**Motivation:**

A convex is a shape in which no line segment between two points within the shape can cross the shapes boundary. The convex hull algorithm is then an algorithm over a set of points which calculates the smallest convex shape which can be used to contain all the points.

Advantages of computing convex hull:

-Aid visualization of data

-reduce polygons to convex polygons for faster computation (i.e. physics simulation)

-generate shape from raw data for use in calculations (i.e 3d scan into 3d object)

**Problems to be solved:**

To calculate the convex hull a region must be calculated that contains each point provided by a complex point object which is passed into the convex hull function.

A number of algorithms will be implemented for solving the convex hull problem. These algorithms will be compared on the basis of two parameters- accuracy and performance. While accuracy will compare whether all the points are correctly bounded, performance will measure time as well as the scalability(10^k, 0<=k<=9) of the algorithms.

**Data types (classes) and operations (methods) that will be provided in interface(s)**

Classes:

* Convex Hull Class //to compute the convex hull for the given input
* Region Class //to return a region object after calculating correct set of points

Methods are defined in detail below.

**Semantics description for each operation**

**ConvexHull Class:**

Region computeConvexHull(complexPoint x) {}

Region computeConvexHull(Region x) {}

orientation( , , ,) {} // Method to implement orientation needed for alg like jarvis and graham scan.

**Point Class**:

*Methods*:

BigRational getX() // returns a big number of the x coordinate

BigRational getY() // returns a big number of the y coordinate

**Complex Point Class**: for this class the manner in which the points are returned is crucial for the algorithm as to what ordering the points may have, an ordering could be used to speed up the computation

*Methods*:

*vector<Point>* getPoints() // returns all the points of the Complex Point object (can be unordered)

LargeNumber Class: this class is used to hold all calculation results and point information

*Methods:*

BigRational operator-(BigRational in)

BigRational  operator+(BigRational  in)

bool operator>(BigRational  in)

bool operator<(BigRational  in)

**Region Class**: the convex hull computation will return a Region Object after calculating the correct set of points

*Methods*:

Region(vector<Segment>)

**Constraints**: this will contain the various constraints that exist while defining a spatial region. This will include constraints such as self-intersecting lines, disconnected boundary, etc. Once the convex hull is computed using the said algorithm, this will be tested against these constraints to verify that it satisfies the conditions for spatial regions.

*Methods*:

bool checkIntersection(complexRegion R) //check for self intersecting lines

bool checkBoundary(complexRegion R) //checks for continuous connected boundary

**Interface requirements from other groups.**

For the interface requirements these classes and methods must be defined in a agreed upon header file. Such a file could be named “Topic#.h” to simplify which header file includes what. This will include the class for ConvexHull which will allow a user to compute convex hull using any of the algorithms.

From the other groups, we would need the following interfaces:

* **BigRational.h** //this should contain the implementation of big number which can be used for mathematical calculations while computing convex hull
* **ComplexObject.h** //this should contain the implementation for complex points, lines and regions