Assignment 2: Camera

Comp175: Introduction to Computer Graphics – Spring 2014

Algorithm due: **Monday February 17th** at 11:59pm
Project due: **Monday February 24th** at 11:59pm


```
Your Names: Marcella Hastings
            Louis Rassaby

Your CS Logins: mhasti01
                lrassa01
```


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 implment Algebra.h, you will need to know (in principle) how to do these operatoins. 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 of the vector $(1, 1, 1)$ 45° around the x-axis produces the following matrix:

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

Rotation around the y-axis produces

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

Finally, rotating around the z-axis creates

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

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.*

$$M_1 = \texttt{rotY\_mat}(\theta)$$

$$\text{where } \theta = \begin{cases} \arctan \frac{a_z}{a_x} & a_x > 0 \\ \pi + \arctan \frac{a_z}{a_x} & a_x < 0 \\ \frac{\pi}{2} & a_x = 0 \end{cases}$$

$$M_2 = \texttt{rotZ\_mat}(\phi)$$

$$\text{where } \phi = \begin{cases} \arctan \frac{a_y}{a_x} & a_x > 0 \\ \pi + \arctan \frac{a_y}{a_x} & a_x < 0 \\ \frac{\pi}{2} & a_x = 0 \end{cases}$$

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

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?

Translate the world using the vector from h to the origin, do the rotation, translate the world back to its original position. If T is the translation matrix to move h to the origin, this is the transformation $T^{-1} \cdot R \cdot T \cdot p$.

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 five matrices, 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_1 : This is the perspective unhinging matrix, which adjusts the viewing window. For $c = -\frac{near}{far}$,

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

M_2 : This is the scaling matrix, which scales to the viewing frustrum (the area that is being projected onto). It does this by scaling the back clipping plane to $z = -1$ and scaling the corners to fit the view model.

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

.

M_3 : This is the rotation matrix that moves points from the uvw coordinate system to the xyz system.

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

where $\langle u_x, u_y, u_z, 0\rangle$, etc. form the basis vectors of the uvw coordinate system.

M_4 : This is the translation matrix, which moves the eye point of the camera to the origin.

$$M_4 = \begin{bmatrix} 1 & 0 & 0 & -e_x \\ 0 & 1 & 0 & -e_y \\ 0 & 0 & 1 & -e_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where $E = \langle e_x, e_y, e_z, 1\rangle$ is the initial position of the eye piece of the camera.

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? $E' = \langle e_u + 1, e_v, e_w\rangle$
2. One unit down? $E' = \langle e_u, e_v - 1, e_w\rangle$
3. One unit forward? $E' = \langle e_u, e_v, e_w - 1\rangle$

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:

Note: In the following answers, the new u, v, and w vectors are written in matrix form: $[uvw]$.

1. Adjusting the "spin" in a clockwise direction by θ radians?
   Transform out of camera coordinate space.
   The transformation matrix for the spin move is

   $$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

   Invert the first transformation.

2. Adjusting (rotating) the "pitch" to face upwards by θ radians?
   Transform out of the camera coordinate space.

   Apply the following pitch transformation matrix:

   $$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

   Invert the first transformation.

3. Adjusting the "yaw" to face right by θ radians?
   Transform out of the camera coordinate space.

   Use the 'yaw' transformation matrix.

   $$\begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

   Invert the first transformation.

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.

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

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

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.

Obviously, the *exact* technique will not translate. However, it is true that there is a closed-form inverse for several specific types of 4x4 matrix. For scaling:

$$M_T = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M_T^{-1} = \begin{bmatrix} 1/s_x & 0 & 0 & 0 \\ 0 & 1/s_y & 0 & 0 \\ 0 & 0 & 1/s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The inverse rotation matrix varies based on the axis of rotation.

Around x:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Around y:

$$\begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Around z:

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The perspective unhinging transformation necessarily also has a closed-form inverse by virtue of being a 4x4 matrix.

In any case, this technique will not usually be efficient because it generally makes sense to combine transformations. It would be more expensive to invert and recombine separate matrices in their own operations than to just perform the inverse calculation for a combined transformation matrix.

6 How to Submit

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

```
provide comp175 a2-alg
```