Assignment 2 – Gestural UI
CAP5937

Due: 9/19/07 11:59pm

As graduate students, you must learn to take a research paper and be able to implement the algorithms described for your own use. The focus of this second assignment is twofold. First, you will have to read a paper (I will give you two options) and use it to implement an important part of this assignment. Second, you will learn how far you can go in recognizing gestures and simple sketches without machine learning. In this assignment you are going to create a simple gestural drawing application where users will be able to sketch out 2D geometric primitives and manipulate them.

Requirements

The main goal of your Gestural UI drawing application is to let users sketch and manipulate simple 2D geometric primitives. The application should contain the following functionality

2D Primitive Functionality

1. Square
2. Rectangle
3. Triangle
4. Ellipse
5. Arrow

2D Primitive Manipulation Functionality

1. Erase primitives using a scribble gesture
2. Support translation of primitives
3. Support scaling of primitives
4. Support grouping primitives together

2 – 4 should be invoked with a gesture of your choice.

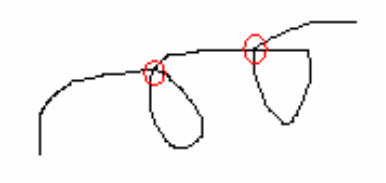
Strategy

First, you are NOT allowed to use the InkAnalyzer of GestureAnalyzer that you worked with in assignment 1.

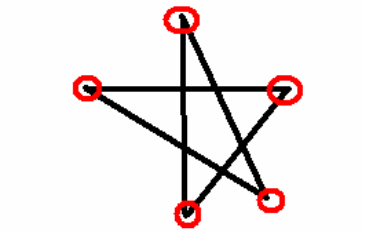
Second, for simplicity you can assume that all primitives are drawn with a single stroke (for extra credit you can try to recognize primitives that have multiple strokes).

Third, you may want to use punctuated gestures (i.e., multi-stroke gestures such as lassoing ink strokes followed by a tap gesture to indicate you want to translate these strokes) to help in distinguishing one gesture from another but only do this if you absolutely need to.

To assist yourself in performing the recognition, you will have to implement two important functions. The first one will find how many self-intersections a stroke has. As an example the following drawing has 2 self intersections



The second will find how many cusps a stroke has. Cusps are places in a stroke with high curvature (note that the starting and ending points of an ink stroke are considered cusps as well). . As an example, the following has 5 cusps.



For self-intersections, you should use simple geometry to perform the calculation.

For cusp detection, you will have the option of implementing the a technique found in either Sezgin's Master's thesis (see Feature Point Detection and Curve Approximation for Early Processing of Free-Hand Sketches) or in Levent Burak Kara's PhD thesis (see Automatic Parsing and Recognition of Hand-Drawn Sketches for Pen-Based Computer Interfaces). Both approaches are fairly straightforward. Of course, you are welcome to come up with your own method.

Keep in mind with both of these functions that you will have to do some preprocessing such as smoothing and filtering.

Other routines that you may want to consider for helping you with recognition is direction of curvature and testing whether a part or all of an ink stroke is a straight line.

Deliverables

You must submit a zip file containing your source and any relevant files needed to compile and run your application. Also include a README file describing what works and what does not in your application, any known bugs, and any problems you encountered. To submit, you can email me your zip file.

Grading

Grading will be loosely based on the following:

80% correct functionality
20% documentation