COMS 4160: Problems on Transformations and OpenGL

Ravi Ramamoorthi

1. Write the homogeneous 4x4 matrices for the following transforms:
   - Translate by +5 units in the X direction
   - Rotate by 30 degrees about the X axis
   - The rotation, followed by the translation above, followed by scaling by a factor of 2.
2. In 3D, consider applying a rotation R followed by a translation T. Write the form of the combined transformation in homogeneous coordinates (i.e. supply a 4x4 matrix) in terms of the elements of R and T. Now, construct the inverse transformation, giving the corresponding 4x4 matrix in terms of R and T. You should simplify your answer (perhaps writing T as [Tx,Ty,Tz] and using appropriate notation for the 9 elements of the rotation matrix, or using appropriate matrix and vector notation for R and T). Verify by matrix multiplication that the inverse times the original transform does in fact give the identity.
3. Derive the homogeneous 4x4 matrices for gluLookAt and gluPerspective
4. Assume that in OpenGL, your near and far clipping planes are set at a distance of 1m and 100m respectively. Further, assume your z-buffer has 9 bits of depth resolution. This means that after the gluPerspective transformation, the remapped z values [ranging from -1 to +1] are quantized into 512 discrete depths.
   - How far apart are these discrete depth levels close to the near clipping plane? More concretely, what is the z range (i.e. 1m to ?) of the first discrete depth?
   - Now, consider the case where all the interesting geometry lies further than 10m. How far apart are the discrete depth levels at 10m? Compare your answer to the first part and explain the cause for this difference.
   - How many discrete depth levels describe the region between 10m and 100m? What is the number of bits required for this number of depth levels? How many bits of precision have been lost? What would you recommend doing to increase precision?
5. Consider the following operations in the OpenGL pipeline: *Scan conversion or Rasterization*, *Projection Matrix*, *Transformation of Points and Normals by the ModelView Matrix*, *Dehomogenization (perspective division)*, *clipping*, *Lighting calculations*. Briefly explan what each of these operations are, and in what order they are performed and why.

Answers

1. Homogeneous Matrices A general representation for 4x4 matrices involving rotation and translation is

$$\begin{pmatrix} R_{3\times 3} & T_{3\times 1} \\ 0_{1\times 3} & 1_{1\times 1}, \end{pmatrix} \tag{1}$$

where R is 3×3 rotation matrix, and T is a 3×1 translation matrix.

For a translation along the X axis by 3 units $T = (5, 0, 0)^t$, while R is the identity. Hence, we have

$$\begin{pmatrix} 1 & 0 & 0 & 5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{2}$$

In the second case, where we are rotating about the X axis, the translation matrix is just 0. We need to remember the formula for rotation about an axis, which is (with angle θ),

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \sqrt{3}/2 & -1/2 & 0 \\ 0 & 1/2 & \sqrt{3}/2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{3}$$

Finally, when we are combining these transformations, S*T*R, we apply the rotation first, followed by a translation. It is easy to verify by matrix multiplication, that this simply has the same form as equation 1 (but see the next problem for when we have R*T). The scale just multiplies everything by a factor of 2, giving

$$\begin{pmatrix} 2 & 0 & 0 & 10 \\ 0 & \sqrt{3} & -1 & 0 \\ 0 & 1 & \sqrt{3} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{4}$$

It is also possible to obtain this result by matrix multiplication of S*T*R

$$\begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \sqrt{3}/2 & -1/2 & 0 \\ 0 & 1/2 & \sqrt{3}/2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 0 & 0 & 10 \\ 0 & \sqrt{3} & -1 & 0 \\ 0 & 1 & \sqrt{3} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{5}$$

2. Rotations and translations Having a rotation followed by a translation is simply T*R, which has the same form as equation 1. The inverse transform is more interesting. Essentially $(TR)^{-1} = R^{-1}T^{-1} = R^t * -T$, which in homogeneous coordinates is

$$\begin{pmatrix} R^t_{3\times3} & 0_{3\times1} \\ 0_{1\times3} & 1 \end{pmatrix} \begin{pmatrix} I_{3\times3} & -T_{3\times1} \\ 0_{1\times3} & 1 \end{pmatrix} = \begin{pmatrix} R^t_{3\times3} & -R^t_{3\times3}T_{3\times1} \\ 0_{1\times3} & 1 \end{pmatrix}. \tag{6}$$

Note that this is the same form as equation 1, using R' and T' with $R' = R^t = R^{-1}$ and $T' = -R^tT$.

Finally, we may verify that the product of the inverse and the original does in fact give the identity.

$$\begin{pmatrix} R^t_{3\times3} & -R^t_{3\times3}T_{3\times1} \\ 0_{1\times3} & 1 \end{pmatrix} \begin{pmatrix} R_{3\times3} & T_{3\times1} \\ 0_{1\times3} & 1 \end{pmatrix} = \begin{pmatrix} R^t_{3\times3}R_{3\times3} & R^t_{3\times3}T_{3\times1} - R^t_{3\times3}T_{3\times1} \\ 0_{1\times3} & 1 \end{pmatrix} = \begin{pmatrix} I_{3\times3} & 0_{3\times1} \\ 0_{1\times3} & 1 \end{pmatrix} \tag{7}$$

3. gluLookAt and gluPerspective I wrote this answer earlier to conform in notation to the Unix Man Pages. Some of you might find it easier to just understand this from the lecture slides in the transformation lectures than this derivation and may want to skip over this section if you already understand the concepts.

gluLookat defines the viewing transformation and is given by *gluLookAt(eyex, eyey, eyez, centerx, centery, centerz, upx, upy, upz)*, corresponds to a camera at *eye* looking at *center* with up direction *up*. First, we define the normalized viewing direction. The symbols used here are chosen to correspond to the definitions in the man page.

$$F = \begin{pmatrix} C_x - E_x \\ C_y - E_y \\ C_z - E_z \end{pmatrix} \qquad \mathbf{f} = F / \parallel F \parallel . \tag{8}$$

This direction $\mathbf{f}$ will correspond to the $-Z$ direction, since the eye is mapped to the origin, and the lookat point or center to the negative z axis. What remains now is to define the X and Y directions. The Y direction corresponds to the up vector. First, we define $UP' = UP/ \parallel UP \parallel$ to normalize. However, this may not be perpendicular to the Z axis, so we use vector cross products to define $X = -Z \times Y$ and $Y = X \times -Z$. In our notation, this defines auxiliary vectors,

$$\mathbf{s} = \frac{\mathbf{f} \times UP'}{\parallel \mathbf{f} \times UP' \parallel} \qquad \mathbf{u} = \frac{\mathbf{s} \times \mathbf{f}}{\parallel \mathbf{s} \times \mathbf{f} \parallel}. \tag{9}$$

Note that this requires the UP vector not to be parallel to the view direction. We now have a set of directions $\mathbf{s}, \mathbf{u}, -\mathbf{f}$ corresponding to X, Y, Z axes. We can therefore define a rotation matrix,

$$M = \begin{pmatrix} \mathbf{s}_x & \mathbf{s}_y & \mathbf{s}_z & 0 \\ \mathbf{u}_x & \mathbf{u}_y & \mathbf{u}_z & 0 \\ -\mathbf{f}_x & -\mathbf{f}_y & -\mathbf{f}_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \tag{10}$$

that rotates a point to the new coordinate frame.

However, gluLookAt requires applying this rotation matrix about the eye position, *not* the origin. It is equivalent to *glMultMatrixf(M) ; glTranslateD(-eyex, -eyey, -eyez) ;* This corresponds to a translation T followed by a rotation R. We know (using equation 6 as a guideline for instance), that this is the same as the rotation R followed by a modified translation $R_{3\times3}T_{3\times1}$. Written out in full, the matrix will then be

$$G = \begin{pmatrix} \mathbf{s}_x & \mathbf{s}_y & \mathbf{s}_z & -\mathbf{s}_x\mathbf{e}_x - \mathbf{s}_y\mathbf{e}_y - \mathbf{s}_z\mathbf{e}_z \\ \mathbf{u}_x & \mathbf{u}_y & \mathbf{u}_z & -\mathbf{u}_x\mathbf{e}_x - \mathbf{u}_y\mathbf{e}_y - \mathbf{u}_z\mathbf{e}_z \\ -\mathbf{f}_x & -\mathbf{f}_y & -\mathbf{f}_z & \mathbf{f}_x\mathbf{e}_x + \mathbf{f}_y\mathbf{e}_y + \mathbf{f}_z\mathbf{e}_z \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{11}$$

gluPerspective defines a perspective transformation used to map 3D objects to the 2D screen and is defined by *gluPerspective(fovy, aspect, zNear, zFar)* where *fovy* specifies the field of view angle, in degrees, in the y direction, and *aspect* specifies the aspect ratio that determines the field of view in the x direction. The aspect ratio is the ratio of x (width) to y (height). *zNear* and *zFar* represent the distance from the viewer to the near and far clipping planes, and must always be positive.

First, we define $f = \cot(fovy/2)$ as corresponding to the focal length or focal distance. A 1 unit height in Y at $Z = f$ should correspond to $y = 1$. This means we must multiply Y by f and corresponding X by $f/aspect$. The matrix has the form

$$M = \begin{pmatrix} \frac{f}{aspect} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & A & B \\ 0 & 0 & -1 & 0 \end{pmatrix}. \tag{12}$$

We will explain the form of the above matrix. The form of terms $f/aspect$ and f has already been explained. The term -1 in the last line is needed to divide by the distance Z as required in perspective, and the negative sign is because OpenGL conventions require us to look down the $-Z$ axis.

It remains to find A and B. Those are chosen so the near and far clipping planes are taken to -1 and $+1$ respectively. Indeed, the entire viewing volume or frustum is mapped to a cube between -1 and $+1$ along all axes. Using the matrix, we can easily formulate that the remapped depth is given by

$$z' = \frac{Az+B}{-z} = -A - \frac{B}{z}, \tag{13}$$

where one must remember that points in front of the viewer have negative z as per OpenGL conventions. Now, the required conditions $z = -zNear \Rightarrow z' = -1$ and $z = -zFar \Rightarrow z' = 1$ have,

$$-A + \frac{B}{zNear} = -1 \qquad -A + \frac{B}{zFar} = +1 \tag{14}$$

Solving this system gives

$$A = \frac{zFar + zNear}{zNear - zFar} \qquad B = \frac{2 \cdot zFar \cdot zNear}{zNear - zFar}, \tag{15}$$

and the final matrix

$$G = \begin{pmatrix} \frac{f}{aspect} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{zFar+zNear}{zNear-zFar} & \frac{2 \cdot zFar \cdot zNear}{zNear-zFar} \\ 0 & 0 & -1 & 0 \end{pmatrix}. \tag{16}$$

Z-buffer in OpenGL The purpose of this question is to show how depth resolution degrades as one moves further away from the near clipping plane, since the remapped depth is nonlinear (and reciprocal) in the original depth values.

Equation 13 gives us the formula for remapping z. We just need to find A and B, which we can do by solving, or plugging directly into equation 15 using $zNear = 1$ and $zFar = 100$. We obtain $A = -101/99 \approx -1.02$ and $B = -200/99 \approx -2.02$. The remapped value is then given by

$$z' = 1.02 - \frac{2.02}{\mid z \mid}. \tag{17}$$

Note that for mathematical simplicity, you might imagine the far plane at infinity, so we don't need the .02.

For the remaining parts of the question, it is probably simplest to just use differential techniques. We can obtain

$$dz' \approx \frac{2}{\mid z \mid^2} d \mid z \mid \;\Rightarrow\; \mid dz \mid = \frac{\mid z \mid^2}{2} \mid dz' \mid \tag{18}$$

To consider one depth bucket, we simply need to set $\mid dz' \mid = 1/256 \approx 0.004$. Now, using the equation above, setting $\mid z \mid = 1$, we get $\mid dz \mid \approx 0.002$. In other words, the first depth bucket ranges from a depth of $1m$ to a depth of approximately $1.0019m$, and we can resolve depths $2mm$ apart.

Now, consider $\mid z \mid = 10$, and plug in above. We know that $\mid dz \mid \sim \mid z \mid^2$, so $\mid dz \mid \approx 0.2$, and the depth buckets around $z = 10m$ are in $20cm$ increments and we lose resolving power quadratically, with a danger that many different objects may go into the same depth bucket. This brings us to the fundamental point of this problem that depth levels are closer near the near plane and depth resolution decreases far away.

Finally, we consider the depth levels between $10m$ and $100m$. Using equation 17, $10m$ transforms to $1.02 - 2.02/10 \approx 0.82$. Thus, only a range of 0.18 remains. Hence, we only have $0.18 * 256 = 46$ depth buckets, or less than 6 bits of precision. We have lost more than 3 bits of precision. To increase precision, we should move the near clipping plane further out if interesting geometry is in the $10m - 100m$ range.

Order of OpenGL operations The operations are performed in the following order. You might wish to peruse Appendix A of the OpenGL manual.

1. *Modelview Matrix:* Each vertex's spatial coordinates are transformed by the modelview matrix M as $\mathbf{x}' = M\mathbf{x}$. Simultaneously, normals are transformed by the inverse transpose, $\mathbf{n}' = M^{-t}\mathbf{n}$ and renormalized if specified.

2. *Lighting Calculations:* The lighting calculations are performed next per vertex. They operate on the transformed coordinates, so must occur after the modelview transform. They are performed before projection, because the lighting isn't affected by the projection matrix. Remember that lighting is therefore performed in eye coordinates. Lighting *must* be performed *before* clipping, because vertices that are clipped could influence shading inside the region of interest (such as a triangle, one of whose vertices is clipped). A user-defined culling algorithm can help avoid unnecessary computations.

3. *Projection Matrix:* The projection matrix is then applied to project objects into the image plane.

4. *Clipping:* Clipping is then done in the homogeneous coordinates against the standard viewing planes $x = \pm w, y = \pm w, z = \pm w$. Clipping is done before dehomogenization to avoid the perspective divide for vertices that are clipped anyway.

5. *Dehomogenization or Perspective divide:* Perspective divide by the fourth or homogeneous coordinate w then occurs.

6. *Scan conversion or Rasterization:* Finally, the primitive is converted to fragments by scan conversion or rasterization.