Questions

1. The mapping from the projection plane (x, y) to the image pixel space (x_p, y_p) was discussed in lecture 6 and in Exercises 6 Questions 5,6. This is the window-to-viewport transformation.

   What is the mapping between continuous texture coordinates (s, t) and the discrete "texel space" (s_p, t_p) on which the texture image $T(s_p, t_p)$ is defined ?

   Note that the window-to-viewport transformation and the transformation between pixel and texel coordinates are both homographies.

2. (a) Consider a 3D triangle that is defined by vertices:

   $$\mathbf{p}_1 = (-5, 2, -7), \quad \mathbf{p}_2 = (-6, 4, -8), \quad \mathbf{p}_3 = (-7, 5, -3)$$

   and suppose that we project this triangle onto the $z = -2$ plane, using perspective projection with the center of projection at the origin. We also texture map this triangle, and choose texture coordinates such that

   $$(s, t) = (0, 0) \rightarrow \mathbf{p}_1$$

   $$(s, t) = (1, 0) \rightarrow \mathbf{p}_2$$

   $$(s, t) = (0, 1) \rightarrow \mathbf{p}_3.$$

   Give the homography that corresponds to the mapping from (s, t) to the projection plane coordinates (x, y).

   (b) Suppose we were to consider *orthographic projection* in the z direction, rather than perspective projection. What would the homography be then?

   (c) Suppose that for (a) you were given other (s, t) texture coordinates for the vertices of the triangle, say particular values $(s_i, t_i), i = 1, 2, 3$. How would you compute a homography in that case?

3. When does the inverse of a homography NOT exist ?

4. Recall the slanted plane example from the lecture. Let the *horizon* be the image projection of points on the plane at infinity. (Technically, points at infinity are not *on* the slanted plane. Rather, points at infinity are only defined in the limit.)

   For this example, where is the horizon in the image?

5. Suppose we inverse map an image pixel to the texture domain (s_p, t_p) using the inverse homography. We would like to interpolate the value of $T(s_p, t_p)$ using values from the surrounding four texture pixels:

   $$(floor(x_p), floor(y_p)), (floor(x_p), ceil(y_p)), (ceil(x_p), floor(y_p)), (ceil(x_p), ceil(y_p)).$$

   This can be done using interpolation, as described in the texture slides 43, 44.

Let's present this interpolation problem in a more general way than just texture mapping. For simplicity of notation, consider a unit square $(\alpha, \beta) \in (0,1) \times (0,1)$. Suppose $I(\alpha, \beta)$ is defined at the four corners of this unit square, i.e. we are given $I(0,0)$, $I(0,1)$, $I(1,0)$, $I(1,1)$. How would you interpolate to get values at any (α, β) within the unit square?

(a) One way to do it is to draw a diagonal across the square - say from upper left to lower right, and decide if the point is above the diagonal or below. If (α, β) is above the diagonal, then it would use the triangle defined above the diagonal and linearly interpolate across the triangle. Similarly, if the point is below the diagonal, it could use the lower triangle.

Can you think of any problems with this method? Specifically, what happens as the point (α, β) crosses the diagonal ?

(b) Another way to interpolate (α, β) in a square, given the values at the corners of a square is called *bilinear interpolation*. This method was sketched out in the lecture. Give a formula for this method, which gives you $I(\alpha, \beta)$ as a function of the values of $I(\ )$ at the corners of the unit square.

6. Where in the OpenGL pipeline are homographies for texture mapping computed, and how are they used?

7. [**ADDED APRIL 26**]

To compute hidden surface removal using the depth buffer method, the depth z of each fragment must be computed. Where and how is the depth of a fragment computed?

Solutions

1. It is just a scaling.

$$\begin{bmatrix} s \\ t \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{N_s} & 0 & 0 \\ 0 & \frac{1}{N_t} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_p \\ t_p \\ 1 \end{bmatrix}$$

2. (a) Since $(s,t) = (0,0)$ maps to $\mathbf{p}_1$, we have $(x_0, y_0, z_0) = \mathbf{p}_1 = (-5, 2, -7)$.

$$\mathbf{p}_2 - \mathbf{p}_1 = (-6, 4, -8) - (-5, 2, -7) = (-1, 2, -1).$$

Similarly,

$$\mathbf{p}_3 - \mathbf{p}_1 = (-7, 5, -3) - (-5, 2, -7) = (-2, 3, 4).$$

Thus, the transform from $(s,t,1)$ to positions $(x,y,z,1)$ is

$$\begin{bmatrix} -1 & -2 & -5 \\ 2 & 3 & 2 \\ -1 & 4 & -7 \\ 0 & 0 & 1 \end{bmatrix}$$

We project the triangle onto the $z = -2$ plane, via perspective projection, and so the projection matrix is:

$$\begin{bmatrix} -2 & 0 & 0 & 0 \\ 0 & -2 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Multiplying these two matrices (3×4 and 4×3)

$$\begin{bmatrix} -2 & 0 & 0 & 0 \\ 0 & -2 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} -1 & -2 & -5 \\ 2 & 3 & 2 \\ -1 & 4 & -7 \\ 0 & 0 & 1 \end{bmatrix}$$

would give the homography (3×3).

(b) If we were to consider *orthographic projection* in the z direction, rather than perspective projection, then the projection matrix we use would be

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

which amounts simply to dropping the z value. So, the homography would be:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -1 & -2 & -5 \\ 2 & 3 & 2 \\ -1 & 4 & -7 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -1 & -2 & -5 \\ 2 & 3 & 2 \\ 0 & 0 & 1 \end{bmatrix}.$$

(c) Your first thought might be to apply a sequence of translation, rotation, and scaling to bring the (s_i, t_i) vertices to $(0,0), (0,1), (0,2)$. However, it is not always possible to do that.

A more general approach is to compute a mapping from triangle (0,0), (1,0), (0,1) to $(s_i, t_i), i = 1, 2, 3$.

$$\begin{bmatrix} s' \\ t' \\ 1 \end{bmatrix} = \begin{bmatrix} s_2 - s_1 & s_3 - s_1 & s_1 \\ t_2 - t_1 & t_3 - t_1 & t_1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s \\ t \\ 1 \end{bmatrix}.$$

Note that the upper left 2×2 is more general than a scaling and rotation.

We want the inverse of this mapping:

$$\begin{bmatrix} s \\ t \\ 1 \end{bmatrix}. = \begin{bmatrix} s_2 - s_1 & s_3 - s_1 & s_1 \\ t_2 - t_1 & t_3 - t_1 & t_1 \\ 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} s' \\ t' \\ 1 \end{bmatrix}$$

which will map the three texture coordinates given in this question (c) to the canonical texture coordinate triangle in (a). We then multiply this mapping by the homography in (a) and we're done.

Note that this solution uses the idea of barycentric coordinates, that is, mapping from a canonical triangle to a more general triangle. See lecture 11 page 2.

3. The homography (3×3 matrix) is invertible, as long as that the camera position does not lie in the plane of the polygon. In that (failure) case, the 3D plane containing the polygon will be projected to a line.

4. Points on the horizon are image points (x, y) which (inverse) map to points in the texture space at infinity:

$$\begin{bmatrix} s \\ t \\ 0 \end{bmatrix} \equiv \mathbf{H}^{-1} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Which points (x, y) lie on the horizon? Going back to the lecture notes and taking the third row of $\mathbf{H}^{-1}$, we get

$$(0, \frac{\tan\theta}{z_0\ f}, \frac{1}{z_0}) \cdot (x, y, 1) = 0$$

and so

$$y = -\frac{f}{\tan\theta}.$$

where you recall that $f < 0$. This is the equation of the horizon in the image plane. Notice that if the surface not slanted at all ($\theta = 0$), then $\tan\theta = 0$ and so the horizon would be the set of points at infinity in the image plane i.e. $(x, y, 0)$.

5. (a) As (α, β) crosses the diagonal, there is a sudden switch in contribution from $I(0,0)$ to $I(1,1)$. This can give a discontinuity in the interpolated value.

   (b) *Bilinear interpolation* avoids the discontinuity of the previous question. As described in the slides, it works in two two steps. First, we interpolate along the lines $\alpha = 0$ and $\alpha = 1$ to get

$$I(0,\beta) = (1-\beta)I(0,0) + \beta I(0,1)$$

   and

$$I(1,\beta) = (1-\beta)I(1,0) + \beta I(1,1) \ .$$

   Then, we interpolate between $(0,\beta)$ and $(1,\beta)$:

$$\begin{aligned} I(\alpha,\beta) &= (1-\alpha)\ I(0,\beta) + \alpha\ I(1,\beta) \\ &= (1-\alpha)((1-\beta)\ I(0,0) + \beta\ I(0,1)) + \alpha((1-\beta)\ I(1,0) + \beta\ I(1,1)) \\ &= (1-\alpha)(1-\beta)\ I(0,0) + (1-\alpha)\beta\ I(0,1) + \alpha(1-\beta)\ I(1,0) + \alpha\beta I(1,1)) \end{aligned}$$

   The interpolation function is said to be *bilinear*, namely for fixed β it gives a linear interpolation in α, and for fixed α it gives a linear interpolation in β

   You can easily check that the interpolation function is of the form

$$I(\alpha,\beta) = A\alpha\beta + B\alpha + C\beta + D$$

   where A, B, C, D are sums of $I(0,0),\ I(1,0),\ I(0,1),\ I(1,1)$.

   Analogous to the quadric representation in lecture 3 p. 4), we have

$$I(\alpha,\beta) = \begin{bmatrix} \alpha & 1 \end{bmatrix} \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} \beta \\ 1 \end{bmatrix}.$$

6. A homography is used to compute the texture coordinates of interior points of a polygon. i.e. texture coordinates (s,t) are given for each vertex of a polygon but not for the interior points. This computation of the texture coordinates is done by the rasterizer.

   The rasterizer generates a fragment, one for each image pixel in the image projection of the polygon. Each fragment has image coordinates (x_p, y_p) and texture coordinates (s_p, t_p). The texture coordinates are computed by the inverse of the homography.

7. The rasterizer linearly interpolate the depths from the given depths of the vertices. Note that the rasterizer receives points in normalized device coordinates, so the depths have been normalized to [0,1].

   Minor point: linear interpolation of depth values does not give the correct depth values since "depth" in the normalized device coordinates is not true depth, but rather it is some non-linearly transformed depth (because of perspective, it is related to 1/z rather than z). Fortunately this doesn't create a problem for the depth buffer algorithm since the non-linear transform preserves depth ordering within the view volume. However, note that if you need the z value for other shading effects, then you should really consider the non-linearity.