Projection Transformations

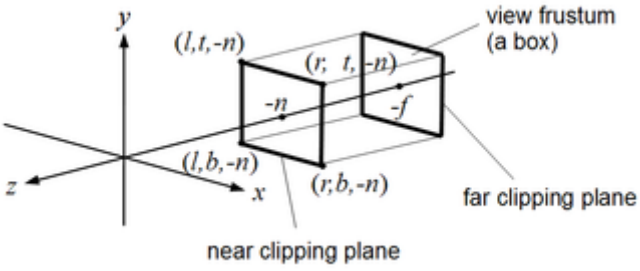
The purpose is to deform the frustum into a canonical box defined by the points (-1,-1,1) to (1,1,-1) (normalized device coordinates) which are independent of device or application program. By doing this, the subsequent operations such as clipping, hidden surface removal (culling), and transforming to screen coordinates (viewport) are easy to do and are identical regardless of the type of projection and/or size of the original frustum, computer, etc.

Below, we use the following notation to specify the positions of the various planes defining the frustum:

f = front, n = near, t = top, b = bottom, r = right, l = left

Remember: `near` is the positive *distance* of the near plane from the camera. Since the camera is looking down the negative z axis, the z value of the near plane is at `-near`

Orthographic:



Transforms frustum into cube with corners ((-1,-1,1) and (1,1,-1)

This matrix can be derived from a translation and scale. The translation is done first; it moves the frustum so that its center sits at the origin. The scale, then scales everything about the origin so that the frustum has dimensions 2x2x2:

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

The result is that the orthographic transformation is:

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

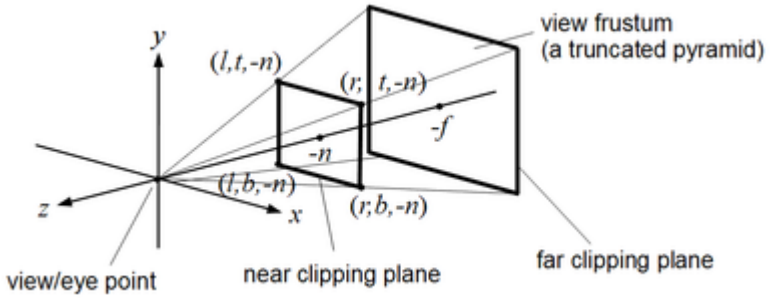
Prove to yourself that this works by applying the above transformation to the corners of the frustum. For example, the point (r,t,-n) should be transformed into (1,1,-1).

Here is the code for doing this from Angel's mat.h file:

```
mat4 Ortho( const GLfloat left, const GLfloat right,
            const GLfloat bottom, const GLfloat top,
            const GLfloat zNear, const GLfloat zFar )
{
    mat4 c;
    c[0][0] = 2.0/(right - left);
    c[1][1] = 2.0/(top - bottom);
    c[2][2] = 2.0/(zNear - zFar);
    c[3][3] = 1.0;
    c[0][3] = -(right + left)/(right - left);
    c[1][3] = -(top + bottom)/(top - bottom);
    c[2][3] = -(zFar + zNear)/(zFar - zNear);
    return c;
}
```

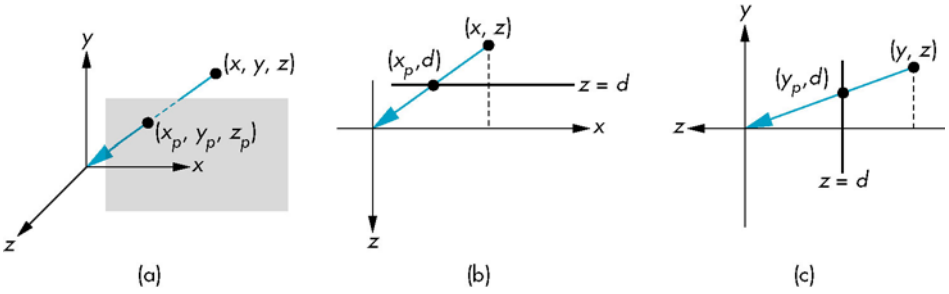
http://en.wikibooks.org/wiki/GLSL_Programming/Vertex_Transformations

Perspective Transform:



Again, we want to transform the frustum into cube with corners ((-1,-1,1) and (1,1,-1).

To understand how to do this transform, consider the following.



We use similar triangles to obtain the relationship $\frac{y_p}{d} = \frac{y}{z}$ or $y_p = y / (z/d)$. Similar for $x_p = x/(z/d)$ to give projected point on the plane at z=d

so $P' = \begin{pmatrix} \frac{x}{z/d} \\ \frac{y}{z/d} \\ d \\ 1 \end{pmatrix}$

Note:

- Perspective transform is nonlinear
- It depends on z-value, that is, transform is different for every point, which is problematic. In order to get rid of the z dependence, the division by z/d is done later and is called the "perspective division"

The resulting matrix for transforming the skewed frustum into a rectangular box is

$MP = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ z/d \end{pmatrix}$, Note: M is the same for each point since d is fixed

Later, we need to divide by (w=z/d) to get the point back in the proper format for homogeneous coordinates (i.e. having a 1 in the last position) to give P' shown above.

The resulting perspective transform (based on the above nonlinear transform combined with translation and rescaling) gives:

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

One can show that this works by showing its effect on the corners of the above frustum, that is, it transforms the frustum into a cube with sides of length 2 and centered at origin.

```
mat4 Frustum( const GLfloat left, const GLfloat right,
              const GLfloat bottom, const GLfloat top,
              const GLfloat zNear, const GLfloat zFar )
{
    mat4 c;
    c[0][0] = 2.0*zNear/(right - left);
    c[0][2] = (right + left)/(right - left);
    c[1][1] = 2.0*zNear/(top - bottom);
    c[1][2] = (top + bottom)/(top - bottom);
    c[2][2] = -(zFar + zNear)/(zFar - zNear);
    c[2][3] = -2.0*zFar*zNear/(zFar - zNear);
    c[3][2] = -1.0;
    return c;
}
```

Note – above will require perspective division when done. This division is done automatically by openGL so the programmer doesn't have to do it explicitly.

Below (Perspective code) is the same as Frustum code above but it assumes that:
left = - right, and bottom = - top.

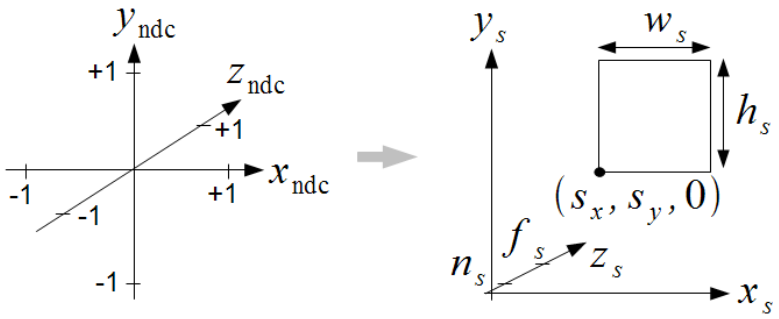
```
mat4 Perspective( const GLfloat fovy, const GLfloat aspect,
                  const GLfloat zNear, const GLfloat zFar)
{

    GLfloat top   = tan(fovy*DegreesToRadians/2) * zNear;
    GLfloat right = top * aspect;

    mat4 c;
    c[0][0] = zNear/right;
    c[1][1] = zNear/top;
    c[2][2] = -(zFar + zNear)/(zFar - zNear);
    c[2][3] = -2.0*zFar*zNear/(zFar - zNear);
    c[3][2] = -1.0;
    return c;
}
```

Viewport Transform

The viewport transform converts from the normalized device coordinates to the screen (or pixel) space.



Which is just a scale and transform in the xy plane. At this point, once can project down onto the xy plane, essentially just removing the z coordinate.

$$Viewport = \begin{pmatrix} w_s/2 & 0 & 0 & s_x + w_s/2 \\ 0 & h_s/2 & 0 & s_y + h_s/2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$