

# Matrices in OpenGL

Dr. Thomas W. Sederberg

8 February 2005

## 1 Eye Position

In native, OpenGL the eye is always located at the origin and looks in the  $-z$  direction. However, the glu function

```
void gluLookAt( GLdouble Ex, GLdouble Ey, GLdouble Ez, GLdouble Ax,
GLdouble Ay, GLdouble Az, GLdouble Upx, GLdouble Upy, GLdouble Upz );
```

creates a  $4 \times 4$  matrix that transforms world coordinates into eye coordinates (again, in this coordinate system, the Eye is at the origin and the view direction is along the  $-z$  axis. Figure 1 illustrates the parameters used in `gluLookAt()`:  $E = (E_x, E_y, E_z)$  is the eye position, given in world coordinates;  $A = (A_x, A_y, A_z)$  is the look-at point, also in world coordinates; and  $Up = (Up_x, Up_y, Up_z)$  is the up vector, in world coordinates.

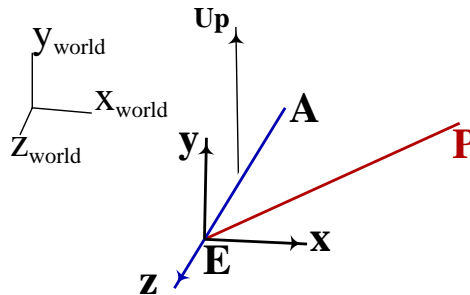


Figure 1: View Frustum

The vectors  $\mathbf{x} = (x_1, x_2, x_3)$ ,  $\mathbf{y} = (y_1, y_2, y_3)$ , and  $\mathbf{z} = (z_1, z_2, z_3)$  (all in world coordinates) are found as follows:

$$\begin{aligned}\mathbf{z} &= \frac{\mathbf{E} - \mathbf{A}}{\|\mathbf{E} - \mathbf{A}\|} \\ \mathbf{x} &= \frac{\mathbf{Up} \times \mathbf{z}}{\|\mathbf{Up} \times \mathbf{z}\|} \\ \mathbf{y} &= \mathbf{z} \times \mathbf{x}.\end{aligned}$$

Denote by  $\mathbf{P}_W = (x_W, y_W, z_W, 1)$  the world coordinates of an arbitrary point  $\mathbf{P}$  and by  $\mathbf{P}_E = (x_E, y_E, z_E, 1)$  the eye coordinates of  $\mathbf{P}$ . We can convert from  $\mathbf{P}_W$  to  $\mathbf{P}_E$  as follows:

$$x_E = (\mathbf{P}_W - \mathbf{E}) \cdot \mathbf{x}; \quad y_E = (\mathbf{P}_W - \mathbf{E}) \cdot \mathbf{y}; \quad z_E = (\mathbf{P}_W - \mathbf{E}) \cdot \mathbf{z}.$$

This can be written in matrix form  $M_E \mathbf{P}_W = \mathbf{P}_E$ , or

$$\begin{bmatrix} x_1 & x_2 & x_3 & -\mathbf{E} \cdot \mathbf{x} \\ y_1 & y_2 & y_3 & -\mathbf{E} \cdot \mathbf{y} \\ z_1 & z_2 & z_3 & -\mathbf{E} \cdot \mathbf{z} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} W_x \\ W_y \\ W_z \\ 1 \end{bmatrix} = \begin{bmatrix} E_x \\ E_y \\ E_z \\ 1 \end{bmatrix}$$

## 2 glFrustum

The `glFrustum` function creates a matrix that performs a perspective projection. The parameters are:

```
void glFrustum( GLdouble left, GLdoubleright, GLdouble bottom, GLdouble top,
GLdouble znear, GLdouble zfar );
```

$z_{near}$  and  $z_{far}$  are the distances from the origin to the near and far clipping planes. These values are always positive, so therefore the near clipping plane is  $z = -z_{near}$  and the far clipping plane is  $z = -z_{far}$ . The parameters *left*, *right*, *top*, *bottom* define vertices of the view frustum, as shown in the figure.

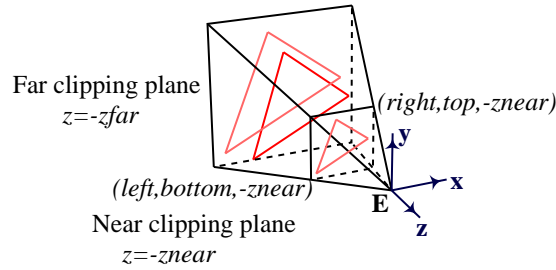


Figure 2: View Frustum

`glFrustum()` creates a transformation matrix  $M_C$  that maps the view frustum into a cube  $[-1, 1] \times [-1, 1] \times [-1, 1]$ :

$$M_C = \begin{bmatrix} \frac{2 \cdot z_{near}}{right - left} & 0 & \frac{right + left}{right - left} & 0 \\ 0 & \frac{2 \cdot z_{near}}{top - bottom} & \frac{top + bottom}{top - bottom} & 0 \\ 0 & 0 & -\frac{z_{far} + z_{near}}{z_{far} - z_{near}} & -\frac{2 \cdot z_{far} \cdot z_{near}}{z_{far} - z_{near}} \\ 0 & 0 & -1 & 0 \end{bmatrix}. \quad (1)$$

The precision of the  $z$ -buffer is influenced by the values of  $z_{near}$  and  $z_{far}$ . If the ratio  $r = \frac{z_{far}}{z_{near}}$  gets too large, roundoff errors in the hidden surface algorithm can become evident. Thus, never set  $z_{near} = 0$ .

The current projection matrix can be retrieved using the command  
**void** glGetFloatv(**GL\_PROJECTION\_MATRIX**, **GLfloat** \*params);

A point in the Eye coordinate system can be mapped to the cube by multiplying by  $M_C$ :  $M_C \mathbf{P}_E = \mathbf{P}_C$ , or

$$\begin{bmatrix} \frac{2 \cdot z_{near}}{right-left} & 0 & \frac{right+left}{right-left} & 0 \\ 0 & \frac{2 \cdot z_{near}}{top-bottom} & \frac{top+bottom}{top-bottom} & 0 \\ 0 & 0 & -\frac{z_{far}+z_{near}}{z_{far}-z_{near}} & -\frac{2 \cdot z_{far} \cdot z_{near}}{z_{far}-z_{near}} \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} E_x \\ E_y \\ E_z \\ 1 \end{bmatrix} = \begin{bmatrix} C_x \\ C_y \\ C_z \\ C_w \end{bmatrix}$$

Since  $\mathbf{P}_C$  is in homogeneous coordinates, a division by  $C_w$  is required in order to project to Cartesian coordinates:  $\left(\frac{C_x}{C_w}, \frac{C_y}{C_w}, \frac{C_z}{C_w}\right)$ .

### 3 glOrtho

Orthographic projection uses a slightly modified version of  $M_C$ :

$$M_C = \begin{bmatrix} \frac{2}{right-left} & 0 & 0 & -\frac{right+left}{right-left} \\ 0 & \frac{2}{top-bottom} & 0 & -\frac{top+bottom}{top-bottom} \\ 0 & 0 & -\frac{2}{z_{near}-z_{far}} & -\frac{z_{near}+z_{far}}{z_{near}-z_{far}} \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (2)$$

In an orthographic view, the values of  $z_{near}$  and  $z_{far}$  are taken to be *positive* distances from the eye. Since we are always looking down the *negative*  $z$ -axis of the eye coordinate system, the actual  $z$ -coordinates of the near and far clipping planes are  $-z_{near}$  and  $-z_{far}$ , respectively. This explains why the element of the third row and third column in  $M_C$  is  $-\frac{2}{z_{near}-z_{far}}$ .

### 4 Pixel Coordinates

The cube can be thought of as a unit viewport; the final transformation that we need is to map to the true viewport in pixel coordinates. Assume that the true viewport has center  $V_{cx}, V_{cy}$  and half-widths  $V_x, V_y$ . Then one final matrix is needed to map to the viewport:  $M_V \mathbf{P}_C = \mathbf{P}_V$ :

$$\begin{bmatrix} V_x & 0 & 0 & V_{cx} \\ 0 & V_y & 0 & V_{cy} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} C_x \\ C_y \\ C_z \\ C_w \end{bmatrix} = \begin{bmatrix} P_x \\ P_y \\ P_z \\ P_w \end{bmatrix}$$

The actual pixel coordinates are now  $\left(\frac{P_x}{P_w}, \frac{P_y}{P_w}\right)$  and the  $z$  value for the point, to be used in Z-buffering, is  $\frac{P_z}{P_w}$ . Points on the near clipping plane end up with a  $z$  value of  $-1$  in the pixel coordinate system, and points on the far clipping plane have  $z$  coordinates of  $1$  in that coordinate system. Think of the  $z$  coordinate as a normalized distance from the eye. Thus, when doing  $z$ -buffering, you want to retain the color of points with the *smallest*  $z$ -coordinate.

It is possible to create a single  $4 \times 4$  matrix  $M_{VCE}$  that will map a point in world coordinates directly into homogeneous pixel coordinates:

$$M_{VCE}\mathbf{P}_W = \mathbf{P}_V \quad \text{where} \quad M_{VCE} = M_V M_C M_E \quad (3)$$