

CS334: Assignment #1 – Linear Algebra

Out: Sept 4, 2008

Back/Due: Sept 18, 2008

Objective:

This objective of this assignment is to implement a 3D vector class, a 3D point class, a 4x4 matrix class, and operations on these data elements. This assignment will be implemented on top of the previous assignment and will be demonstrated by implementing a virtual trackball navigation tool for simple wireframe object rendering. The classes you create will provide a good basis from which all your future graphics programs will be built (so design them well!).

Specifics:

1. 3D vector/point class (remember, they are not the same...)
 - Stores 3 floats. You may want to instead template this class to accept any data type. This may be useful to you in later projects for simplifying code that involves 3D array bounds (int) or when needing more floating point precision (double). You may also want to implement the storage as an array so that you can take advantage of OpenGL function calls like `glVertex3fv`, which is more efficient than just `glVertex3f`.
 - Constructor from 3 floats.
 - Read/write access to elements with square brackets operator.
 - Normalization (for vector)
 - Length (for vector)
 - Dot product (for vector)
 - Cross product (for vector)
 - Multiplication/division with scalar (use operator).
 - Addition/subtraction with another vector/point (use operators).
2. 4x4 matrix class
 - Stores 16 floats
 - Constructor from 16 floats.
 - Read/write access to rows and columns.
 - Matrix transposition.
 - Matrix times column vector.
 - Matrix times column point.
 - Matrix multiplication.
 - Set up as a
 - x,y,z rotation matrix
 - translation matrix
 - uniform scale matrix
 - identity (should be default)
3. Virtual Trackball

- Support rendering of simple OpenGL/GLUT primitives (e.g., box, cylinder) and control of the viewing position using a virtual trackball interface.
- You may assume the object is initially located at the origin and the camera is at a reasonable distance and looking down the negative-z axis.

Mouse/Button Control:

To control the mouse and button events, you can use GLUT's callback mechanism to register callbacks for when the mouse is moved and/or a button is pressed. This is how you can implement the virtual trackball. The code fragment from the previous assignment has such callbacks setup for refresh. You need to add for mouse moved and for mouse pressed.

Testing:

Show an “object” in the middle of the screen. Use the GLUT utilities to render the object in wireframe (i.e., use `glutWireSphere`, `glutWireTeapot`). Use GLUT to enable choosing from 1 of 2 object primitives (sphere or teapot). Using the left mouse-button, enable rotating the object about its center. Using the right mouse-button, enable translating the object in the XY plane. The GUI (with GLUT) should support providing:

- a rotation speed factor
- a translation speed factor
- the selection of the object primitive (use radio buttons: sphere or teapot)
- specifying the field-of-view of the perspective projection used (see `gluPerspective`).
- specifying the distance from the camera to the object.
- **Extra credit 10%:** add the ability to change the “z” (or depth) to the object (using the mouse as well, enable this with the middle mouse button or when both left and right buttons are pressed).

By default use a field of view of 60 degrees and initialize other parameters to a reasonable value so that the object is visible and occupies most of the window. You may use all of OpenGL but for all math and matrix operations (e.g., except perspective projection matrices for which you may use the OpenGL `gluPerspective` call) you must use your library of object classes, including for all virtual trackball computations. Once your matrix is computed, you can load it to OpenGL using `glLoadMatrixf`.

Important note: remember if you use column-major vectors, you must transpose your matrix **before** giving it to OpenGL.

If you have more questions, please see myself or the TA.