

Prefootnote: See HB Section 11-6 for the way to declare the keyboard and mouse callback functions. The prototype given in an old edition of the Red Book
`glutKeyboardFunc(void(*func)unsigned char key, int x, int y);`
is wrong.

The main objective of Project 2 is to practice programming of linear and translation transformations in 2-d; and then *experiance by seeing* the effects of your work. Secondary but necessary objectives are to code a graphic model data structure, program OpenGL operations with data from the model, and program event-driven user input.

Use `proj2` for the project name when you submit it using `turnin-csi422`.

Function

The viewing should be configured the same way as `double.c` from the Red Book

Unlike pure GL programs, it will print messages on the controlling terminal in addition to displaying graphics.

When the program starts, or when the `z` key is pressed, it displays, in the center of the display window, the **U** logo or a more creative centered and non-symmetric figure. The logo, like the square, should be about half the width and height of the display window.

When a mouse button is pressed, the program will print a message like
`myMouseFunc(button=0, state=0, x=685, y=226)`
that shows the values of the arguments to the `glutMouseFunc` callback. It should also record the coordinates of the mouse position.

When a mouse button is released, the program will print a message like
`myMouseFunc(button=0, state=1, x=690, y=203)`
that shows the values of the arguments to the `glutMouseFunc` callback. It should calculate and print the displacement vector from the mouse position of the previous button press. The displacement should be expressed using coordinates for which the positive `x` direction is right, and the positive `y` direction is **UP** (not down, which is what positive mouse position coordinate `y` means). For example
`Mouse Dx=15, Dy=23`
can be printed when the mouse was dragged up and to the right.

Also, when a mouse button is released, the transformation specified by the program's **transformation mode** with parameters given by the one or both components of the mouse displacement shall be applied to the *figure currently being displayed*. The display should be cleared and the transformed figure be drawn.

When each of the following keys is pressed, the program goes into the indicated transformation mode. The transformation for each mode is then explained.

t translate This is the initial transformation mode. The logo is translated in the `x` and

y directions by offset parameters proportional to the corresponding components of the mouse displacement.

r rotate Only the x coordinate of the mouse displacement is used to determine the angle parameter. The logo is rotated around the world coordinate origin by an angle proportional to the mouse displacement coordinate.

k scale The logo is scaled relative to the origin by x and y parameter factors proportional to the respective mouse displacement coordinates.

h shear The logo is sheared relative to the origin by parameters sh_x and sh_y proportional to the respective mouse displacement coordinates. See the “Shear” paragraph in HB Chapter 5.

Note the user is likely to transform the logo completely out of the clipping window so it will no longer be visible. The user can restore visibility by pressing the z key.

Structure

The program should have a static array of subarrays or structures that is initialized to hold the vertex coordinates for the master model of the logo.

The program should have another static array called the “working vertex array” of subarrays or structures that will hold the vertex coordinates to use for drawing the logo. The program initialization and z key operation will copy all the coordinates from the master model into this array.

The program should have static GLint variables to save the mouse coordinates after a mouse button press. They are needed in the subtractions done to compute the mouse displacement after a mouse button release. You might use additional static variables for mouse state, so that you can make your code more straightforward.

It should also have a static variable of type GLubyte for the transformation mode, so that the key’s character value passed to the KeyboardFunc callback can be used by the MouseFunc callback (or a function it calls) to choose among the four transformations.

You must code the application of each transformation to the vertices in the working vertex array yourself. Class notes and HB Chapter 5 contain relevant the mathematical concepts and formulas. You can implement the formulas by using your own matrix and vector operations or by just coding the formulas. (OpenGL matrix facilities MUST NOT BE USED for this purpose. Indeed, they are clumsy for this purpose because the OpenGL matrix transformation functions compose the new with the current transformation in the wrong direction for us.)

Here are some tips. You can do the projects in C++ also.

1. C Code to print the specified event messages is

```
#include <stdio.h>
```

```

.....
printf("myMouseFunc(button=%d, state=%d, x=%d, y=%d)\n",
      button, state, x, y );
/* prints myMouseFunc(button=n, state=n, x=n, y=n) */

printf("Mouse Dx=%d, Dy=%d\n", Dxvalue, Dyvalue);
/* prints Mouse Dx=15, Dy=23 */

```

2. In C, you can't use `for(int i = 0; etc.` since all declarations must be at the beginning of a block. Just move the `int i` up.
3. Arrays or subarrays cannot be copied with one assignment statement. (However, structures or classes can, even if they contain arrays.) So, to copy a vector with two components, you must code 2-3 assignment statements or a length 2 or 3 loop.
4. The code to apply a transformation to a vertex will change the two (or 3) components. Be sure the code only uses the old values in the transformation formulas. You will need temporary variables for some cases.

Here's the problem: Interpreting the equations

$$X = AX + BY$$

$$Y = CX + DY$$

as C/C++ assignments is wrong because in the calculation for second assignment (to Y), the NEW value of X is used.

5. Use `GLfloat` or `GLdouble` for vertex coordinates and for numbers in the transformation formulas, not integers, for accuracy. Note the visible world coordinate range is ± 100 .
6. Include the math library header file with `#include <math.h>` to get the declarations of `sin()` and `cos()`.
7. Store coordinates of points in 2-d arrays of floats, then the `glVertex2fv` form of `glVertex` to save typing and run time argument passing.
8. Use `'z'`, `'t'`, `'r'`, `'k'` and `'h'` for the char constants to test the input character.

Project Steps

1. Start. `double.c` example and discussion from the Red Book. Download it and make sure you can compile and run it...Seek help IMMEDIATELY if you cannot.
2. Make your own model. Polygon tables in HB 3-15, Vertex Arrays in HB 3-17, Icosahedron model in Ch. 2 of Red Book illustrates the kind of model you should program. Yours should be in 2-d and simpler.

Declare the master vertex array and the initializations that define the vertices of your logo.

Declare the working array.

Write a function to copy the master values into the working array.

A separate function should be used to draw the logo by calling OpenGL primitives to access the working array. This small, specialized function should then be called by the display callback.

3. Control a gl transformation with the mouse. Get `double.c` to rotate the square an amount given by the mouse motion instead of an animation.

First, put into the `MouseFunc` callback the code to (1) print the messages when the mouse buttons are pressed and released. Recompile and observe the numbers printed when you experiment with the mouse.

Next, declare the static variables to hold the mouse position and add to the `MouseFunc` callback the code to calculate the displacement coordinates. Test it by coding and observing the mouse displacement messages.

Finally, write code to use mouse x displacement, converted to a float value, to update the value of `spin` in the original program. Then verify that the OpenGL `glRotatef` will rotate the logo by an amount specified by the mouse drag amount.

Eventually, you will remove the use of OpenGL matrix facilities. But the original `double.c` program is a good place to debug your strategy to obtain an amount to change an angle from a mouse drag.

4. Develop the keyboard controls. Read HB Ch. 11, Study 11-6.

Be sure to declare a static `char` variable to store the code of the pressed key.

At this point, think about what you will make your program do if the user presses an invalid key. The behavior is “unspecified” but it should be “reasonable” (not crash or go into an infinite loop, etc.)

5. Develop your own model transformation functions. Read HB Ch. 5 (to study for midterm too). Learn how to apply 2-d transformations and adapt the `transformVerts2D` function of HB 5-15, and the transformations treated in HB 5-5.

It would be a good idea to write 4 separate functions, one for each kind of transformation.

Like the code in `double.c`, your mouse function should call `glutPostRedisplay()` after it transforms the working array vertices so the display window is cleared and the transformed logo is drawn.

6. Integrate (put it together). Test, experiment, adjust transformation from screen coordinates passed to the mouse callback into floating point numbers to control the transformation. What works well for rotation and translation will behave badly for shearing and scaling.

Midterm 1 Accumulated Readings and Exam Syllabus

During the first midterm, you will be expected to use Hearn and Baker's textbook particularly for details on using a few OpenGL functions, and other details as necessary. So, concentrate on the *order in which various operations are done* and *what they mean*.

You are also invited to bring one 8 1/2 by 11 inch sheet of notes. No other materials allowed!

Named Paragraphs, numbered sections and (principles) in Hearn and Baker:

Within Chapter 2: Coordinate Representations. Graphics Functions. Basic OpenGL Syntax. Display Window Management Using GLUT. Within A Complete OpenGL Program, emphasize: (display and other callback functions).

Within Chapter 3: 3-1 Coordinate Reference Frames. (Converting from world to screen coordinates as done in class.) 3-2 Specifying a Two-Dimensional World-Coordinate Reference Frame in OpenGL. 3-3 OpenGL Point Functions. 3-4 OpenGL Line Functions. 3-5 Line-Drawing Algorithms. 3-7 Setting Frame Buffer Values. Within 3-9, (equations for circles). Polygon Tables. 3-16 OpenGL Polygon Fill-Area Functions (including arrays and/or structures for storing your Project 2 model.) 3-20 (Bitmap and outline fonts compared). 3-24 OpenGL Display-Window Reshape Function (how to preserve aspect ratio during resizing).

Within Chapter 4: 4-1 State Variables. 4-2 Color and Gray Scale. 4-5 Line Attributes. 4-7 OpenGL Point-Attribute Functions. 4-8 OpenGL Line Attribute Functions. 4-9 Fill-Area Attributes (and the formula $\mathbf{P} = t\mathbf{F} + (1 - t)\mathbf{B}$ for blending).

Practically all of Chapter 5 (it's not that much because the 3-d material is mostly a repetition of the 2-d material). Omit 5-6, 5-7 and Quaternion for now.

Chapter 6: Two dim. viewing as done in class. When should `glViewport()` set the viewport to the window size; and when should it not? Within 6-7, equation (6-13)

Chapter 11: GLUT mouse and keyboard functions in 11-6

Red Book: Double Buffering and Animation from Chapter 2.

Forms and manipulation of equations for lines and circles as done in class.