Transformations

CS 3451: Project 1B

1 - Objective

The goal of this project is to understand through implementation the basic underpinnings of a simple graphics library that is similar in design to openGL.

2 - Deadline

Your project solution should be submitted on T-Square by 11:55PM on Wednesday, February 3, 2016.

3 - Process

3.1 Download the base source

Download and unzip the folder with the base code for this project.

3.2 Re-purpose your previous project

You should first paste the Matrix class from your previous project into the Matrix tab in the source code. You can also use your previously designed Transformation functions to help you with those in this project.

3.3 Project description

As stated in the project objective, you will be implementing a basic graphics library which we will call openGT, of a similar style to the popular openGL library which you will use for future projects. You will implement several routines which allow a user to manipulate and display 3D objects in a scene. The provided source code provides you with empty methods for each of the following routines: (See the Source Code section for more information.)

1. **gtInitialize()**

   The gtInitialize command should initialize your matrix stack to have just a single matrix on the stack. This matrix should be the identity matrix.

2. **gtPushMatrix() / gtPopMatrix()**

   The gtPushMatrix command replicates the matrix at the top of the matrix stack and places this new matrix on top of the stack. This new top matrix is now the current transformation matrix. The gtPopMatrix command pops the top matrix off the stack, causing the next matrix down to become the current transformation matrix. An error message should be printed if a pop is attempted when only one matrix is on the stack. As described under the gtInitialize command, the matrix stack is initially created with an identity matrix as the only matrix on the stack. Your stack only needs to handle up to 10 matrices on it at any one time. **You are not allowed to use Java built-in Stacks for this assignment. ArrayLists and arrays are allowed.**

3. **gtTranslate(float x, float y, float z)**

   Multiply the current transformation matrix on the right by a matrix specifying a translation of (tx, ty, tz). The current transformation matrix is defined to be the top matrix on the matrix stack.

4. **gtScale(float sx, float xy, float sz)**

Multiply the current transformation matrix on the right by a matrix specifying a (possibly non-uniform) scaling of (sx, sy, sz).

5. **gtRotate(char axis, float theta)**

   Multiply the current transformation matrix on the right by a matrix that specifies a rotation of theta degrees about the axis (‘x’,’y’, or ‘z’). The rotation is counter-clockwise as one looks from the axis towards the origin. For example, the command gtRotate (‘x’, 30.0) specifies a 30 degree rotation counter-clockwise around the x-axis.

6. **glRotate(float theta, float axisX, float axisY, float axisZ)**

   Multiply the current transformation matrix on the right by a matrix that specifies a rotation of theta degrees about the axis <axisX, axisY, axisZ>. The rotation is counter-clockwise as one looks from the axis twoards the origin. For example, glRotate(30.0, 1, 1, 0) rotates the transformation matrix 30 degrees counter-clockwise around a vector pointing to the upper-right.

7. **gtOrtho(float left, float right, float bottom, float top)**

   Specifies that a parallel projection will be performed on subsequent vertices. The direction of projection is assumed to be along the z-axis. The four values passed to this routine describe a box to which all lines will be clipped. The "left" and "right" values specify the minimum and maximum x values that will be mapped to the left and right edges of the framebuffer. The "bottom" and "top" values specify the y values that map to the bottom and top edges of the framebuffer. The eye point is assumed to be facing the negative z-axis.

8. **gtPerspective(float fov)**

   Specifies that a perspective projection will be performed on subsequent vertices. The center of projection is assumed to be the origin, and the viewing direction is along the negative z-axis. The value "fov" is an angle in degrees that describes the field of view. In order to make it easier to write this routine, we will assume that all screen sizes will be square, so you don't need to worry about the vertical and horizontal field-of-view being different.

   OpenGL uses a separate matrix to do projection that is different than the current transformation matrix and its associated stack. This means that in OpenGL, you can specify projections at any time before you draw lines and polygons. We will do the same for our assignment. Which ever projection that you specify (gtOrtho or gtPerspective) should be the last transformation that is applied to the line endpoints, regardless of where those procedure calls appear with respect to other transformations.

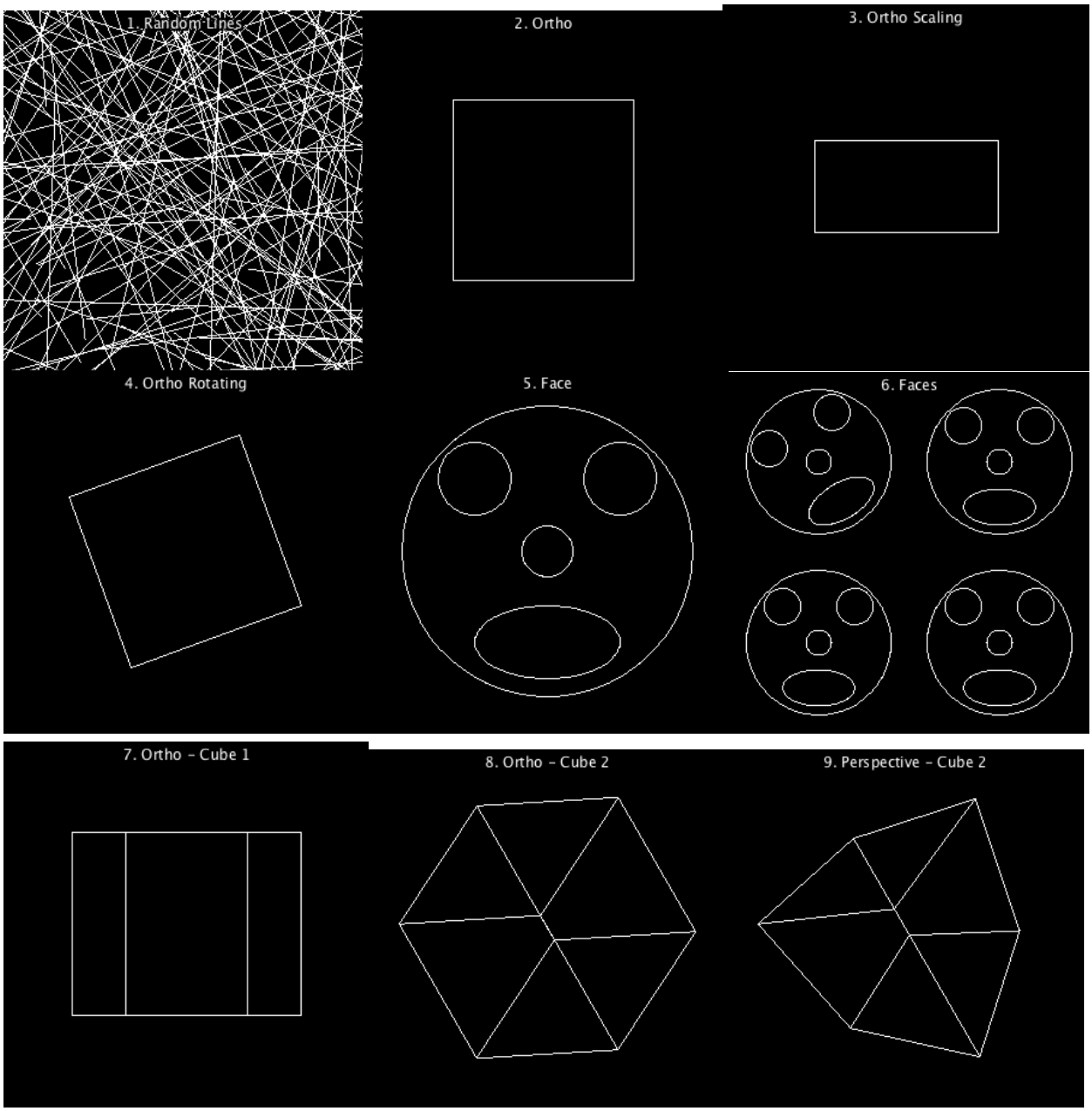
9. **gtBegin(), gtEnd(), gtVertex3f(float x, float y, float z)**

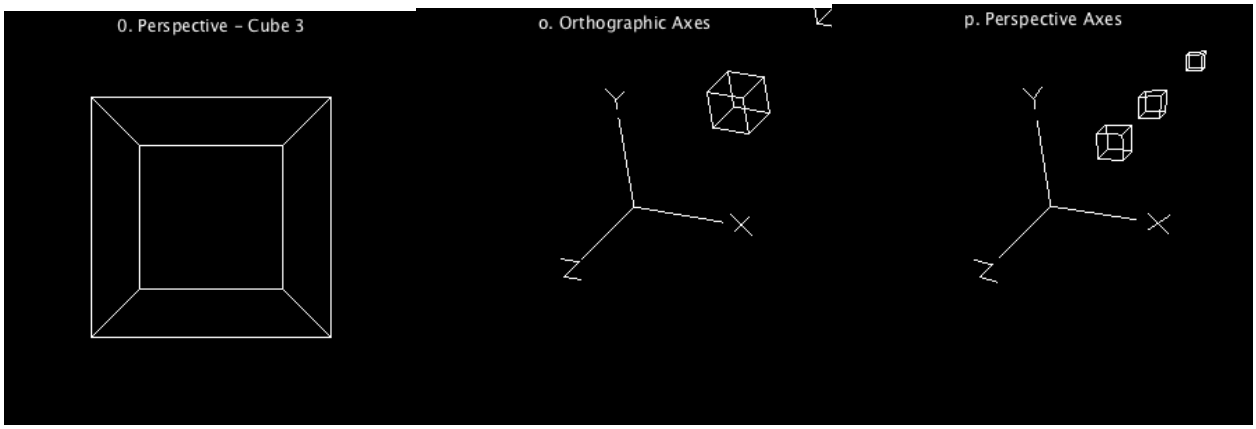
   The gtBegin and gtEnd commands signal the start and end of a list of endpoints for line segments that are to be drawn. Each call to the routine gtVertex3f between these two commands specifies a 3D vertex that is a line endpoint. White lines are drawn between successive odd/even pairs of these vertices. If, for example, the four vertices v1, v2, v3, v4 are given in four sequential gtVertex3f commands then two line segments will be drawn, one between v1 and v2 and another between v3 and v4.

   The vertices of the lines are modified in turn by the current transformation matrix and then by which ever projection was most recently described (gtOrtho). Only one of gtOrtho is in effect at any one time. These projections do not affect the matrix stack and the current transformation matrix! Your gtBegin, gtVertex3f and gtEnd commands must be able to draw any number of lines. You should draw the lines as soon as both vertices are given to you (using gtVertex3f), so there is no need to store more than two vertex positions at any time.

Note: The user command sets producing the images below (0-9, o, p) assume that the positive y axis points upward. This is not true by default in processing. If your images are upside down, you need to flip the y axis.

3.4 Results


1. Random Lines
2. Ortho
3. Ortho Scaling
4. Ortho Rotating
5. Face
6. Faces
7. Ortho - Cube 1
8. Ortho - Cube 2
9. Perspective - Cube 2



3.5 Source code

The provided source code contains all of the methods and framework necessary to draw 2D lines on the processing screen given the x and y location of two points in screen coordinates. Upon running the program, a supplied set of user instructions can be called with the number keys (0-9) and the letters 'o' and 'p'. You will use these test cases to debug and validate your library routines.

Pressing the '1' key will call the first test case to draw lines between random points on the screen. Use this function to see how the provided draw_line function is used. Pressing keys '2' and higher will only work once you have supplied the relevant code. For some of the higher numbered examples, pressing the spacebar will rotate the objects in the scene.

You should modify the source code only in the Matrix and matlib tabs and comment your code (include your name in the header). The source code is written in Processing. Visit "Processing.org/reference/" for more information on built in functions and structure. Please not that you are not allowed to use built in processing functions or Java Stacks to accomplish the tasks listed in the project description. When in doubt, ask.

3.6 Authorship Rules

The code that you turn in entirely your own. You are allowed to talk to other members of the class and to the Professor and the TA about general implementation of the assignment. It is, for example, perfectly fine to discuss how one might organize the data for a matrix stack. It is also fine to seek the help of others for general Processing/Java programming questions. You may not, however, use code that anyone other than yourself has written. Code that is explicitly not allowed includes code taken from the Web, from books, or from any source other than yourself. The only exception to this rule is that you should use the GT Graphics Library routines and the test code that we provide, and the relevant parts of your code from Project 1A. You may NOT use other library routines for matrices and stacks. You should not show your code to other students. Feel free to seek the help of the Professor and the TA's for suggestions about debugging your code.

3.7 Submission

In order to run the source code, it must be in a folder named after the main file. When submitting any assignment, leave it in this folder, compress it and submit via T-square.