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

Problem Set 3

Please Target This Date: Midnight, Friday, October 17, 2008
and reserve some of your weekend time to study for the Midterm.
Grace Period until Midnight, Saturday, October 18, 2008.
This coding problem set has 40 points, about 4% of total grade.
Maximum score is 50% on late submissions.

MIDTERM exam is 1:00pm on Wednesday 22 October 2008, in LH115.

Remember: The homework will be graded on a RedHat Linux system. Put all your files into one `tar` file, do not upload files individually. Do NOT send executables; they will be returned ungraded for resubmission. GRADING is based on this written problem statement, not on demonstration code that may be shown.

We will work in `(GLUT_DOUBLE | GLUT_RGBA)` mode, with a two-dimensional right-handed coordinate system that corresponds closely to pixel locations. You can redraw with each Display callback and use glutSwapBuffers() now. Remove references to the Depth Buffer if they are in your template for some reason. The `reshape(w,h)` function supplied in the template has been configured for *right-handed* Cartesian coordinates. One line of transform code in the supplied template mouse handlers turns the "upside down" GLUT mouse/screen coordinates right side up. Therefore, you can draw pixels directly in the transformed mouse coordinates in all the following exercises. You may want to use the variables `main_height`, `main_width`. It is recommended to get in the habit of implementing a "help" text printed when the user presses the 'h' key.

Left Multiplication Survival Kit:

```
float mat[16];
...
glGetFloatv(GL_MODELVIEW_MATRIX,mat);
glLoadIdentity();
glTranslatef((float)(X-X1),(float)(Y-Y1),0.0);
glMultMatrixf(mat);
```

3.1 (15 points) **Polygon create/edit, Identity Modelview Matrix**

Task: *Interactive Polygon Entry.* In the mode selected by the '1' key, or after hitting the lower-case 'r' reset key, go into Polygon entry mode, activating a set of OpenGL glut callbacks implementing the features below. **Note**: all this assumes that the Modelview matrix is the identity matrix. This will change in the next part. The tasks for this part are the following:

- **Left Click/Add and Drag.** Insert a new vertex for a wire-frame-polygon starting at the mouse-down point and dragging to the desired endpoint, saving this point on the polygon using the value of the mouse-up point. Save the points in sequence in an array. Hint: Look at `GL_LINE_LOOP` . Note: you will of course need at least three points to see a polygon appear.

  You **may** clear and redraw everything with each call to the Display callback. HINT: Because of the nature of the Line Loop object, a single array can be used to add a vertex each time the mouse clicks. But be careful to check the array *bounds*.

- **No Buttons Down: Passive Motion Vertex Highlighting.** Detect proximity (within a number of pixels that you choose) of any vertex (using the Euclidean distance formula). Highlight the vertex by drawing a larger or different colored representation of the object part to make it apparent to the user. Hint: An elegant solution is to XOR the same object with a larger value of `glPointSize()`.

- **No Buttons Down: Passive Motion Nearest-Edge plus Nearest-Point-on-Edge Highlighting.** If no actual vertex is near the mouse cursor, determine if any line segment is near the mouse cursor (again, you pick a distance). If so, create a highlight for the edge itself (e.g., a larger XOR-ed object) as well as the *nearest point on the edge* to the mouse position. Hint: be careful to highlight only when the mouse point is near the polygon line *segement*, not the infinite abstract line.

Highlighting requires some tricky bookkeeping: you need to have it *off* when the mouse is not near anything, it must remain *on* when the mouse is within the threshold number of pixels, and it must *turn off* as soon as the mouse cursor moves too far away again.

3.2 (15 points) **Moving the object with matrices.**

Task: If the '2' key is depressed, go into a mode that uses mouse button press-and-drags to execute translations, scaling, and rotations using the GL_MODELVIEW matrix, leaving the polygon vertex values untouched. Be careful to precede this action with a call to glMatrixMode(GL_MODELVIEW), *and* to avoid problems with part 3.3, check that you return to GL_PROJECTION mode when needed.

- **Left Button: Translate.** When this button is pressed and dragged while down, translate the whole object in *user coordinates* to follow the mouse cursor, leaving the object at its new position when the mouse is released. (See included Left Multiplication Code.)
- **Middle Button (preferred) or Left+Right: Scale, center of mass fixed point.** When this button is pressed and dragged while down, scale the whole object proportional to the x,y motions of the mouse cursor, leaving the object at its new scale when the mouse is released. Be careful not to allow zero scale values, and to use small numbers near 1.0 to scale the ModelView matrix (see notes). Use the polygon center of mass as the fixed point
- **Right Button: Rotate, center of mass fixed point.** When this button is pressed and dragged while down, rotate the whole object by an angle proportional to the mouse motion (x, or y, or both - your choice), leaving the object at its new orientation when the mouse is released. Use the polygon center of mass as the fixed point
- **Reset.** Capital “R” key resets the transformation to the identity without resetting the polygon.

3.3 (10 points) **OpenGL Selection Locates Points after Arbitrary 2D Matrix Transformations.**

When the “3” key is hit, go into a new mode that replaces the fixed-coordinate system passive-mouse-motion callback, etc., by a highlighting/dragging system that does not refer to the coordinates at all, but uses the OpenGL *Selection and Feedback* system.

Read Chapter 13, Selection and Feedback, particularly Example 13-3. Note: a slightly obsolete version of this chapter (in the edition printed when it was Chapter 12) can be found online at `http://www.opengl.org/documentation/red_book/`. An enhanced and corrected version of `picksquare.c` (Example 13-3) is included in your template code.

Task: Mouse Near a Vertex. Use the methods of Chapter 13 to determine when the passive mouse callback (no buttons down) is near one of the vertices *without* using a direct proximity calculation; light up the vertex when it is near. Using the button-down and button-dragged mouse callbacks, test if the passive motion callback believes the mouse is near a vertex; if so, identify the vertex and *drag the lit vertex* in the “local” rectangular coordinates. This modifies the vertex coordinates in their *local* coordinate system only; it is all right if this does not correspond to user coordinates.

(Why study this? Because this way you do not even have to ask what the components of the matrix multiplying the vertex are to find where the vertex or other object is located. Later we will look at yet another set of utilities that can map all these operations to the user coordinates.)

Instructions: Use the keyboard “1”, “2”, or “3” key to select the section of the problem to be activated. ’r’ resets the whole system, ’R’ reset only the transformation matrcies in part 3.2. You may wish to have separate display and control programs, in the same or separate files, or maybe just an internal switch statement. In any case, 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 ps3. The submitted file should be called `ps3xxyy.tar`.