Questions

1. I claimed in the lecture that when mapping to a new coordinate system by the matrix $\mathbf{M}$, the surface normal should be mapped using $\mathbf{M}^{-T}$ i.e. the transpose of the inverse (or equivalently, the inverse of the transpose).

   (a) What is the inverse of a 4×4 translation matrix $\mathbf{M}$?

   (b) **[updated March 25]** Show that if $\mathbf{M}$ is defined by a rotation and/or translation, then for any surface normal vector $\mathbf{n}$ that is represented as a point at infinity,

   $$\mathbf{n} = (n_x, n_y, n_z, 0)$$

   $\mathbf{M}^{-T}\mathbf{n}$ and $\mathbf{Mn}$ represent the same *unit* surface normal. That is, when we normalize these two transformed vectors, we get the same unit length vector.

   (c) Show that if $\mathbf{M}$ also includes a scaling matrix, then there is no guarentee that the relation in (b) holds.

   (d) In an OpenGL vertex shader, is it ok to use the `GL_MODELVIEW` matrix when mapping the surface normal ? If not, why not?

   (In OpenGL 1.x, the user never sees how the normal is mapped, but in modern graphics programming one needs to map the normal oneself.)

2. (a) How would you do texture mapping with a bicubic? What challenges arise that do not arise with a polygon or quadric?

   (b) Does the same problem arise for Phong shading a bicubic? If so, explain why. If not, why not?

3. Bump mapping is based on the following model of a surface $\mathbf{p}_{bump}(s,t)$,

   $$\mathbf{p}_{bump}(s,t) = \mathbf{p}(s,t) + b(s,t)\mathbf{n}(s,t)$$

   where $\mathbf{n}(s,t)$ is the normal vector on $\mathbf{p}(s,t)$ and $b(s,t)$ is the height of the bumps.

   Consider the specific case of adding a bump map to a rectangle, lying in the plane $y = y_0$. Assume the normal of the rectangle is the unit y vector.

   (a) Write the above equation for this specific case. Be sure to specify the 3D coordinates of points on the rectangle.

   (b) How you would compute the surface normal on this bump mapped surface?

4. Suppose you wanted to automatically enhance the outline of a smooth surface surface by coloring it black. Would it be better to do this coloring using a vertex shader or a fragment shader ?

5. Highlights are not isolated points, but rather define small neighborhoods on the surface. The area of a highlight depends on the shininess. But it also depends on how curved the surface is. Why?

Answers

1. (a) Verify that the translation matrix on the right is the inverse of the translation matrix on the left by multiplying it out and seeing that the result is indeed the identity matrix.

$$\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -t_x \\ 0 & 1 & 0 & -t_y \\ 0 & 0 & 1 & -t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(b) The case of rotation should be easy since $\mathbf{R}^{-1} = \mathbf{R}^T$ and so $\mathbf{R}^{-T} \equiv (\mathbf{R}^{-1})^T = \mathbf{R}$.

The case of translation is much more subtle. Here,

$$\mathbf{Mv} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \\ 0 \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ v_z \\ 0 \end{bmatrix}$$

and

$$\mathbf{M}^{-T}\mathbf{v} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -t_x & -t_y & -t_z & 1 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \\ 0 \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ v_z \\ -t_x v_x - t_y v_y - t_z v_z \end{bmatrix}$$

So in the first case, the point at infinity maps to itself, whereas in the second case it maps to a finite point that is along the line in the same direction (v_x, v_y, v_z). Note that these two 4D vectors do not represent the same 3D vector since the first is a point at infinity and the second is typically a finite point (unless $\mathbf{t}$ and $\mathbf{v}$ happen to be orthogonal). However the directions of these two vectors *are* the same, which means that when you normalize the two vectors i.e. find a unit vector in the direction of each of them, you get the same (unit) vector.

If this is true for rotations and translation, then it is true for a composition of these mappings. e.g. If you do the translation first and then the rotation, the claim still holds since the translated vectors are in the same direction and rotation will take two vectors that have the same direction and map them to two vectors that have the same direction.

(c) It is easy to come up with a counter example. Take the following scaling matrix and its inverse, where $a \neq 1$.

$$\mathbf{M} = \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{M}^{-T} = \begin{bmatrix} \frac{1}{a} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Multiplying a point at infinity by each of these matrices gives different answers.

(d) No, its not ok. If the `GL_MODELVIEW` includes a scaling e.g. if you have an ellipsoid, then the normals will not be mapped properly.

2. (a) The main problem is that for any pixel (x_p, y_p) in your image, you need to find the coorsponding texture coordinate (s, t) so that you can lookup the corresponding value (s_p, t_p) in the texture map. However, there is no obvious way to do this inverse mapping since the bicubic function is not linear (and so there is no homography).

   There are two solutions people use. One solution is to approximate the bicubic by triangulating the (s, t) space. This essentially does surface subdivision (lecture 11). But it is costly in terms of space since it requires many more vertices. A second solution is to use numerical methods. Given an (x_p, y_p), run an iterative algorithm to find a corresponding (s, t). This comes down to finding roots of a degree 3 polynomial.

   (b) Not exactly the same problem arises, but a similar problem arises. To shade a bicubic with Phong shading, you need to interpolate the surface normal across the image projection of the surface. It is not obvious how to do that properly. If you just interpolate the normals from the control point vertices, you will get a smooth change in the normals but there is no guarentee that the result will be correct. This is just like question (a), if you had interpolated the texture coordinates (s, t) from the control points. (Here I am implicitly assuming that the texture coordinates and bicubic parameters are the same. This need not be the case in general.)

3. (a)

$$\begin{aligned}
\mathbf{p}(s,t) &= (s, y_0, t) \\
\mathbf{n}(s,t) &= (0,1,0) \\
\mathbf{p}_{bump}(s,t) &= \mathbf{p}(s,t) + b(s,t)(0,1,0) \\
&= (s, y_0 + b(s,t), t)
\end{aligned}$$

   (b)

$$\begin{aligned}
\frac{\partial \mathbf{p}_{bump}(s,t)}{\partial s} &= \frac{\partial \mathbf{p}(s,t)}{\partial s} + \frac{\partial b(s,t)}{\partial s}\mathbf{n}(s,t) \\
&= (1,0,0) + \frac{\partial b(s,t)}{\partial s}(0,1,0)
\end{aligned}$$

   and

$$\begin{aligned}
\frac{\partial \mathbf{p}_{bump}(s,t)}{\partial t} &= \frac{\partial \mathbf{p}(s,t)}{\partial t} + \frac{\partial b(s,t)}{\partial t}\mathbf{n}(s,t) \\
&= (0,0,1) + \frac{\partial b(s,t)}{\partial t}(0,1,0)
\end{aligned}$$

   So,

$$\frac{\partial \mathbf{p}_{bump}(s,t)}{\partial s} \times \frac{\partial \mathbf{p}_{bump}(s,t)}{\partial t} = (0,-1,0) + \frac{\partial b(s,t)}{\partial s}(1,0,0) + \frac{\partial b(s,t)}{\partial t}(0,0,1)$$

This vector is in the direction of the surface normal. To get a unit surface normal you need to normalize.

4. It would be better to use a fragment shader. The condition you want to test is that the surface normal at that fragment is roughly perpendicular to the viewing direction $(0, 0, -1)$. If it is, then color the point black, i.e. RGB = (0,0,0).

   What is wrong with using a vertex shader for this? To use a vertex shader for computing RGB values, you need to know the normal at this vertex. Then the fragment shader will interpolate the colors from these vertices (i.e. "smooth" shading). This would have several problems. First, if the surface is a mesh, then you would need to have a vertex whose normal was perpendicular to the viewing direction. Since there are far fewer vertices than fragments typically, you would have to get lucky. Second - and this is a more serious problem - the rasterizer would then interpolate the black outline into the interior of the object. This is probably not what you want.

5. First let's make sure we understand why the size of the highlight depends on the shininess (exponent). As the shininess is increasesd, the area of the highlight becomes smaller. Intuitive, polishing an apple increases the shininess and this leads to a sharper highlight. The reason why the highlight becomes sharper is that a large exponent will nearly zero out the specular terms for which $\mathbf{n} \cdot \mathbf{H}$ is different from 1, where $\mathbf{H}$ is the Blinn-Phong half vector.

   What about the curvature of the surface? For a more highly curved surface, the area of the highlight is smaller. The reason is that higher curvature means that the surface normal varies more quickly across the surface, The more quickly $\mathbf{n}$ varies, the less likely that $\mathbf{H} \cdot \mathbf{n}$ can maintain a value near 1. Thus, curved surfaces tend to produce smaller highlights.