

Homework 3

Assigned Sunday, September 20, 2009

Due Tuesday, September 29, 2009 at 11:59pm

In this homework, you will get to experiment with the 2D and 3D transformations we have been talking about in class.

1. Experimentation with 2D Transformations (30 pts)

In this portion of the assignment, you will write code that will allow you to apply a series of 2D transformations to the 2D model of a house. First, you will begin by writing a `Matrix3x3` class that allows you to express rotations, translations, scales and shears in 2D as we discussed in class. Specifically you must:

1. Create a "*" operator that allows you to multiply the matrix by a vector of class `Vector2D` and returns a new vector
2. Create a "*" operator that allows you to multiply the matrix by another matrix and return a new matrix
3. Create an "=" operator that allows you to set the matrix to a set of new values
4. Create an identity method that sets the matrix to the identity matrix
5. Create methods to specify various transformations using the matrix. In particular, you will have to implement the matrices for translations, rotations, scaling, and shears.
6. For debugging, add a `print()` method that prints out the values of the matrix

Once this has been done, you can verify that your matrix class works by creating matrices to transform the vertices of the house. To do this, multiply the vertices by the transformation matrix in the draw loop and the house should appear transformed in the result.

Now implement a modelview matrix similar to that of OpenGL by making a global matrix called `MV` (modelview). Set up your draw loop so that the vertices are always multiplied by the `MV` before being drawn. We want to have functions similar to `glTranslate()`, so fill in the code for the following functions: `myTranslate()`, `myRotate()`, `myRotateAboutPoint()`, `myScale()`, `myShearX()` and `myShearY()`, in order to apply the transformation directly to the vertex. When you call one of these functions, you are going to be right-multiplying the `MV` matrix by the new transformation. The result is that the house model will be transformed by the matrices in the reverse order that they are issued, very much as is done in OpenGL. Note that the user should also be able to directly multiply matrices together and set them to the modelview matrix like this: $MV = R1 * T1 * R2$.

Once you confirm this all works, demonstrate that everything works by creating a new draw function called `drawNeighborhood()` that draws an array of 10 x 10 houses all appropriately scaled and uniformly separated. This is an example of instancing, where you keep the vertices the same but modify the modelview matrix accordingly to draw

each instance of the house. Finally, add a little pizzazz to your program by adding some animation: add a rotation to each house in the neighborhood so that they all rotate clockwise about their centers in unison upon each draw loop. Upon each drawScene() call, the houses will rotate a small angle θ from their previous position.

To get you started use the files located in the 2DTransforms.zip. After you are done, put all the files I need to compile your code (including the project solution) into a directory called 2DTransforms.

2. Experimentation with 3D Transformations (30 pts)

In this portion of the homework, you will write a viewer for a 3D model that takes in the mouse input and then allows you to rotate about an object in 3D. This kind of viewer is a helpful interface for a modeling program because it allows the artists to rotate the model they are working and look at it from every angle.

To do this implementation, think of the model as being at the origin of the world space and that the viewer's eye is moving around it in a sphere of a specific radius. The sphere is centered about a point in space (let's call it `eye_center`) that the user can translate. The position of the eye on the sphere can be represented by the angles θ and ϕ . The vector represented from the eye position to `eye_center` is the view direction.

We will be using glut to handle the mouse inputs. To write your viewer, you will have to implement the following functions:

`dragView()`: When the left button is pressed you should rotate about the `eye_center` by updating the θ and ϕ angles.

`doZoom()`: When the middle mouse button is held down, the user should be able to move into the scene. Do this by modifying the radius of the eye sphere.

`doTranslate()`: When the right mouse button is held down, the user can translate the center of the view by translating the `eye_center` position. This will let the user look at different parts of the model.

You will also have to write the respective `begin*()` functions that are called when the mouse button is first pressed down. To get you started, use the project included in the ModelViewer.zip file. In it, I am providing you with a file called `mouse.cpp` that already has the appropriate glut callbacks for handling the mouse inputs. All you have to do is fill in the functions and set up the appropriate projection and modelview matrices. I also included my own version of ModelViewer.exe with `printf()` debugging enabled so that you can see what you should be getting for every value of `eye_center`, θ , ϕ , and `eye_distance` (the radius of the sphere). The behavior of your code should be as close as possible to my own.

Extra Credit: Right now the viewing program simply draws a boring quadrangle. Find a more interesting model on the internet (make sure that it is copyright free or draw it yourself) and modify the draw loop so that it is rendered [10 pts].

3. VR Viewer (40 pts)

In this portion of the homework, you will be implementing a simple VR viewing program that allows us to view cube environment map in an interactive way. You may have used similar applications such as Quicktime VR, which are very popular on the web today. To accomplish this, you will use OpenGL to do the rendering.

To understand what you are supposed to do, first check out the VRviewer application available in the VRviewer.zip file from the website. This includes the VRviewer.exe, the glut DLL and a directory of sample environment map images, courtesy of Paul Debevec (<http://www.debevec.org/Probes/>). You can run the sample code in the following way:

```
VRviewer.exe Uffizi/uffizi
```

You will see a view of the Uffizi gallery in Florence. Drag the mouse to move the viewpoint around the environment. The purpose of the program is to give the user a “virtual reality” experience in a new environment.

Your task for this part of the homework is to write this program, and you will need to do this from scratch. However, to make your task easier, you are welcome to use some of the image classes that I have given you in the past, and you can re-use the mouse control you wrote for the 2nd part of this homework. The program is actually very simple to write. You will need to render a texture mapped cube with the viewer in the center. The view angle should be controlled by moving the mouse. For best results, set projection matrix to perspective with a 45-degree field-of-view. The walls should be texture-mapped with the appropriate environment map images provided in the directory. These environment maps are labeled according to the direction they belong, i.e. +x, -x, +y, etc. Make sure you orient the images correctly so that the seams are not visible in your program!

I will expect your application to have the same functionality as my own so make sure that it works the same way. This program should be very simple to write if you are familiar with OpenGL (it took me about ten minutes), but the first time it might take you longer.

Extra Credit: For extra credit, photograph your own environment map around campus and set it up to run on your application. Make sure you line the images properly to minimize the seams. [20pts]

Submission Instructions

As in the last assignment, you are to write your code for these three problems in C or C++ so that it compiles and runs in the Visual Studio environment available on the PCs in the computer lab ECE 211. Put the first part into a directory called 2DTransforms, the second part into a directory called ModelViewer, and the last part into a directory called VRViewer. Zip up these three directories **into a single zip file with your name** and submit them through WebCT before the deadline at 11:59pm on **Tuesday, September 29, 2009**. This is a hard deadline. Read the syllabus regarding late submissions.

Tips and advice

This assignment should be shorter than the last one, but nevertheless get started soon and don't wait until the last minute. Post your questions on the forum early. You can also ask me (and the TA) questions if you have them. Keep an eye on the forum for any announcements regarding this homework. Good luck!