Transformation Matrices

CS 3451: Project 1A

1 - Objective

This first project is designed to familiarize you with the basics of creating transformation matrices and a matrix stack. You will also use this project as the basis for the second part of Project 1.

2 - Deadline

Your project solution should be submitted on T-Square by 11:55PM on Friday, January 26.

3 - Process

3.1 Download the base source

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

3.2 Project description

In order to familiarize you with matrix transformations and operations, you will be completing the empty methods that are modeled on the corresponding commands from OpenGL. You will write code to do the following:

1. Write the code for gtInitialize() that initializes the matrix stack. After this command is called, the only matrix on the stack should be the 4x4 identity matrix.

2. Print the current transformation matrix (the top of the stack) to the screen. The command for this is print_ctm().

Example:
```
[1, 0, 0, 0]
[0, 1, 0, 0]
[0, 0, 1, 0]
[0, 0, 0, 1]
```

3. Perform 4x4 matrix/matrix multiplication, to be used in the next step.

4. Create 4x4 scale, translate, and simple rotation matrices, and multiply the matrix on the top of the stack with this newly-created matrix. The names of the commands that you will implement are gtScale, gtTranslate, gtRotateX, gtRotateY, gtRotateZ. For instance, gtScale will cause this change to the top of the matrix stack: new_ctm = old_ctm * scale_matrix.

5. Implement gtPushMatrix and gtPopMatrix. The gtPushMatrix command duplicates the current transformation matrix and places this copy on the top of the matrix stack. The gtPopMatrix command pops the current transformation matrix off the top of the stack. If there is just one matrix on the stack, gtPopMatrix should print an error message.

The provided source code gives you empty methods for each of the operations listed above. The provided code also tests these matrix commands, calling various commands and then printing the current transformation matrix to show the result. These tests are in the routine mat_test(). See below for sample output from this test.

You should modify the source code in any way you see fit and comment your code (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 data structures. Please note that you are not allowed to use built-in

Processing functions to accomplish the tasks listed in the project description. In particular, you cannot use the matrix functions provided by Processing. When in doubt about what code you may use, ask.

3.3 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's about general implementation of the assignment. 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. The only exception is that you should use the source code that we provide for this project. Code that is explicitly **not** allowed includes code taken from the Web, from books, from other assignments or from any source other than yourself. 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.

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, zip it and submit via T-square.

Results from mat_test()

Below are correct results from the code in mat_test(). Your results should be similar, if you have correctly implemented the matrix commands. Note that very small numbers may print out slightly differently than what is listed below, due to variations in numerical precision, and this is okay. Also, depending on how you write your routines, some of your printed numbers might be integers and some might be floats, and this is fine as well.

```
[1, 0, 0, 0]
[0, 1, 0, 0]
[0, 0, 1, 0]
[0, 0, 0, 1]

[1, 0, 0, 3.0]
[0, 1, 0, 2.0]
[0, 0, 1, 1.5]
[0, 0, 0, 1.0]

[2, 0, 0, 0]
[0, 3, 0, 0]
[0, 0, 4, 0]
[0, 0, 0, 1]

[1, 0.0, 0.0, 0]
[0, -4.371138828673793e-08, -1.0, 0]
[0, 1.0, -4.371138828673793e-08, 0]
[0, 0.0, 0.0, 1]
```

```
[0.9659258127212524, 0, -0.258819043636322, 0]
[0.0, 1, 0.0, 0]
[0.258819043636322, 0, 0.9659258127212524, 0]
[0.0, 0, 0.0, 1]

[0.7071067690849304, -0.7071067690849304, 0, 0]
[0.7071067690849304, 0.7071067690849304, 0, 0]
[0.0, 0.0, 1, 0]
[0.0, 0.0, 0, 1]

[1, 0, 0, 0]
[0, 1, 0, 0]
[0, 0, 1, 0]
[0, 0, 0, 1]

[2.0, 0.0, 0.0, 1.5]
[0.0, 2.0, 0.0, 2.5]
[0.0, 0.0, 2.0, 3.5]
[0.0, 0.0, 0.0, 1.0]

[4.0, 0.0, 0.0, 8.0]
[0.0, 2.0, 0.0, -4.0]
[0.0, 0.0, 0.5, 5.0]
[0.0, 0.0, 0.0, 1.0]

[2, 0, 0, 3.0]
[0, 2, 0, 5.0]
[0, 0, 2, 7.0]
[0, 0, 0, 1.0]

[2, 0, 0, 0]
[0, 2, 0, 0]
[0, 0, 2, 0]
[0, 0, 0, 1]

cannot pop the matrix stack
```