CM20219 Assignment One

Due 30 October 2013

1 Outline

This is a fairly simple assignment to get you started especially those that have not used C++ or a GUI before. As well as learning about homogeneous transformation matrices in 2D, the assignment will serve to help you become acquainted with C++, Qt and OpenGl. It is understood that many students are new to C++.

Add functionality to the given program to allow the user to create new polygons. Next modify the code and user interface so that user is able to type in values to a 3x3 homogeneous 2D transformation matrix and apply it to the vertices of the polygon and display the results.

Add the ability to store matrices and multiply them together in a user chosen order, and then apply them to the polygon/shapes. This could be done simply by, allowing the user to store a maximum of say, five stored matrices, to be applied in the order stated.

A bonus mark will be offered if the matrices can be stored in a stack, so the user can identify any number of stored matrices and push them onto the stack. The push operation : push(m), would multiply m with the current stack top and push the result onto the stack. The user should also be able to "pop" the top matrix from the stack and apply it t the vertices of a chosen shape.

An additional bonus mark will be given if drag and drop is modified to operate with the matrix multiplication.

Marks will also be offered for the functionality of your user interface (see Section **??**).

2 Guidance

To allow the user to add a new polygon, start by creating a new class, inheriting the *shape* class and implementing *drawShape()* and *insideZeroCentredShape()*. If you are struggling with the latter, you could approximate the polygon as a circle. You will need to modify the user interface and *GLWidget* class.

To allow the user to add matrices, use a grid of *doubleSpinBoxes*, or otherwise, either in the existing user interface or (preferably) create a new widget. You can access the contents of *doubleSpinBoxes* from within the class (*Window* or your new widget) using the following:

```
this->doubleSpinBoxName->value()
```

To improve your code, use a *QTransform*. For example, include the <*QTransform*> header file, and use the following:

```
QTransform myMat(1.0, 0.0, 0.0,
                 0.0, 1.0, 0.0,
                 0.0, 0.0, 3.0);
```

Replace the parameters of the matrix with values from the user interface.

Initially, try experimenting with matrix multiplications in your *polygon* class by storing the x and y co-ordinates in a *QPointF*, e.g.:

```
QTransform myMat(1.0, 0.0, 0.0,
                 0.0, 1.0, 0.0,
                 0.0, 0.0, 3.0);

QPointF vertex(1, 1);
vertex = myMat*vertex;

glVertex2f(vertex.x(), vertex.y());
```

(QT automatically converts the point into homogeneous co-ordinates an performs the division step after).

Once you have mastered this, you can use the OpenGL function *glMultMatrixf* to perform the multiplication. To do this, you will need to turn the three by three transform into a four by four matrix (the additional row and column applies to the "z" component and is equal to that in the identity matrix). Next send a pointer to the raw matrix data *inside* the class[1]. For example, you can use the following:

```
QTransform myMat(1.0, 0.0, 0.0,
                 0.0, 1.0, 0.0,
                 0.0, 0.0, 3.0);

QMatrix4x4 myMat4(myMat);
glMultMatrixf(myMat4.data());
```

To carry out multiple matrix operations, you can either multiply the *QTransform*s together or call *glMultMatrixf* multiple times. OpenGL performs transformations in "reverse order". Consider the following:

```
glTranslatef(1.0F, 1.0F, 0.0F);
glRotatef(0.5F, 1.0F, 1.0F, 1.0F);

glPushMatrix();
  glMultMatrixf(m1);
  //Group 1 vertices
glPopMatrix();

glPushMatrix();
  glMultMatrixf(m2);
```

[1]The data lies inside a "private" member of the class, but is not copied and can even be changed through the pointer! The idea of doing this ought to raise a few eyebrows. Unlike managed languages like java, C++ is happy let you "shoot yourself in the foot".

```
  //Group 2 vertices
glPopMatrix();

//Group 3 vertices
```

This will result in the following transformations: $\mathbf{x}' = \mathbf{TRM_1x}$ for group 1 vertices, $\mathbf{x}' = \mathbf{TRM_2x}$ for group 2 vertices and $\mathbf{x}' = \mathbf{TRx}$ for group 3 vertices.

To make the transformations only apply to a given shape, use *glMultMatrixf* in *shape::draw*(), after the translations. This will have the affect of somewhat breaking "drag and drop" functionality. Applying the matrix in *paintGL* before any shapes are drawn will apply to every shape, however this will totally break "drag and drop" functionality. A bonus mark is given if you can fix the "drag and drop" functionality.

The *QStack* class can be used to implement a matrix stack.

3 Marking (total 5% of course credit)

- Functionality
  - Polygon drawing (3 marks)
    - One mark for being able to draw any polygon (other than a circle or quadrilateral).
    - One mark for being able to draw a polygon of size N, where N can be selected in the user interface or as a constant in the source code.
    - One mark for being able to select the polygon (a circular assumption is allowed).
  - Basic matrix operations (3 marks)
    - One mark for being able to store and apply a single matrix to one or more shapes.
    - One mark for being able to store at least three matrices and apply a single chosen matrix to one or more shapes.
    - One mark for being able to store multiple matrices and apply a combinations of at least three matrices, in any order, to one or more shapes.
  - Matrix stack (1 mark)—as above but allowing an arbitrarily large sack of matrices to be applied to one or more shapes
  - Drag and drop with matrix operations (1 mark)—awarded if "drag and drop" functionality works with matrix operations
- User interface (4 marks)
  - One mark for being able to add a new polygon from the user interface
  - One mark for being able to add a new polygon from the user interface with a specified number of sides
  - One mark for being able to apply matrix transforms to the selected shape, rather than all shapes
  - One mark for being able to change the properties (colour, radius/edge size and (if applicable) number of sides) of existing shapes by selecting them

- Programming Style (3 marks)