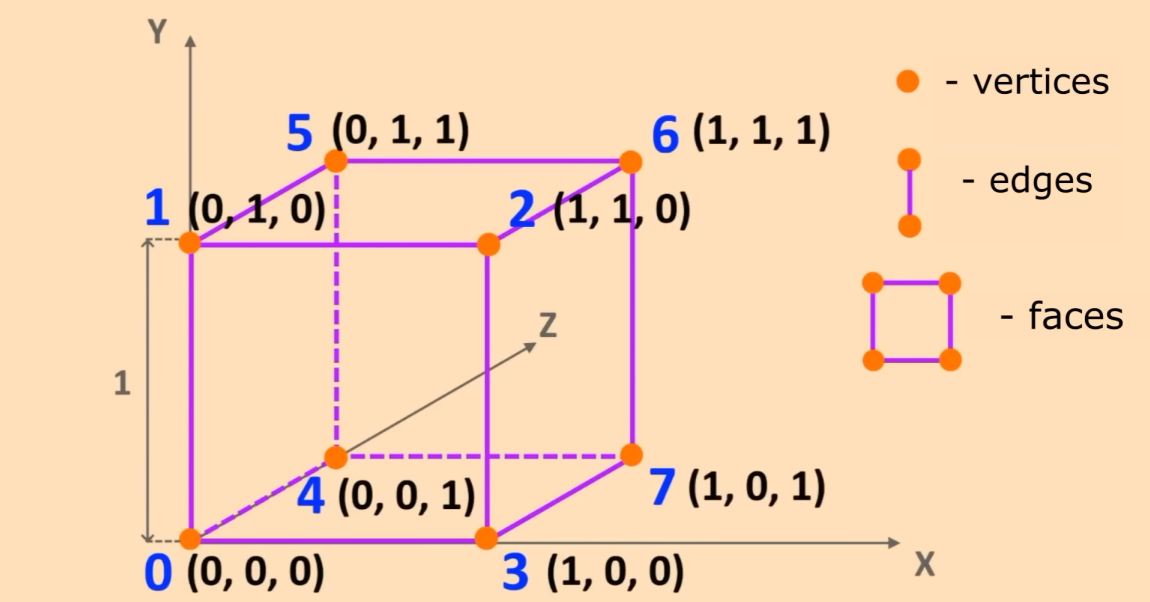


Figure 1:

We can use the coordinate to describe the faces of the cube, namely: $\begin{bmatrix} (0,1,2,3), \\ (4,5,6,7), \\ (0,4,5,1), \\ (2,3,7,6), \\ (1,2,6,5), \\ (0,3,7,4), \end{bmatrix}$ with each number in the vector denoting a vertex number displayed in the graph

Elementary Operations for 3D Objects

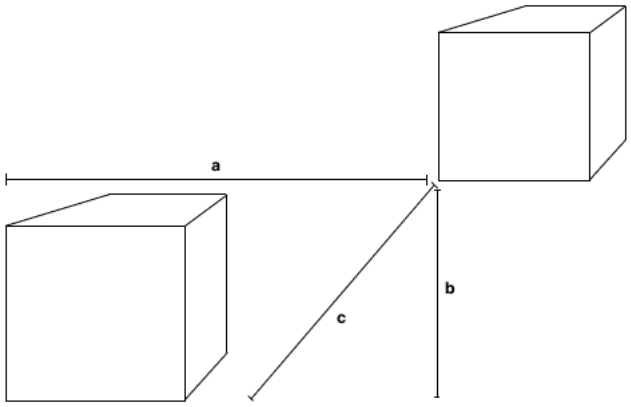
For a 3D object, we want to be able to perform the following elementary operations:

1. translation
2. rotation
3. scaling

We notice that those operations can be represented by linear algebra:

0.1 translation

By translation, we mean shifting the object in the x, y, and z direction by some amount, which is represented by the graph below:



With a 3x1 vector denoting the original position to be $\begin{bmatrix} x \\ y \\ z \end{bmatrix}$ and shifting a unit in the x-axis, b unit in the y-axis, and c unit in z-axis, we can denote the new coordinate to be: $\begin{bmatrix} x+a \\ y+b \\ z+c \end{bmatrix}$ It is tempting to represent this new vector by a matrix multiplication like this: $\begin{bmatrix} 1 & \_ & \_ \\ \_ & 1 & \_ \\ \_ & \_ & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x+a \\ y+b \\ z+c \end{bmatrix}$ However, it's not possible using a 3x3 matrix, since we do not want any resulting term to be a variable of the other coordinate. Here comes the

homogenous coordinate system:
For ease of calculation and representation, we add an extra dimension to our matrix (we want to keep w to 1 in this case, you will see why later in projection matrix):

$$\begin{bmatrix} 1 & \_ & \_ & \_ \\ \_ & 1 & \_ & \_ \\ \_ & \_ & 1 & \_ \\ \_ & \_ & \_ & 1 \end{bmatrix} \quad \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad = \quad \begin{bmatrix} x+a \\ y+b \\ z+c \\ 1 \end{bmatrix}$$ If we to try to find a matrix satisfying this condition, by properties of matrix multiplication, we can see that: $$\begin{bmatrix} 1 & \_ & \_ & a \\ \_ & 1 & \_ & b \\ \_ & \_ & 1 & c \\ \_ & \_ & \_ & 1 \end{bmatrix} \quad \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad = \quad \begin{bmatrix} x+a \\ y+b \\ z+c \\ 1 \end{bmatrix}$$ satisfy such condition. Put into simple term, here's our translation matrix:

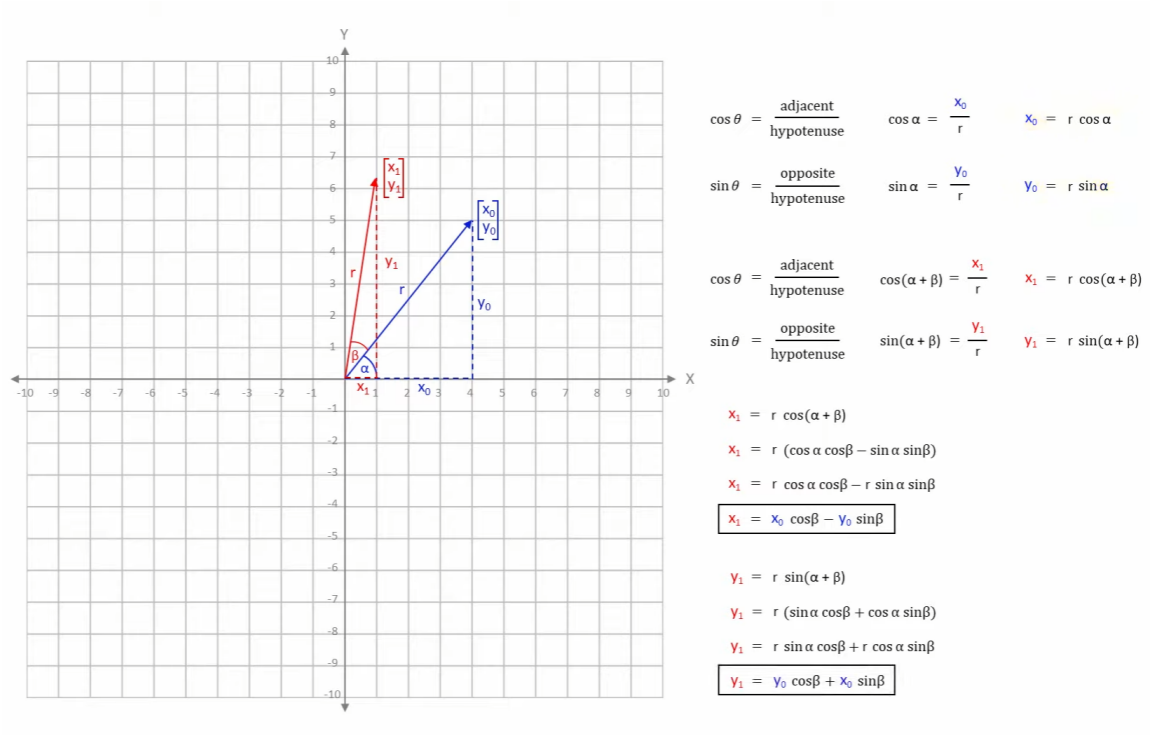
$$T(a,b,c) = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

0.2 rotation

We also want to perform rotation on our object, which can be achieved by furthur seperating the rotation into:

1. rotate around the x-axis
2. rotate around the y-axis
3. rotate around the z-axis

Putting it in 2D first, we have this graph:



We notice that this is a rotation along the z-axis since it's rotating in the xy cut plane. Therefore, we yield our rotational matrix in respect to z axis, and consequently x and y axis shown below (the image doesn't have a homogenous coordinate by the way):

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

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

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

0.3 scaling

The scaling matrix is the simplist among all, we can intuitively come up with the scale matrix to be:

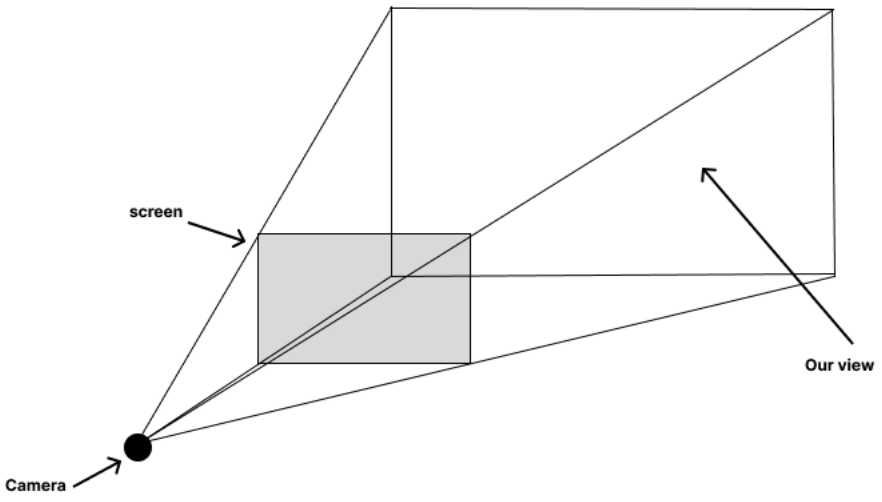
$$S_v(n) = \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

However, we do want to generalize the scaling matrix so that it is able to scale individually on the x-axis, y-axis, and z-axis, luckily, linear algebra gives us the tool to achieve that, it is also very simple:

$$S_v(n_x, n_y, n_z) = \begin{bmatrix} n_x & 0 & 0 & 0 \\ 0 & n_y & 0 & 0 \\ 0 & 0 & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Visualize the camera and projection

Now we want to add our camera, and obviously we want to place ourself at the camera's view, the grey rectangle will be the picture displayed on our screen, the image is shown below:



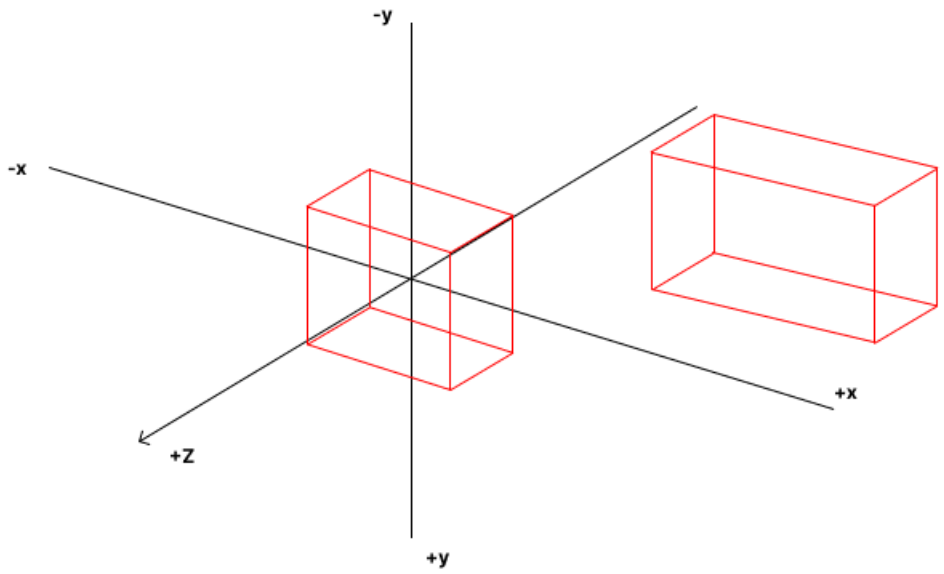
Notice that in our case, x is width, y is height, and z is depth, meaning that our coordinate system is different

Orthographic Projection

As shown in the image above, such behavior is called projection. It's obviously difficult to figure out everything at the same time, however, we can ultilize the fact that composition of linear transformation perserves its linearity. We will begin with something called orthographic projection (a method of projection in which an object is depicted or a surface mapped using parallel lines to project its shape onto a plane)

Imagine we have a rectanglular view space, we need to achieve a couple key objectives:

1. We want to put the center of the view space to the origion
2. We want to limit the finished projection coordinate into x ranging from -1 to 1, y ranging from -1 to 1, and z ranging from 0 to 1
3. We want to know which z value comes first (we do not want to render things that is blocked)



To achieve our first objective, we want to ultilize the translation matrix that we discovered previously. In order to successfully perform the translation, we need to know a couple of values, namely, left, right, top, bottom, far, and near. Our translation matrix will then be:

$$T(\frac{r+l}{2}, \frac{b+t}{2}, n) = \begin{bmatrix} 1 & 0 & 0 & \frac{r+l}{2} \\ 0 & 1 & 0 & \frac{b+t}{2} \\ 0 & 0 & 1 & n \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

To achieve our second objective, we want to ultilize the scaling matrix that we initially discovered:

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

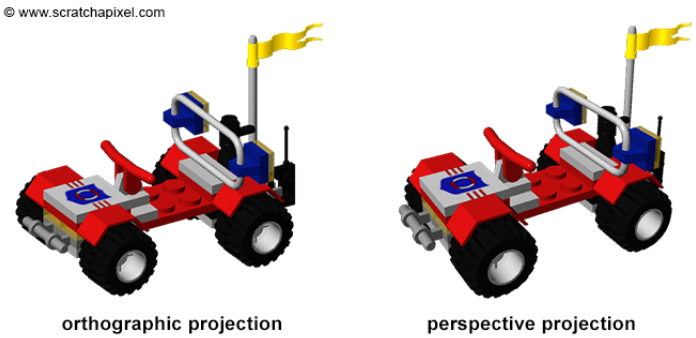
We can also observe that our third objective is already met since we know the exact z value of the projected object (You will see why this is a key objective later)

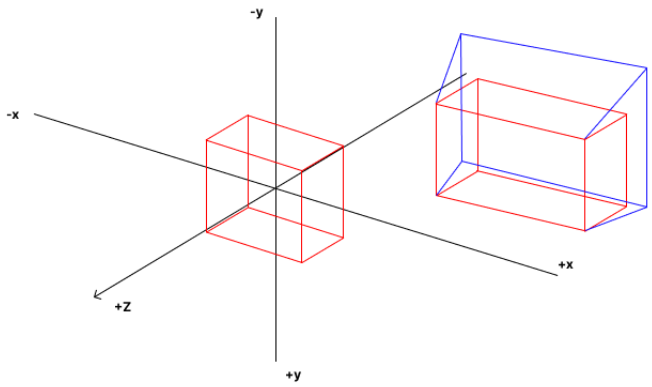
Finally, if we are to composite the matrix, we obtain our orthographic projection matrix:

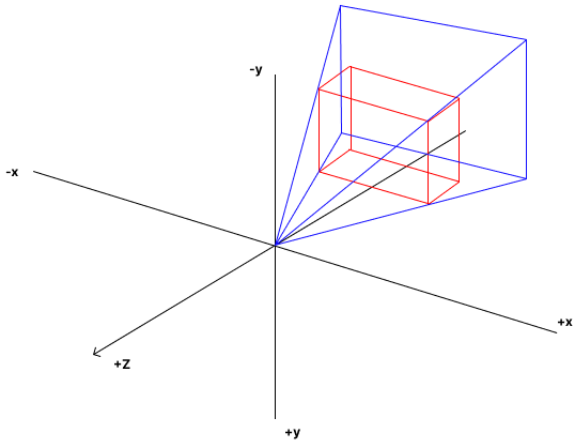
$$T(\frac{r+l}{2}, \frac{b+t}{2}, n) \times S_v(\frac{2}{r-l}, \frac{2}{b-t}, f-n) = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{b-t} & 0 & -\frac{b+t}{b-t} \\ 0 & 0 & \frac{1}{f-n} & -\frac{n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Prospective Projection Matrix (The Hard Part)

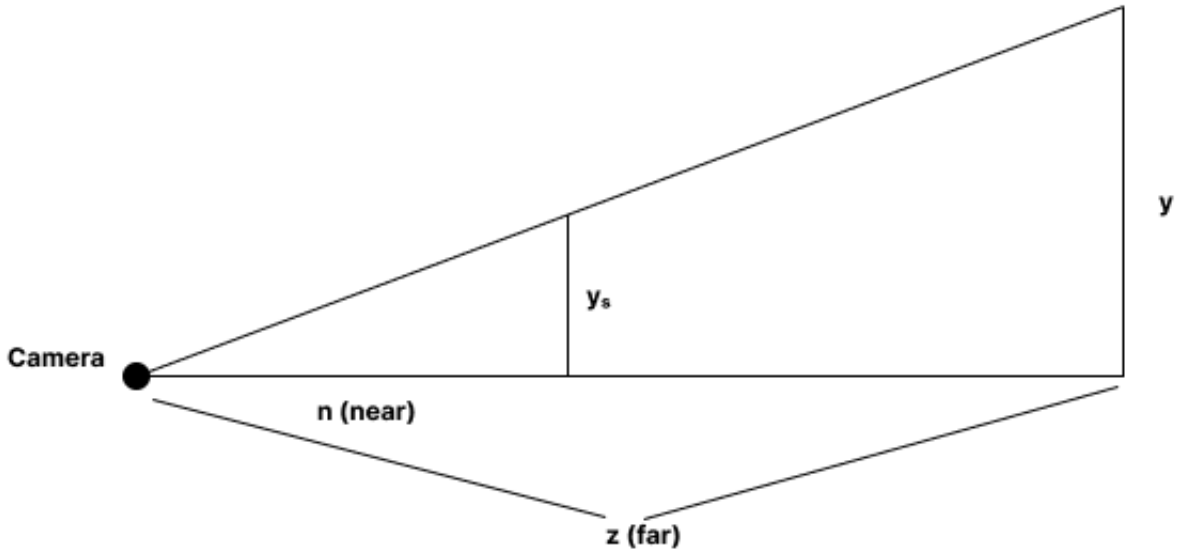
Now instead of a parallel transformation, our view space is a frustum. The difference between orthographic projection vs perspective projection is shown below:







To better understand the problem, we draw a sidevew of the 3D world:



We can observe that there's a pair of similar triangle in this shape. Therefore, we get that:

$$\frac{y_s}{n} = \frac{y}{z}$$
$$y_s = \frac{ny}{z}$$
$$x_s = \frac{nx}{z}$$

Ignoring z for now, we need a matrix such that: $\begin{bmatrix} - & - & - & - \\ - & - & - & - \\ - & - & - & - \\ - & - & - & - \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{nx}{z} \\ \frac{ny}{z} \\ z \\ 1 \end{bmatrix}$ However, we can observe that it is simply not possible for us to get this matrix since it involves division

and we do not know z prior.
Here we introduce anothe property of homogenous coordinates, , which is that we finding the actual x, y, z value of the 4x1 vector, we divide the x, y, and z value by w

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} 2x \\ 2y \\ 2z \\ 2 \end{bmatrix} = \begin{bmatrix} wx \\ wy \\ wz \\ w \end{bmatrix}$$

Here we can ultilize homogenous coordinate: $\begin{bmatrix} - & - & - & - \\ - & - & - & - \\ - & - & - & - \\ - & - & - & - \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} nx \\ ny \\ z^2 \\ z \end{bmatrix} = \begin{bmatrix} \frac{nx}{z} \\ \frac{ny}{z} \\ z \\ 1 \end{bmatrix}$

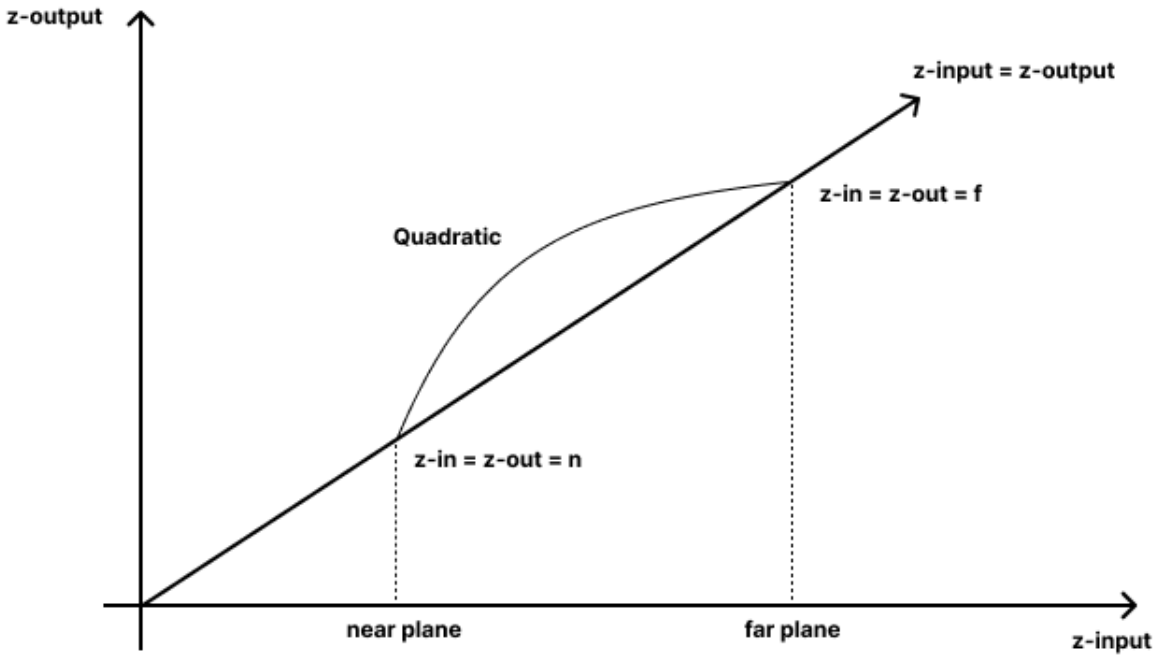
It is easy to find row 1, 2, and 4 for our matrix: $\begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ - & - & - & - \\ 0 & 0 & 1 & 0 \end{bmatrix}$ However, it takes some

derivation to find the third row. First we realize that the first and second term must be 0 since we do not want z to be a variable of x and y, so we can set our 3rd term to be m_1 and 4th term to be m_2 which our matrix becomes: $\begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & m_1 & m_1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} nx \\ ny \\ z^2 \\ z \end{bmatrix}$ and thus $m_1 z + m_2 = z^2$. Among the z-axis, the only two value that we know of(we define of) are the near plane and far plane, which we yield a system of equations:

$$\begin{cases} m_1 n + m_2 = n^2 \\ m_1 f + m_2 = f^2 \end{cases}$$

$$\begin{cases} m_1 = f + n \\ m_2 = -fn \end{cases}$$

We can see that this is a quadratic equation which is only true when $z = n$(near plane) and $z = f$(far plane) To explore the intermediate behavior of z, we have a graph of z-input and z-output below:



Although the result is non-linear and contains error, it doesn't affect our goal:

1. We want to put the center of the view space to the origion
2. We want to limit the finished projection coordinate into x ranging from -1 to 1, y ranging from -1 to 1, and z ranging from 0 to 1
3. We want to know which z value comes first (we do not want to render things that is blocked)

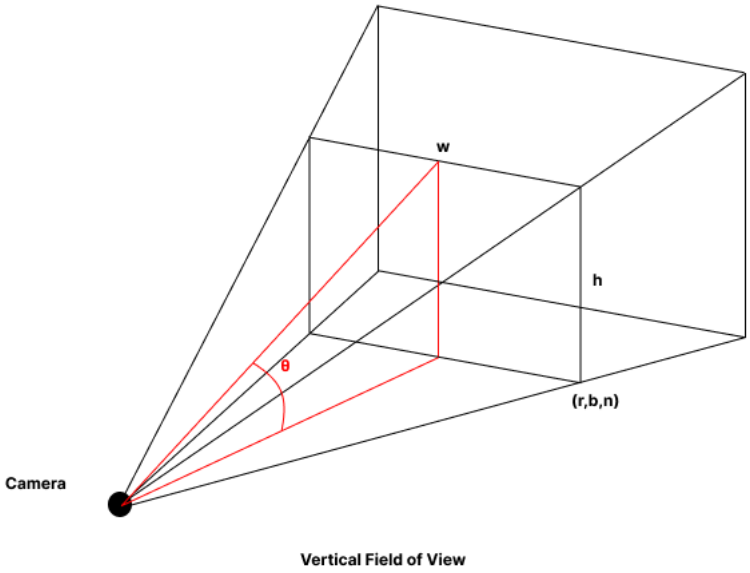
Error in z value doesn't affect the order of z-value, so we can still use the resulting value to calculate which item is blocked.
Therefore, since linearity is perserved when multiplying matrices, we yield our general perspective projection matrix (finally):

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

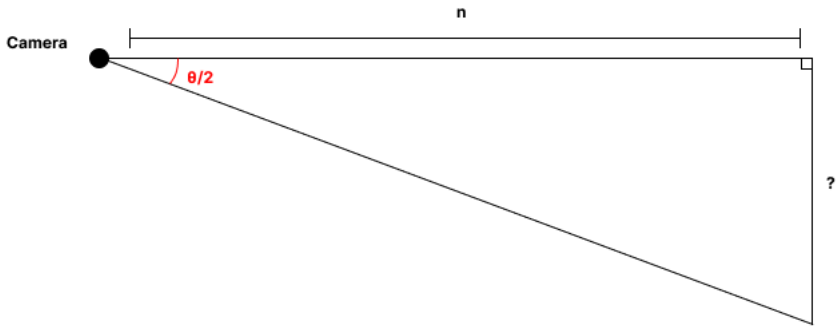
Luckily, it's common for the camera to be place at the very center of the view, which means that $r = -l$ and $t = -b$, therefore, we simplify the projection matrix to be:

$$\begin{bmatrix} \frac{n}{r} & 0 & 0 & 0 \\ 0 & \frac{n}{b} & 0 & 0 \\ 0 & 0 & \frac{f}{f-n} & -\frac{fn}{f-n} \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

We are not over yet however, from our perspective matrix, we can observe that besides the value of near plane and far plane, we also need to know right and bottom of our view, which is not very convenient, instead, we use something called field of view. The field of view is that part of the world that is visible through the camera at a particular position and orientation in space.



Vertical Field of View



By trigonometry, we can easily find that $b = n \cdot tan(\frac{\theta}{2})$, obtaining b, we can multiply the aspect ratio of the screen which is $\frac{w}{h}$, so $r = \frac{nw}{h} \cdot tan(\frac{\theta}{2})$, substituting the values to the projection matrix, we get:

$$P(f, n, \theta) = \begin{bmatrix} \frac{h}{w \cdot tan(\frac{\theta}{2})} & 0 & 0 & 0 \\ 0 & \frac{1}{tan(\frac{\theta}{2})} & 0 & 0 \\ 0 & 0 & \frac{f}{f-n} & -\frac{fn}{f-n} \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

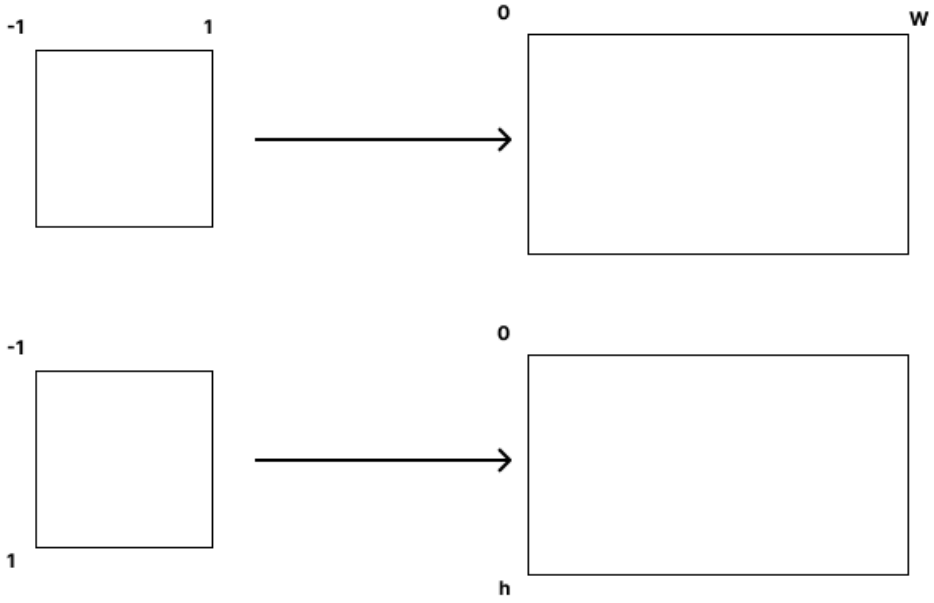
Putting Everything Together

To put everything together, let's recap all the transformation needed:

1. With the user granted the ability to move the camera, we always want the camera to stay at the origin, we must apply translation matrix
2. With the user granted the ability to rotate the camera, we always want the camera to stay at the origin, we must apply rotation matrix
3. Apply the perspective projection matrix

4. In order to convert the normalized device coordinates (x from -1 to 1, y from -1 to 1, z from 0 to 1), we want to apply a viewport transformation matrix

For step 4, we want a matrix that perform such transformation:



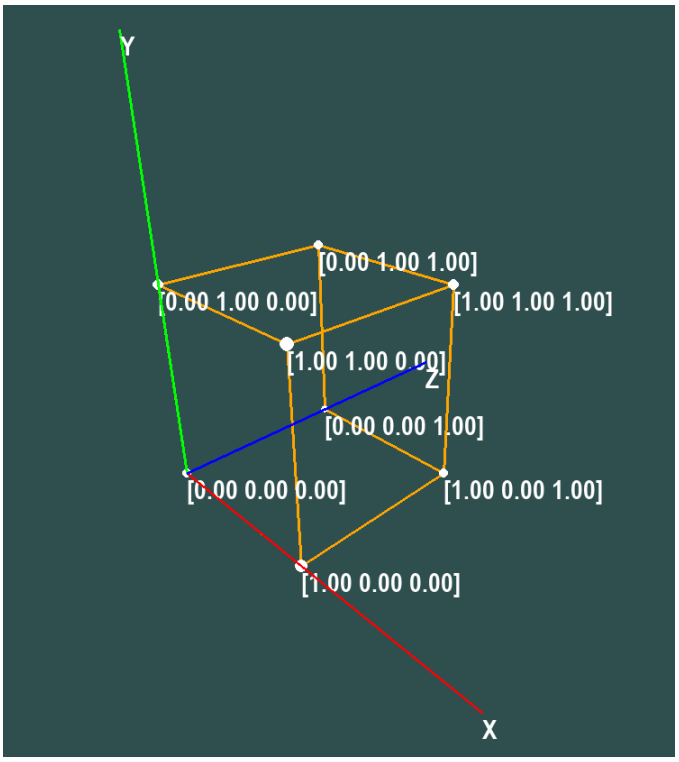
If we observe it closely, this transformation involves a scaling and a translation, which we can then derive the to screen matrix:

$$V(h,w) = \begin{bmatrix} 1 & 0 & 0 & \frac{w}{2} \\ 0 & 1 & 0 & \frac{h}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{w}{2} & 0 & 0 & 0 \\ 0 & -\frac{h}{2} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{w}{2} & 0 & 0 & \frac{w}{2} \\ 0 & -\frac{h}{2} & 0 & \frac{h}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Finally, notice that what we are essentially doing is transforming the object's coordinate into our view, instead of transforming the camera into the object's view. So given the coordinate of a object's vertex $\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$, the to-screen coordinate is:

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = V \times P \times R \times T \times \begin{bmatrix} x \\ y \\ z \\ w' \end{bmatrix}$$

Rendering the x,y points, we get:

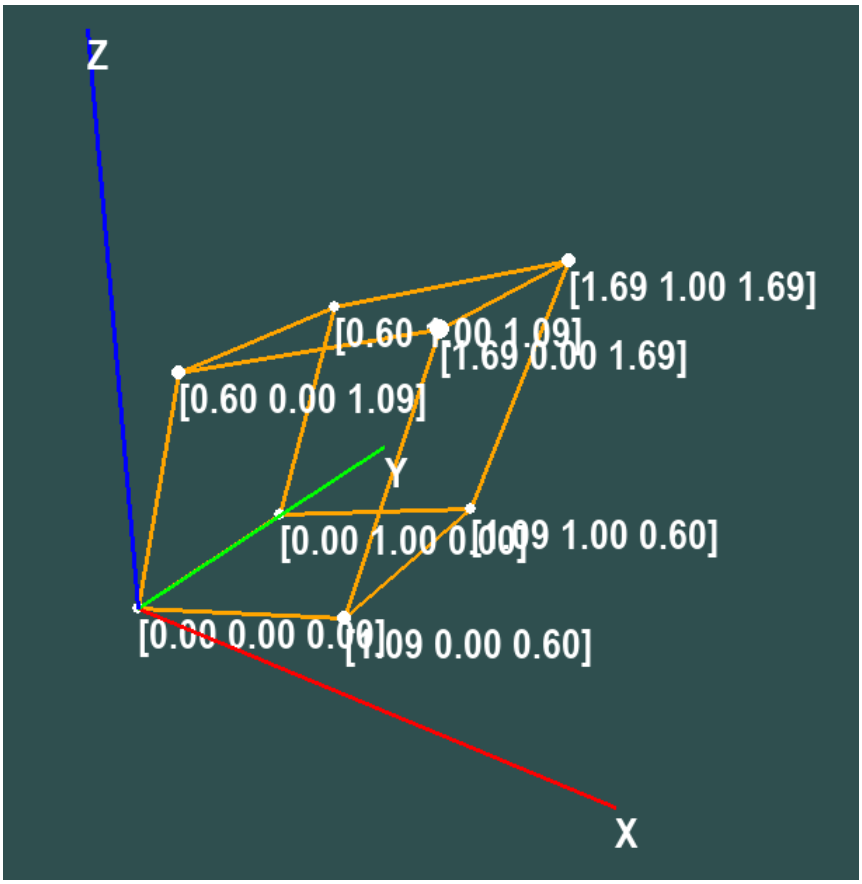


we can see that one issue is by the width, height, depth model, our y axis and z axis is flipped, it's not required, but to make it look more intuitive, we apply another matrix to change its axis:

$$\mathrm{C} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Moreover, if we want to visualize linear transformation A, we can define our transformation matrix A and apply that at the very beginning, which resulted in the final formula:

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = V \times P \times R \times T \times C \times A \times \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



Conclusion

There's a couple of key points at the very end:

1. We cannot change the order of matrix multiplication, and the order is determined in a way such that we transform the object's coordinate into our camera's coordinate
2. We can see the power of linear algebra in this case, it allows us to composite matrix and describe many operations with clearity and elegance
3. Objects that are out of the normalized device coordinate will be clipped
4. We need to normalize the coordinate either after applying P or applying V
5. We need to manually define a near plane and far plane, notice by the quadratic shape, the bigger the difference between them, the higher the precision
6. We did not use z to determine blocking in this case (too hard to code), but we have the data needed to achieve it

7. Animation is achieved by $A' = I + (A - I) \cdot \dfrac{1}{time\ interval}$
8. Different graphics APIs have different z-boundaries, but it doesn't affect computation (OpenGL goes from -1 to 1, DirectX goes from 0 to 1, Vulkan goes from 0 to 1, etc)