Appendix 4: OpenGL Projection Matrix

The goal of this article is not to provide a mathematically vigorous derivation of the projection matrix. The goal is to provide an explanation that helps understand the matrix.

The projection matrix transforms the vertices from the view space to the NDC space, through a perspective projection. The geometry setup that conforms with that pipeline is shown in Figure 5-1.

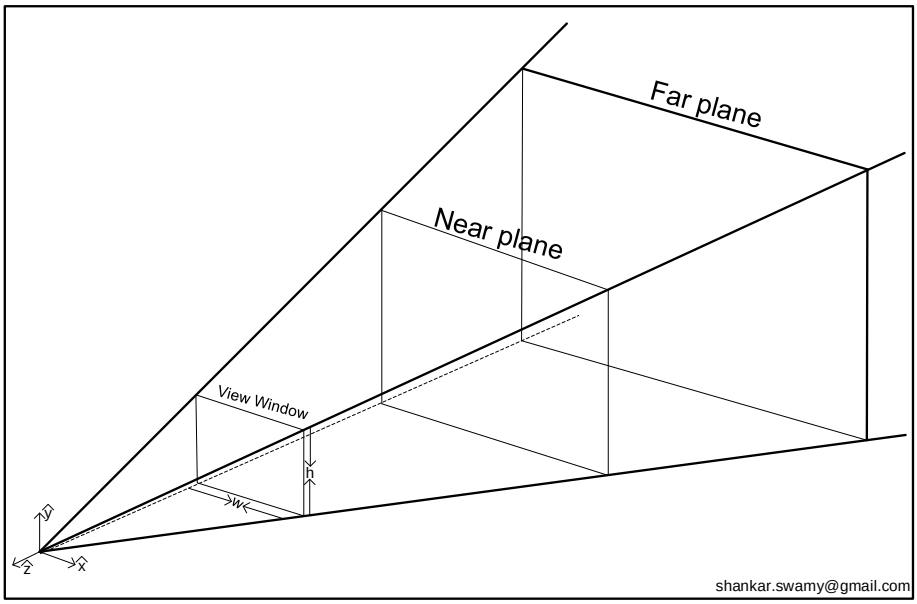

Figure 8-8 View frustum geometry

Essentially the OpenGL projection matrix does three transformations:

1. Take the vertex from the view space apply projection transformation to the vertex
2. Apply perspective division
3. Transform the vertices to the NDC space.

Splitting up the perspective projection into a projection step and a separate perspective division has two advantages.

First, this enables clipping the primitives in the homogeneous space which is affords the most efficient clipping algorithm. What is lost in efficiency by the extra perspective division is more than offset by the gains made by clipping the primitives in the homogeneous space.

Secondly, for the projection transformation the same function in the code or the same hardware used for orthographic projection can be reused.

The view frustum in the view space, the source of the vertices and the NDC space, the destination of the vertices after the projection transformation are shown in Figure 5-2.

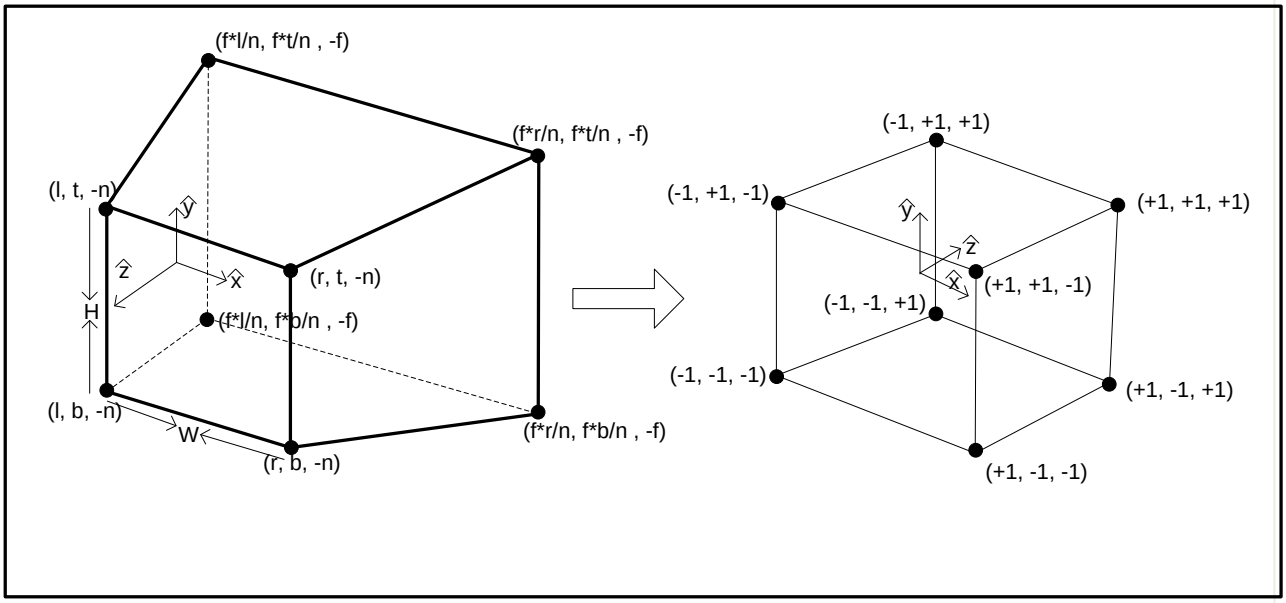

Figure 8-9 View frustum and the NDC space

NDC space is a cube bounded by (-1, -1, -1) x (1, 1, 1,). Notice that the sense of z-direction in the view space (frustum) and the NDC space are opposite to each other, which means that during the projection transformation, the z-axis needs to be flipped to reverse the sign. The two spaces projected on to the plane looking down along the negative x-axis are shown in Figure 5-3. An almost identical Figure for the projection on to the plane looking down along the y-axis is not shown.

From the Figure, we notice that (r, t, -n) in {View} maps to {1, 1, -1), and (r, t, -f) in {View} maps to (1, 1, 1).

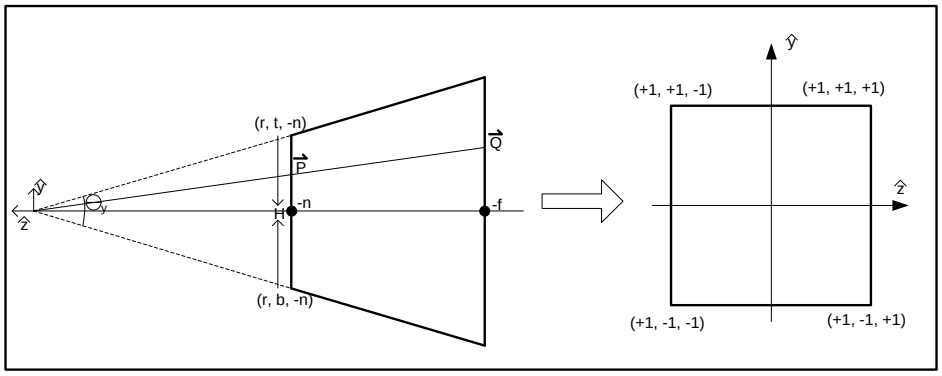

Figure 8-10 View frustum and the NDC space projected on to x-plane

We seek the matrix that transforms a vertex in the view space to the NDC space with perspective projection:

$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \begin{bmatrix} {}^{v}\vec{p}_x \\ {}^{v}\vec{p}_y \\ {}^{v}\vec{p}_z \\ 1 \end{bmatrix} = \begin{bmatrix} {}^{ndc}\vec{p}_x \\ {}^{ndc}\vec{p}_y \\ {}^{ndc}\vec{p}_z \\ 1 \end{bmatrix} \qquad \text{A 4.1}$$

and equivalently:

$$M_{proj}\ {}^{v}\vec{p} = {}^{ndc}\vec{p} \qquad \text{A 4.2}$$

The 4x4 matrix M_{proj} is the OpenGL projection matrix, ${}^{v}\vec{p}$ is the vector in the view space and ${}^{ndc}\vec{p}$ is the vector transformed to the NDC space.

A perspective projection is termed *asymmetric* if the $W \neq H$ or $\theta_y \neq \frac{\pi}{2}$ or both (W, H and θ_y are shown in Figure 5-3 and Figure 5-4).

First consider the symmetric projection. That is, $\theta_x = \theta_y = \pi/4$, and $W = H$ *i.e.* a square cross-section for the truncated pyramid.

We start with a matrix that will result in the perspective or the foreshortening. The matrix that produces foreshortening along the z-axis is given by:

$$M_p = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \qquad \text{A 4.3}$$

The matrix given in equation A 4.5 the perspective matrix for foreshortening along the z-axis. However in OpenGL the perspective matrix is taken as:

$$M_p = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix} \qquad \text{A 4.4}$$

In the OpenGL convention, {View} the z-coordinate is directed away from the scene which is the opposite sense of the z-axis in the conventional z-direction in the {Perspective}. For the perspective to be along –z-axis, we need to flip the z-coordinate and hence the minus sign in the perspective term (fourth row of the matrix) in A 4.4.

The matrix should essentially transform the truncated pyramid into a rectangular box (Figure 5-2**Error! Reference source not found.** and Figure 5-3). We have

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ -z \end{bmatrix} \equiv \begin{bmatrix} -\frac{x}{z} \\ -\frac{y}{z} \\ -1 \\ 1 \end{bmatrix} \qquad \text{A 4.5}$$

Using the above matrix, the eight corners of the truncated pyramid tans from as shown in the table below:

Table 2 Coordinates of the corners of the truncated pyramid forming the view frustum

Vertex Number	${}^{view}\vec{p}$	$M_p\ {}^{view}\vec{p}$
0	(l, t, -n, 1)	$(\frac{l}{n}, \frac{t}{n}$, -1, 1)
1	(l, b, -n, 1)	$(\frac{l}{n}, \frac{b}{n}$, -1, 1)
2	(r, b, -n, 1)	$(\frac{r}{n}, \frac{b}{n}$, -1, 1)
3	(r, t, -n 1)	$(\frac{r}{n}, \frac{t}{n}$, -1, 1)
4	$(\frac{f*l}{n}, \frac{f*t}{n}$, -f, 1)	$(\frac{l}{n}, \frac{t}{n}$, -1, 1)
5	$(\frac{f*l}{n}, \frac{f*b}{n}$, -f, 1)	$(\frac{l}{n}, \frac{b}{n}$, -1, 1)
6	$(\frac{f*r}{n}, \frac{f*b}{n}$, -f, 1)	$(\frac{r}{n}, \frac{b}{n}$, -1, 1)
7	$(\frac{f*r}{n}, \frac{f*t}{n}$, -f, 1)	$(\frac{r}{n}, \frac{t}{n}$, -1, 1)

Notice that this maps vertex pairs (0, 4), (1, 5), (2, 6), (3, 7) each to the same point. For the transformation, this implies that all points along a line from the origin, between $z = -n$ to $z = -f$, all map to the same point after the transformation. Referring to Figure 5-3, this means, for example, that all points on the line segment $\overline{PQ}$ would map to $\vec{P}$. So all points within the truncated viewing pyramid are projected properly to the

rectangle at the near $z = -n$ plane and by scaling that rectangle we have the properly perspective projected transformation. This will suffice if we do not need any form of visible surface determination.

However we do need the visible surface determination for rendering, and hence instead of mapping the points within the truncated viewing pyramid to a plane, we need to map them to the box bounded by $(-1,-1,-1) \times (1,1,1)$ - Figure 5-2. More precisely, we would like $z = -n$ to be mapped to -1 and $z = -f$ to be mapped to +1. To achieve this without affecting the transformations of the x and y coordinates in the M_{proj} matrix, we have to parameterize the elements (3,3) and (3, 4) of the matrix. Say, the two are respectively parameterized a a and b. We have:

$$M_p = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & -1 & 0 \end{bmatrix} \qquad \text{A 4.6}$$

With the matrix thus modified, the z −coordinate now transforms as: $z \rightarrow \frac{az+b}{-z}$, which gives $-n \rightarrow \frac{-an+b}{n}$, and $-f \rightarrow \frac{-af+b}{f}$. Given that we would like these points to map to -1 and +1 respectively, we should have:

$$\begin{aligned} \frac{-an+b}{n} &= -1 \\ \frac{-af+b}{f} &= +1 \end{aligned} \qquad \text{A 4.7}$$

which give:

$$\begin{aligned} a &= -\frac{f+n}{f-n} \\ b &= -\frac{2fn}{f-n} \end{aligned} \qquad \text{A 4.8}$$

This gives:

$$M_p = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \qquad \text{A 4.9}$$

With this matrix any point in the truncated view pyramid transforms as

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ -\frac{z}{(f-n)}\left\{(f+n)+\frac{2fn}{z}\right\} \\ -z \end{bmatrix}$$

$$\equiv \begin{bmatrix} -\frac{x}{z} \\ -\frac{y}{z} \\ \frac{1}{(f-n)}\left\{(f+n)+\frac{2fn}{z}\right\} \\ 1 \end{bmatrix} \qquad \text{A 4.10}$$

The equation A 4.10 assumes that the axis of the view frustum is aligned along the –z-axis as shown in Figure 5-1. However a more generic case is shown in Figure 5-4.

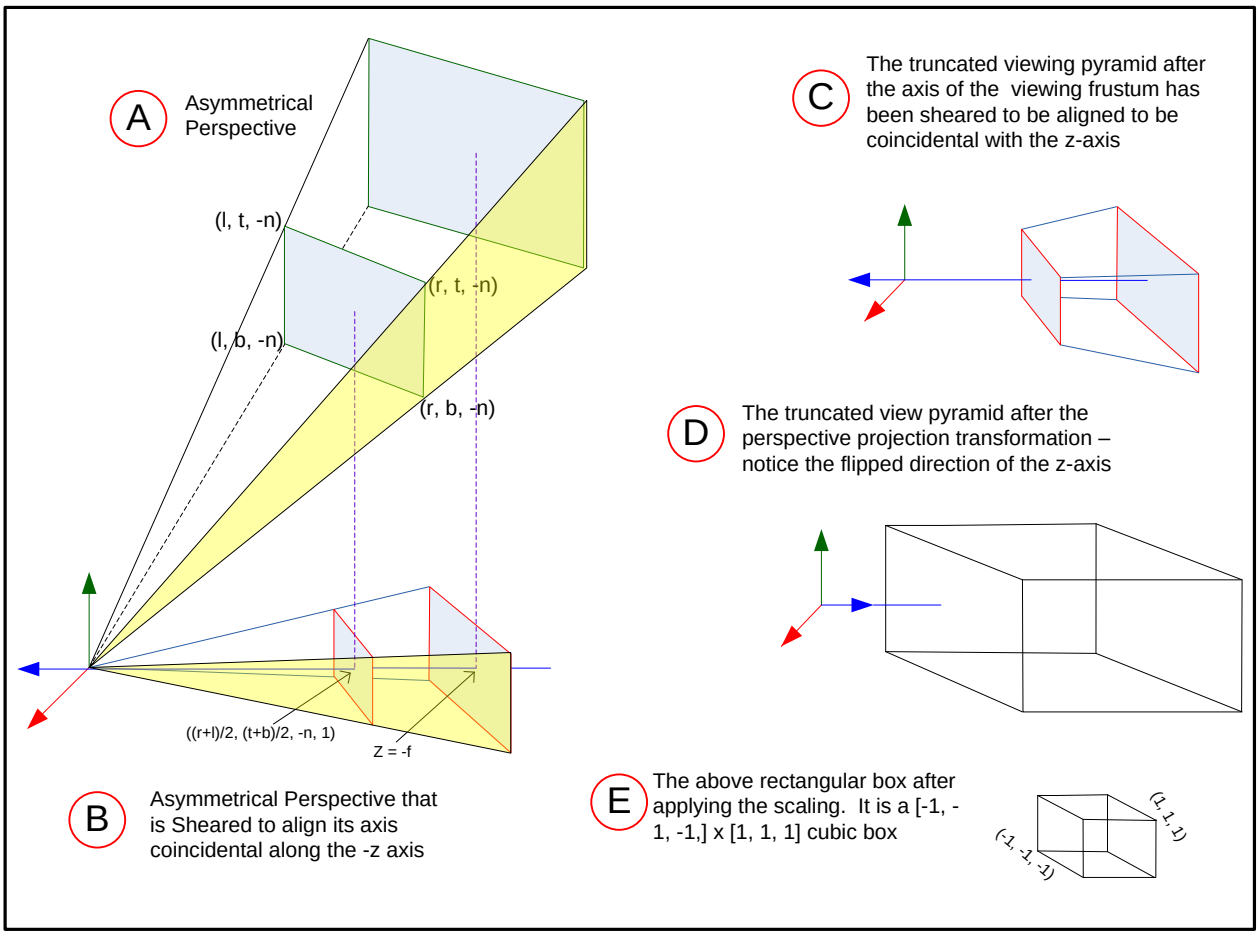

Figure 8-11 Two views of an asymmetric perspective projection view frustum

The figure shows two views of a view frustum which is not aligned with the z-axis. But matrix M_{view} as given in equation A 4.10 holds for a view frustum that is aligned with its axis coincident with the z-axis. Before we apply the transformation given by equation A 4.10 we need to transform the view frustum to make its axis coincidental with the z-axis.

Notice in Figure 5-4 – A and B, that the tops of the original and the transformed frustums do not move. For the rest of the points in the original frustum, the movement is proportional to its z-coordinate. This shearing transformation takes the vertices bounding the truncated view frustum at the $z = -n$ to transform as shown in the table below.

Notice that this shearing transformation leaves the z-coordinate invariant and thus this is a shearing along the x- and y-directions. Thus any point within the asymmetric frustum is should first be transformed by the matrix for shearing along the x- and y-directions as shown below.

$$\begin{bmatrix} 1 & 0 & \alpha & 0 \\ 0 & 1 & \beta & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x + \alpha z \\ y + \beta z \\ z \\ 1 \end{bmatrix} \qquad \text{A 4.11}$$

Table 3 Asymmetric perspective vertices correlated with vertices after *shearing* transformation

Vertices from the asymmetric perspective	Vertices after the shearing transformation
$(l, t, -n, 1)$	$\left(-\frac{r-l}{2}, \frac{t-b}{2}, -n, 1\right)$
$(l, b, -n, 1)$	$\left(-\frac{r-l}{2}, -\frac{t-b}{2}, -n, 1\right)$
$(r, b, -n, 1)$	$\left(\frac{r-l}{2}, -\frac{t-b}{2}, -n, 1\right)$
$(r, t, -n, 1)$	$\left(\frac{r-l}{2}, \frac{t-b}{2}, -n, 1\right)$

With, say $(l, b, -n,\ 1) \rightarrow \left(-\frac{r-l}{2}, -\frac{t-b}{2}, -n, 1\right)$ (or one of the other three vertices in the table given earlier), we get:

$$\alpha = (r+l)/2n$$
$$\beta = (t+b)/2n$$

A 4.12

Thus the points in the viewing frustum should first be transformed by the shearing matrix

$$M_{shear} = \begin{bmatrix} 1 & 0 & \frac{r+l}{2n} & 0 \\ 0 & 1 & \frac{t+b}{2n} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

A 4.13

If the view frustum is first sheared to align its axis with the z-axis and then projected, we get all points inside the truncated pyramid to be transformed into a rectangular box, with perspective projection at $z = -n$ plane. For the rectangular box to be the NDC space, we require the box to be bounded by $[-1, -1, -1] \times [1, 1, 1]$. We need to scale the points so that the resulting box is of appropriate extents, and that scaling should happen before the perspective projection for the points to project appropriately. Further, the scaling is needed along the x and y axes only. The scaling along the z-direction is properly handled by projection matrix. The required matrix has the form:

$$M_{sacle} = \begin{bmatrix} S_{xx} & 0 & 0 & 0 \\ 0 & S_{yy} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

A 4.14

The so called perspective projection matrix in OpenGL consolidates these three into a single matrix:

$$M_{proj} = M_p * M_{scale} * M_{shear}$$

A 4.15

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} * \begin{bmatrix} S_{xx} & 0 & 0 & 0 \\ 0 & S_{yy} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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

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

In the above equation only unknowns are S_{xx} and S_{yy}. To fix up those values, we notice that this matrix should transform all the points in the truncated viewing pyramid to within the cubic box bounded by $[-1,-1,-1] \times [1,1,1]$, and we pick a convenient boundary point whose transformation is known – like (l, t, -n, 1). This point in the {View} should transform into (-1, -1, -1, 1) in {NDC}.

From equation A 4.15, a generic point $(x, y, z, 1) \in \{NDC\}$ transforms as

$$\left(-S_{xx}\left(\frac{x}{z}+\frac{r+l}{2n}\right),\ -S_{yy}\left(\frac{y}{z}+\frac{t+b}{2n}\right), \frac{f+n}{f-n}+\frac{2fn}{f-n}.\frac{1}{z}\right)$$

Applying this transformation to x and y coordinates of the point (l, b, -n, 1) gives

$$-S_{xx}\left(\frac{l}{-n}+\frac{r+l}{2n}\right) = -1 \qquad \text{A 4.16}$$
$$-S_{yy}\left(\frac{y}{z}+\frac{t+b}{2n}\right) = -1$$

These equations yield $S_{xx} = \frac{2n}{r-l}$, and $S_{yy} = \frac{2n}{t-b}$. This substituted in equation A 4.15 gives

$$M_{proj} = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \qquad \text{A 4.17}$$

This is the OpenGL perspective projection matrix. This delivers the vertices from the {Camera/View} space to {Clip}. Multiplication by this matrix followed by a perspective division delivers the vertices in the {NDC}. That is, if the {Clip} coordinates are $({}^{c}x,\ {}^{c}y,\ {}^{c}z,\ {}^{c}w)$ then {NDC} coordinates are given by $\left(\frac{{}^{c}x}{{}^{c}w}, \frac{{}^{c}y}{{}^{c}w}, \frac{{}^{c}z}{{}^{c}w}, 1\right) \equiv \left({}^{ndc}x, {}^{ndc}y,\ {}^{ndc}z, 1\right)$. But the above matrix delivers the coordinates from the {View} directly into the {NDC} – that is, with the perspective division already done. Within the {NDC} we can determine whether or not the primitive needs to be clipped. If the clipping becomes necessary, the vertices of the primitives need to be taken back to the {Clip} and then clipped and the vertices of the clipped primitive need to be transformed back to the {NDC} again.

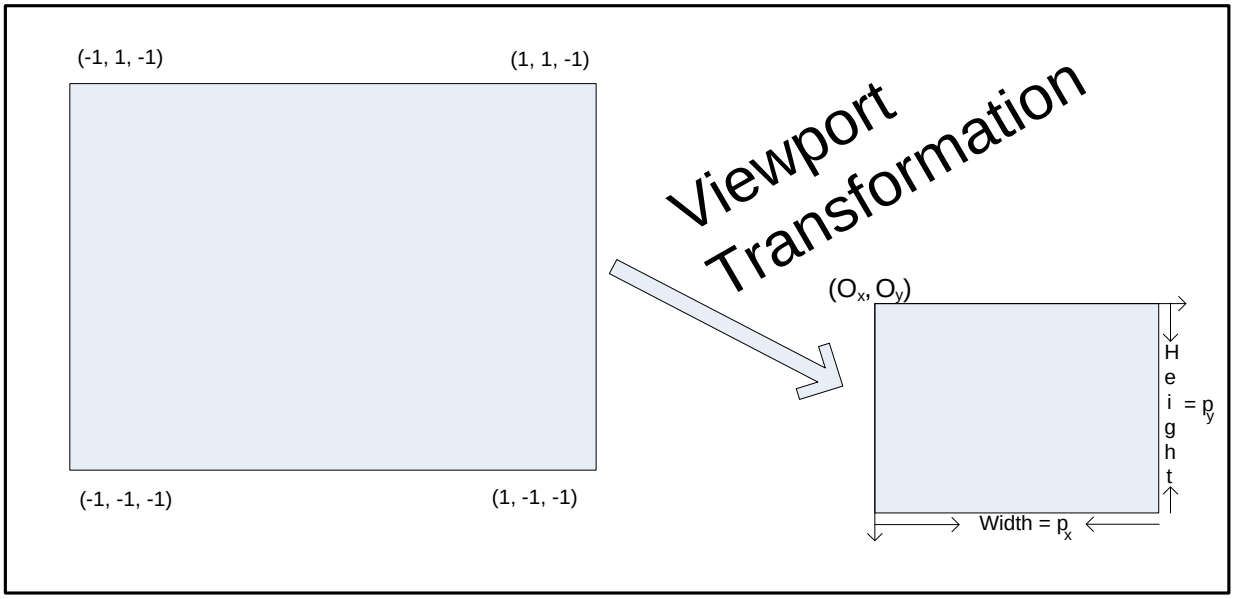

Figure 8-12 OpenGL Viewport Transformation

In the pipeline, this is followed up with a viewport transformation. OpenGL supports the *glViewort(…)* call that takes the viewport origin coordinates, the width and the height as the arguments. This call determines the transformation used to transform the pixel coordinates in {NDC} to viewport pixel coordinates.

The viewport transformation maps the coordinates expressed in units of pixels form the {NDC} to {Viewport}: $\left({}^{ndc}x, {}^{ndc}y, {}^{ndc}z, 1\right) \rightarrow ({}^{w}x, {}^{w}y, {}^{w}z, 1)$ where $\left({}^{ndc}x, {}^{ndc}y, {}^{ndc}z\right) \in [-1, 1]$ and

$$\blacksquare\left({}^{w}x \in [O_x, O_x + p_x] \; @ \; {}^{w}y \in [O_y, O_y + p_y] \; @ \; {}^{w}z \in [0, 1.0]\right)$$

To get this mapping, we first offset each coordinate by unity so that the range is reset from [-1, 1] to [0, 2]. Then we scale it appropriately – based on the viewport dimensions for the x- and y- coordinates and to the range [0, 1.0] in the last case. This mapping is given by the transformations

$$\begin{aligned} {}^{w}x &= O_x + \frac{p_x}{2}\left({}^{ndc}x + 1\right) \\ {}^{w}x &= O_y + \frac{p_y}{2}\left({}^{ndc}y + 1\right) \\ {}^{w}z &= \frac{{}^{ndc}z + 1}{2} \end{aligned} \qquad \text{A 4.18}$$

This equation seems simple enough to be more efficient if implemented directly as opposed to being implemented as a matrix. In reality whether it is implemented as a matrix or as a direct transformation depends on how the rest of the pipeline is implemented. Given that the matrix operations themselves are implemented as composite of multiply and add operations means most likely this is implemented as a matrix operation. For completeness the viewport transformation matrix is given below.

$$M_{viewport} = \begin{bmatrix} \frac{p_x}{2} & 0 & 0 & O_x + \frac{p_x}{2} \\ 0 & \frac{p_y}{2} & 0 & O_y + \frac{p_y}{2} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{A 4.19}$$

A note on the zbuffer resolution

The z-coordinate transformed to the {NDC} is given by equation A 4.17 as: ${}^{ndc}z = \frac{f+n}{f-n} + \frac{2fn}{f-n}\ {}^{view}z$. The ${}^{ndc}z$ is plotted against ${}^{view}z$ for various values of 'n' at a constant value of *f* – shown below.