PROJECT GOAL: Write a restricted OpenGL library.

The goal of the project is to *compute* all the transformation matrices with your library instead of using OpenGL library. Since you are relying on OpenGL for final rasterization, you have to let OpenGL know what the final transformation matrix is, so that it transforms the vertices and triangles and does the final rasterization. There are available third party libraries online, but you are NOT allowed to use them. Recall the assignment policies included in your first assignment description, third party codes are only allowed for extra features and importing/exporting files (I will try to specify it when it is needed).

1 Preparation

Before you start your OpenGL library, you should notice that this time the assignment package includes several new files which are named as "my_gl" and "i_my_gl". They basically serve as a skeleton of the implementation of your library. To start your work, first browse these new files to get a basic idea about the contents of them. At the beginning of the i_my_gl.cpp file, you can see there are a few structures and constants have already been defined for you. Feel free to use them (and you should) for the rest of the assignment. You are also allowed to modify them according to your programming style or design issues. After these predefined contents, there are three empty-body functions:

void matrix_multiply(const GLdouble *b) void
cross_product(GLdouble *ax, GLdouble *ay, GLdouble *az,
    GLdouble bx, GLdouble by, GLdouble bz,
GLdouble cx, GLdouble cy, GLdouble cz)
void normalize(GLdouble *x, GLdouble *y, GLdouble *z)

By just look at the names, they should easily remind you your linear algebra knowledge. Go ahead to implement them and then you can call them whenever you need to for your later work.

The rest part of the i_my_gl.cpp file contains functions which are used as "internal" functions. Thus, you will not call them directly from your application code. You have to use the wrapper functions contained in the my_gl.cpp file to call these internal functions. All wrapper functions perform at least three operations: (a) Call the appropriate "internal function". (b) Get the final matrix from the stack using I_my_glGetMatrix function. (c) Use glLoadMatrix (actual OpenGL) function to load the matrix into appropriate OpenGL stack.

There is a my_gl.h file provided in the package, in which there are prototypes of all the wrapper functions and a few of macros in the format "#define <glFuntion> <my_glFunction>". For example,

#define glLoadIdentity my_glLoadIdentity

All these macros are currently commented out. When they are enabled, they make the corresponding functions you implemented override the original OpenGL functions.

2 Matrix and Stack Manipulation Functions

The core work of this assignment is to implement the internal functions in the i_my_gl.cpp file. Follow the rules at the end of this description during your work. As you can see, all these functions share names with existing OpenGL (some of them are from glu) transformation or stack-accessing functions. Thus, to implement them perfectly, you need to study the blue book: OpenGL Reference Manual http://www.glprogramming.com/blue/ to understand how they work. Note: the reference manual does not explain the gluLookAt function clearly enough, so take a look at the programming guide chapter 3 for more details:
 http://glprogramming.com/red/chapter03.html#name2

void I_my_glLoadIdentity(void) void
I_my_glPushMatrix(void)
void I_my_glPopMatrix(void)

void I_my_glLoadMatrixf(const GLfloat *m)
void I_my_glLoadMatrixd(const GLdouble *m)

void I_my_glTranslated(GLdouble x, GLdouble y, GLdouble z) void
I_my_glTranslatef(GLfloat x, GLfloat y, GLfloat z);

void I_my_glRotated(GLdouble angle, GLdouble x, GLdouble y, GLdouble z) void
I_my_glRotatef(GLfloat angle, GLfloat x, GLfloat y, GLfloat z);

void I_my_glScaled(GLdouble x, GLdouble y, GLdouble z) void
I_my_glScalef(GLfloat x, GLfloat y, GLfloat z);

void I_my_glGetMatrixf(GLfloat *m)
void I_my_glGetMatrixd(GLdouble *m)

void I_my_gluLookAt(GLdouble eyeX, GLdouble eyeY, GLdouble eyeZ, GLdouble centerX,
                    GLdouble centerY, GLdouble centerZ, GLdouble upX, GLdouble upY,
GLdouble upZ)

void I_my_glFrustum(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top,
GLdouble zNear,   GLdouble zFar)
void I_my_gluPerspective(GLdouble fovy,   GLdouble aspect,
GLdouble zNear, GLdouble zFar)
(NOTE: "glu" instead of "gl" for LookAt and Perspective)

3 Matrix Mode Function for Operations

The above functions operate on the "current stack". The "current stack" is one of the
ModelView stack and the Projection stack. To determine which stack your program is currently
working on, you also need to implement the following function
void I_my_glMatrixMode(GLenum mode);
This function will allow you to switch between the two matrix modes, GL_MODELVIEW and
GL_PROJECTION.

4 Test Case 1: Assignment 1

In the package, a tiny modified version of your assignment 1 is also included. Use it to finish
this test case. Assignment 1 should call your wrapper functions instead of the OpenGL
functions. To achieve this, it includes the my_gl.h file in main.cpp and sceneModule.cpp files
now. Later you may or may not want to include it in the inputModule.cpp as well depending on
how you organize your program. Do not include this my_gl.h file in the files that you write for
this assignment (assignment 2), especially the wrapper function file, since you do not want to
substitute the actual OpenGL functions that have to be called with your function names.

Now you want to test your own version of the following functions one by one:

glMatrixMode
glLoadIdentity
glLoadMatrixf
glLoadMatrixd
glTranslated
glTranslatef
glRotated glRotatef
glFrustum
gluPerspective

To achieve this, verify your functions in the following way:

i)   Whenever you use glLoadIdentity() for a MODEL_VIEW matrix, initialize your matrix
to a 4x4 identity matrix.

ii)   Replace every call to any OpenGL transformation function with the following sequence of calls. The following example is given for glTranslate. Perform similar changes to other calls you have implemented.

Create two local 16-element array variables: tempmatrix and newmatrix;
Replace every glTranslate(tx,ty,tz) with:

> *// This operates the OpenGL way, retrieves the result and undoes it*
> *glPushMatrix( );*
> *glTranslated(tx, ty, tz);*
> *glGetFloatv(GL_MODELVIEW_MATRIX, tempmatrix);*
> *glPopMatrix( );*
>
> *// This uses your my_gl* functions, and will have effect*
> *my_glTranslated(tx ,ty, tz);*
> *my_glGetMatrix(newmatrix); printMat(newmatrix) ;*
> *printMat(tempmatrix);*
> *if(compareMat(newmatrix, tempmatrix)) printf("Matrix is wrong");*
> *// compareMat raises the flag if the matrices don't match.*

iii)   For all this to work properly, you must make sure to comment the list of #define statements in my_gl.h, so that no automatic substitutions are performed and you control which version (gl or my_gl) of the transformations is called in each moment.

iv)   printMat and compareMat are functions not included in the package, but can be easily made by yourself. PrintMat should just print out the values of all 16 entries of the given matrix, and compareMat should compare two matrices element by element. GlGetFloatv can be replaced with glGetDoublev if your tempmatrix and newmatrix are arrays of double type. And the first parameter GL_MODELVIEW_MATRIX should be replaced by GL_PROJECTION_MATRIX when you test projection functions. Check glGet in the reference manual for details about how glGetFloatv and glGetDoublev work.

v)   Notice that the first three functions in the Part 1 should also be fully tested (in case you are using them).

The above change is going to print a lot of matrices. Once you have verified your compareMat function correctly compares two matrices, you can remove the printMat statements. It does not need to be part of your submission.

Once you have verified that there is no statement displayed that "Matrix is wrong" (in other words, your matrix in newmatrix and OpenGL's matrix in tempmatrix are the same all the time), then you should replace OpenGL's computation of transformations with your equivalent computation of the transformations by uncommenting the corresponding macros.

Check if your output is the same as before (before substituting gl* with my_gl*). If so, then you have successfully emulated OpenGL transformation calls using your own my_gl transformation calls.

5 Test Case 2: gluLookAt Function

In the sceneTransformation function of Assignment 1, several glTranslate and glRotate functions are called to do the model/view transformation controlled by mouse motions. In this test case, replace these transformation function calls with a single gluLookAt function call. First try to work on OpenGL's gluLookAt function to understand how it works. You need to set the arguments you sent to gluLookAt carefully to make it work exactly in the same way as Assignment 1. GluLookAt basically translates and rotates the camera instead of directly manipulate on the cube, which implies it works in the opposite way as the previous approach. (Again, read the chapter 3 of the programming guide mentioned in Part 2 for more details.) The motion controlled by the middle mouse button can be tricky here. If you feel too difficult to figure it out, you can skip the middle mouse button, it does not count any points in this assignment. After you make OpenGL's gluLookAt function works perfectly, enable your own gluLookAt function by uncomment it in my_gl.h to verify your implementation and fix potential bugs.

6 Test Case 3: Animation

In this test case, you are going to test all the rest functions:

glScaled glScalef
glPushMatrix
glPopMatrix

Again, first try use the OpenGL versions of these functions to achieve the following task. Modify your program to display one more cube (call the drawScene function twice in your display function). Let us call these two cubes Spinner and Revolver, and they should satisfy the following constraints:

a)   Spinner should rotate about an axis through its center, parallel to any edge.
b)   Revolver rotates about the same axis as Spinner revolving around Spinner.
c)   Revolver DOES NOT rotate about an axis through its own center.
d)   The frequency of Spinner rotating about its own axis should be the same as that of the rotating Revolver. This means, for every complete revolution of Revolver around Spinner, Spinner should itself have completed 360 degrees. e)   Spinner should be larger than Revolver.

Your program should support interactive viewpoint manipulation, i.e., the user should be able to use the mouse select viewpoint position in space. - Use glLookAt for this.

Hints that MIGHT be useful:

a)   Use glTranslate to make these two cubes rendered in different place
b)   Use glRotate to rotate them
c)   Use glScale to make them in different sizes
d)   OpenGL uses a local co-ordinate system, and POST-MULTIPLIES successive transformations.
e)   Use glPushMatrix and glPopMatrix to utilize the stack to do different sets of transformations separately on Spinner and Revolver.
f)   To do animation, you need to use glutIdleFunc(). The way it works is glutIdleFunc() continuously calls the function you pass to it as argument while the system is idle. When this function modifies your scene slightly each time it is called, the entire scene becomes an animation. Most likely in this program this function will be your display function. You may also want to do it in another way (rather than call display). It is up to you. Regardless which way you do it, the key point is the function called by glutIdleFunc() should involves some variables which will be changed each time it is called, and these variables will affect the transformations you do on the Spinner and the Revolver.

Now you want to verify your own functions for this animation program.

RULES:
1. Allow a maximum of 16 matrices to be pushed.
2. Report error if the stack is empty when a my_glPopMatrix function is called and continue.
3. Report error if the stack is full (16 elements) when a my_glPushMatrix is called and continue.
4. *m* is a pointer to the array of 16 consecutive values (linear array) of the matrix (4x4 matrix) in *column major order.*
5. Implement a "static" array of matrices so that consecutive calls to your matrix manipulation routines will be accumulated.
6. Include gl.h to make use of the data types GLfloat* and GLdouble*.
7. The meaning of each of the above functions (except my_glGetMatrix*) takes the same semantic meaning as the functions in OpenGL library (without "my_"). Reference OpenGL blue book for these functions.
8. my_glGetMatrix* function returns through *m* the matrix you have on the top of the stack.
9. All internal computation of composition of matrices should use GLdouble.
10. Use I_my_glFrustum to perform I_my_gluPerspective.