

Perspective Projection

There are four parameters involved.

1. Eye
2. View Direction
3. View Up Vector
4. Normal to the Image Plane

Case I: Eye is at $(0, 0, 0)$, View Direction and Normal to the Image Plane is coincident with the z axis and View Up vector is coincident with y axis.

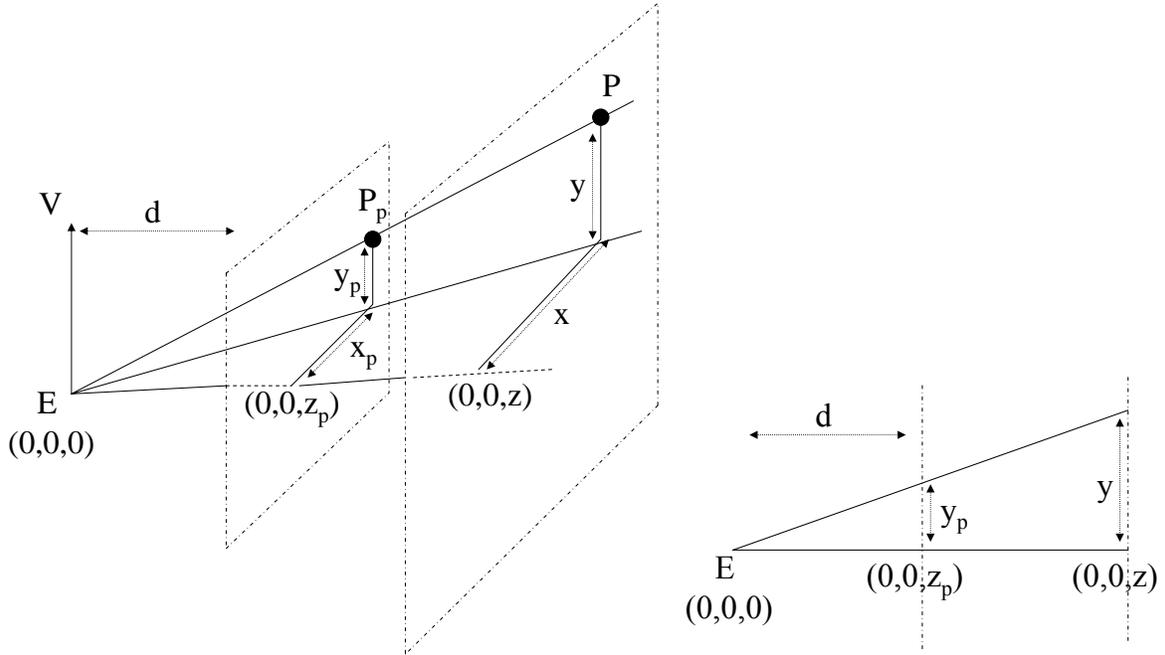


Figure 1: Left: 3D diagram for case I. Right: 2D version by projection on yz plane.

In this case, by similar triangles we get,

$$\frac{x_p}{x} = \frac{y_p}{y} = \frac{z_p}{z}$$

From this we get,

$$x_p = \frac{x}{z} z_p; \quad y_p = \frac{y}{z} z_p$$

This can be expressed in the form of matrix as,

$$\begin{pmatrix} x_p \\ y_p \\ z_p \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{d} & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

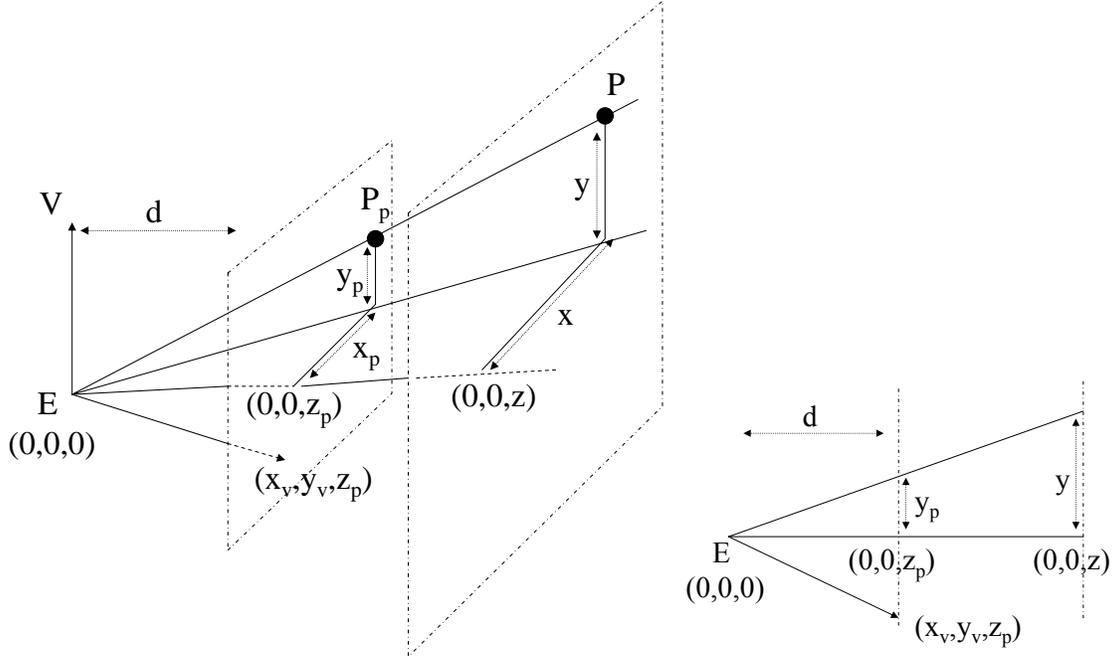


Figure 2: Left: 3D diagram for case II. Right: 2D version.

In other notation,

$$P_p = M(d).P$$

Case II: Eye is at $(0, 0, 0)$, Normal to the Image Plane is coincident with the z axis and View Up vector is coincident with y axis, View Direction defined by (x_v, y_v, z_p) .

How to make (x_v, y_v, z_p) coincident with $(0, 0, z_p)$?

A shear can do the job. Hence, we want to find the shear matrix that satisfies

$$\begin{pmatrix} 0 \\ 0 \\ z_p \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & a & 0 \\ 0 & 1 & b & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_v \\ y_v \\ z_p \\ 1 \end{pmatrix}$$

From this we find,

$$a = \frac{-x_v}{d}; \quad b = \frac{-y_v}{d}$$

This shear matrix is define as

$$Sh(-x_v d, -y_v d) = \begin{pmatrix} 1 & 0 & \frac{-x_v}{d} & 0 \\ 0 & 1 & \frac{-y_v}{d} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

So, the final projection equation is

$$P_p = Sh^{-1}.M(d).Sh.P$$

Case III: The image plane is no longer perpendicular to z . The plane's direction is given by $N = (x_n, y_n, z_n)$. V need not be orthogonal to N and need not coincide with y .

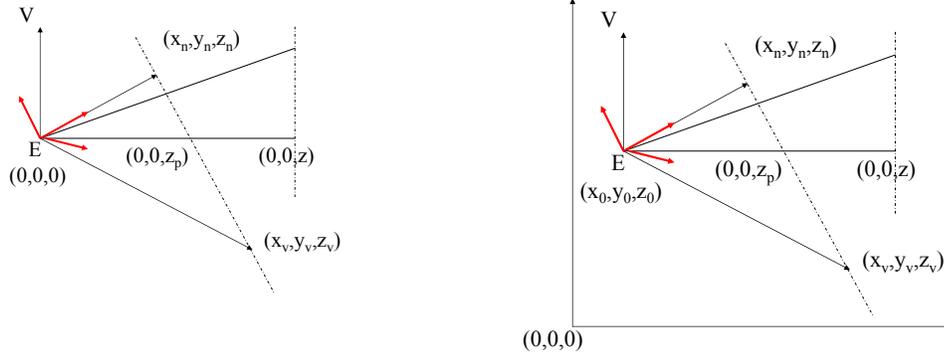


Figure 3: Left: 2D version of case III. Right: 2D version of case IV.

First we define a coordinate system at E so that N coincides with z axis.

$$u'_z = \frac{N}{|N|}$$

$$u'_x = \frac{V}{|V|} \times u'_z$$

$$u'_y = u'_z \times u'_x$$

Rotation required to coincide N with z axis is given by

$$R(N, V) = \begin{pmatrix} u_{x1} & u_{x2} & u_{x3} & 0 \\ u_{y1} & u_{y2} & u_{y3} & 0 \\ u_{z1} & u_{z2} & u_{z3} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Note that after this rotation, the case reduces to case II. But we need to find where the view vector hits the plane at distance $|N|$ from the eye along z direction. This point is achieved by

$$\begin{pmatrix} x'_v \\ y'_v \\ |N| \\ 1 \end{pmatrix} = R \cdot \begin{pmatrix} x_v \\ y_v \\ z_v \\ 1 \end{pmatrix}$$

So, the final projection matrix is given by,

$$P_p = R^{-1} \cdot Sh^{-1} \cdot M(|N|) \cdot Sh \cdot R \cdot P$$

Case IV: The most general case is where the Eye is not at $(0, 0, 0)$ but at (x_0, y_0, z_0) . The transformation then is given by

$$P_p = T(x_0, y_0, z_0) \cdot R^{-1} \cdot Sh^{-1} \cdot M(|N|) \cdot Sh \cdot R \cdot T(-x_0, -y_0, -z_0) \cdot P$$

Note that the P_p thus given out of the above equation is in the world coordinates within which the eye, normal etc are define. Now, when we are talking about image generation, we would like to define the P_p in a coordinate

system, called the coordinate system of the eye, where E is the origin, the view up vector is aligned with the Y axis, the view direction is perpendicular to the image plane. Note that the matrix $Sh.R.T(-x_0, -y_0, -z_0)$ gives this transformation which would bring the world coordinates to the coordinate system of the eye. The perspective projection matrix $M(|N|)$ is applied after that which achieves the perspective projection in the coordinate system of the eye. Then the matrix $T(x_0, y_0, z_0).R^{-1}.Sh^{-1}$ is used to transform it back to the world coordinate system. So, if we want to get the perspective transformation in the eye's coordinate system, all we need to do is not take it back to the world coordinate system by applying the inverse transformations. Thus the perspective projection P_p^E in the eye's coordinate system is given by

$$P_p^E = M(|N|).Sh.R.T(-x_0, -y_0, -z_0).P$$

Note that R depends only on direction of the normal and direction of the view up vector. The length of the normal $|N|$ is only important for the matrix M .

In graphics, the rotation and translation is defined in the world coordinate system. This is given by a function call

gluLookAt(*Eye Coordinate: E, View Up Vector: V, Normal to the Image Plane: N*)

Hence, with this command the matrix $R(N, V).T(-E)$ is applied to all objects to get them in eye's coordinate system. Note that now the image plane is perpendicular to the z axis. The distance of the image plane from the eye is given by $|N|$. Let $|N| = n$. Note that the transformation mentioned above brings the z of any point to n which is the image plane.

Note that ideally image plane is infinite. When generating an image in the eye, we do not have an infinite image plane. So, we need to limit the plane somehow. This is done by defining a window (r, l, t, b) on the image plane in the *eye's coordinate system*. The defined (r, l, t, b) says that the eye would get the image in the rectangular window whose left, right, top and bottom ends are defined by $r, l, t,$ and b respectively.

Note that indirectly, the center of this window now defines the view direction and is given by $(\frac{r+l}{2}, \frac{t+b}{2}, n)$. Thus, as before, the shear matrix required to bring the view direction $(\frac{r+l}{2}, \frac{t+b}{2}, n)$ to be coincident with $(0, 0, n)$ is given by $Sh(\frac{r+l}{2n}, \frac{t+b}{2n})$.

This says that the range of x_p^E and y_p^E achieved by the transformation $M(n).Sh(\frac{r+l}{2n}, \frac{t+b}{2n}).R(N, V).T(-E)$ will vary between $[-\frac{r-l}{2}, \frac{r-l}{2}]$ and $[-\frac{t-b}{2}, \frac{t-b}{2}]$ respectively, and the z will be n . The problem with this is we cannot limit our framebuffer since it is dependent on (r, l, t, b) . So, what we would like to do is to x_p^E and y_p^E to range between $[-1, 1]$ only. This means we would like

$$\frac{r-l}{2} \rightarrow 1, \quad \text{in } x \text{ direction}$$

$$\frac{t-b}{2} \rightarrow 1, \quad \text{in } y \text{ direction}$$

So, for this we need a scaling matrix defined by $Sc(\frac{2}{r-l}, \frac{2}{t-b}, 1)$. Thus,

$$\begin{pmatrix} x_p^E \\ y_p^E \\ z_p^E \\ 1 \end{pmatrix} = Sc(\frac{2}{r-l}, \frac{2}{t-b}, 1).M(n).Sh(\frac{r+l}{2n}, \frac{t+b}{2n}).R(N, V).T(-E) \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

This can be written in terms of the 3D coordinates in eye's coordinate system as,

$$\begin{pmatrix} x_p^E \\ y_p^E \\ z_p^E \\ 1 \end{pmatrix} = Sc(\frac{2}{r-l}, \frac{2}{t-b}, 1).M(n).Sh(\frac{r+l}{2n}, \frac{t+b}{2n}) \begin{pmatrix} x^E \\ y^E \\ z^E \\ 1 \end{pmatrix} \quad (1)$$

From this equation we get

$$\begin{pmatrix} x_p^E \\ y_p^E \\ z_p^E \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{2nx^E}{z^E(r-l)} \\ \frac{2ny^E}{z^E(t-b)} \\ n \\ 1 \end{pmatrix}$$

However, there lies a problem in this, at least for the computers. To generate a scene in graphics, we take a triangle representation of a model. While rendering, every vertex of the triangle is projected on the image plane and the color within this projected triangle is filled up. This filling up is called rasterization. But, the problem is, since the z always maps to n , from the projected point, you cannot know which one is in front of the other. Note, every point in the image plane has the color from the closest object, everything behind is occluded. So, what we would need to do is to take objects in back to front order and draw. But this does not work when objects are intersecting each other. So, the only option left, is to take corresponding point that lies on the same ray from the two objects, find where they hit the objects and find who is in front of whom. But, this is extremely inefficient, especially when in real scene several such situations may arise.

A better way is if we can somehow encode this relationship in z_p^E so that just by seeing the z_p^E present in the framebuffer from rasterization of previous triangle can be compared with the projection of the point that is being rendered currently to decide whether that image coordinate should get the color from this point or not. To achieve this we define a far plane beyond which we do not want to consider objects. Let this far plane be at f . Note the (r, l, t, b, n, f) thus defines a view volume, objects within which are considered for rendering. So, now we may say that we want a transformation which satisfies

$$\begin{pmatrix} x_p^E \\ y_p^E \\ z_p^E \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{2nx^E}{z^E(r-l)} \\ \frac{2ny^E}{z^E(t-b)} \\ z^E \\ 1 \end{pmatrix} \quad (2)$$

so that the z values are retained for us to decide on occlusion. Though this is not exactly perspective transformation, we are only changing the value of z which is anyway not important for the perspective transformation, but to resolve occlusions.

But, there is a problem here too. One thing that is critical here is does linear interpolation of Z in screen space give us linear interpolation of points in the object space. Or in other words, is the following equation correct?

$$\frac{X_0 + t(X_1 - X_0)}{Z_0 + t(Z_1 - Z_0)} = s_0 + t(s_1 - s_0) \quad (3)$$

$$\frac{(1-t)X_0 + tX_1}{(1-t)Z_0 + tZ_1} = (1-t)s_0 + ts_1 \quad (4)$$

$$= \frac{(1-t)X_0Z_1 + tX_1Z_0}{Z_0Z_1} \quad (5)$$

$$(6)$$

where $s = x_p^E$, $X = x^E$, and $Z = z^E$. Clearly, this is not the same. In fact, if you plot the curve that the interpolation of Z in screen space would achieve in object space, instead of traversing the object, it would project on a curved path as shown in Figure 4.

So, now that we know that interpolation of Z won't work, let us what kind of interpolation in screen space, if

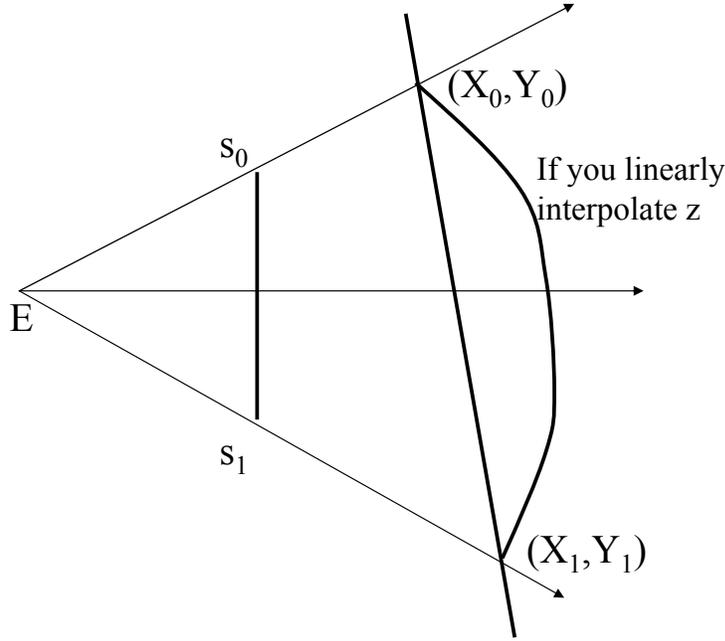


Figure 4: Problem with interpolation of z .

any, would give us correct depth values. For that let us find the u which would give the following

$$\frac{X_0 + t(X_1 - X_0)}{Z_0 + t(Z_1 - Z_0)} = s_0 + u(s_1 - s_0) \quad (7)$$

$$\frac{(1-t)X_0 + tX_1}{(1-t)Z_0 + tZ_1} = \frac{X_0}{Z_0} + u \left(\frac{X_1}{Z_1} - \frac{X_0}{Z_0} \right) \quad (8)$$

$$\frac{\frac{(1-t)X_0 + tX_1}{(1-t)Z_0 + tZ_1} - \frac{X_0}{Z_0}}{\frac{X_1 - X_0}{Z_1 - Z_0}} = u \quad (9)$$

$$\frac{Z_1 t}{Z_0(1-t) + tZ_1} = u \quad (10)$$

$$(11)$$

From this, we find t as,

$$\frac{1}{u} = \frac{Z_0 + t(Z_1 - Z_0)}{Z_1 t} \quad (12)$$

$$= \frac{Z_0}{Z_1 t} + \frac{Z_1 - Z_0}{Z_1} \quad (13)$$

$$\frac{1}{u} - \frac{Z_1 - Z_0}{Z_1} = \frac{Z_0}{Z_1 t} \quad (14)$$

$$\frac{uZ_0}{Z_1 - u(Z_1 - Z_0)} = t \quad (15)$$

$$(16)$$

Let the depth of the point at parameter t be denoted by Z_t .

$$Z_t = Z_0 + t(Z_1 - Z_0) \quad (17)$$

$$= Z_0 + \frac{uZ_0}{Z_1 - u(Z_1 - Z_0)}t(Z_1 - Z_0) \quad (18)$$

$$= \frac{Z_0Z_1}{Z_1 - u(Z_1 - Z_0)} \quad (19)$$

$$= \frac{1}{\frac{1}{Z_0} - u(\frac{1}{Z_0} - \frac{1}{Z_1})} \quad (20)$$

Therefore,

$$\frac{1}{Z_t} = \frac{1}{Z_0} + u(\frac{1}{Z_1} - \frac{1}{Z_0}) \quad (21)$$

$$= \frac{1}{Z_0}(1 - u) + u\frac{1}{Z_1} \quad (22)$$

$$(23)$$

This shows that to find correct depth values we have to taken reciprocal of the Z 's, interpolate using screen space u and then again take its reciprocal. Thus, the interpolation needs to be done is $\frac{1}{Z}$ space.

Hence, in Equation 2, instead of z^E , we would like to store $\frac{1}{z^E}$ so that we can interpolate them. Thus, what we want is,

$$\begin{pmatrix} x_p^E \\ y_p^E \\ z_p^E \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{2nx^E}{z^E(r-l)} \\ \frac{2ny^E}{z^E(t-b)} \\ \frac{1}{z^E} \\ 1 \end{pmatrix} \quad (24)$$

Next, comes the same question of normalizing. $\frac{1}{z^E}$ can be between $\frac{1}{n}$ and $\frac{1}{f}$ which gives us unbounded volume which is difficult for computers to handle. So, we would like to scale the depth of $\frac{1}{n}$ to $\frac{1}{f}$ to -1 to 1 . This can be done by a very simple linear 1D transformation. These can be achieved by two steps.

1. Translate the center in the z -direction to origin. This is given by

$$-\frac{\frac{1}{n} + \frac{1}{f}}{2} = -\frac{f + n}{2nf}$$

2. Scale it to match the length of -1 to $+1$. This can be achieved by a scaling of

$$\frac{2}{\frac{1}{f} - \frac{1}{n}} = \frac{2nf}{n - f}$$

Hence, the whole transformation to $\frac{1}{z^E}$ is given by

$$z_p^E = \left(\frac{1}{z^E} - \frac{f + n}{2nf} \right) \frac{2nf}{n - f} \quad (25)$$

$$= \left(\frac{2nf}{(n - f)z^E} + \frac{f + n}{f - n} \right) \quad (26)$$

$$= \frac{\frac{2nf}{(n-f)} + \frac{f+n}{f-n}z^E}{z^E} \quad (27)$$

Now, the desired point we have is given by,

$$\begin{pmatrix} x_p^E \\ y_p^E \\ z_p^E \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{2nx^E}{z^E(r-l)} \\ \frac{2ny^E}{z^E(t-b)} \\ \frac{\frac{2nf}{(n-f)} + \frac{f+n}{f-n} z^E}{z^E} \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{2nx^E}{(r-l)} \\ \frac{2ny^E}{(t-b)} \\ \frac{2nf}{(n-f)} + \frac{f+n}{f-n} z^E \\ z^E \end{pmatrix} \quad (28)$$

Now, this transformation can be achieved by our $Sc(\frac{2}{r-l}, \frac{2}{t-b}, 1)$ followed by another matrix called $D(n, f)$ which is a function of only the near and far plane, n and f respectively.

$$D(n, f) = \begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & \frac{f+n}{f-n} & \frac{2fn}{f-n} \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

So, the $Sc.M$ in Equation 1 is replaced by $D.Sc$ to give us the desired z . Thus the complete transformation is given by,

$$\begin{pmatrix} x_p^E \\ y_p^E \\ z_p^E \\ 1 \end{pmatrix} = D(n, f).Sc(\frac{2}{r-l}, \frac{2}{t-b}, 1).Sh(\frac{r+l}{2n}, \frac{t+b}{2n}).R(N, V).T(-E) \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

As we mentioned before, the transformation of $R(N, V).T(-E)$ is given with **gluLookAt** function call. The transformation $D(n, f).Sc(\frac{2}{r-l}, \frac{2}{t-b}, 1).Sh(\frac{r+l}{2n}, \frac{t+b}{2n})$ depends only on r, l, t, b, n and f and is applied with a function call

glFrustum(r, l, t, b, n, f)

The matrix generated by this call is given by $D(n, f).Sc(\frac{2}{r-l}, \frac{2}{t-b}, 1).Sh(\frac{r+l}{2n}, \frac{t+b}{2n})$ as

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