glTexGen and glClipPlane

Andreas Bærentzen

November 12, 2002

This is a brief note (actually an elaboration of a long email) that I wrote to explain how glTexGen and glClipPlane work. The documentation for automatic texture coordinate generation and clip plane specification is a bit confusing. I think this is because we are moving back and forth between eye and object coordinates. I hope this note makes things more clear.

glClipPlane Let us look at the problem of setting an appropriate clipping plane. Assume that the plane is defined by the normal $\mathbf{n} = [p_x p_y p_z]$ and a point $\mathbf{p}_0$. In that case, we can describe the plane by the 4D vector

$$\mathbf{P} = [p_x p_y p_z p_d] \tag{1}$$

where $p_d = -\mathbf{n} \cdot \mathbf{p}_0$.

Clipping always takes place in eye coordinates, i.e. after the modelview matrix has been mulitplied onto the coordinates.

However, the user frequently wants to specify the plane equation in object coordinates. Hence, the plane equation is multiplied by the inverse modelview matrix. This corresponds to specifying the clip plane in object coordinate, i.e. before modelview transformation.

The clipping equation looks as follows:

$$dist = [p_x p_y p_z p_d] M^{-1} \cdot [x_e y_e z_e w_e] \geq 0 \tag{2}$$

where $\cdot$ denotes dot product. In other words, if dist is zero or positive, the point passes, otherwise it is clipped. Note that, in the equation, we left multiply the plane onto the inverse modelview matrix. This is the same as if we had right multiplied $[x_e y_e z_e w_e]$ onto the inverse modelview matrix. In either case, the result is to simply "undo" the modelview matrix.

So far so good, but if we now change the modelview matrix *after we have specified the clip plane*, the clip plane remains unaltered. So, if effect, the equation looks like the following

$$dist = [p_x p_y p_z p_d] {M_A}^{-1}, M_B [x_o y_o z_o w_o]^T) >= 0 \tag{3}$$

where $[x_o y_o z_o w_o]$ are the point object coordinates, M_A is the modelview matrix that was active when the clip plane was specified, and M_B is the matrix that was active when transforming the point in question. This has the important implication that when the clip plane has been specified, it is constant w.r.t. the position of the eye.

glTexGen At this point let us look at glTexGen, because precisely the sample principle is true of glTexGen when we use GL_EYE_LINEAR. In this case, the equation for each of the coordinates is precisely as the above equation. Our point in eye coordinates is transformed back into object coordinates with the inverse modelview transformation, and the generated texture coordinate is equal to the distance to a plane. For instance, the code below

```
float gls[] = {1,0,0,0};
glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_EYE_LINEAR);
glTexGenfv(GL_S, GL_EYE_PLANE, gls);
glEnable(GL_TEXTURE_GEN_S);
```

Means that the s texture coordinate becomes

$$s = [p_x p_y p_z p_d] M^{-1} \cdot [x_e y_e z_e w_e] \tag{4}$$

where $[p_x p_y p_z p_d] = [1\ 0\ 0\ 0]$ ... but this plane is, clearly, just the plane through the origin with the (unit) normal coinciding with the x axis. It is now obvious that the distance is simply the x coordinate.

So we don't have to think in terms of planes. If we do the same for t, r, and q the setup simply corresponds to multiplying the vector of eye coordinates onto the identity matrix – having first multiplied them unto the inverse modelview matrix.

If the modelview matrix is the same as when the automatic texture generation was set up, we are doing more work than necessary. However, if the modelview matrix changes, the texture coordinate generation stays constant with respect to the eye.

Object coordinates To make the texture coordinate constant with regard to the object, we can use object coordinates in the specification of the texture coordinate generation.

```
float gls[] = {1,0,0,0};
float glt[] = {0,1,0,0};
float glr[] = {0,0,1,0};
float glq[] = {0,0,0,1};
glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR);
glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR);
glTexGeni(GL_R, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR);
glTexGeni(GL_Q, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR);
glTexGenfv(GL_S, GL_OBJECT_PLANE, gls);
glTexGenfv(GL_T, GL_OBJECT_PLANE, glt);
glTexGenfv(GL_R, GL_OBJECT_PLANE, glr);
glTexGenfv(GL_Q, GL_OBJECT_PLANE, glq);
glEnable(GL_TEXTURE_GEN_S);
glEnable(GL_TEXTURE_GEN_T);
glEnable(GL_TEXTURE_GEN_R);
glEnable(GL_TEXTURE_GEN_Q);
```

Now, the object coordinates are used directly, and the equation for s is

$$s = [x_o y_o z_o w_o] \cdot [p_x p_y p_z p_d] \tag{5}$$

and we observe that no modelview matrix is involved, so the texture generation follows the object.

It is really quite simple