User Interface Engineering: Homework N
The $1 Gesture Recognizer

Instructor: Otmar Hilliges
TAs: Fabrizio Pece and Jie Song
Due Date: 25 March 2015

1 Instructions

- For this homework, you should submit your completed work by committing all your files in the course svn.
- All questions marked with a **Q** require a submission. For the implementation part of the homework please stick to the function headers described and provided literature.

2 The $1 Recognizer (80 pts)

During class, you have learned about the "$1 Gesture Recognizer" ($1 hereafter). $1 is a simple, yet powerful stroke-based gesture recognizer which can be easily implemented without the aid of special libraries or tool-kits. The key idea behind the $1 is pretty simple: given a collection of (normalised) gesture templates, for each new (normalised) test input find its closest match by comparing its shape to the entire collection. **In this homework, you will implement the $1**.

To guide you in implementing the $1, you have two main resources: the skeletal code which we provide you, and the original paper:

Jacob O. Wobbrock, Andrew D. Wilson, and Yang Li. Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes. In *Proceedings of the 20th annual ACM symposium on User interface software and technology (UIST '07)*.

which again is provided with the coursework sheet. Before to start implementing the $1, we *strongly* recommend to read the paper.

In the appendix of the original paper you can find some handy pseudo-code. In both this document and in the skeletal code we use the same notation used in the paper and pseudo-code.

The skeletal code is divided in three main files:

- main.cpp: This is the main file, which handles the mouse/keyboard input, and runs the windows manager (based on OpenGL/GLUT). Before to start implementing anything else, we suggest you make yourself familiar with this file. An helper function is provided to find out which input are accepted.
- Gesture.hpp: This is the core of the program, and implements a single gesture. You are required to fill the relevant stub functions in here.
- OneDollarRecognizer.hpp: This class serves two purposes: it collects a set of gestures, and performs the gesture matching. Two methods for load and save a gesture database are provided. You are required to fill the relevant stub functions in here.

Compile the code. The source files and a CMakeLists.txt are provided.

- Unix: make sure you have cmake installed on your machine. Then open a terminal, navigate to the source directory and type the following:

  mkdir build

  cd build

  cmake ../.

  make

  This will produces the $oneDollar\_bin$ executable.

- Windows:
  - Create an empty Visual Studio solution file, and add the provided sources.
  - Download GLUT from the following location: `https://user.xmission.com/~nate/glut.html`.
  - In the project properties, specify the include directories and linking directories to the GLUT folder you have just downloaded (and relevant subfolder, if any).
  - You also need to link against the glut32.lib provided in the folder you have downloaded.
  - You may need to copy the glut32.dll in the location where your executable is created. Alternatively, you can add the GLUT folder to your system PATH.

2.1 Step 1: Resample (12 pts)

Q 1.1 Implement the RESAMPLE function (and all other relevant functions) from the paper's APPENDIX A. To do so, modify the stub functions:

```
std::vector<Point2D> Gesture::resample(std::vector<Point2D> path, int n)
double Gesture::pathLength(std::vector<Point2D> path)
```

2.2 Step 2: Rotate (12 pts)

Q 1.2 Implement the ROTATE-TO-ZERO function (and all other relevant functions) from the paper's APPENDIX A. To do so, modify the stub functions:

```
void Gesture::rotateToZero(std::vector<Point2D>& path)
void Gesture::rotateBy(std::vector<Point2D>& path, Point2D C, double theta)
Point2D Gesture::centroid(std::vector<Point2D> path)
```

2.3 Step 3: Scale to Square (12 pts)

Q 1.3 Implement the SCALE-TO-SQUARE function (and all other relevant functions) from the paper's APPENDIX A. To do so, modify the stub functions:

```
void Gesture::scaleToSquare(std::vector<Point2D>& path, double nSize)
void Gesture::translateToOrigin(std::vector<Point2D>& path)
Rect Gesture::boundingBox(const std::vector<Point2D>& path)
```

2.4 Step 4: Build gesture Database (12 pts)

Q 1.4 Modify the main.cpp file to add a variety of gesture templates. We suggest to record at least three templates per gesture. You can record multiple templates per gesture by giving to each template exactly the same name. As a start, try the "triangle", "x" and "rectangle" from the Figure 1 in the paper. To do so, modify the stub functions:

```
void Gesture::normaliseGesture(int numResampled, int squareSize)
void OneDollarRecognizer::addGestures(Gesture gest)
```

You can record new gesture templates using the code provided in main.cpp (hint: see myKeyboardFunc).

2.5 Step 5: Match (32 pts)

Q 1.5 Implement the RECOGNIZE function (and all other relevant functions) from the paper's APPENDIX A. To do so, modify the stub functions:

```
Gesture OneDollarRecognizer::recognize(Gesture gest, double* score)
double Gesture::distanceAtBestAngle(Gesture gest)
double Gesture::distanceAtAngle(double nAngle, Gesture gesture)
double Gesture::pathDistance(std::vector<Point2D> path, Gesture gesture)
```

You can record new test gestures using the code provided in main.cpp (hint: see myKeyboardFunc).

3 Questions (20 pts)

- **Q 3.1** (5 pts) Can the $1 recognizer handle 1D gestures, such as vertical or horizontal line? Motivate your answer.
- **Q 3.2** (5 pts) Can the $1 recognizer handle gestures that differ only by orientation? Motivate your answer.

- **Q 3.3** (10 pts) The $1 Recognizer cannot handle multi-stroke gestures. How would you extend the original algorithm to allow for multi-stroke gestures?

4 Extension (10 pts)

- **Q 4.1** The $1 recognizer can handle a large variety of unistroke and can be easily implemented. However, the recognition time can be highly affected by the size of the template database, due to the iterative search method used for the recognition. To alleviate this problem, Yang Li introduced "Protractor" [1], a fast and accurate gesture recogniser that builds upon the $1 recognizer, but uses a closed-form template matching to speed up (and make more robust) the recognition. **Your task is to implement Protractor**. You can finds full details of the method in the literature provided.

[1] Yang Li, *Protractor: A Fast and Accurate Gesture Recognizer*, Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI 2010), 2010.