Assignment 2: Camera

Comp175: Introduction to Computer Graphics – Spring 2016

Algorithm due: **Wednesday Feb 24th** at 11:59pm
Project due: **Wednesday March 2nd** at 11:59pm

```
Your Names: ________Matt Asnes___________

            ________Fury Sheron__________

Your CS Logins: ______masnes01___________

                ______lshero01___________
```

1 Instructions

Complete this assignment only with your teammate. You may use a calculator or computer algebra system. All your answers should be given in simplest form. When a numerical answer is required, provide a reduced fraction (i.e. 1/3) or at least three decimal places (i.e. 0.333). Show all work; write your answers on this sheet. This algorithm handout is worth 3% of your final grade for the class.

2 Axis-Aligned Rotation

Even though you won't have to implement Algebra.h, you will need to know (in principle) how to do these operations. For example, you should know how to create a rotation matrix...
`Matrix rotX_mat(const double radians)`
`Matrix rotY_mat(const double radians)`
`Matrix rotZ_mat(const double radians)`

Each of these functions returns the rotation matrix about the x, y, or z axis (respectively) by the specified angle.

[**1 point**] Using a sample data point (e.g., (1, 1, 1)), find out what happens to this point when it is rotated by 45 *degrees* using each of these matrices.

Rotation about x via `Matrix rotX_mat`:

$$\begin{bmatrix} \cos(45°) & -\sin(45°) & 0 & 0 \\ \sin(45°) & \cos(45°) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

Rotation about y via `Matrix rotY_mat`:

$$\begin{bmatrix} \cos(45°) & 0 & \sin(45°) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(45°) & 0 & \cos(45°) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

Rotation about z via `Matrix rotZ_mat`:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(45°) & -\sin(45°) & 0 \\ 0 & \sin(45°) & \cos(45°) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

3 Arbitrary Rotation

Performing a rotation about an arbitrary axis can be seen as a composition of rotations around the bases. If you can rotate all of space around the bases until you align the arbitrary axis with the x axis, then you can treat the rotation about the arbitrary axis as a rotation around x. Once you have transformed space with that rotation around x, then you need to rotate space back so that the arbitrary axis is in its original orientation. We can do this because rotation is a rigid-body transformation.

To rotate a point p, this series of rotations looks like:

$$p' = M_1^{-1} \cdot M_2^{-1} \cdot M_3 \cdot M_2 \cdot M_1 \cdot p$$

where M_1 rotates the arbitrary axis about the y axis, M_2 rotates the arbitrary axis to lie on the x axis, and M_3 rotates the desired amount about the x axis.

3.1 Rotation about the origin

[1.5 points] Assuming we want to rotate $p = (p_x, p_y, p_z, 1)$ about a vector $a = \langle a_x, a_y, a_z, 0\rangle$ by λ radians, how would you calculate M_1, M_2, and M_3 in terms of the functions `rotX_mat`, `rotY_mat`, and `rotZ_mat`?

Hint: This is tough! You can (and should) compute angles to use along the way. You can accomplish arbitrary rotation with `arctan` *and/or* `arcos`*, for instance. Approach this problem step by step, it's not extremely math heavy. You should need only one* `sqrt` *call.*

$$M_1 = \texttt{rotY\_mat}\Big(-\tan^{-1}\Big(\frac{a_z}{a_x}\Big)\Big)$$

$$M_2 = \texttt{rotZ\_mat}\Big(-\tan^{-1}\Big(\frac{a_y}{\sqrt{a_x^2 + a_z^2}}\Big)\Big)$$

$$M_3 = \texttt{rotX\_mat}(\lambda)$$

$$M_2^{-1} = \texttt{rotZ\_mat}\Big(\tan^{-1}\Big(\frac{a_y}{\sqrt{a_x^2 + a_z^2}}\Big)\Big)$$

$$M_1^{-1} = \texttt{rotY\_mat}(\tan^{-1}\Big(\frac{a_z}{a_x}\Big))$$

3.2 Rotation about an arbitrary point

[1 point] The equation you just derived rotates a point p about the origin. How can you make this operation rotate about an arbitrary point h?

We can translate the world such that h is the origin, then do our rotation, and translate back at the end.

4 Camera Transformation

To transform a point p from world space to screen space we use the *normalizing transformation.* The normalizing transformation is composed of four matrices[1], as shown here:

$$p' = M_1 \cdot M_2 \cdot M_3 \cdot M_4 \cdot p$$

Each M_i corresponds to a step described in lecture. Here, p is a point in world space, and we would like to construct a point p' relative to the camera's coordinate system, so that p' is its resulting position on the screen (with its z coordinate holding the depth buffer information). You can assume that the camera is positioned at $(x, y, z, 1)$, it has look vector *look* and up vector *up*, height angle θ_h, width angle θ_w, near plane *near* and far plane *far*.

[1/2 pt. each] Briefly write out what each matrix is responsible for doing. Then write what values they have. Make sure to get the order correct (that is, matrix M_4 corresponds to only one of the steps described in lecture.)

M_4 is a translation from the camera position $(x, y, z, 1)$ to the origin:

$$M_4 = \begin{bmatrix} 1 & 0 & 0 & -x \\ 0 & 1 & 0 & -y \\ 0 & 0 & 1 & -z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

M_3 is a rotation of the camera's coordinate system to the world's. As in the slides, we can define (u, v, w) with $\vec{w} = \frac{-\vec{look}}{\|\vec{look}\|}$, $\vec{u} = \frac{\vec{up}\times\vec{w}}{\|\vec{up}\times\vec{w}\|}$, $\vec{v} = \vec{w} \times \vec{u}$, so that:

$$M_3 = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ w_x & w_y & w_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

[1]Technically, there are five matrices. The last step is the 2D viewport transform, which is omitted here because OpenGL takes care of this step for us.

M_2 is a scaling matrix,

$$M_2 = \begin{bmatrix} \frac{1}{\tan\left(\frac{\theta_w}{2}\right) far} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan\left(\frac{\theta_h}{2}\right) far} & 0 & 0 \\ 0 & 0 & \frac{1}{far} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

M_1 is an unhinging transform to transform from a perspective into a plane; we define $c = \frac{-near}{far}$ and assign:

$$M_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{c+1} & \frac{-c}{c+1} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

For the assignment you need to perform rotation operations on the camera in its own virtual uvw coordinate system, e.g. spinning the camera about its v-axis. Additionally, you will need to perform translation operations on the camera in world space.

[1/2 pt. each] How (mathematically) will you translate the camera's eye point E:

1. One unit right?

$$E' = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} E$$

2. One unit down?

$$E' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} E$$

3. One unit forward?

$$E' = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix} E$$

In the space (u, v, w) the rows of a vector or point correspond to u, v and w in that order, so to shift one unit right we add 1 to the u component of a vector; similarly, since v points up, we subtract 1 to move down, and since w points away from the look vector we subtract one to move further in the direction we are looking.

You can either move in and out of the camera coordinate space to perform these transformations, or you can do arbitrary rotations in world space. Both answers are acceptable.

[1/2 pt. each] How (mathematically) will you use the u, v, and w vectors, in conjunction with a rotation angle θ, to get new u, v, and w vectors when:

1. Adjusting the "spin" in a clockwise direction by θ radians?

$$\vec{w'} = -\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \vec{w}$$

2. Adjusting (rotating) the "pitch" to face upwards by θ radians?

$$\vec{u'} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \vec{u}$$

3. Adjusting the "yaw" to face right by θ radians?

$$\vec{v'} = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \vec{v}$$

5 Inverting Transformations

[1 point] Computing a full matrix inverse isn't always the most efficient way of inverting a transformation if you know all the components. Write out fully (i.e., write out all the elements) a 4x4 translation matrix M_T and its inverse, M_T^{-1}. Our claim should become obvious to you.

If we want to translate something x_0 units in the x direction, y_0 units in the y direction, and z_0 units in the z direction,

$$M_T = \begin{bmatrix} 1 & 0 & 0 & x_0 \\ 0 & 1 & 0 & y_0 \\ 0 & 0 & 1 & z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Then the inverse translation, and also the inverse matrix, is:

$$M_T^{-1} = \begin{bmatrix} 1 & 0 & 0 & -x_0 \\ 0 & 1 & 0 & -y_0 \\ 0 & 0 & 1 & -z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

We can check that $M_T M_T^{-1} = I$ with quick mental computation. This is clearly an easy transformation to invert, while the actual process of inverting a 4×4 matrix involves a lot more work.

Extra credit: Will this same technique work for scale and rotation matrices? If so, write out the inverse scale and inverse rotation matrices. If not, explain why not. What about the perspective "unhinging" transformation? Will this technique be as efficient? Explain.

For the rotation in perspective/parallel camera spaces, we can convert by swapping our notion of u, v, w and x, y, z, i.e.:

$$M_{rotate}^{-1} = \begin{bmatrix} x_u & x_v & x_w & 0 \\ y_u & y_v & y_w & 0 \\ z_u & z_v & z_w & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

If we just want to do an arbitrary rotation inverse, if the rotation was by θ degrees we could just rotate by $-\theta$; this reverses the sign of the *sin* functions but not *cos* in the rotation matrices.

For the scale to scale camera world space, if we know θ_w and θ_h and far, then we can change the main column to be the multiplicative inverses of the regular scale matrix's elements to undo the multiplication (scale) that they do:

$$M_{scale}^{-1} = \begin{bmatrix} \tan\left(\frac{\theta_w}{2}\right) far & 0 & 0 & 0 \\ 0 & \tan\left(\frac{\theta_h}{2}\right) far & 0 & 0 \\ 0 & 0 & far & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

For an arbitrary scale we do the same thing; the inverse of a scale with main diagonal $S_x, S_y, S_z, 1$ will be the matrix with main diagonal $\frac{1}{S_x}, \frac{1}{S_y}, \frac{1}{S_z}, 1$

For the perspective unhinging transform the inverse should be fairly easy, since we have so many zero elements that essentially only a 2×2 submatrix that matters for the inverse; this gives us:

$$M_{unhinge}^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & -\frac{1+c}{c} & \frac{1}{c} \end{bmatrix}$$

Most of these shortcuts will be computationally preferable to a full 4x4 matrix inversion, especially those with a high number of nonzero elements.

6 How to Submit

Hand in a PDF version of your solutions using the following command:

```
provide comp175 a2-alg
```