Projection and Line Drawing

CS 3451: Project 1B

1 - Objective

The goal of this project is to learn the underlying techniques used in a graphics library that is similar in design to OpenGL. In particular, you will implement transformation, projection, and mapping to the screen of user-provided lines. You will test out your code with the help of the provided routines for drawing a few simple images. You will also create a routine that draws your initials on the screen in perspective.

2 - Deadline

Your project solution should be submitted on T-Square by 11:55PM on Friday, September 15, 2017.

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 copy the matlib.py routines from your previous project into a tab in the source code.

3.3 Project description

As stated in the project objective, you will be implementing a basic graphics library that is similar in style to the popular openGL library. Part of these routines are the ones that you wrote for Project 1A. These routines should act as they did in that earlier project:

gtInitialize()

gtPushMatrix(), gtPopMatrix()

gtTranslate(x, y, z)

gtScale(sx, xy, sz)

gtRotateX(theta), gtRotateY(theta), gtRotateZ(theta)

You will also implement several new routines which allow a user to manipulate and display 3D lines. The provided source code contains empty methods for each of the following routines:

gtOrtho(left, right, bottom, top, near, far)

Specifies that an orthographic projection will be performed on subsequent vertices. The direction of projection is assumed to be along the z-axis. The six 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 "near" and "far" values are ignored for this project. The eye point is assumed to be facing the negative z-axis.

gtPerspective(fov, near, far)

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. The "near" and "far" values are ignored for this project.

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 operation that is applied to the line endpoints, regardless of where those procedure calls appear with respect to the other transformations.

gtBeginShape(), gtEndShape(), gtVertex(x, y, 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 gtVertex between these two commands specifies a 3D vertex that is a line endpoint. Black 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 gtVertex 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 first modified by the current transformation matrix, and then by which ever projection was most recently described (gtOrtho or gtPerspective). Only one of gtOrtho or gtPerspective is in effect at any one time. These projections do not affect the matrix stack and the current transformation matrix! Your gtBegin, gtVertex and gtEnd commands must be able to draw any number of lines. You can draw the lines as soon as both vertices are given to you (using gtVertex), or you can store all of the vertices and draw all of the lines when gtEnd is called. Which way you use is up to you. To draw the lines, use the built-in line() routine in Processing.

Note: The user command sets producing the images below (0-9) 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 Source code

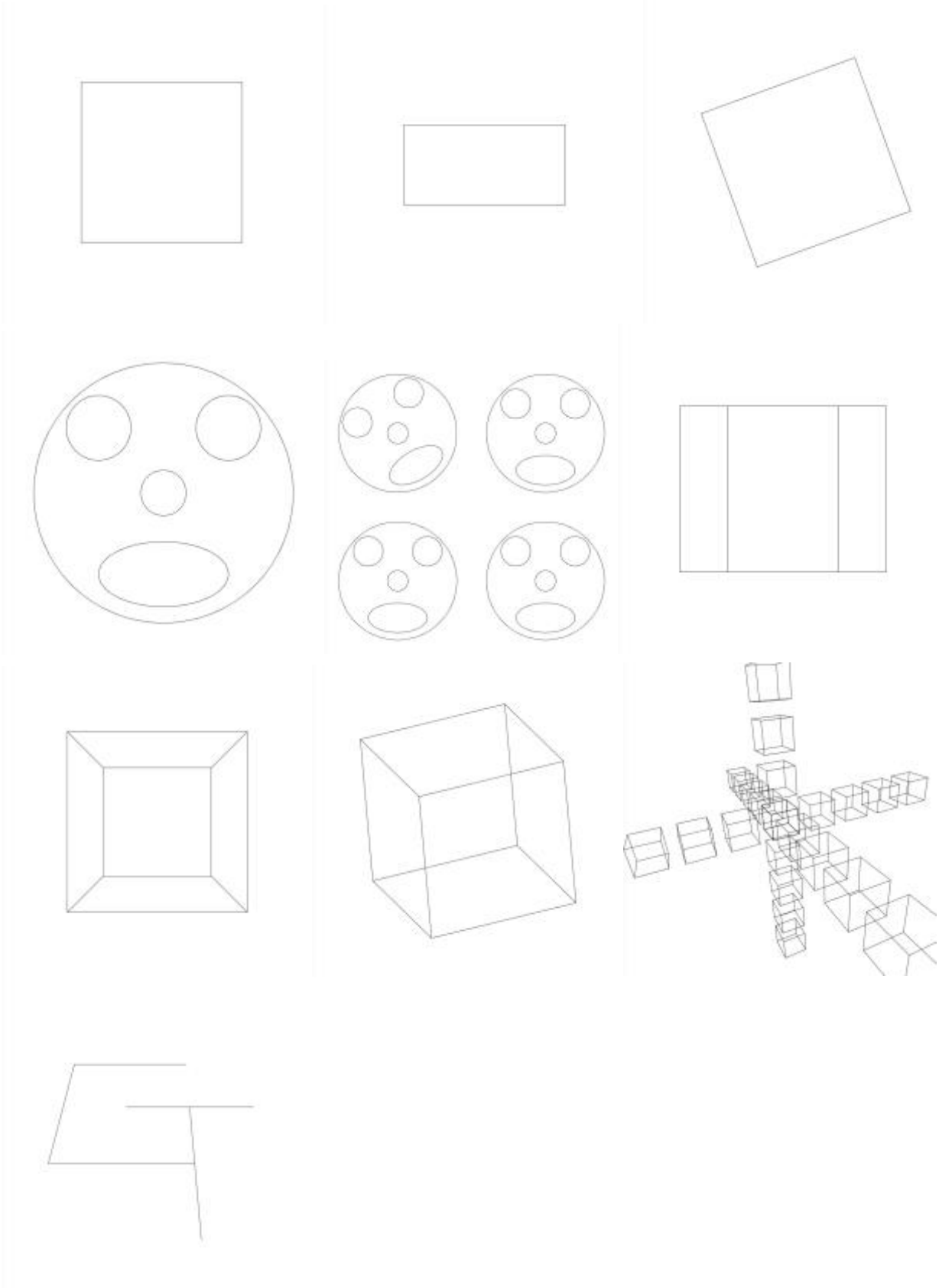
The provided source code contains various routines to draw a few simple images. Upon running the program, these supplied set of routines can be called with the number keys (0-9). You will use these test cases to debug and validate your library routines.

Pressing the '1' key will call the first test case to draw a square on the screen. This test function should be used to test out your gtOrtho routine. Pressing keys '2' and higher will test out more of the routines that you should provide.

The '0' key will run a routine called persp_initials(). In this routine, you should write a set of commands to draw your initials on the screen, using the commands that you have created (gtPerspective(), gtBeginShape(), gtVertex(), and so on). That is, you should draw the first letters of your first and last name. You must use perspective projection, and you also must tilt your initials enough so that it is obvious that they are being seen in perspective. In the example below, Greg Turk has drawn the initials GT. You can be as simple or as elaborate as you like in drawing your initials, so long as these letters can be correctly identified.

You should modify the source code in the drawlib, matlib and initials tabs, and include your name in the header. The source code is written in Python Processing. Visit "py.processing.org/reference/" for more information on built in functions and structure. Please not that you are not allowed to use built in matrix stack and transformation functions in Processing to accomplish the tasks listed in the project description. The only drawing routine that you should use in your code is line(). When in doubt, ask.

3.5 Results



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 you can store the information needed to describe the projection the user has requested. It is also fine to seek the help of others for general Processing/Python 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 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 that are built into Processing. 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. Make sure you include all of the python code that you wrote, as well as the provided file drawing_test.pyde.