PERSPECTIVE PROJECTION

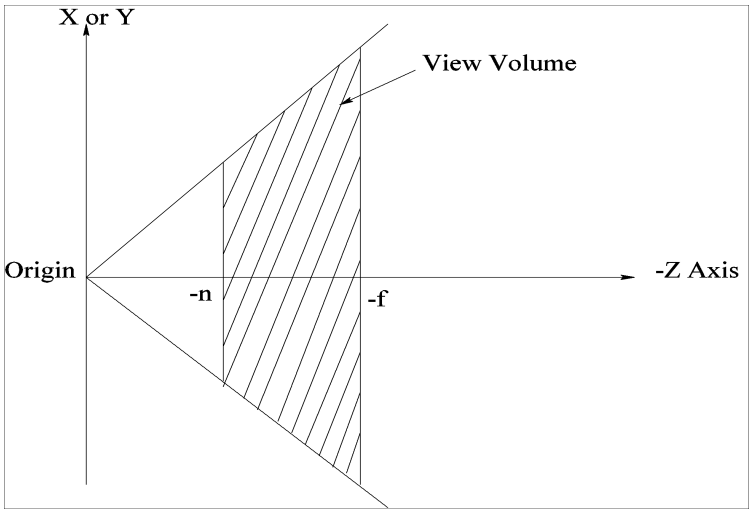
Gopi Meenakshisundaram

1) What does the perspective projection matrix assume?

- Eye is at the origin.
- View-up vector is along the Y-axis.
- View Plane is parallel to the XY plane

2) What does glFrustum (perspective projection) achieve?

- View frustum is along the negative Z-axis
- The view frustum planes are at 45 degrees from the view frustum axis. In effect, the total angle between the view frustum planes are 90 degrees along X and Y axes.
- Find x,y, and z coordinate of any object point on the screen and for correct visibility computation. (Explained in detail later).



3) How do you satisfy the assumptions? (Question 1)

The gluLookAt function achieves the first three requirements. It is split into two functions RT. The T (translation) brings eye to the origin. The R (rotation) rotates such that the view-up vector aligns with the Y axis, and the projection plane is parallel to the XY plane. We assume that this transformation is done before projection transformation is computed. So we do not take into account the gluLookAt transformations (RT) as part of projection transformations.

4) How do you do the first transformation required by glFrustum?

By Shear Matrix

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

5) How do you do the second transformation required by glFrustum? (Refer Qusetion 2).

By Scaling

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

Answers to Questions 4 and 5 bring the view frustum similar to the diagram shown earler.

The above two requirements are easy to describe geometrically. The next requirement is based on the requirement that is not directly related to perspective projection. It is required by later stages of the graphics pipeline like rasterization. So we have to jump ahead a bit to describe what is required by the rasterization, and then come back to compute the appropriate transformation to achieve that.

6) What is the requirement of the rasterization stage?

Rasterization is done on the window. Given the end points of a line, it finds the intermediate pixels that forms the line by linearly interpolating the x,y coordinates of the end points. Similarly, it can fill a triangle, given its three vertices, by bi-linearly interpolating the end points.

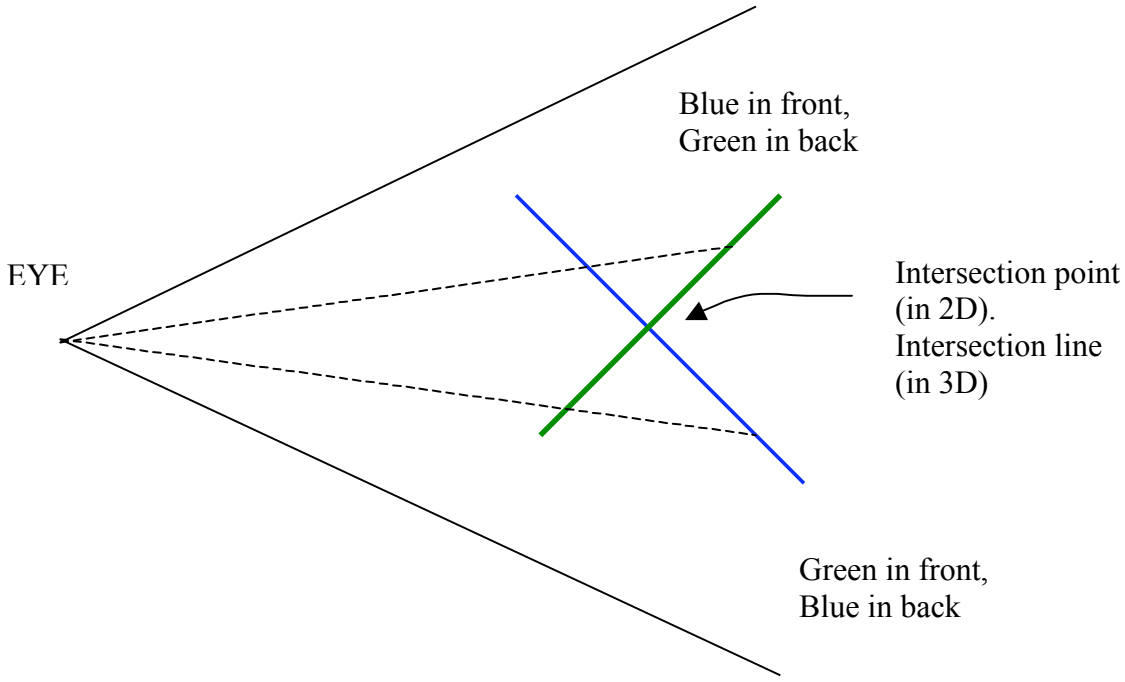
First, the object space 3D coordinates have to be transformed into screen 2D coordinates. Since rasterization stage works only in 2D on the screen, it is enough if we transform the 3D x and y coordinates correctly to 2D x and y screen coordinates. We can ignore the Z value. It is like taking a photograph. We don't care about the depth. We are just taking a 2D image of the 3D scene.

But unfortunately, unlike photographs, where an object in the front automatically occludes the object in the back, in graphics, we have to find which object is in the front and which object is in the back to get a correct image. One way to achieve this is to draw the objects that are in the back first, and then draw objects that are in the front. Thus the objects in the front will overwrite that are in the back yielding correct visibility. Since the triangles are drawn in order, again it is enough for the rasterization stage to know only the x and y coordinates.

But, unfortunately again, during rasterization, you don't always get the triangles in back-to-front order. Triangles come in arbitrary order. You are not allowed to store all the triangles, compute their back-to-front order, and then draw. So the only way to get the correct ordering is to find the correct Z value to figure out which object is in the front and which is in the back, as and when it arrives to the rasterization stage. In fact, you have to know the correct Z value for all points in the triangle, not just at the vertices.

Since we have Z values only for the end points of the triangle or lines, we have to interpolate this Z to get the correct Z values for pixels in-between. We need the correct Z value everywhere for the following reason. Refer to the figure below. If a blue triangle and a green triangle stab each other, we need to draw this intersection line. On one side of the intersection line green is visible, and blue is below green, and on the other side, this relationship is reversed. We need to know the exact Z coordinates everywhere on the triangle to resolve this back-to-front ordering and relationship.

This shows that interpolation of Z is required. Remember we are already interpolating the x any y coordinates, and that too with the same function (linear interpolation). So the question is, can we use the same linear interpolation function on the Z values also? The answer is NO!



7) Why can't we do a linear interpolation of Z from the Z values of the end points, as we do in the cases of X and Y?

To answer this question, we need to know why we interpolate and what is the relationship between the transformation of end points, and the interpolation of intermediate points.

Transforming all points: Ideally, we should consider the triangle (and the object as a whole) consists of infinite number of 3D points, and we should think that we are transforming (projecting) all these points to 2D coordinates on the screen. (This transformation is a projective transformation that uses similar triangles property:

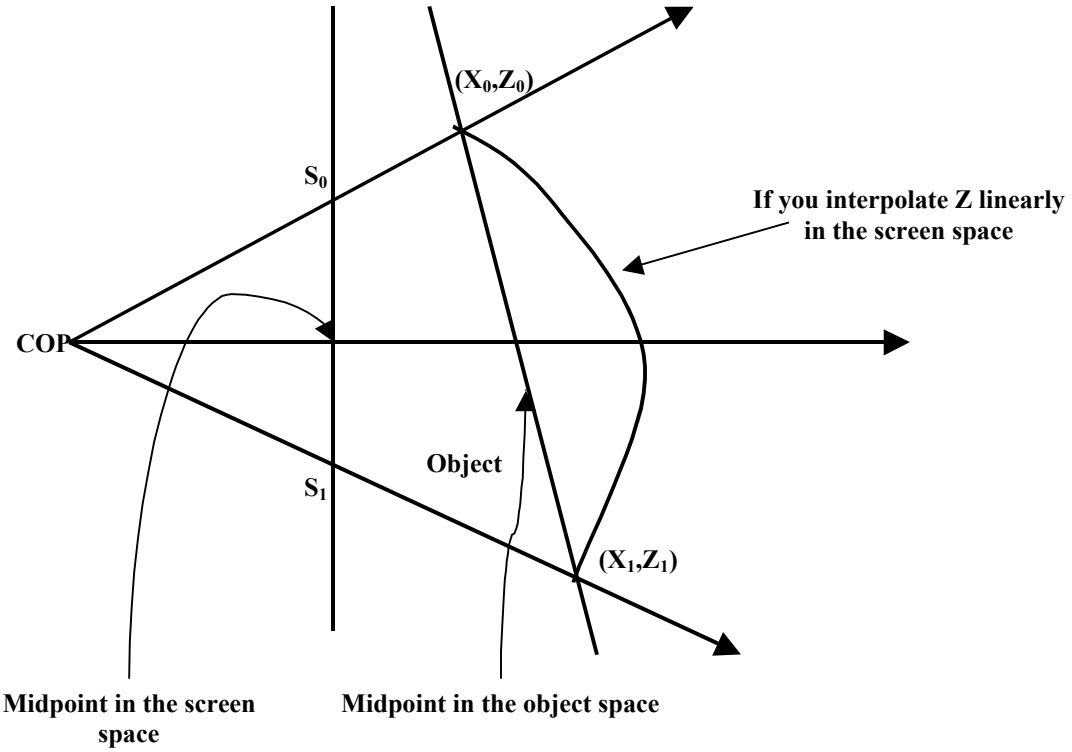
x'=x/(-z), and y'=y/(-z).

Interpolating interior points: Instead we transform only the vertices of the triangle (or line) and linearly interpolate this projected value to get all projected points.

We are saving a lot of time and space by doing just the interpolation instead of transformation of infinite number of points. On the other hand, we have to show that the interpolation is same as transformation. In this context we have to know what is transformed and what is not.

There are two kinds of data: the transformed data, and the attributes. The (x',y') screen coordinates got by transforming a 3D point, are the transformed data. All other data, including the z value, are attributes of this point. Other attributes include color, texture coordinate, normal vector, etc. Unlike x and y, the attributes do not undergo any transformation. In other words, the x and y coordinates of the 3D point is different from those of the projected point. But the attributes of the 3D point are exactly the same as that of the projected point. For example, the color of the 3D point should not change after being projected to a 2D screen. In the same way, the z value stored with the 2D point as an attribute should be same as the z coordinate of the 3D point. (Remember, for a 2D projected point, z value is just another attribute like color, and does not dictate the position of the point. But the z value for a 3D point is a *coordinate* and is used to find the position of that 3D point.)

Consider a straight line. We project the end points and store the actual z value as attribute with these projected end points. Then we do linear interpolation of these z attributes and find the z attribute for all other projected intermediate points on the screen. Ideally, the interpolated z value of the projected line should be the z coordinate of the 3D straight line. But the linearly interpolated z attributes of the intermediate points are found to have the z value that traces a curve as shown in the figure.



8) Since linear interpolation of Z values does not yield correct result, what exactly do we have to interpolate to get the correct z value as attributes for the intermediate projected points?

If S_0 has Z value stored as Z_0, S_1 has z value stored as Z_1 and if you interpolate linearly the Z values for all the screen space points between S_0 and S_1, the corresponding Z values will get mapped on to a curve in the object space.

The pt. $S_0 = \frac{X_0}{Z_0}$, $S1 = \frac{X_1}{Z_1}$

Any point between S0 and S1, say S(u)

$$S(u) = \frac{X(t)}{Z(t)}$$

$$\frac{X_0}{Z_0}(1-u) + \frac{X_1}{Z_1} = \frac{X_0(1-t) + X_1(t)}{(Z_0(1-t) + Z_1(t))}$$

Factoring (-1) out of both sides of the equation, and removing it, we get,

$$\frac{X_0}{Z_0}+\left(\frac{X_1}{Z_1}-\frac{X_0}{z_0}\right)u=\frac{X_0(1-t)+X_1(t)}{Z_0(1-t)+Z_1(t)}$$

$$\left(\frac{Z_0X_1-X_0Z_1}{Z_1Z_0}\right)u=\frac{X_0Z_0(1-t)+X_1Z_0(t)-X_0Z_0(1-t)-X_0Z_1(t)}{Z_0(Z_0(1-t)+Z1t)}$$

$$\left(\frac{Z_0X_1-X_0Z_1}{Z_1Z_0}\right)u=\frac{t(Z_0X_1-X_0Z_1)}{Z_0(Z_0(1-t)+Z_1t)}$$

$$u=\frac{Z_1t}{Z_0(1-t)+Z_1t}$$

We want to solve for t

$$(Z_0(1-t)+Z_1t)u=Z_1t$$

$$Z_0u+tu(Z_1-Z_0)=Z_1t$$

$$t(Z_1-Z_1u+Z_0u)=Z_0u$$

$$\boxed{t=\frac{Z_0u}{Z_1(1-u)+Z_0u}}$$

$$Z(t)=Z_0(1-t)+Z_1t$$

$$Z(u)=Z_0\left(1-\frac{Z_0u}{Z_1(1-u)+Z_0u}\right)+Z_1\left(\frac{Z_0u}{Z_1(1-u)+Z_0u}\right)$$

$$=\frac{Z_0(Z_1(1-u)+Z_0u-Z_0u)+Z_1Z_0u}{Z_1(1-u)+Z_0u}$$

$$=\frac{Z_1Z_0(1-u)+Z_1Z_0u}{Z_1(1-u)+Z_0u}=\frac{Z_1Z_0}{Z_1(1-u)+Z_0u}$$

$$\frac{1}{Z(u)}=\frac{(1-u)1}{Z_0}+u\frac{1}{Z_1}$$

So the actual Z of the interpolated points is computed by linearly interpolating $\frac{1}{Z}$ and taking the reciprocal of the interpolated value.

Similar to Z interpolation, other attributes like color and texture coordinates have to be interpolated in the above way. But, very rarely one does color interpolation by linearly interpolating the reciprocal. Since humans are not very sensitive to minor color changes, most systems cheat by linearly interpolating the color instead of linearly interpolating the reciprocal of the color values. On the other hand, no one cheats on interpolating the texture coordinates.

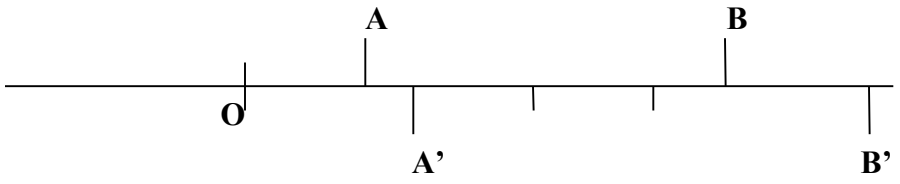
As part of efficient and correct computation of interpolation, a function of $\frac{1}{Z}$ is stored at the vertices and the depth buffer, instead of Z. Another reason for using $\frac{1}{Z}$ instead of Z is to get more resolution for objects that are close to near plane (closer Z) and less resolution for objects that are far (close to the far plane). Further, now, you can set the far plane to -infinity and just store zero for –1/f. For the same reason, you should never set the near plane to 0 (origin).

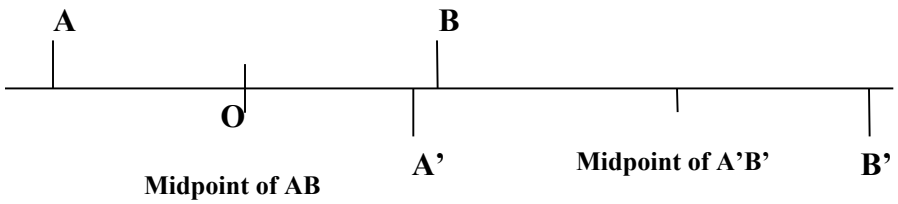
9) What function of $\frac{1}{Z}$ is stored in the depth buffer?

The values of $\frac{1}{Z}$ range from $\frac{1}{-n}$ to $\frac{1}{-f}$. This range is scaled between +1 and -1. This is what is stored at the vertices and depth buffer.

10) How is this transformation done?

Let the given range be A to B (in our case $\frac{1}{-n}$ and $\frac{1}{-f}$). Let us assume that this range has to be mapped to $A' - B'$ (in our case +1 to –1).





Steps

1. Translate the range AB such that the midpoint of AB is coincident with the origin.
2. Scale the translated range AB by $\frac{(B'-A')}{(B-A)}$.

3. Translate the scaled range to (A' + B') / 2

Note that the mid point of AB has been mapped to the mid point of A'B'.
Now applying the above steps to our given problem:

$A = \frac{1}{-n}$ B = $\frac{1}{-f}$

Mid-Point of AB : $\frac{A+B}{2} = -\frac{\left(\frac{1}{n}+\frac{1}{f}\right)}{2} = -\frac{f+n}{2nf}$

A' = +1 B' = -1

Mid-Point of AB : $\frac{A'+B'}{2} = 0$

First translate: -mid-point of AB = $\frac{f+n}{2nf}$

Next scale: $\frac{(B'-A')}{(B-A)} = +\frac{2fn}{n-f}$

Then translate: + mid-point of A'B' = 0

Therefore, any given value $\left(\frac{1}{-Z}\right)$ is transformed into: $\left(\frac{1}{-Z}+\frac{f+n}{2nf}\right)\left(\frac{2fn}{n-f}\right)$

$$= \frac{\left(\frac{f+n}{n-f}\right)(+Z)-\left(\frac{2fn}{n-f}\right)}{-Z}$$

We know that, after Perspective Projection new X = $\frac{X}{-Z}$, Y = $\frac{Y}{-Z}$

The homogeneous point, (x,y,z,w) is same as the homogeneous point (x/w, y/w, z/w, 1). Therefore, the homogeneous point after the perspective transformation is:

$$\left[X, Y, \frac{f+n}{f-n}(Z) + \frac{2fn}{f-n}, -Z\right]$$

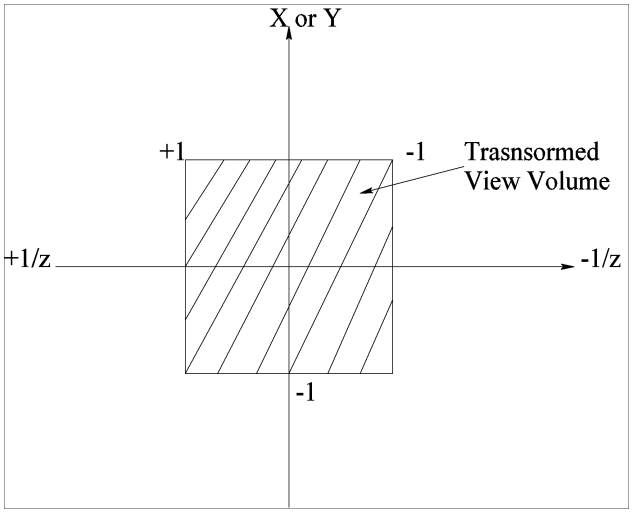
So, the matrix for perspective transformation, that transforms a point (X,Y,Z, 1) to the above perspectively transformed point is given by:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{f+n}{f-n} & \frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Transforming a point using the above matrix achieves the following: X and Y values are divided by the Z *distance* of point from the origin. Z value is converted into a function of 1/Z and that too within the range +1 to –1. Since the view frustum was already set to 45 degrees on all sides of the viewing axis (-Z axis), the |X| value (or the |Y| value) of any point on the four edges forming the view frustum is same as the |Z| value of that point. So dividing the X and Y values of the points on those edges, by its –Z will make it +1 and –1. For any other point inside the view volume, the X/-Z and Y/-Z will be between +1 and –1. In essence, all the three coordinates, X,Y, and Z will have the range +1 to –1.

Note that when the object space coordinate *approaches* the origin (0,0,0), the transformed X and Y coordinates approach +1 and –1, and the transformed Z approaches infinity. Thus, the view frustum is transformed into an orthographic frustum of parallel projection.

11) How would the transformed view volume look like?



7) What is the final matrix for glFrustum(l,r,b,t,n,f) command

Perspective*Scale*Shear =

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

Note that the Z and W are affected only by the perspective-matrix. Compare these results with the man pages of glFrustum.