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

Problem Set 1

Due by 5pm Friday, 26 September 2008

This coding exercise has 36 points, about 3% of total grade.

Do the following programming exercises, with

glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE).

Turn in your final code in a tar file `ps1mylogin.tar` (NO executables) to username **b581** at **cs.indiana.edu**. Please include the Makefile.

(*Note*: `extern double y0, y1;` are in `math.h`, which can cause a common naming conflict, `'y0' redeclared as different kind of symbol` , if you try to use these as variables.

(*Note*: You will need to check the calling sequences of a number of Glut and OpenGL procedures. While these are in the Red Book and in the CS B581 utility directory at docs/glut-3.2spec.pdf, the unix **man** command should also work most places. Among the functions you may want to look up are:

glutMouseFunc, glutMotionFunc, glutPassiveMotionFunc, glutKeyboardFunc, glRasterPos2i, glBitmap, glPixelZoom, glDrawPixels, glCopyPixels.

1. (10 points) **OpenGL rubberband line.** When your application opens, or when the "a" key is pressed on the keyboard, do the following:

   - *Use the glutMouseFunc() callback.* When the left mouse button goes down (check "state" for `GLUT_UP or GLUT_DOWN`), record the mouse location.
   - While the mouse button is held down, using glutMotionFunc(), save the mouse position and draw a line from the position where the mouse button went down to the current mouse location. As the mouse pointer is moved, more lines will be drawn. Stop drawing when the mouse goes up (state is `GLUT_UP`). Do not erase older lines.
   - Each time the "a" key is pressed, set a flag that signals the display callback to clear the entire window *just once* and prepare to start drawing lines again when the mouse is pressed and dragged.

   The purpose of this is to watch the destructive nature of drawing without refreshing, and prepare for a future exercise in non-destructive Exclusive Or usage.

2. (10 points) **Pixel operations.** Activate this mode when the "p" is depressed. to use **glRasterPos2i, glBitmap, glPixelZoom, glDrawPixels and** `GL_POINTS` .

   - You probably want to specify the pixel mode in your initialization, e.g.
     `glPixelStorei(GL_UNPACK_ALIGNMENT, 1);`.

- **LEFT = points** . Whenever the left mouse button is pressed, draw a point at the current mouse location using glBegin, etc. Suggestions: vary the colors, vary the size using glPointSize().
- **MIDDLE = bitmaps** . Whenever the middle mouse button is pressed, draw a bitmap of your choosing using glBitmap, etc. at the current mouse location. Suggestions: vary the colors, or even vary the bitmaps.
- **RIGHT = color pixels** . Whenever the right mouse button is pressed, draw a block of color pixels at the mouse location using something like
  `glDrawPixels(8,8, GL_RGB,GL_UNSIGNED_BYTE,rgbPixels);`
  Suggestions: vary the size using glPixelZoom(xscale,yscale).
- Do not clear the window while drawing in this mode, but clear and start fresh when the "p" key is pressed.

3. (16 points) **Mouse Controlled Screen Zoom.** When the "z" key is depressed on the keyboard, use a different display function, or a different branch of the display function, to do the following:

   - Set up a nested grid of rectangles that cover the entire screen, all centered at the screen center (which you should calculate accurately and have available to the program). *Optional:* You can sprinkle the screen with equally spaced bitmap images in addition to the rectangles. Remember you have to fill in the center if it gets empty, keeping the density of information uniform (see below).
   - *Use the glutMouseFunc() callback.* Record the point at which the mouse button went down.
   - *Use the glutMotionFunc() callback.* When the left mouse button goes down and you start dragging the mouse, *clear the screen each time the mouse value changes*, record the mouse position, and *scale the world of nested rectangles that you have created as follows:*
     - Scale by changing the rectangle sizes by the distance from the mouse-down point to the current point.
     - Make the rectangles *bigger* if the mouse is moving away from the center. Create *new* rectangles to fill in empty space in the middle.
     - Make the rectangles *smaller* if the mouse is moving towards the center. Destroy rectangles whose size shrinks to nearly zero. Don't do anything if the mouse is within a few pixels of the center.

- Optional experiment: see what happens if you use the *ratio* of the initial distance to the center divided by the current distance to the center to scale the grid. We can discuss the pros and cons of the different methods in class.
- Each time the "z" key is pressed, clear the window and start over.
- This will flash and blink, since we cannot change from Single buffer mode to Double buffer mode in a single application, at least not easily. Don't worry about it — we will fix this in later exercises.

Instructions: Use the keys as described above to select the section of the problem to be activated. You may wish to have separate display and control programs, in the same or separate files, or maybe just an internal switch statement.

You can create the program with the "make -f Makefile.linux" command.

In any case,submit a working LINUX version with a LINUX makefile named `Makefile.linux` 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 ps1. The submitted file should be called `ps1xxyy.tar`. Email the file to the username **b581** at **cs.indiana.edu**