CS51 – Final project Specification

Convex Hull and Sweep Line Algorithm

-Vinay Mishra (vmishra@g.harvard.edu)

HUID: 10886759

1 OVERVIEW

In this project, I am planning to study two problems related to computational geometry. The two problems are – 'finding the convex hull' and 'determining whether any pair of segments intersects' in two dimension.

The convex hull of given set Q of points is smallest convex polygon P for which each point in Q is either on the boundary of P or in its interior. I would use the Graham's scan algorithm for computing the convex hull of set of n points. This algorithm uses the 'rotational sweep' technique to process vertices in the order of the polar angles they form with a reference vertex.

The second problem will be address with the 'sweep line' algorithm, in which an imaginary vertical sweep line passes through the given set of geometric objects. The intersection of sweep line with the objects provide the relationship among the objects and at pre-computed event points sweep line processes the event point to change sweep line status.

My goal of this project is to implement these two algorithms with a given set of points and visualize the output using OpenGL graphics. Since I am working alone on this project, I would first implement the convex hull algorithm with visualization and then move on to implementing second algorithm if time permits.

2 FEATURE LIST

I will use the algorithms provided in this book. I would use 'Graham's scan' algorithm for finding convex hull and 'sweep line' algorithm for determining the line intersection. Input data points would be in two dimension only. I would use OpenGL graphics to visualize the output. I may use OpenGL animation functionality as well. I would like to do the input check diagnostics as the time permits.

3 TECHNICAL SPECIFICATION

I am planning to modularize the project in the following way:

a) Reading input: The input will be a text file which would specify the data type for points in 2-D e.g. int, float or double. Each line in input file will correspond to one point with x and y coordinates separated by a space. If any discrepancies found in the input file program will exit with an error message.
   For the second algorithm the line will be represented by two immediate points of input file. Format of this input file will remain same, but total number of points should be even number. If any discrepancies found in the input file program will exit with an error message.
b) Creating the input data structure: The input points will be stored in the data structure.

For the convex hull algorithm, the input is a set of points and output is also a set of points which has the equal or less than input size.
For sweeping line algorithm, the input is set of lines and output is set of lines with reduced input size. We make an assumption for the input that no input segment is vertical and no three input segment intersect at a single point.
The user will be prompted to execute either the first algorithm implementation or the second algorithm.
I am planning to provide parameterized classes for Line and Point, parameterized on the data type (e.g. int or float) and space dimension (e.g. 2D or 3D). As an example:

```
template <typename T, size_t N>
class Line
{
public:
        Line()
        {
                mStart = Point<T, N>::Point();
                mEnd = Point<T, N>::Point();
        }
}
```

The methods related to operations on these entities will be provided e.g. for adding one point with another

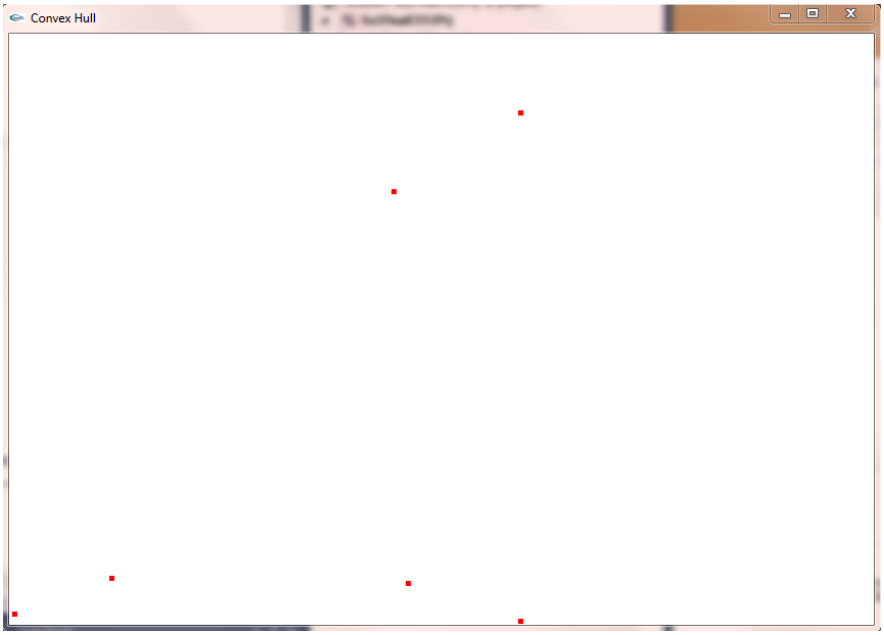
```
void operator+=(const Point& iPoint){

        for (int i = 0; i < N; ++i)

                mPosition[i] += iPoint[i];
}
```

The interface of graphics draw for these points or lines which will be implemented by these geometrical entities.

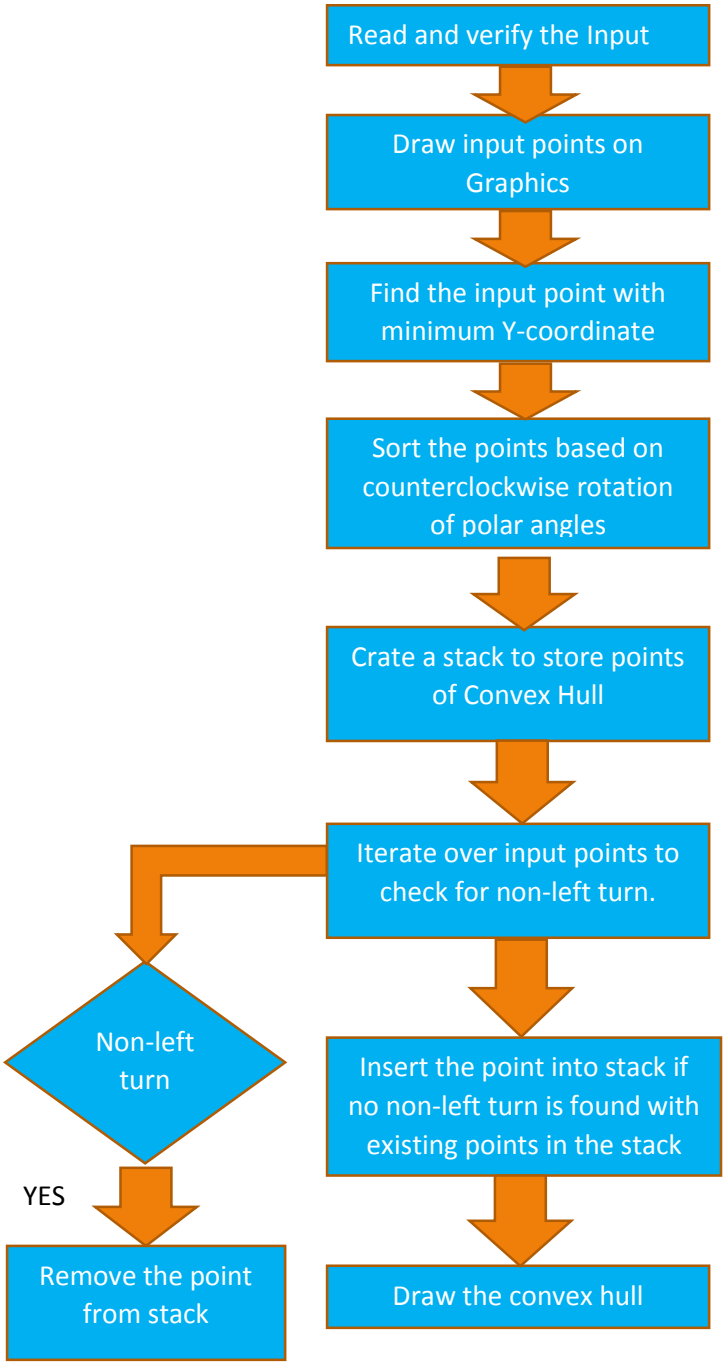
Interface virtual draw();

There will be certain more methods needed e.g. finding angle between two lines, checking if a line is vertical, dot product of two lines etc.

c) Implementing the algorithm: I would use the algorithms provided in the book: Introduction to Algorithms, 3rd edition. The pseudo code is provided on page 1025 and page 1031.
d) Visualizing the output: The OpenGL graphics library will be used. A quick implementation of drawing points after reading inputs from a file provided me the following output:


Convex Hull

4 ALGORITHM DESIGN FOR CONVEX HULL



5 FUNCTION OUTLINE (CONVEX HULL)

a. Input points data format:

2dpointsdata.txt

```
1  float
2  50.0        498.0
3  100.0       100.0
4  20.0        300.0
5  20.0        150.0
6  51.0        151.0
7  82.0        152.0
8  123.0       153.0
9  30.0        5.0
10 40.0        70.0
11 100.0       290.0
```

b. Drawing the input points (using opengl):

```
//draw input points for finding the Convex Hull.
void drawPoints(vector<PointFloat2D>& iListPoints)
```

c. Check for non-left turn of two line segments:

```
bool isNonLeftTurn(const PointFloat2D& pt0, const PointFloat2D& pt1, const PointFloat2D& pt2)
```

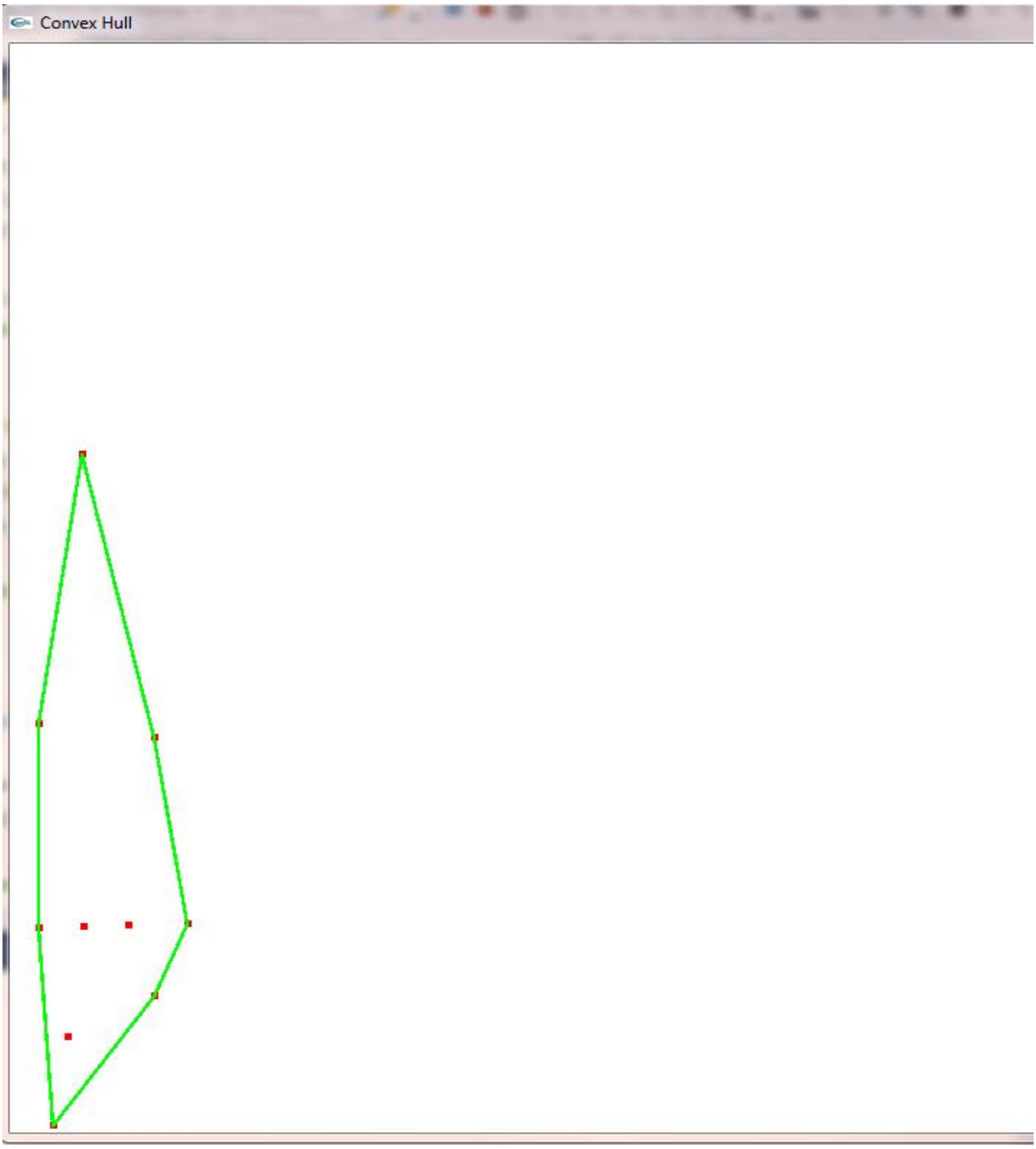
d.
```
void nextToStackTop(stack<PointFloat2D>& iStack, PointFloat2D& oPoint)
```

e.
```
//compare the direction of two line segments based on cross product of two vectors, origin of both vectors is at (0,0)
bool compareDirection(const PointFloat2D& vect1, const PointFloat2D& vect2)
```

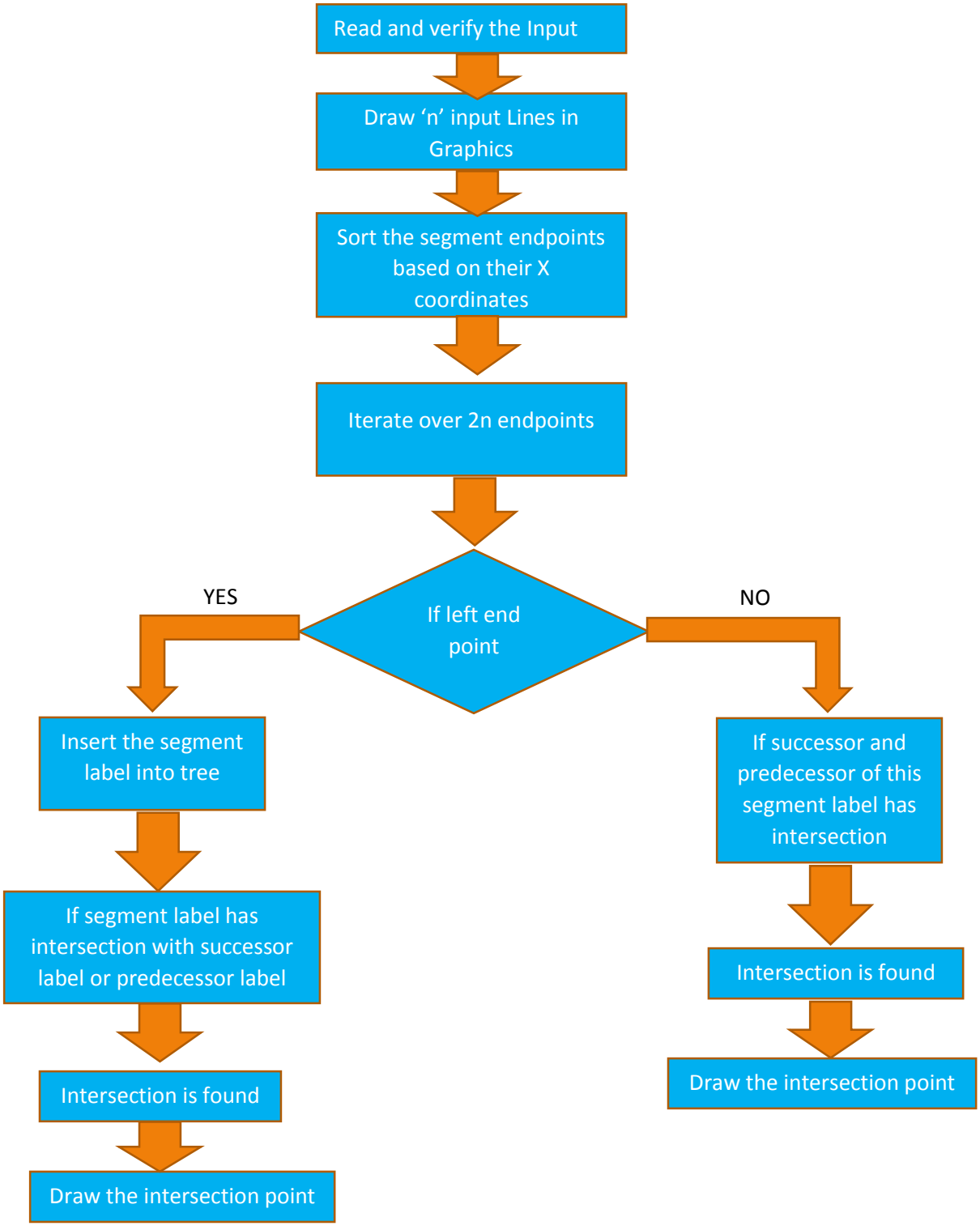
f.
```
//Minimum Y element, if there is a tie then lesser X value element is before the other one.
bool compare_MinY(const PointFloat2D& a, const PointFloat2D& b)
```

g.
```
//Find and draw convex hull points.
void findAndDrawConvexHull(vector<PointFloat2D>& iListPoints)
```

For the given input points, this is the output as convex hull:


Convex Hull

6 Algorithm Design for Sweep Line to Find Any Line Segment Intersection



7 FUNCTION OUTLINE (SWEEP LINE ALGORITHM)

a. Drawing the input lines (using opengl):

```
//draw input points for finding the Convex Hull.
void drawLines(vector<PointFloat2D>& iListPoints)
```

b. `//create Input Line segments`

```
void createLineSegments(vector<PointFloat2D>& iListInputPointsFloat2D,
vector<LineFloat2D>& oLineVec)
```

c. `//Compare the X coordinate of points.`

```
bool compare_MinX(const PointFloat2D& a, const PointFloat2D& b)
```

d. `//compare the line segments based on the Y-coordinate of the end points.`

```
bool compare_MaxY(const LineFloat2D& line1, const LineFloat2D& line2)
```

e. `void findIntersection()`