OpenGL Projection Matrix

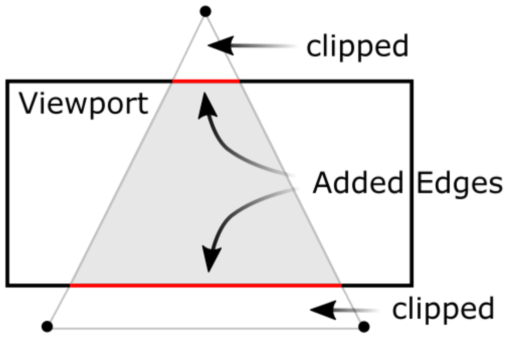
Related Topics: OpenGL Transformation, OpenGL Matrix

Updates: The MathML version is available here.

Overview

A computer monitor is a 2D surface. A 3D scene rendered by OpenGL must be projected onto the computer screen as a 2D image. GL_PROJECTION matrix is used for this projection transformation. First, it transforms all vertex data from the eye coordinates to the clip coordinates. Then, these clip coordinates are also transformed to the normalized device coordinates (NDC) by dividing with *w* component of the clip coordinates.

Therefore, we have to keep in mind that both clipping (frustum culling) and NDC transformations are integrated into **GL_PROJECTION** matrix. The following sections describe how to build the projection matrix from 6 parameters; *left*, *right*, *bottom*, *top*, *near* and *far* boundary values.



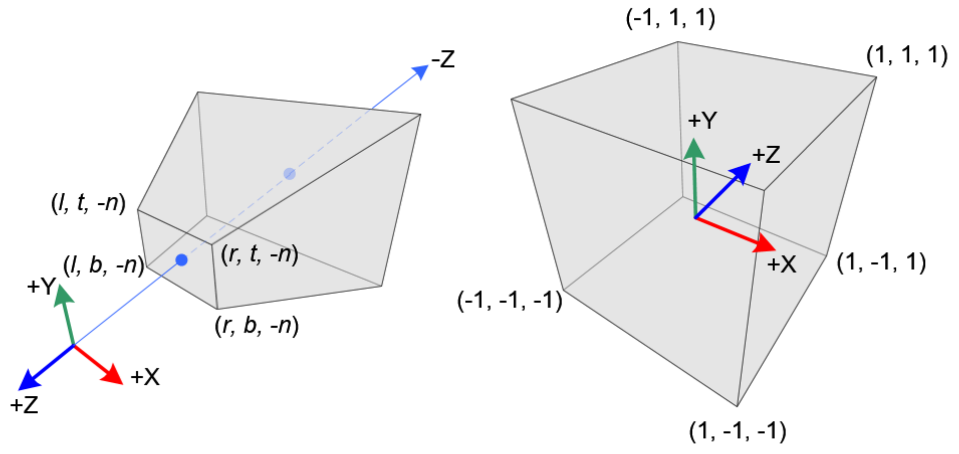
A triangle clipped by frustum

Note that the frustum culling (clipping) is performed in the clip coordinates, just before dividing by w_c. The clip coordinates, x_c, y_c and z_c are tested by comparing with w_c. If any clip coordinate is less than $-w_c$, or greater than w_c, then the vertex will be discarded.

$$-w_c < x_c, y_c, z_c < w_c$$

Then, OpenGL will reconstruct the edges of the polygon where clipping occurs.

Perspective Projection

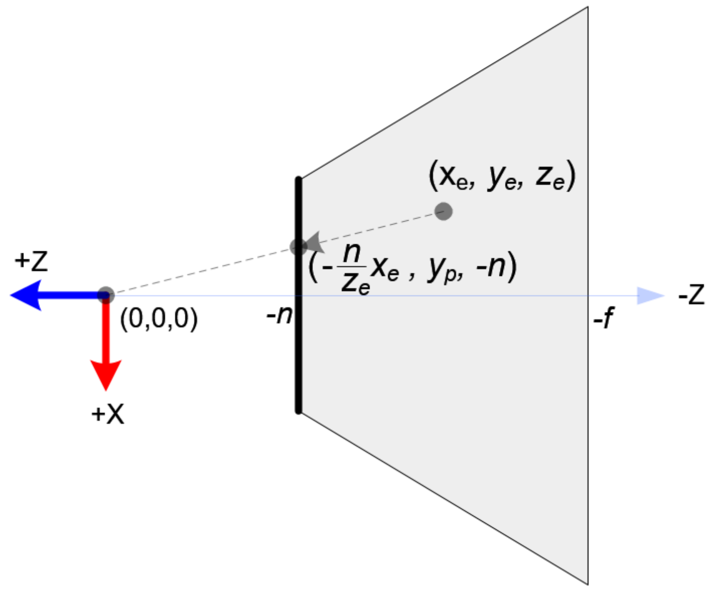


Perspective Frustum and Normalized Device Coordinates (NDC)

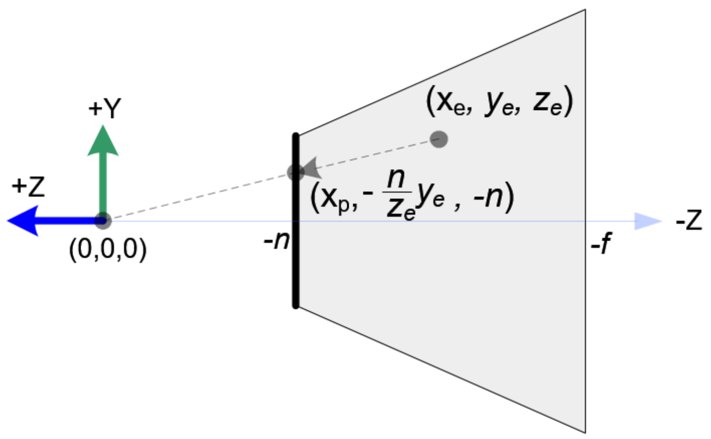
In perspective projection, a 3D point in a truncated pyramid frustum (eye coordinates) is mapped to a cube (NDC); the range of x-coordinate from [l, r] to [-1, 1], the y-coordinate from [b, t] to [-1, 1] and the z-coordinate from [-n, -f] to [-1, 1].

Note that the eye coordinates are defined in the right-handed coordinate system, but NDC uses the left-handed coordinate system. That is, the camera at the origin is looking along -Z axis in eye space, but it is looking along +Z axis in NDC. Since **glFrustum()** accepts only positive values of *near* and *far* distances, we need to negate them during the construction of GL_PROJECTION matrix.

In OpenGL, a 3D point in eye space is projected onto the *near* plane (projection plane). The following diagrams show how a point (x_e, y_e, z_e) in eye space is projected to (x_p, y_p, z_p) on the *near* plane.



Top View of Frustum



Side View of Frustum

From the top view of the frustum, the x-coordinate of eye space, x_e is mapped to x_p, which is calculated by using the ratio of similar triangles;

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

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

From the side view of the frustum, y_p is also calculated in a similar way;

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

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

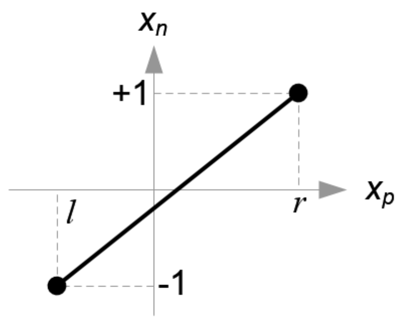
Note that both x_p and y_p depend on z_e; they are inversely propotional to $-z_e$. In other words, they are both divided by $-z_e$. It is a very first clue to construct GL_PROJECTION matrix. After the eye coordinates are transformed by multiplying GL_PROJECTION matrix, the clip coordinates are still a homogeneous coordinates. It finally becomes the normalized device coordinates (NDC) by divided by the w-component of the clip coordinates. (*See more details on OpenGL Transformation.*)

$$\begin{pmatrix} x_{clip} \\ y_{clip} \\ z_{clip} \\ w_{clip} \end{pmatrix} = M_{projection} \cdot \begin{pmatrix} x_{eye} \\ y_{eye} \\ z_{eye} \\ w_{eye} \end{pmatrix}, \quad \begin{pmatrix} x_{ndc} \\ y_{ndc} \\ z_{ndc} \end{pmatrix} = \begin{pmatrix} x_{clip}/w_{clip} \\ y_{clip}/w_{clip} \\ z_{clip}/w_{clip} \end{pmatrix}$$

Therefore, we can set the w-component of the clip coordinates as $-z_e$. And, the 4th of GL_PROJECTION matrix becomes (0, 0, -1, 0).

$$\begin{pmatrix} x_c \\ y_c \\ z_c \\ w_c \end{pmatrix} = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} x_e \\ y_e \\ z_e \\ w_e \end{pmatrix}, \qquad \therefore w_c = -z_e$$

Next, we map x_p and y_p to x_n and y_n of NDC with linear relationship; [l, r] ⇒ [-1, 1] and [b, t] ⇒ [-1, 1].



Mapping from x_p to x_n

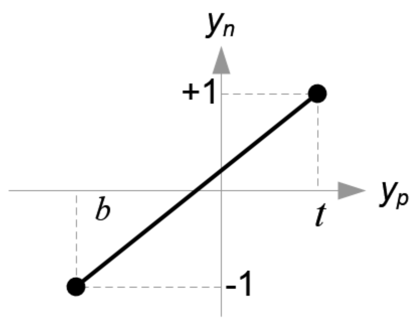
$$x_n = \frac{1-(-1)}{r-l} \cdot x_p + \beta$$

$$1 = \frac{2r}{r-l} + \beta \qquad \text{(substitute } (r, 1) \text{ for } (x_p, x_n))$$

$$\beta = 1 - \frac{2r}{r-l} = \frac{r-l}{r-l} - \frac{2r}{r-l}$$

$$= \frac{r-l-2r}{r-l} = \frac{-r-l}{r-l} = -\frac{r+l}{r-l}$$

$$\therefore x_n = \frac{2x_p}{r-l} - \frac{r+l}{r-l}$$



Mapping from y_p to y_n

$$y_n = \frac{1-(-1)}{t-b} \cdot y_p + \beta$$

$$1 = \frac{2t}{t-b} + \beta \qquad \text{(substitute } (t, 1) \text{ for } (y_p, y_n))$$

$$\beta = 1 - \frac{2t}{t-b} = \frac{t-b}{t-b} - \frac{2t}{t-b}$$

$$= \frac{t-b-2t}{t-b} = \frac{-t-b}{t-b} = -\frac{t+b}{t-b}$$

$$\therefore y_n = \frac{2y_p}{t-b} - \frac{t+b}{t-b}$$

Then, we substitute x_p and y_p into the above equations.

$$\begin{aligned} x_n &= \frac{2x_p}{r-l} - \frac{r+l}{r-l} \qquad \left(x_p = \frac{nx_e}{-z_e}\right) \\ &= \frac{2 \cdot \frac{n \cdot x_e}{-z_e}}{r-l} - \frac{r+l}{r-l} \\ &= \frac{2n \cdot x_e}{(r-l)(-z_e)} - \frac{r+l}{r-l} \\ &= \frac{\frac{2n}{r-l} \cdot x_e}{-z_e} - \frac{r+l}{r-l} \\ &= \frac{\frac{2n}{r-l} \cdot x_e}{-z_e} + \frac{\frac{r+l}{r-l} \cdot z_e}{-z_e} \\ &= \underbrace{\left(\frac{2n}{r-l} \cdot x_e + \frac{r+l}{r-l} \cdot z_e\right)}_{x_c} \Big/ - z_e \end{aligned}$$

$$\begin{aligned} y_n &= \frac{2y_p}{t-b} - \frac{t+b}{t-b} \qquad \left(y_p = \frac{ny_e}{-z_e}\right) \\ &= \frac{2 \cdot \frac{n \cdot y_e}{-z_e}}{t-b} - \frac{t+b}{t-b} \\ &= \frac{2n \cdot y_e}{(t-b)(-z_e)} - \frac{t+b}{t-b} \\ &= \frac{\frac{2n}{t-b} \cdot y_e}{-z_e} - \frac{t+b}{t-b} \\ &= \frac{\frac{2n}{t-b} \cdot y_e}{-z_e} + \frac{\frac{t+b}{t-b} \cdot z_e}{-z_e} \\ &= \underbrace{\left(\frac{2n}{t-b} \cdot y_e + \frac{t+b}{t-b} \cdot z_e\right)}_{y_c} \Big/ - z_e \end{aligned}$$

Note that we make both terms of each equation divisible by -z_e for perspective division (x_c/w_c, y_c/w_c). And we set w_c to -z_e earlier, and the terms inside parentheses become x_c and y_c of the clip coordiantes.

From these equations, we can find the 1st and 2nd rows of GL_PROJECTION matrix.

$$\begin{pmatrix} x_c \\ y_c \\ z_c \\ w_c \end{pmatrix} = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} x_e \\ y_e \\ z_e \\ w_e \end{pmatrix}$$

Now, we only have the 3rd row of GL_PROJECTION matrix to solve. Finding z_n is a little different from others because z_e in eye space is always projected to -n on the near plane. But we need unique z value for the clipping and depth test. Plus, we should be able to unproject (inverse transform) it. Since we know z does not depend on x or y value, we borrow w-component to find the relationship between z_n and z_e. Therefore, we can specify the 3rd row of GL_PROJECTION matrix like this.

$$\begin{pmatrix} x_c \\ y_c \\ z_c \\ w_c \end{pmatrix} = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & A & B \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} x_e \\ y_e \\ z_e \\ w_e \end{pmatrix}, \qquad z_n = z_c / w_c = \frac{Az_e + Bw_e}{-z_e}$$

In eye space, w_e equals to 1. Therefore, the equation becomes;

$$z_n = \frac{Az_e + B}{-z_e}$$

To find the coefficients, *A* and *B*, we use the (z_e, z_n) relation; (-n, -1) and (-f, 1), and put them into the above equation.

$$\begin{cases} \dfrac{-An+B}{n} = -1 \\ \\ \dfrac{-Af+B}{f} = 1 \end{cases} \rightarrow \begin{cases} -An+B = -n & (1) \\ -Af+B = f & (2) \end{cases}$$

To solve the equations for *A* and *B*, rewrite eq.(1) for B;

$$B = An - n \qquad (1\text{'})$$

Substitute eq.(1') to *B* in eq.(2), then solve for A;

$$-Af + (An - n) = f \qquad (2)$$

$$-(f-n)A = f + n$$

$$A = -\frac{f+n}{f-n}$$

Put *A* into eq.(1) to find *B*;

$$\left(\frac{f+n}{f-n}\right)n + B = -n \qquad (1)$$

$$B = -n - \left(\frac{f+n}{f-n}\right)n = -\left(1 + \frac{f+n}{f-n}\right)n = -\left(\frac{f-n+f+n}{f-n}\right)n$$

$$= -\frac{2fn}{f-n}$$

We found *A* and *B*. Therefore, the relation between z_e and z_n becomes;

$$z_n = \frac{-\frac{f+n}{f-n} z_e - \frac{2fn}{f-n}}{-z_e} \qquad (3)$$

Finally, we found all entries of GL_PROJECTION matrix. The complete projection matrix is;

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

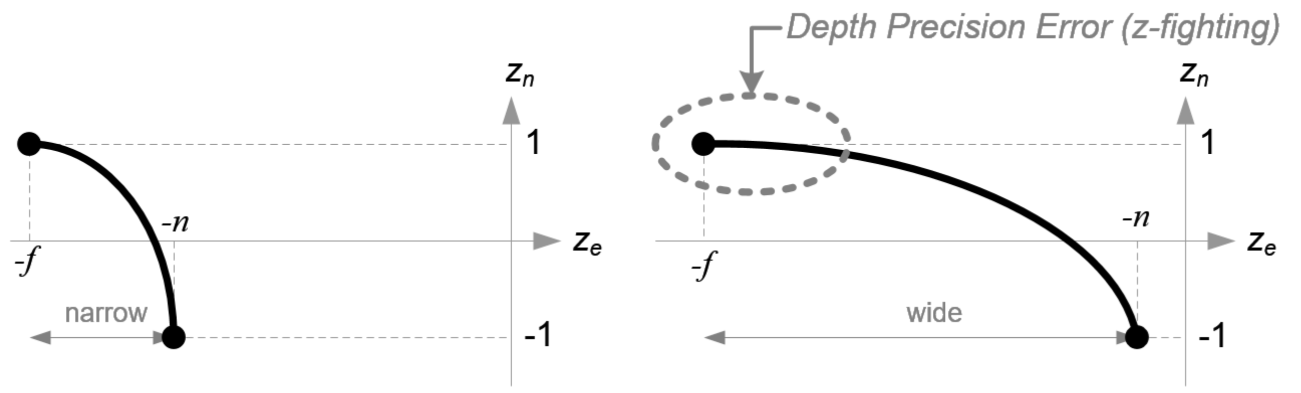
OpenGL Perspective Projection Matrix

This projection matrix is for a general frustum. If the viewing volume is symmetric, which is $r = -l$ and $t = -b$, then it can be simplified as;

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

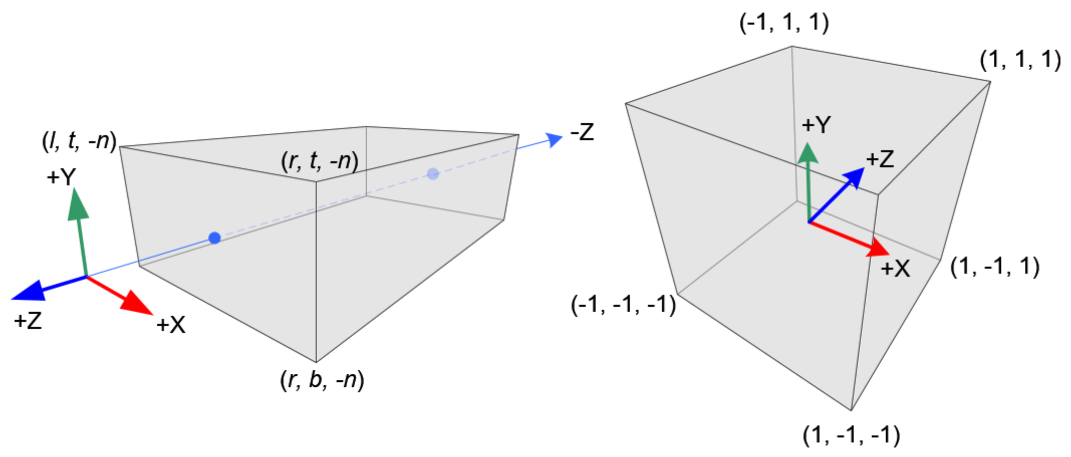
$$\begin{pmatrix} \frac{n}{r} & 0 & 0 & 0 \\ 0 & \frac{n}{t} & 0 & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Before we move on, please take a look at the relation between z_e and z_n, eq.(3) once again. You notice it is a rational function and is non-linear relationship between z_e and z_n. It means there is very high precision at the *near* plane, but very little precision at the *far* plane. If the range [-n, -f] is getting larger, it causes a depth precision problem (z-fighting); a small change of z_e around the *far* plane does not affect on z_n value. The distance between *n* and *f* should be short as possible to minimize the depth buffer precision problem.



Comparison of Depth Buffer Precisions

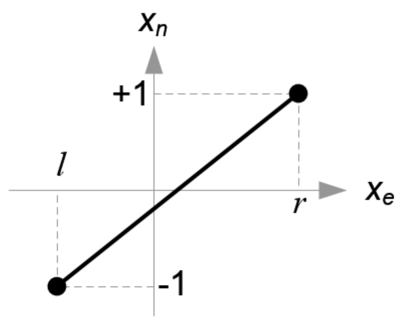
Orthographic Projection



Orthographic Volume and Normalized Device Coordinates (NDC)

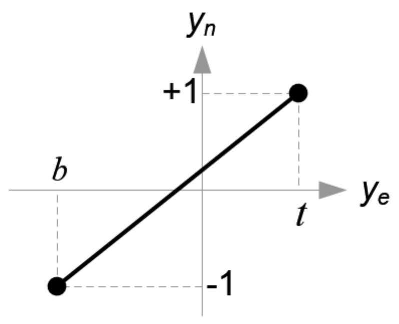
Constructing GL_PROJECTION matrix for orthographic projection is much simpler than perspective mode.

All x_e, y_e and z_e components in eye space are linearly mapped to NDC. We just need to scale a rectangular volume to a cube, then move it to the origin. Let's find out the elements of GL_PROJECTION using linear relationship.



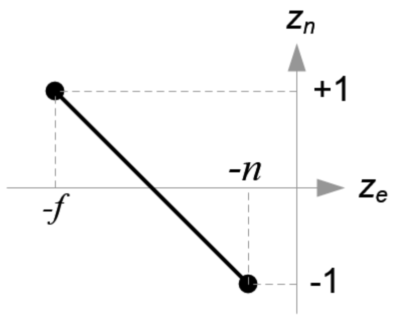
Mapping from x_e to x_n

$$
\begin{aligned}
x_n &= \frac{1-(-1)}{r-l} \cdot x_e + \beta \\
1 &= \frac{2r}{r-l} + \beta \qquad (\text{substitute } (r,1) \text{ for } (x_e, x_n)) \\
\beta &= 1 - \frac{2r}{r-l} = -\frac{r+l}{r-l} \\
\therefore x_n &= \frac{2}{r-l} \cdot x_e - \frac{r+l}{r-l}
\end{aligned}
$$



Mapping from y_e to y_n

$$
\begin{aligned}
y_n &= \frac{1-(-1)}{t-b} \cdot y_e + \beta \\
1 &= \frac{2t}{t-b} + \beta \qquad (\text{substitute } (t,1) \text{ for } (y_e, y_n)) \\
\beta &= 1 - \frac{2t}{t-b} = -\frac{t+b}{t-b} \\
\therefore y_n &= \frac{2}{t-b} \cdot y_e - \frac{t+b}{t-b}
\end{aligned}
$$



Mapping from z_e to z_n

$$
\begin{aligned}
z_n &= \frac{1-(-1)}{-f-(-n)} \cdot z_e + \beta \\
1 &= \frac{2f}{f-n} + \beta \qquad (\text{substitute } (-f, 1) \text{ for } (z_e, z_n)) \\
\beta &= 1 - \frac{2f}{f-n} = -\frac{f+n}{f-n} \\
\therefore z_n &= \frac{-2}{f-n} \cdot z_e - \frac{f+n}{f-n}
\end{aligned}
$$

Since w-component is not necessary for orthographic projection, the 4th row of GL_PROJECTION matrix remains as (0, 0, 0, 1). Therefore, the complete GL_PROJECTION matrix for orthographic projection is;

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

OpenGL Orthographic Projection Matrix

It can be further simplified if the viewing volume is symmetrical, $r = -l$ and $t = -b$.

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

$$\begin{pmatrix} \frac{1}{r} & 0 & 0 & 0 \\ 0 & \frac{1}{t} & 0 & 0 \\ 0 & 0 & \frac{-2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

←Back

Stéphane Champailler • 4 months ago • edited

Hello, nice explanantion. However, I find the fiugre of the perspective projection unclear. In the perspective frustum (left part), the eye is located at the origin, which can be further guessed by looking at the figure titled "
Note that the eye coordinates are defined in the right-handed coordinate
system, but NDC uses the left-handed coordinate system. That is, the
camera at the origin..." I guess that the eye is at (0,0,0) but somehow it's a bit disturbing to me.
IOW, in the perspective frustum, the full frustum volume is in front of the eye whereas in the NDC, only half of it seems so, which is a bit confusing.

1 ^ | ˅ • Reply • Share ›

songho Mod ➦ Stéphane Champailler • a month ago

Yes, the eye is at (0,0,0) on the left-side image. However, please focus on the viewing frustum in the diagram, which any objects in this volumn will be rendered. Otherwise clipped. And forget about the position of the eye. (Technically, the eye is outside of the volumn, so it is clipped out, and is not at the center of the cube.)

^ | ˅ • Reply • Share ›

Hirosam • 4 months ago

Hey songho! Thanks a lot for making this paper! I could actually understand how a projection matrix is formed with this! Thanks again! I got excited and made this graph that shows visually the difference between the non-linear and a theoretical linear of the ze and zn relation. It is really fun, and you can find interesting things messing with it around.

The only thing I am still not too sure, is how would I create a frustum given FOV (Field of View) angle. Do you have any tips? Thanks!

1 ^ | ˅ • Reply • Share ›

songho Mod ➦ Hirosam • 4 months ago • edited

@Hirosam, thank you for sharing th graph for the depth buffer. It is a great idea and useful to understand it.

You can compute the viewing frustum using the FOV and aspect ratio (width/height). Please check the code snippet at the bottom of OpenGL Transformation page.

1 ^ | ˅ • Reply • Share ›

Ruotong Jia • 6 months ago

Hi Songho, great effort in putting all these notes together. Could you please put together the basic formulation of the problem, such as what is near, what is far plane, what is the clipped frame, etc.? These fundamental concepts will make the logic flow complete, and the notes will be more beginner friendly. Thanks

1 ^ | ⌄ • Reply • Share ›

songho Mod ➦ Ruotong Jia • 5 months ago

Thank you for the suggesions. I will add more fundamental concept.

^ | ⌄ • Reply • Share ›

lonely • 6 months ago

hi songho. Thanks for your articles. However in order to fully understand, i want to know reason why we can set "w-component of the clip coordinates as -ze". i can understand similar triangle stuff and homogeneous coordinates but can't understand why Wc=-Ze with just knowing Xp and Yp depend on Ze and they are inversely propotional to -Ze.

1 ^ | ⌄ • Reply • Share ›

songho Mod ➦ lonely • 5 months ago • edited

Consider it as homogeneous coordinates (x,y,w) of 2D (X,Y).
X = x/w
Y = y/w
And, the negative sign comes from the eye space; all visible vertices are defined where Z is negative.

^ | ⌄ • Reply • Share ›

S B • 7 months ago • edited

Hi songho, thanks for sharing your knowledge! much appreciated. I just wanted to clarify my understanding. The projection matrix basically scales and flips the world coords from the pov of the camera into a 2x2x2 cube. When it flips the z coord it converts it from a right hand to left hand system correct? And during the perspective divide is there any possibility of Wc being zero? I dont think thats an issue is it? Thanks for your help.

1 ^ | ⌄ • Reply • Share ›

songho Mod ➦ S B • 6 months ago

Yes, OpenGL defines NDC in left-handed, so z-coord is flipped. Since the near and far distances are positive values, and the visible vertices are within the near and far planes, the Z coords cannot be zero, where Wc = -Ze.

1 ^ | ⌄ • Reply • Share ›

S B ➦ songho • 6 months ago • edited

thanks :)

^ | ⌄ • Reply • Share ›

flashTHREE • 7 months ago • edited

Is it possible to say that we need to transform Ze range of [-n, -f] to Zclip range of [-w,w]? In that case Zclip = -(-2*Ze/(f-n) - (f+n)/(f-n))*Ze will do the job. It's a different formula but it also works:
Zndc = Zclip/w = -2*Ze/(f-n) - (f+n)/(f-n) will be in a range [-1,1].

^ | ⌄ • Reply • Share ›

Alessandro Bellia • 8 months ago • edited

Hi, shouldn't 'l' be defined as a positive number? So, when you demonstrate the mapping from xp to xn, shouldn't it be "r+l" (i.e. r - (-l)) instead of "r-l"? Thanks in advance.
EDIT: Never mind. According to the pictures, 'l' is actually negative xd

1 ^ | ⌄ • Reply • Share ›

Steven • 8 months ago

Hi songho, I would really like to use your second image that is labeled "perspective frustum and NDC" for my school project! Could I have permission to use it for my project?

1 ^ | ⌄ • Reply • Share ›

songho Mod ➦ Steven • 8 months ago

Yes, you can use the diagrams in this page for your project without any restrictions.

^ | ⌄ • Reply • Share ›

Shep ➦ songho • 8 months ago

Thank you!

^ | ⌄ • Reply • Share ›

lin zhu • 8 months ago

Hi,I really like your post! Thanks for sharing your knowlege!!!

But I am confused by your expression, you mentioned " we have to keep in mind that both clipping (frustum culling) and NDC transformations are integrated into GL_PROJECTION matrix. " in the overview, and in my view,the projection matrix only transform eye coordinates to clipping coordinates,and the perspective division just does the NDC transformations.

Maybe I misunderstood your meaning.
Thanks for the answer :)

1 ^ | ˅ • Reply • Share ›

songho Mod ↪ lin zhu • 8 months ago

I meant the elements in the projection matrix are calculated for the perspective division as well, so NDC are nicely fit between -1 and +1 after the division.

^ | ˅ • Reply • Share ›

Matt • 8 months ago

Hi, I love the article really helped me to get better grasp of the process.

However in order to fully understand, I believe it is more important to understand why you do certain steps not how. I understand why you do most of the steps the only one I struggle with is, why do you map x_p and y_p to x_n and y_n. What does this step acomplish? Why are we maping to (-1,1)?

The only reason I could think off is, that we map to (-1,1) because later we "leave out" the division by -z which is accomplished later by dividing with w_c, so we are effectively maping to (-1 * w_c, 1 * w_c), but it seems like a stretch.
Thanks for the answer

1 ^ | ˅ • Reply • Share ›

songho Mod ↪ Matt • 8 months ago • edited

Since the pyramid frustum becomes a cube, 6 clipping plane equations becomes simpler for the clipping process; Xc+Wc=0, Yc+Wc=0, Zc+Wc=0, and so on. (All clipping planes are perpendicular to axes)

Note that OpenGL performs the clipping processs right after multiplying the projection matrix but before NDC, so Xc, Yc, Zc and Wc are

Note that OpenGL performs the clipping process right after multiplying the projection matrix but before NDC, so Xc, Yc, Zc and Wc are clip coordinates.

^ | ⌄ • Reply • Share ›

Matt ➦ songho • 8 months ago

Yeah I did some more digging and I belive I understand now.

I think the mistake I was making was, that I thought that the transition from x_e to x_p already maps the to the projection plane and clips it but it is not the case. So I belive that by maping x_p to x_n we are specifying the cliping plane. Am I correct?

1 ^ | ⌄ • Reply • Share ›

songho Mod ➦ Matt • 8 months ago

Yes, mapping Xp to Xn is a part of constructing the projection matrix, and this projection matrix is used for both clipping (frustum culling) and normalization (NDC).

^ | ⌄ • Reply • Share ›

Ahbar Siddiqui • 9 months ago • edited

Hey can you tell me whats the difference here between projected coordinates and clipped coordinates it feels to me like they are both the same thing in a specified range and that is for n =1??

2 ^ | ⌄ • Reply • Share ›

songho Mod ➦ Ahbar Siddiqui • 9 months ago

In OpenGL, the clip coordinate is the result of multiplying the projection matrix. And it is normalized -1 ~ +1. (because it is used for clipping)

The projected coordinates is normally defined as a 2D plane, however the clip coordinates are still 4D.

1 ^ | ⌄ • Reply • Share ›

中梓星音 • 9 months ago

Hello! Thanks very much for your paper. It was amazing!
Could you please also add the details of infinity far plane projection? It is provided in GLM as glm::infinitePerspective.

1 ^ | ⌄ • Reply • Share ›

songho Mod ↪ 中梓星音 • 9 months ago

Thank you for the suggestion. I will add an additional note for this.

| 1 • Reply • Share ›

Brandon Mak • a year ago

Hello Songho! I love this article. Every single step is so well explained (from view space all the way to NDC). But I failed to understand 1 part:

1) I can see how Xn and Yn are reliant on Ze. Geometrically, this is quite easy to visualize. However, I cannot understand why Zn is reliant on We. I understand Zn HAS to rely on some other coordinate other than itself because we need a unique z-value. But the dependence on We seems to be very arbitrary and I fail to find a geometric representation for it. Could you perhaps explain why there is this dependence and how it was derived?

1 | • Reply • Share ›

songho Mod ↪ Brandon Mak • a year ago • edited

When a 3D point is projected onto a screen, it becomes a 2D point, losing 3rd dimension. So, we need a 4th dimension in order to see 3D (*4D is projected to 3D*).

This is how a human eye can see 3D, and we use Homogeneous Coordinates to solve it mathematically by adding one additional coordinate w, for a 3D point, it can be (x, y, z, w) in Homogeneous space. And, this point is converted back to Cartesian by dividing by w-coord, (x/w, y/w, z/w). Notice the division by w for every component. That is what we are doing here for the perspective projection, and we simply set the w to -z.

Please see the following 2D Homogeneous diagram

see more

2 ^ | ˅ • Reply • Share ›

Brandon Mak ➦ songho • a year ago

Is that why there's a coefficient 'B' when trying to find Zn? I can understand from your explanation why it needs to divide by Wc (which is -Ze), but the "+BWe" is what confuses me. It seems like that's a different w value (and it's always 1).

1 ^ | ˅ • Reply • Share ›

songho Mod ➦ Brandon Mak • a year ago • edited

Yes, **We** is 1 for Cartesian coordinate, so **Zn** equation becomes;
Zn = (AZe + B) / -Ze
And, **B** is depending on the near/far values.

However, you can play with the w value of a vertex, for example, if you set w=2, then it simulates the vertex twice away from the camera.

1 ^ | ˅ • Reply • Share ›

Brandon Mak ➦ songho • a year ago

Ahhhh okay! This makes alot more sense now. Thank you!

^ | ˅ • Reply • Share ›

Fabian Hachenberg • a year ago • edited

About
z-fighting: Is it really enough to simply minimize |f - n|? Shouldn't I also move f beyond the farthest object so the part of the curve where the slope is vanishing is not actually used in my scene?

1 ^ | ˅ • Reply • Share ›

songho Mod ➦ Fabian Hachenberg • a year ago • edited

It should be helpful for avoiding z-fighting if a whole scene is closer the near plane or an extra far value beyond the farthest object. But, it is still depending on the distance between f and n.
If the range (f-n) is really big, it still suffers from z-fighting. And, the far plane is used to clip the scene (to reduce the number of polys to

render), so it may produce a rendering performance issue and improper clipping problem.

1 ^ | v • Reply • Share ›

Beez Zaaen • a year ago

Why do you take near and far planes as negative when they should always be positive?You could do the whole calculation on the positive z space right?

1 ^ | v • Reply • Share ›

songho Mod ➦ Beez Zaaen • a year ago • edited

The near and far values of glFrustum() are the distances from the camera, not the actual coordinates. Since the camera is located at the orgin and is looking along -Z axis, the actual coordinates of the near and far planes are (0,0,-n) and (0,0,-f) respectively.

2 ^ | v • Reply • Share ›

beez zaen • a year ago

why do you take near and far planes as negative when they should always be positive?

1 ^ | v • Reply • Share ›

Yingnan Wang ➦ beez zaen • 9 months ago

Camera is assumed to be placed at (0,0,0) and look toward the -z direction in right hand system

1 ^ | v • Reply • Share ›

Ken Brooks • a year ago

Songho - this makes a lot of sense, except for one thing: when doing simulated scenery, it seems to me that I would really want the depth to be infinite: no far clipping plane. I want mountains, planets, stars, to be visible as well as nearby objects. Presuming that the scenery can be designed intelligently so as to avoid z-fighting among distant objects.

How are these needs typically handled?

1 ^ | v • Reply • Share ›

songho Mod ➦ Ken Brooks • a year ago

Please search "infinite projection matrix" or "infinite far plane" to find how to construct the projection matrix. Z-fighting is simply a

Please search "infinite projection matrix" or "infinite far plane" to find how to construct the projection matrix. Z-fighting is simply a precision error of floating-point numbers. Double precision float should overcome this issue, but there are some techniques to avoid it.

1 ^ | ∨ • Reply • Share ›

cv-learn ➦ Ken Brooks • a year ago

I'm not Songho, but in case you need a quick answer or a reference to look at, I suggest looking up for infinitePerspective() function in GLM library. This function creates a perspective projection for view frustum with far plane with z = infinity.

1 ^ | ∨ • Reply • Share ›

songho Mod ➦ cv-learn • a year ago

Thank you for pointing out GLM library. I will take a look.

^ | ∨ • Reply • Share ›

Goktug • a year ago • edited

The simulation platform in which I and my team work benefits from OpenGL, but the other platform which we intend to integrate with that simulation platform performs camera calibration operations by following the multiplication of intrinsic and extrinsic matrices in following formula:
w[u, v, 1] = K * E * [X, Y, Z, 1]

u and v are window (pixel) coordinates. K and E intrinsic and extrinsic matrices, while w refers to depth scale. By reading this OpenGL tutorial carefully and making some calculations, I can derive intrinsic and extrinsic matrices. The problem is depth scale. I guess that depth scale in that formula actually refers to clip coordinate wc, since If we divide the formula above by 1/w, right hand side of the equation is scaled and normalized just like in NDC computation in this paper.

However, the second platform that we will use requests us the value of w as a parameter. what is the exact value of w (guessing exact value of wc) according to this paper ? I cannot find. Please help me.

1 ^ | ∨ • Reply • Share ›

eduardtejerocirera • a year ago

Hi **@songho**! I would like to know why we choose to use the **zn = (A·ze + B) / -ze** fromula (which results in a non-linear function, as you show in the graph) instead of using a linear function (maybe just **zn = A·ze + B**), which would mantain the prescision constant between near and far planes.
What is the reason for choosing this non-linear formula? I know GPU divides our point by we (-ze in our case) and, thus, we're somehow forced to have a -ze in the denominator, but I guess we could somehow avoid it. Thanks!

1 ^ | ⌄ • Reply • Share ›

songho Mod ↪ eduardtejerocirera • a year ago • edited

The key is understanding Homogeneous coordinate; a 3D point (x,y,z) in Cartesian is (x,y,z,w) in Homogeneous. And, a homegeneous coord is converted back to Cartesian by dividing by w component; (x/w, y/w, z/w).

-Ze is actually the w-component in homogeneous, and dividing by -Ze is converting from homegeneous to 3D cartesian.

It is necessary because we project 3D points onto a 2D plane. Without homegeneous, we lose the z-component, which is the depth information.

4 ^ | ⌄ • Reply • Share ›

eduardtejerocirera ↪ songho • a year ago

I understand why we need to do the division by -ze (to project 3D points onto 2D screen). What I'd like to know is why we map x and y coordinates linearly (as you demonstrate) and z NON-linearly. That is, what is the reason for choosing a non-linear relationship between ze and zn (unlike xn and xe or yn and ye)? Thanks again!

2 ^ | ⌄ • Reply • Share ›

中梓星音 ↪ eduardtejerocirera • a year ago

Indeed you can avoid z NON-linearly and at the same time keep the linearly of x and y BY HAND.
The point is if you use (x,y,z,w) and w not equals 1, OpenGL will devide it to 1 in background.
So before return in vertex shader, you can do the calculation and return (x/w, y/w, z, 1), then OpenGL will not divide them.
In this case, the perspective matrix may be a little bit different since you need to build a linear way to get z_n. (This means you should recalculate the formula of A and B in the post by Mr. songho)

Why we use the non-linearly z map may be: The most time, it looks nothing wrong and fits our needs, and it is convenience. It is a very good question to doubt about the z non linear problem :)

This is also to say the perspective map is a non-linear transform. We just found a linear way to describe them and we have to abandon something not necessary.

4 ^ | ⌄ • Reply • Share ›

songho Mod ↪ eduardotejerocirera • a year ago • edited

Zn = (AZe +BWe) / -Ze is actually a rational function for the varable Ze. We didn't intentionally choose it as a non-linear function for Z, but it simply becomes a non-linear rational function.

1 ^ | ˅ • Reply • Share ›

Shuvo Sarker • a year ago

Thank you, **@songho**.
I have a question. I came to this article from here. It says that depth buffer value ranges from 0 to 1. But in your calculation you mapped z values in the NDC from -1 to 1. Is there any other conversion going on? Or is there anything I missed?

4 ^ | ˅ • Reply • Share ›

songho Mod ↪ Shuvo Sarker • a year ago • edited

Yes, the depth buffer ranges from 0 to 1. But, NDC in OpenGL ranges from -1 to 1.

NDC will be transformed once again to Window (Screen) coordinates. (It is called Viewport Transform.) Then, the z (depth) value finally becomes from 0 to 1.
Please check the Window Coordinates section to see how to transform NDC to Window Coordinate. OpenGL also provides glDepthRange() function to adjust the depth value conversion.

1 ^ | ˅ • Reply • Share ›

Shuvo Sarker ↪ songho • a year ago

Thank you. But there is an error when I follow the link you provided (Window Coordinates).

2 ^ | ˅ • Reply • Share ›

Load more comments

Subscribe **Add Disqus to your site** **Do Not Sell My Data** DISQUS