B581 — Computer Graphics, Fall 2008 (A. Hanson)

Problem Set 4

Late after Midnight, Monday 10 November 2008. 24 hour grace period.
This problem set has 36 points.

20 points maximum score on any late submission.

Submission Instructions: The homework will be graded on a Linux system. Put all your files into one `tar` file, do not upload files individually. Do **not** upload executables; they will be returned ungraded for resubmission. **Grading** is based on this written problem statement, not on demonstration code that may be shown or made available. Any demos shown are for illustration only.

Submit a working Linux version with a Linux Makefile and the complete source code *(no executables)* in a tar file as follows:
Use "tar cvf ps$< N >< login >$.tar *" to create a tar file.
$< N >$ – the number of the assignment
$< login >$ – your user name
For example: A student whose username is xxyy works on ps4. The submitted file should be called `ps4xxyy.tar`.

We will work in `(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH)` mode, with double buffering (use `glutSwapBuffers()`) and will use both two-dimensional and three-dimensional coordinate systems.

The default coordinate system is a floating point 3D system from -1.0 to 1.0, although you are free to experiment with this. The Depth Buffer is activated, although you will not be using it in any "deep" way.

See the OGL Red Book Chapter 3, especially Example 3-8, if you choose to use `gluProject()` or `gluUnProject()`, the tutorial file `https://www.cs.indiana.edu/classes/b581/pickNpoint.html`, and the notes on the last page of this problem statement. You may also wish to remember Chapter 13, Selection and Feedback methods.

4.1 (5 points) **Implement Midterm Scaling Problem.** Activate with the "1" key.

Recall that this requires implementing a fixed point scaling with signed scaling depending on the sign of the difference between the current mouse-dragged point and the mouse-down point. The 'r' key should reset the Modelview matrix.

In this implementation, only the first mouse-down event actually is guaranteed to be a fixed point; later mouse-down points will not be fixed points.

One change: Create a grid or lattice of straight lines so that you can tell where the mouse went down and see where the first fixed point is.

4.2 (7 points) **Do Midterm Scaling Problem Correctly.** When the "2" key is pressed, fix the error in 4.1.

In 4.1, you only had to locate the scaling fixed point the first time the mouse was pressed and dragged. In this part, you must make the system scale about the actual mouse-down point as the fixed point EVERY TIME. See the code hints above and on the last page; you may want to use some combination of the functions:

```
GLint gluProject( GLdouble objX, GLdouble objY, GLdouble objZ,
                  const GLdouble *model, const GLdouble *proj,
                  const GLint *view,
                  GLdouble* winX, GLdouble* winY, GLdouble* winZ )

GLint gluUnProject( GLdouble winX, GLdouble winY, GLdouble winZ,
                    const GLdouble *model, const GLdouble *proj,
                    const GLint *view,
                    GLdouble* objX, GLdouble* objY, GLdouble* objZ )
```

to find the rescaled center and use the correct fixed point *each* time the mouse is pressed and dragged.

4.3 (12 points) *Implement the bug catcher.* For the "3" key — Establish a $(-1.0, 1.0)$ coordinate system. Use **double drand48()** or equivalent (you need to include "stdlib.h"), which gives uniformly distributed random numbers between zero and one, to place a "bug" on a 2D screen with $(0, 0)$ at the center (this is very important — the transforms below depend on this). The mouse will control the location of the "catcher" that you will draw and move around to try to catch the bug.

Task - warp the bug catcher: Make the "real" mouse location and the actual mouse location differ by one of several transformations relative to the center of the screen, specified below. Let (mx, my) be the mouse coordinates transformed to the $(-1.0, 1.0)$ screen coordinates; that is, if the screen is 100 by 100 and the bare (right-handed) mouse coordinates are (60, 70), then $mx = 0.2,\ my = 0.4$. Let (x, y) be the *location of the catcher*, i.e. DRAW a moving catcher at that location, and test to see if the catcher is close enough to grab the "bug." Do NOT draw anything at the actual mouse location (mx, my)! Cycle through the following transformations when the space bar is hit, printing the transform on the console:

(a) $(x, y) = (\pm mx, \pm my)$ four cases of reflection

(b) $(x, y) = (\pm my, \pm mx)$ (swapped reflection, normals)

Note: Except for one case, these will never be where the mouse is.

FEEDBACK MODE: When the 'a' key is hit, *draw lines* from the center of the screen to both the actual mouse point and the transformed point and draw a pointer at the real mouse location (mx, my) (usually you draw nothing).

GAME BEHAVIOR: Whenever the transformed (x, y) point and its "catcher" get sufficiently close to the "bug," do something disastrous like flashing the bug, then erase it and draw a new one at a new random location.

Optional for fun: Consider making a timer and print out your times for catching the bug. Do your times improve after you turn on Feedback Mode for a while, and then turn it off to play the game again?

4.4 (12 points) **3D - create a rotating teapot.**
The "4" key establishes a mode with a 3D coordinate system, and draws a 3D wireframe Teapot (look at glutWireTeapot). When the mouse is pressed and dragged, rotate the teapot about the top of the lid as the fixed point.

- **Left/Middle/Right Mouse:** Rotate about the x, y, or z axis, respectively.
- **Shift Left/Middle/Right Mouse:** Left (user-system) Translate along the x, y, or z axis, respectively. When you translate, you must keep the *new* tip of the lid as the fixed point whenever you rotate. You can do this *many* ways, including just recording the translations by hand, or being fancy with Project/Unproject.

Optional for fun: See the file `/l/www/classes/b581/spline-algorithms.pdf` for a summary of the simplified treatment of the general spline algorithms. On key "5", implement any 2D or 3D spline.

Hints and warnings:

- 3D translation and rotation of the entire scene ought to "stick", no matter how many times or in which order they're interactively performed.
- *Clipping in depth.* You need to be careful that you have plenty of coordinate space or else your object may be clipped. The default template can only see $-25 < z < -5$, since `glFrustum(-1.0,1.0,-1.0,1.0,5.0,25.0);`, and the Modelview matrix is initialized to draw at $z = -10$. You need to include this z displacement in your fixed point code.
- `MouseModifiers = glutGetModifiers();` returns a mask with `GLUT_ACTIVE_SHIFT, GLUT_ACTIVE_CTRL, GLUT_ACTIVE_ALT` ; you can only call this in the mouse button routine, so the global variable needs to be set to be checked in other routines.
- the provided template code includes a function drawing 3D coordinate system axes. This is just to get you started with 3D translation and rotation. You must it remove calling this function in your submitted solution code's `display()` callback function, so that it is not drawn when the wireframe teapot surface is displayed.

Implementation Suggestions. See also https://www.cs.indiana.edu/classes/b581/pickNpoint.html.

Left Matrix Multiplication:

```
GLfloat mvmatrix[16];

glGetFloatv(GL_MODELVIEW_MATRIX,mvmatrix);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
TransformWHATEVER();
glMultMatrixf(mvmatrix);
```

Finding the Mouse with UnProject:

```
GLint viewport[4];
GLdouble mvmatrix[16],projmatrix[16];
GLdouble wx,wy,wz;
GLint realY;
GLint XformX, XformY;

glGetIntegerv(GL_VIEWPORT,viewport);
glGetDoublev(GL_MODELVIEW_MATRIX,mvmatrix);
glGetDoublev(GL_PROJECTION_MATRIX,projmatrix);

/*  realY = viewport[3] - (GLint) Y - 1;    */

gluUnProject((GLdouble) X,(GLdouble) Y, 0.0,
             mvmatrix, projmatrix, viewport,&wx,&wy,&wz);

XformX = wx;
XformY = wy;
```

Finding the Window point with Project:

```
   double winX, winY, winZ;
      /* Get window coordinates */
      gluProject(point[xd], point[yd], point[zd],
                 mvmatrix, projmatrix, viewport,
                 &winX, &winY, &winZ);
```