4.1

Tweaking a Vertex's Projected Depth Value

Eric Lengyel

Many games need to render special effects such as scorch marks on a wall or footprints on the ground that are not an original part of a scene, but are created during gameplay. These types of decorative additions are usually decaled onto an existing surface, and thus consist of polygons that are coplanar with other polygons in a scene. The problem is that pixels rendered as part of one polygon rarely have exactly the same depth value as pixels rendered as part of a coplanar polygon. The result is an undesired pattern in which parts of the original surface show through the decaled polygons.

The goal is to find a way to offset a polygon's depth in a scene without changing its projected screen coordinates or altering its texture mapping perspective. Most 3D graphics libraries contain some kind of polygon offset function to help achieve this goal. However, these solutions generally lack fine control and usually incur a per-vertex performance cost. This article presents an alternative method that modifies the projection matrix to achieve the depth offset effect.

Examining the Projection Matrix

Let us first examine the effect of the standard OpenGL perspective projection matrix on an eye space point $\mathbf{P} = (P_x, P_y, P_z, 1)$. To simplify our matrix, we assume that the view frustum is centered about the z-axis in eye space (i.e., the rectangle on the near clipping plane carved out by the four side planes has the property that $left = -right$ and $bottom = -top$). Calling the distance to the near clipping plane n and the distance to the far clipping plane f, we have:

$$\begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} = \begin{bmatrix} nP_x \\ nP_y \\ -\frac{f+n}{f-n}P_z - \frac{2fn}{f-n} \\ -P_z \end{bmatrix}. \quad (4.1.1)$$

To finish the projection, we need to divide this result by its w-coordinate, which has the value $-P_z$. This division gives us the following projected 3D point, which we will call $\mathbf{P}'$.

$$\mathbf{P}' = \begin{bmatrix} -\frac{nP_x}{P_z} \\ -\frac{nP_y}{P_z} \\ \frac{f+n}{f-n} + \frac{2fn}{P_z(f-n)} \end{bmatrix} \quad (4.1.2)$$

Recall that the near clipping plane lies at $z = -n$, and the far clipping plane lies at $z = -f$ since the camera points in the negative z direction. Thus, plugging $-n$ and $-f$ into Equation 4.1.2 for P_z gives us the expected z values of -1 and 1 bounding the normalized clipping volume. Also recall that this mapping from $[-n..-f]$ to $[-1..1]$ is a function of inverse z. This is necessary so that linear interpolation by the 3D hardware of values in the depth buffer remain perspective correct.

Tweaking the Depth Value

It is clear from Equation 4.1.2 that preserving the value of $-P_z$ for the w-coordinate will guarantee the preservation of the projected x- and y-coordinates as well. From this point forward, we shall only concern ourselves with the lower-right 2×2 portion of the projection matrix, since this is the only part that affects the z- and w-coordinates. The projected z-coordinate may be altered without disturbing the w-coordinate by introducing a factor of $1 + \varepsilon$, for some small ε, as follows.

$$\begin{bmatrix} -(1+\varepsilon)\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} P_z \\ 1 \end{bmatrix} = \begin{bmatrix} -(1+\varepsilon)\frac{f+n}{f-n}P_z - \frac{2fn}{f-n} \\ -P_z \end{bmatrix} \quad (4.1.3)$$

After division by w, we arrive at the following value for the projected z-coordinate.

$$P_z' = (1+\varepsilon)\frac{f+n}{f-n} + \frac{2fn}{P_z(f-n)}$$
$$= \frac{f+n}{f-n} + \frac{2fn}{P_z(f-n)} + \varepsilon\frac{f+n}{f-n} \tag{4.1.4}$$

Comparing this to the z-coordinate in Equation 4.1.2, we see that we have found a way to offset projected depth values by a constant $\varepsilon\frac{f+n}{f-n}$.

Choosing an Appropriate Epsilon

Due to the nonlinear nature of the z-buffer, the constant offset given in Equation 4.1.4 corresponds to a larger eye space difference far from the camera than it does near the camera. While this constant offset may work well for some applications, there is no single solution that works for every application at all depths. The best we can do is choose an appropriate ε given an eye space offset δ and a depth value P_z which collectively represents the object that we are offsetting. To determine a formula for ε, let us examine the result of applying the standard projection matrix from Equation 4.1.1 to a point whose z-coordinate has been offset by some small δ.

$$\begin{bmatrix} -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} P_z+\delta \\ 1 \end{bmatrix} = \begin{bmatrix} -\frac{f+n}{f-n}(P_z+\delta) - \frac{2fn}{f-n} \\ -(P_z+\delta) \end{bmatrix} \tag{4.1.5}$$

Dividing by w, we have the following value for the projected z-coordinate.

$$P_z' = \frac{f+n}{f-n} + \frac{2fn}{(P_z+\delta)(f-n)}$$
$$= \frac{f+n}{f-n} + \frac{2fn}{P_z(f-n)} + \frac{2fn}{f-n}\left(\frac{1}{P_z+\delta} - \frac{1}{P_z}\right) \tag{4.1.6}$$

Equating this result to Equation 4.1.4 and simplifying a bit, we end up with:

$$\varepsilon = -\frac{2fn}{f+n}\left(\frac{\delta}{P_z(P_z+\delta)}\right). \tag{4.1.7}$$

A good value of δ for a particular application can be found with a little experimentation. In should be kept in mind that δ is an eye space offset, and thus becomes

less effective as P_z gets larger. For an m-bit integer depth buffer, we want to make sure that:

$$|\varepsilon| \geq \frac{1}{2^m - 1}\left(\frac{f - n}{f + n}\right) \tag{4.1.8}$$

since smaller values of ε will not yield an offset significant enough to alter the integer depth value. Substituting the right side of Equation 4.1.7 for ε and solving for δ gives us:

$$\delta \geq \frac{kP_z^2}{1 - kP_z} \tag{4.1.9}$$

or:

$$\delta \leq \frac{-kP_z^2}{1 + kP_z} \tag{4.1.10}$$

where the constant k is given by:

$$k = \frac{f - n}{2fn\left(2^m - 1\right)}. \tag{4.1.11}$$

Equation 4.1.9 gives us the minimum effective value for δ when offsetting a polygon toward the camera (the usual case), and Equation 4.1.10 gives us the maximum effective value for δ when offsetting a polygon away from the camera.

Implementation

The following sample code demonstrates how the projection matrix shown in Equation 4.1.3 may be implemented under OpenGL. The function `LoadOffsetMatrix` takes the same six values that are passed to the OpenGL function `glFrustum`. It also takes values for δ and P_z that are used to calculate ε.

Source Code

```
#include <gl.h>

void LoadOffsetMatrix(GLdouble l, GLdouble r,
GLdouble b, GLdouble t,
GLdouble n, GLdouble f,
GLfloat delta, GLfloat pz)
{
    GLfloat matrix[16];
```

```
        // Set up standard perspective projection
        glMatrixMode(GL_PROJECTION);
        glFrustum(l, r, b, t, n, f);

        // Retrieve the projection matrix
        glGetFloatv(GL_PROJECTION_MATRIX, matrix);

        // Calculate epsilon with equation (4.1.7)
        GLfloat epsilon = -2.0F * f * n * delta /
        ((f + n) * pz * (pz + delta));

        // Modify entry (3,3) of the projection matrix
        matrix[10] *= 1.0F + epsilon;

        // Send the projection matrix back to OpenGL
        glLoadMatrixf(matrix);
    }
```