DSA – PROJECT

DR. SHALINI L

17-05-2020

Epidemic Confinement using Convex Hull

To find minimum possible danger area which we need to confine to avoid further outbreak of disease with the help of CONVEX HULL.

Bhavya Goel

19BCE0378

DSA – Project

EPIDEMIC CONFINEMENT USING CONVEX HULL

INTRODUCTION

During the 1770s, smallpox killed at least 30% of the West Coast Native Americans. The smallpox epidemic of 1780-1782 brought devastation and drastic depopulation among the plains Indians.

And year 2020, we encountered another devastating pandemic namely COVID-19 by the deadly virus Coronavirus, this virus is also as capable as smallpox in devastation and drastic depopulation across the globe. This strain of virus is even more contagious than smallpox. It spreads with cough or sneeze as droplets from mouth and nose to other people.

So, in order to tackle such an epidemic, we plot a graph with marking of most recent outburst of the disease, which we assume to be the hospitals to mark record number of cases. By using the fact that approx. 5 km of the outburst is the saturated area by virus. So, if the virus spreads uniformly in all directions, so it will form a circle which we will say DANGER AREA (which we will represent by user specified count of coordinates on these circles, all of which will be at an angle of 360 / (user specified count) degree with each other).

Suppose a place where happens to be recorded many such cases of disease, in which danger area may or may not intersect the danger area caused by any other outbreak. So, we will end up having an intersection of circles i.e. bunch of points, those intersection points are points that engulf both the circles.

We will find the minimum number of points which will cover the whole bunch of points inside it, when joined with straight line. And those minimum points will not be anything but the boundary line of the danger area.

So, in this project we will be finding out the boundary line of the final intersected DANGER AREA with the help of CONVEX HULL.

Algorithms used: -

1. Jarvis March Algorithm
2. Graham Scan Algorithm

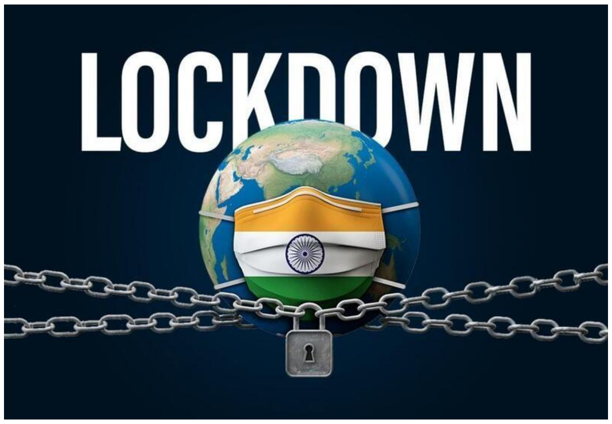
Key Features:

- Marking the area to be kept under strict surveillance, to prevent further outbreak
- Graphical representation of the resulting area
- Calculating the area to be kept under surveillance

Motivation

Learning about convex hull, about how it is being used in various sectors like collision avoidance, and many more image processing algorithms. I realized that this algorithm can be used for the betterment of our society.

As we are already suffering from a global pandemic, it is becoming quite difficult to recognize proper confinement areas, and it is proven that confinement is the only known way at present to control this global pandemic.



As COVID – positive patients are being quarantined in hospitals, their own houses or societies, so, we can figure out the coordinates of such places where chances of being infected by this deadly virus are more. We can also define the vicinity of confinement areas.

So, we can restrict people from entering these confinement areas only when we are aware of the total vicinity of these multiple hotspots, so in order to accomplish this task I used convex hull as my foremost choice.

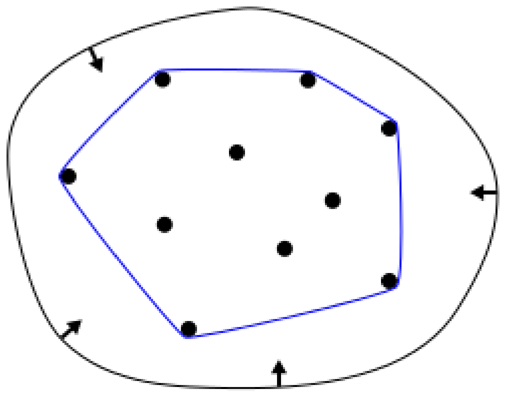
Now, we can feed the multiple coordinates of these hotspot areas, the vicinity of virus spread and we can easily confine the areas to prevent further contamination.

"It's temporary, but it's a tornado-like headwind, so it's going to be powerful for a period of time."

Details

What is Convex Hull?

The convex hull of a set of points is defined as the smallest convex polygon, that encloses all of the points in the set. Convex means that the polygon has no corner that are bent inwards.



A useful way to think about the convex hull is the rubber band analogy. Suppose the points in the set were nails, sticking out of a flat surface. Imagine now, what would happen if you took a rubber band and stretch it around the nails. Trying to contract back to its original length, the rubber band would enclose the nails, touching the ones that stick out the furthest from the center.

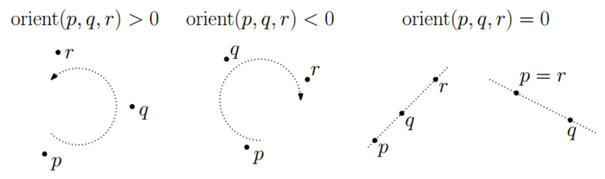
Finding the Convex Hull

Looking at the set of points, human intuition lets us quickly figure out which points are likely to touch the convex hull, and which ones will be closer to the center and thus away from the convex hull. But translating this intuition into code takes a bit of work.

So, here I would be using Graham Scan and Jarvis March algorithms to find the convex hull.

Graham's Algorithm [0(nlogn)]

The idea of how points are oriented plays a key role in understanding graham's algorithm.



Thus, finding out whether the points p, q, r are making a left turn or a right turn is a simple calculation of determinant.

Algorithm:

1. Find the bottom-most point by comparing y coordinate of all points. If there are two points with same y value, then the point with smaller x coordinate value is considered. Let the bottom-most point be P0. Put P0 at first position in output hull.
2. Consider the remaining n-1 points and sort them by polar angle in counterclockwise order around points[0]. If the polar angle of the two points is the same, the put the nearest point first.
3. After sorting, check if two or more points have the same angle. If two more points have the same angle, then remove all same angle points except the point farthest from P0. Let the size of the new array be m.
4. If m is less than 3, return (Convex Hull not possible)
5. Create an empty stack 'boundary' and push points[0], points[1] and points[2] to boundary.
6. Process remaining m-3 points one by one. Do following for every point 'points[i]'
   a) Keep removing points from stack while orientation of following 3 points is not counterclockwise (or the don't make a left turn)
      - Point next to top in stack
      - Point at the top of stack
      - Points[i]

   b) Push points[i] to boundary.
7. Push content of boundary to vector graham-scan.

$$\text{Orient}(p,q,r) = \det \begin{pmatrix} 1 & p_x & p_y \\ 1 & q_x & q_y \\ 1 & r_x & r_y \end{pmatrix}.$$

The above algorithm is divided into two phases:

a. **Phase 1 (Sort Points) :** We first find the bottom-most point. The idea is to pre-process points be sorting them with respect to the bottom-most point. Once the points are sorted, they form a simple closed path. The idea is the use the orientation to compare angles without actually computing them (compare() function)
b. **Phase 2 (Accept or Reject points) :** Once we have the closed path, the next step is to traverse the path and remove concave points on this path. Again, orientation helps here decide which point to remove and which to keep. The first two points in sorted array are always part of Convex Hull. For remaining points, we keep track of recent three points, and find the angle formed by them. Let the three points be prev(p), curr(c) and next(n). if the orientation of these points (considering them in same order) is not counterclockwise, we discard c, otherwise we keep it.

Pseudocode:

```
let points be the list of points
let stack = empty_stack()

find the lowest y-coordinate and leftmost point, called P0
sort points by polar angle with P0, if several points have the same polar angle
then only keep the farthest

for point in points:
    # pop the last point from the stack if we turn clockwise to reach this
point
    while count stack > 1 and ccw(next_to_top(stack), top(stack), point) < 0:
        pop stack
    push point to stack
end
```

Time Complexity:

Let n be the number of input points. The algorithm takes O(nLogn) time if we use a O(nLogn) sorting algorithm.

The first step (finding the bottom-most point) takes O(n) time. The second step (sorting points) takes O(nLogn) time. The third step (every element is pushed or popped once a time) takes O(n) time. Assuming the stack operations take O(1) time. Overall complexity is O(n) + O(nLogn) + O(n) which is **O(nLogn).**

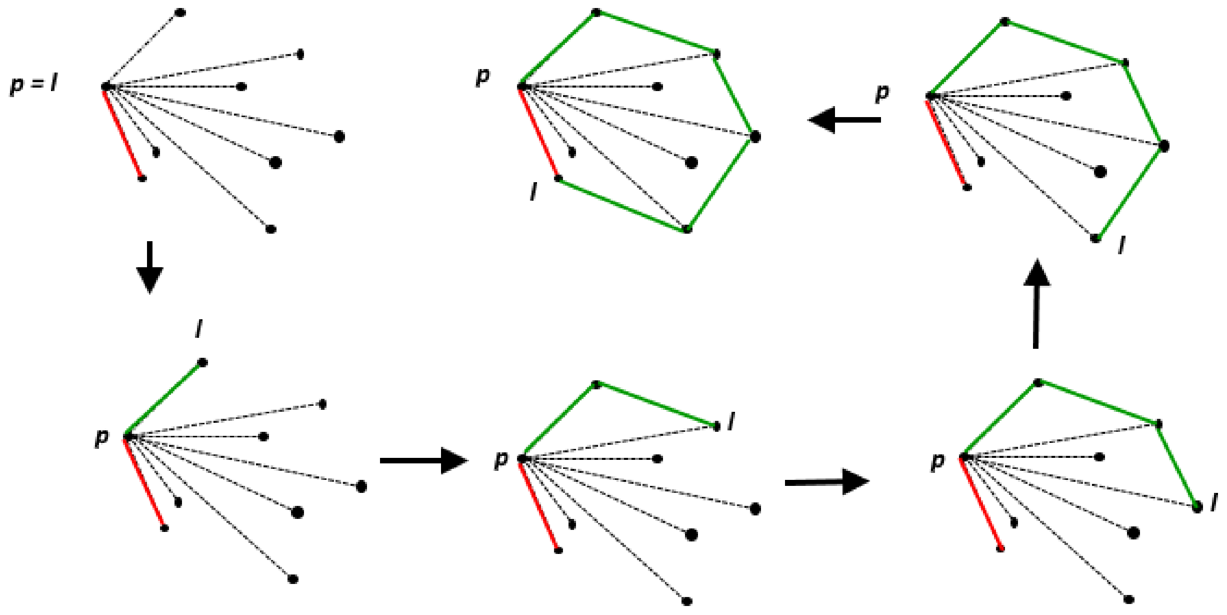
References:

https://www.tutorialspoint.com/Graham-Scan-Algorithm

https://en.wikipedia.org/wiki/Graham_scan

Jarvis March Algorithm [O(n^2)]

The idea or Jarvis March algorithm is that, we start from the leftmost point (or point with minimum x coordinate value) and we keep wrapping points in counterclockwise direction. Given a point p as current point, to find the next point we use orientation(). Next point is selected as the point that beats all other points at counterclockwise orientation, ie next point is q if for any other r, we have “orientation(p,q,r) = counterclockwise”.



The execution of jarvis's March

Algorithm:

1. Initialize p as leftmost point.
2. Do following while we don’t come back to the first (or leftmost) point.
    a) The next point q is such that the triplet (p,q,r) is counterclockwise for any other point r.
    b) Next[p] = q (store q as next of p in output convex hull)
    c) P = q (set p as q for next iteration)

Pseudocode

```
algorithm jarvis(S) is
    // S is the set of points
    // P will be the set of points which form the convex hull. Final set size
is i.
    pointOnHull = leftmost point in S // which is guaranteed to be part of the
CH(S)
    i := 0
    repeat
        P[i] := pointOnHull
        endpoint := S[0]      // initial endpoint for a candidate edge on the
hull
        for j from 0 to |S| do
            // endpoint == pointOnHull is a rare case and can happen only when
j == 1 and a better endpoint has not yet been set for the loop
            if (endpoint == pointOnHull) or (S[j] is on left of line from P[i]
to endpoint) then
                endpoint := S[j]   // found greater left turn, update endpoint
        i := i + 1
        pointOnHull = endpoint
    until endpoint = P[0]      // wrapped around to first hull point
```

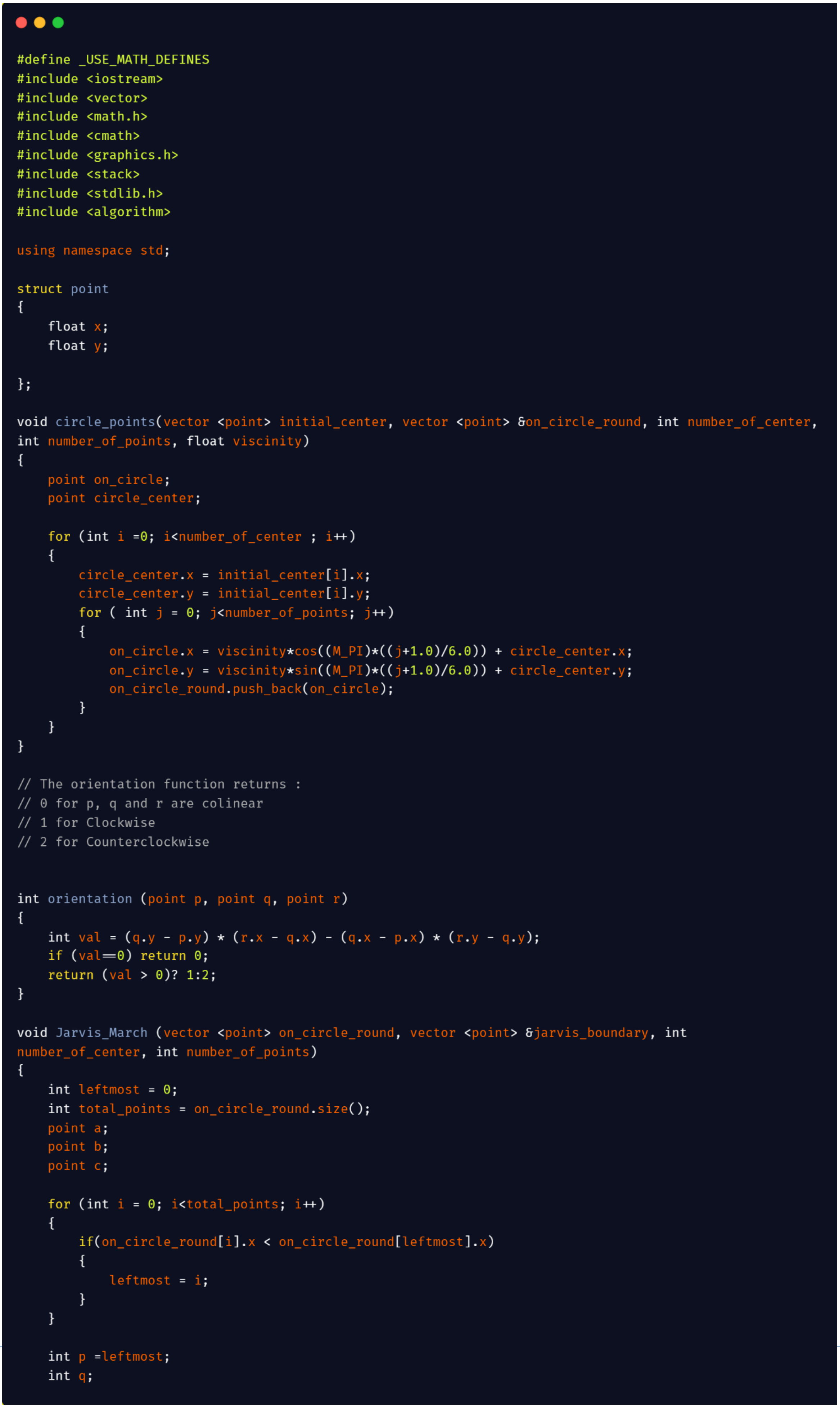
Time Complexity:

For every point in hull we examine all other points to determine the next point. Time Complexity is O(m*n) where n is the number of input points and m is number of output or hull points (m<=n). in worst case, time complexity is O(n^2) when m=n.

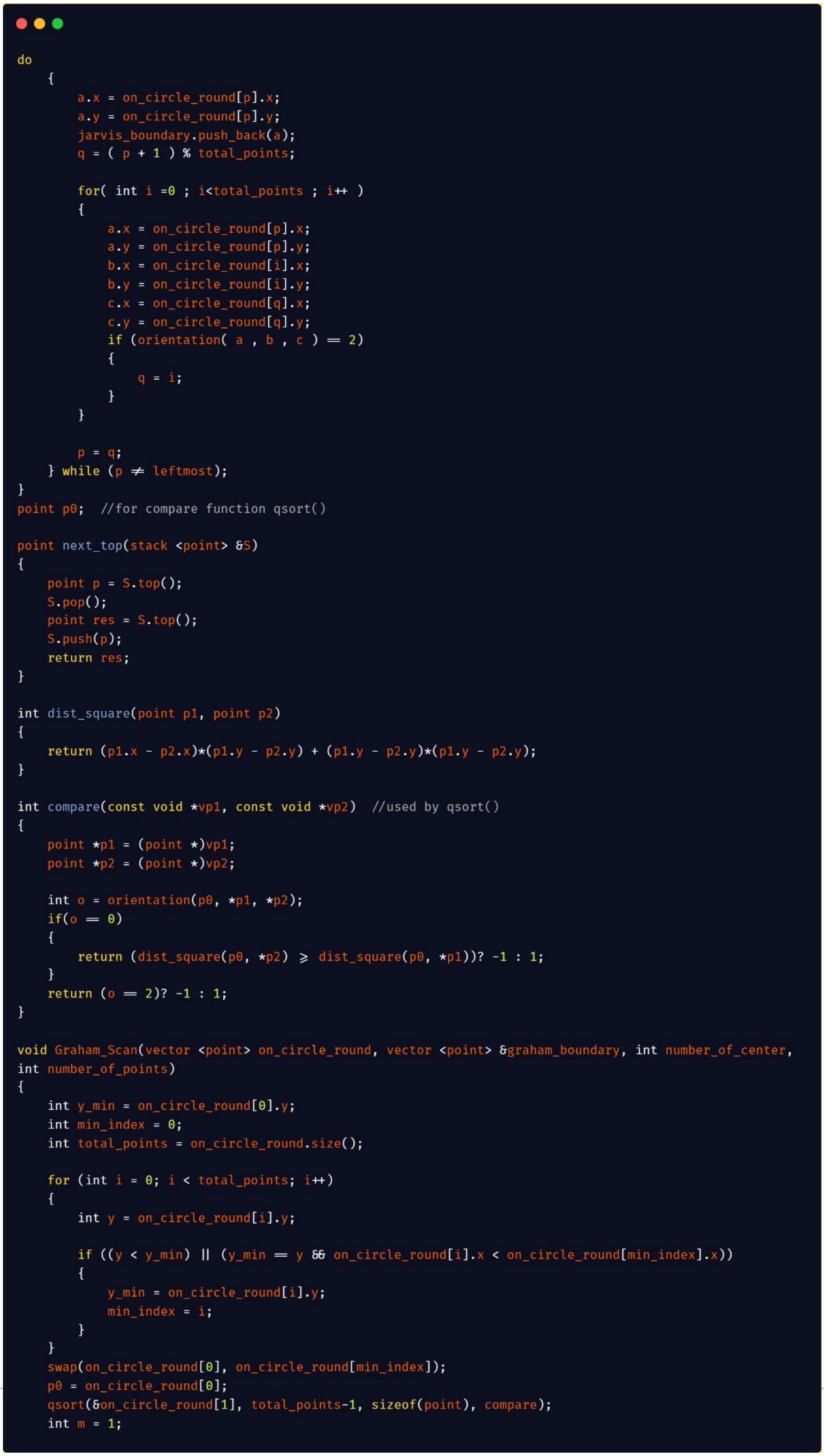
References:

https://www.tutorialspoint.com/Jarvis-March-Algorithm

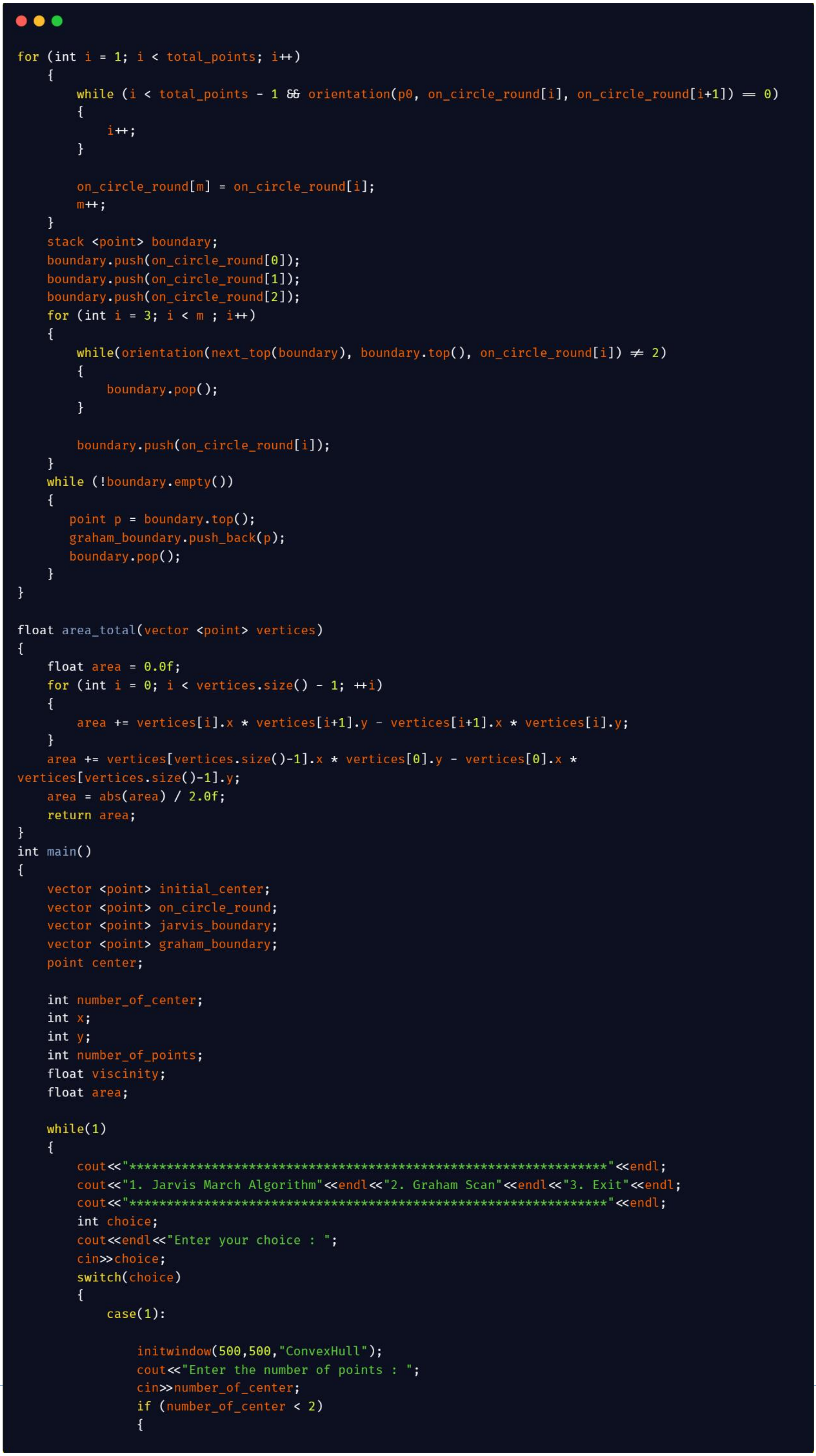
CODE SNIPPETS



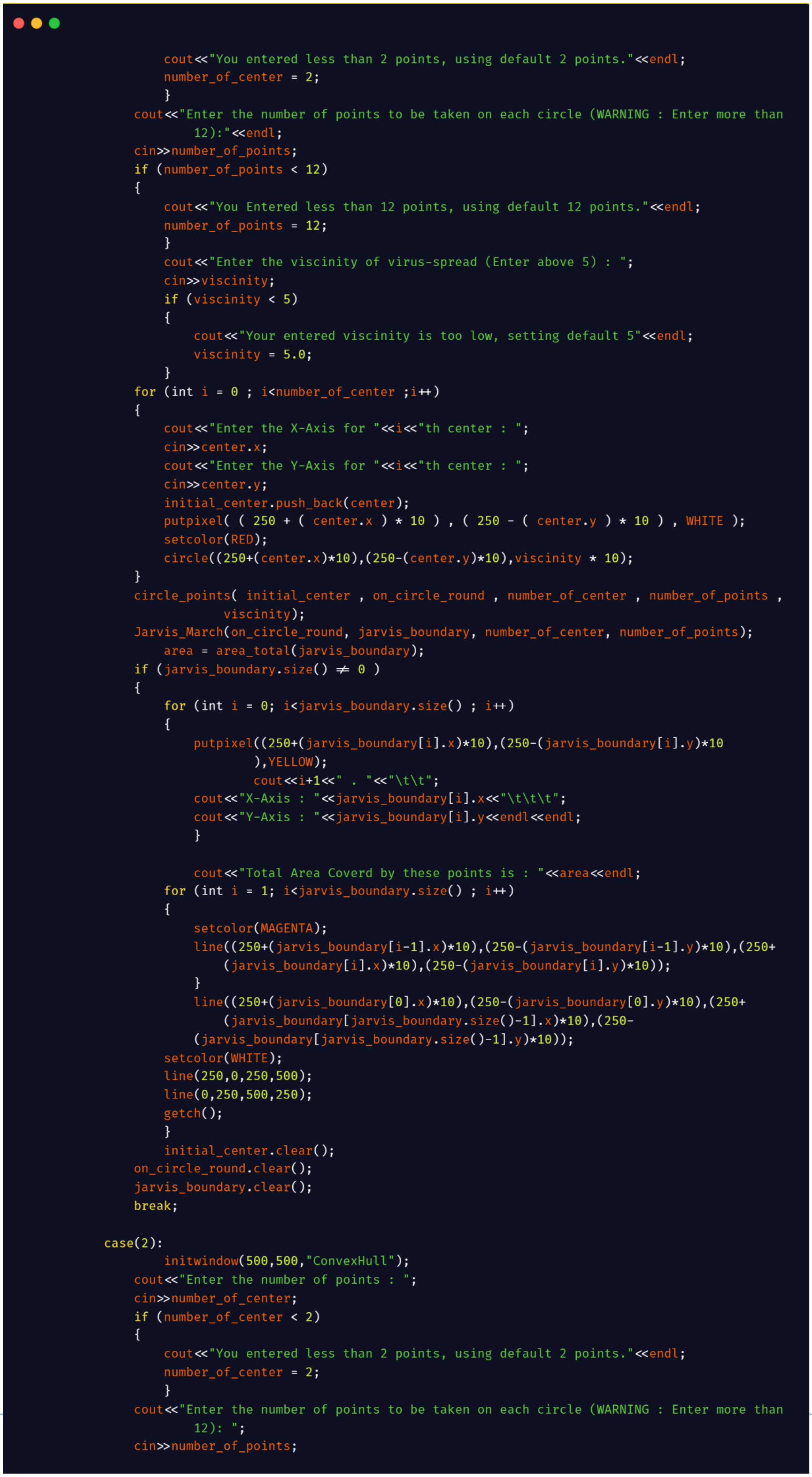
```cpp
#define _USE_MATH_DEFINES
#include <iostream>
#include <vector>
#include <math.h>
#include <cmath>
#include <graphics.h>
#include <stack>
#include <stdlib.h>
#include <algorithm>

using namespace std;

struct point
{
    float x;
    float y;

};

void circle_points(vector <point> initial_center, vector <point> &on_circle_round, int number_of_center,
int number_of_points, float viscinity)
{
    point on_circle;
    point circle_center;

    for (int i =0; i<number_of_center ; i++)
    {
        circle_center.x = initial_center[i].x;
        circle_center.y = initial_center[i].y;
        for ( int j = 0; j<number_of_points; j++)
        {
            on_circle.x = viscinity*cos((M_PI)*((j+1.0)/6.0)) + circle_center.x;
            on_circle.y = viscinity*sin((M_PI)*((j+1.0)/6.0)) + circle_center.y;
            on_circle_round.push_back(on_circle);
        }
    }
}

// The orientation function returns :
// 0 for p, q and r are colinear
// 1 for Clockwise
// 2 for Counterclockwise


int orientation (point p, point q, point r)
{
    int val = (q.y - p.y) * (r.x - q.x) - (q.x - p.x) * (r.y - q.y);
    if (val==0) return 0;
    return (val > 0)? 1:2;
}

void Jarvis_March (vector <point> on_circle_round, vector <point> &jarvis_boundary, int
number_of_center, int number_of_points)
{
    int leftmost = 0;
    int total_points = on_circle_round.size();
    point a;
    point b;
    point c;

    for (int i = 0; i<total_points; i++)
    {
        if(on_circle_round[i].x < on_circle_round[leftmost].x)
        {
            leftmost = i;
        }
    }

    int p =leftmost;
    int q;
```



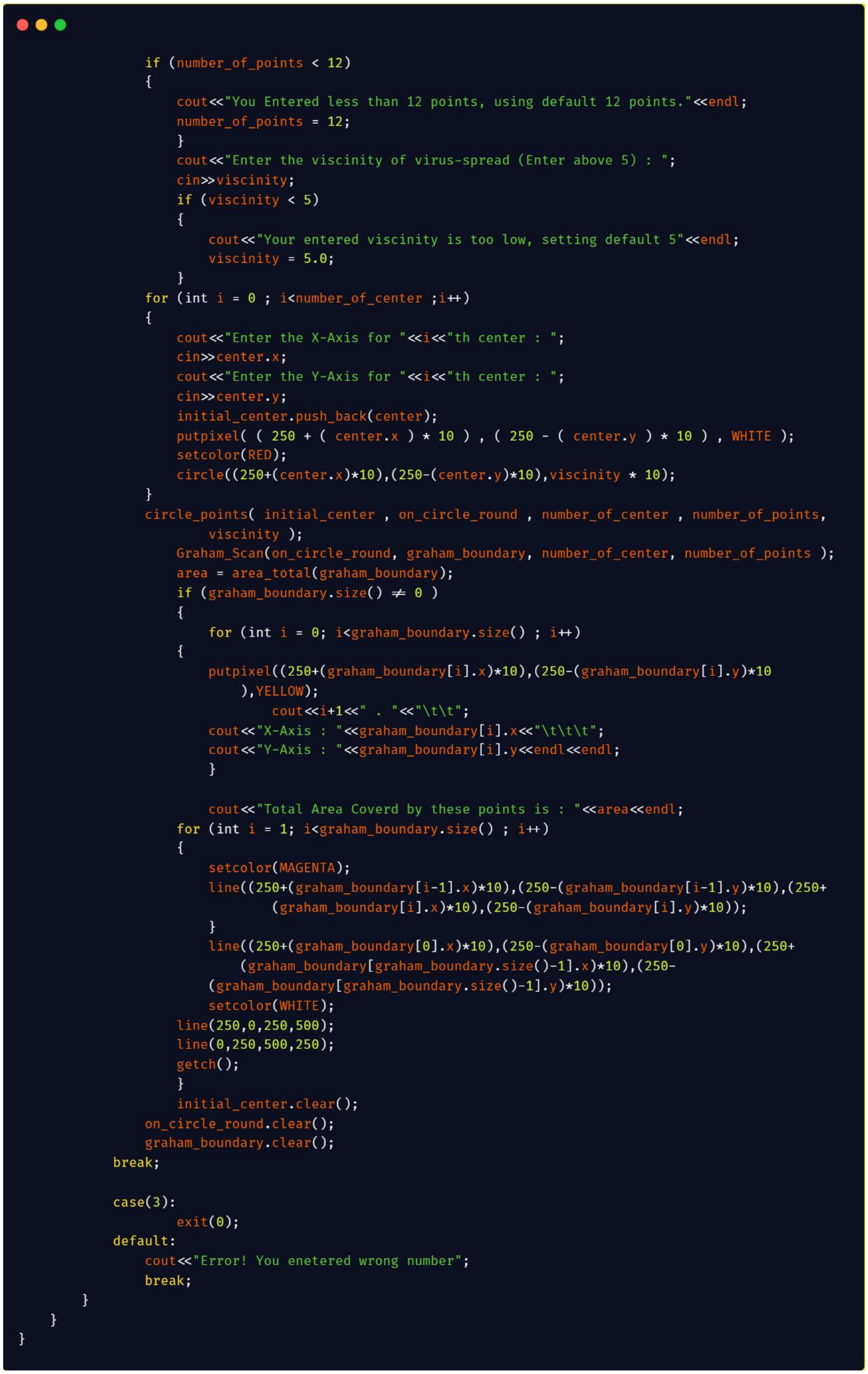
```
do
    {
        a.x = on_circle_round[p].x;
        a.y = on_circle_round[p].y;
        jarvis_boundary.push_back(a);
        q = ( p + 1 ) % total_points;

        for( int i =0 ; i<total_points ; i++ )
        {
            a.x = on_circle_round[p].x;
            a.y = on_circle_round[p].y;
            b.x = on_circle_round[i].x;
            b.y = on_circle_round[i].y;
            c.x = on_circle_round[q].x;
            c.y = on_circle_round[q].y;
            if (orientation( a , b , c ) == 2)
            {
                q = i;
            }
        }

        p = q;
    } while (p != leftmost);
}
point p0;  //for compare function qsort()

point next_top(stack <point> &S)
{
    point p = S.top();
    S.pop();
    point res = S.top();
    S.push(p);
    return res;
}

int dist_square(point p1, point p2)
{
    return (p1.x - p2.x)*(p1.y - p2.y) + (p1.y - p2.y)*(p1.y - p2.y);
}

int compare(const void *vp1, const void *vp2)  //used by qsort()
{
    point *p1 = (point *)vp1;
    point *p2 = (point *)vp2;

    int o = orientation(p0, *p1, *p2);
    if(o == 0)
    {
        return (dist_square(p0, *p2) >= dist_square(p0, *p1))? -1 : 1;
    }
    return (o == 2)? -1 : 1;
}

void Graham_Scan(vector <point> on_circle_round, vector <point> &graham_boundary, int number_of_center,
int number_of_points)
{
    int y_min = on_circle_round[0].y;
    int min_index = 0;
    int total_points = on_circle_round.size();

    for (int i = 0; i < total_points; i++)
    {
        int y = on_circle_round[i].y;

        if ((y < y_min) || (y_min == y && on_circle_round[i].x < on_circle_round[min_index].x))
        {
            y_min = on_circle_round[i].y;
            min_index = i;
        }
    }
    swap(on_circle_round[0], on_circle_round[min_index]);
    p0 = on_circle_round[0];
    qsort(&on_circle_round[1], total_points-1, sizeof(point), compare);
    int m = 1;
```



```
for (int i = 1; i < total_points; i++)
    {
        while (i < total_points - 1 && orientation(p0, on_circle_round[i], on_circle_round[i+1]) == 0)
        {
            i++;
        }

        on_circle_round[m] = on_circle_round[i];
        m++;
    }
    stack <point> boundary;
    boundary.push(on_circle_round[0]);
    boundary.push(on_circle_round[1]);
    boundary.push(on_circle_round[2]);
    for (int i = 3; i < m ; i++)
    {
        while(orientation(next_top(boundary), boundary.top(), on_circle_round[i]) != 2)
        {
            boundary.pop();
        }

        boundary.push(on_circle_round[i]);
    }
    while (!boundary.empty())
    {
       point p = boundary.top();
       graham_boundary.push_back(p);
       boundary.pop();
    }
}

float area_total(vector <point> vertices)
{
    float area = 0.0f;
    for (int i = 0; i < vertices.size() - 1; ++i)
    {
        area += vertices[i].x * vertices[i+1].y - vertices[i+1].x * vertices[i].y;
    }
    area += vertices[vertices.size()-1].x * vertices[0].y - vertices[0].x *
vertices[vertices.size()-1].y;
    area = abs(area) / 2.0f;
    return area;
}
int main()
{
    vector <point> initial_center;
    vector <point> on_circle_round;
    vector <point> jarvis_boundary;
    vector <point> graham_boundary;
    point center;

    int number_of_center;
    int x;
    int y;
    int number_of_points;
    float viscinity;
    float area;

    while(1)
    {
        cout<<"*****************************************************************"<<endl;
        cout<<"1. Jarvis March Algorithm"<<endl<<"2. Graham Scan"<<endl<<"3. Exit"<<endl;
        cout<<"*****************************************************************"<<endl;
        int choice;
        cout<<endl<<"Enter your choice : ";
        cin>>choice;
        switch(choice)
        {
            case(1):

                initwindow(500,500,"ConvexHull");
                cout<<"Enter the number of points : ";
                cin>>number_of_center;
                if (number_of_center < 2)
                {
```

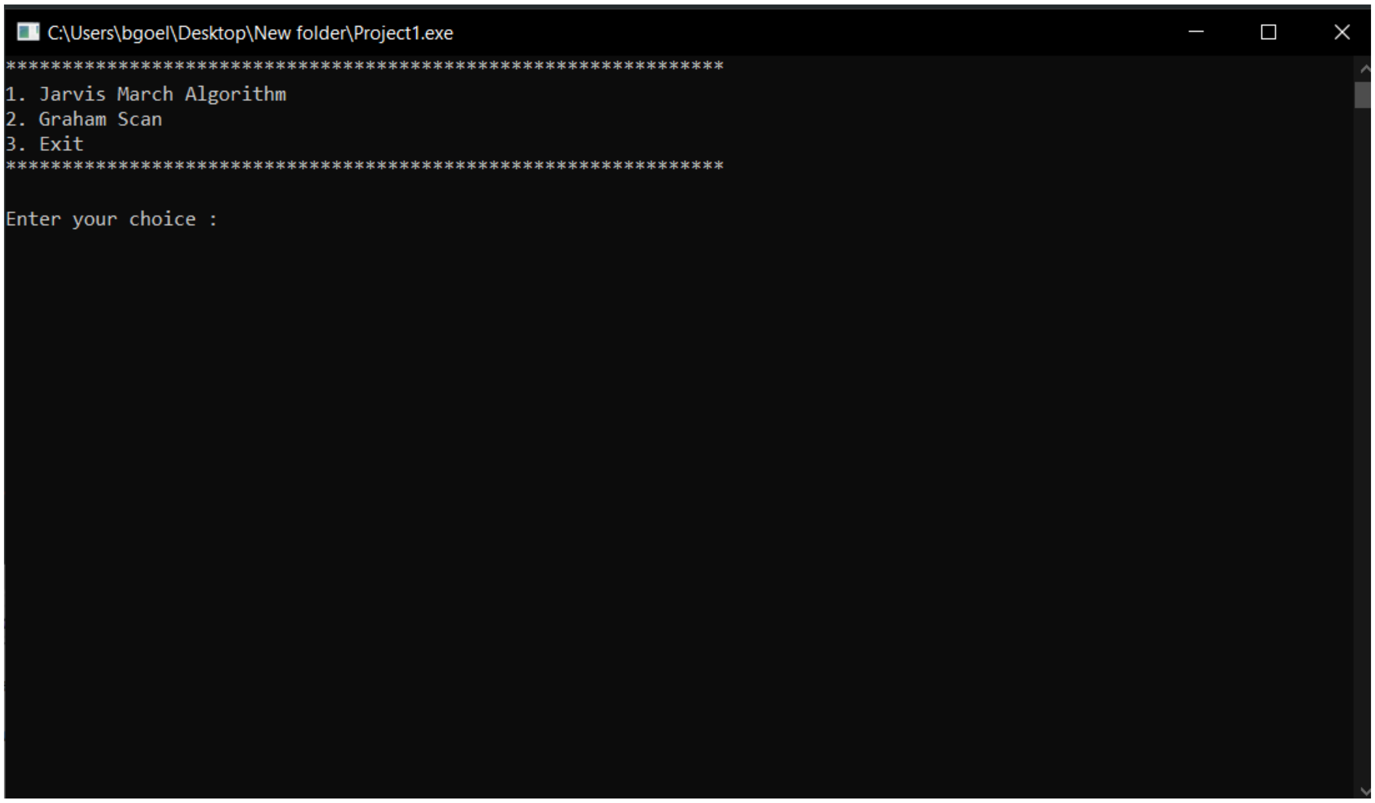


```
        cout<<"You entered less than 2 points, using default 2 points."<<endl;
        number_of_center = 2;
        }
    cout<<"Enter the number of points to be taken on each circle (WARNING : Enter more than
            12):"<<endl;
    cin>>number_of_points;
    if (number_of_points < 12)
    {
        cout<<"You Entered less than 12 points, using default 12 points."<<endl;
        number_of_points = 12;
        }
        cout<<"Enter the viscinity of virus-spread (Enter above 5) : ";
        cin>>viscinity;
        if (viscinity < 5)
        {
            cout<<"Your entered viscinity is too low, setting default 5"<<endl;
            viscinity = 5.0;
        }
    for (int i = 0 ; i<number_of_center ;i++)
    {
        cout<<"Enter the X-Axis for "<<i<<"th center : ";
        cin>>center.x;
        cout<<"Enter the Y-Axis for "<<i<<"th center : ";
        cin>>center.y;
        initial_center.push_back(center);
        putpixel( ( 250 + ( center.x ) * 10 ) , ( 250 - ( center.y ) * 10 ) , WHITE );
        setcolor(RED);
        circle((250+(center.x)*10),(250-(center.y)*10),viscinity * 10);
    }
    circle_points( initial_center , on_circle_round , number_of_center , number_of_points ,
                viscinity);
    Jarvis_March(on_circle_round, jarvis_boundary, number_of_center, number_of_points);
        area = area_total(jarvis_boundary);
    if (jarvis_boundary.size() != 0 )
    {
        for (int i = 0; i<jarvis_boundary.size() ; i++)
        {
            putpixel((250+(jarvis_boundary[i].x)*10),(250-(jarvis_boundary[i].y)*10
                    ),YELLOW);
                    cout<<i+1<<" . "<<"\t\t";
            cout<<"X-Axis : "<<jarvis_boundary[i].x<<"\t\t\t";
            cout<<"Y-Axis : "<<jarvis_boundary[i].y<<endl<<endl;
            }

            cout<<"Total Area Coverd by these points is : "<<area<<endl;
        for (int i = 1; i<jarvis_boundary.size() ; i++)
        {
            setcolor(MAGENTA);
            line((250+(jarvis_boundary[i-1].x)*10),(250-(jarvis_boundary[i-1].y)*10),(250+
                (jarvis_boundary[i].x)*10),(250-(jarvis_boundary[i].y)*10));
            }
            line((250+(jarvis_boundary[0].x)*10),(250-(jarvis_boundary[0].y)*10),(250+
                (jarvis_boundary[jarvis_boundary.size()-1].x)*10),(250-
            (jarvis_boundary[jarvis_boundary.size()-1].y)*10));
        setcolor(WHITE);
        line(250,0,250,500);
        line(0,250,500,250);
        getch();
        }
        initial_center.clear();
    on_circle_round.clear();
    jarvis_boundary.clear();
    break;

case(2):
        initwindow(500,500,"ConvexHull");
    cout<<"Enter the number of points : ";
    cin>>number_of_center;
    if (number_of_center < 2)
    {
        cout<<"You entered less than 2 points, using default 2 points."<<endl;
        number_of_center = 2;
        }
    cout<<"Enter the number of points to be taken on each circle (WARNING : Enter more than
            12): ";
    cin>>number_of_points;
```



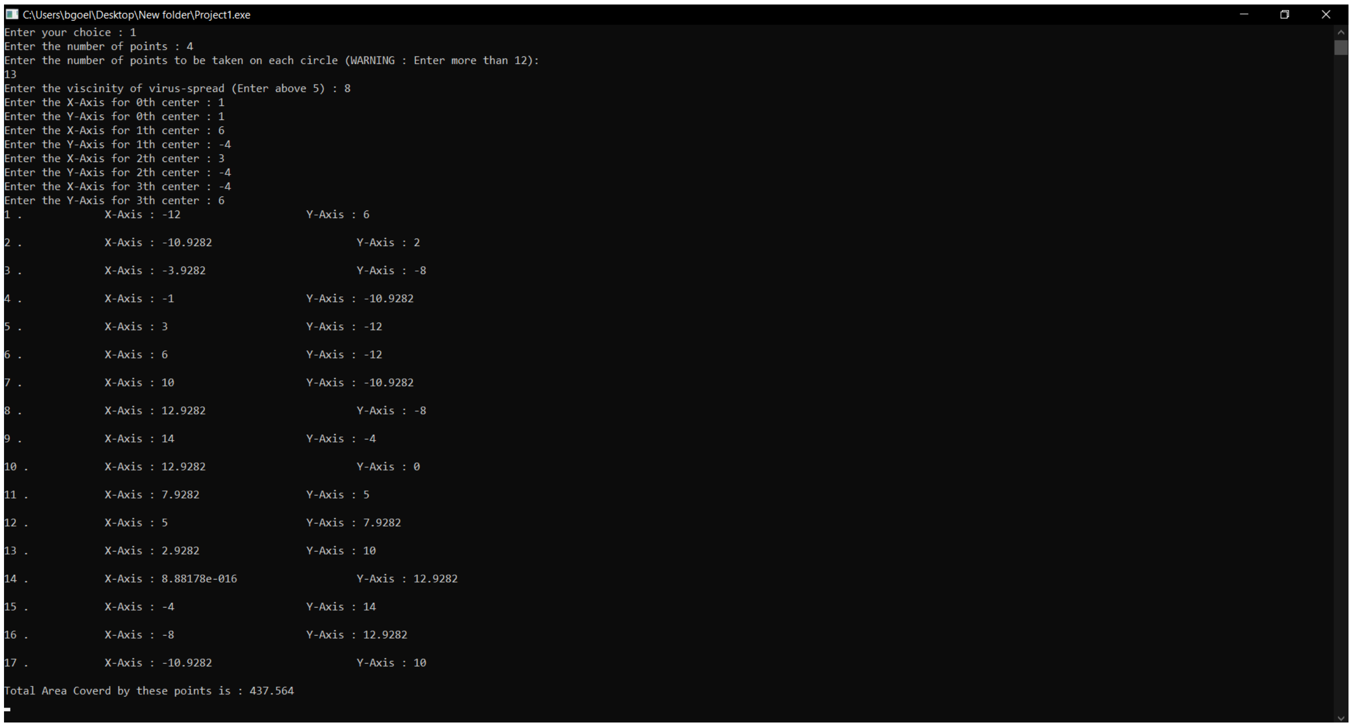
```
                if (number_of_points < 12)
                {
                    cout<<"You Entered less than 12 points, using default 12 points."<<endl;
                    number_of_points = 12;
                    }
                    cout<<"Enter the viscinity of virus-spread (Enter above 5) : ";
                    cin>>viscinity;
                    if (viscinity < 5)
                    {
                        cout<<"Your entered viscinity is too low, setting default 5"<<endl;
                        viscinity = 5.0;
                    }
                for (int i = 0 ; i<number_of_center ;i++)
                {
                    cout<<"Enter the X-Axis for "<<i<<"th center : ";
                    cin>>center.x;
                    cout<<"Enter the Y-Axis for "<<i<<"th center : ";
                    cin>>center.y;
                    initial_center.push_back(center);
                    putpixel( ( 250 + ( center.x ) * 10 ) , ( 250 - ( center.y ) * 10 ) , WHITE );
                    setcolor(RED);
                    circle((250+(center.x)*10),(250-(center.y)*10),viscinity * 10);
                }
                circle_points( initial_center , on_circle_round , number_of_center , number_of_points,
                        viscinity );
                    Graham_Scan(on_circle_round, graham_boundary, number_of_center, number_of_points );
                    area = area_total(graham_boundary);
                    if (graham_boundary.size() != 0 )
                    {
                        for (int i = 0; i<graham_boundary.size() ; i++)
                    {
                        putpixel((250+(graham_boundary[i].x)*10),(250-(graham_boundary[i].y)*10
                            ),YELLOW);
                                cout<<i+1<<" . "<<"\t\t";
                        cout<<"X-Axis : "<<graham_boundary[i].x<<"\t\t\t";
                        cout<<"Y-Axis : "<<graham_boundary[i].y<<endl<<endl;
                        }

                        cout<<"Total Area Coverd by these points is : "<<area<<endl;
                    for (int i = 1; i<graham_boundary.size() ; i++)
                    {
                        setcolor(MAGENTA);
                        line((250+(graham_boundary[i-1].x)*10),(250-(graham_boundary[i-1].y)*10),(250+
                                (graham_boundary[i].x)*10),(250-(graham_boundary[i].y)*10));
                        }
                        line((250+(graham_boundary[0].x)*10),(250-(graham_boundary[0].y)*10),(250+
                            (graham_boundary[graham_boundary.size()-1].x)*10),(250-
                        (graham_boundary[graham_boundary.size()-1].y)*10));
                        setcolor(WHITE);
                    line(250,0,250,500);
                    line(0,250,500,250);
                    getch();
                    }
                    initial_center.clear();
                on_circle_round.clear();
                graham_boundary.clear();
            break;

            case(3):
                    exit(0);
            default:
                cout<<"Error! You enetered wrong number";
                break;
        }
    }
}
```

OUTPUT



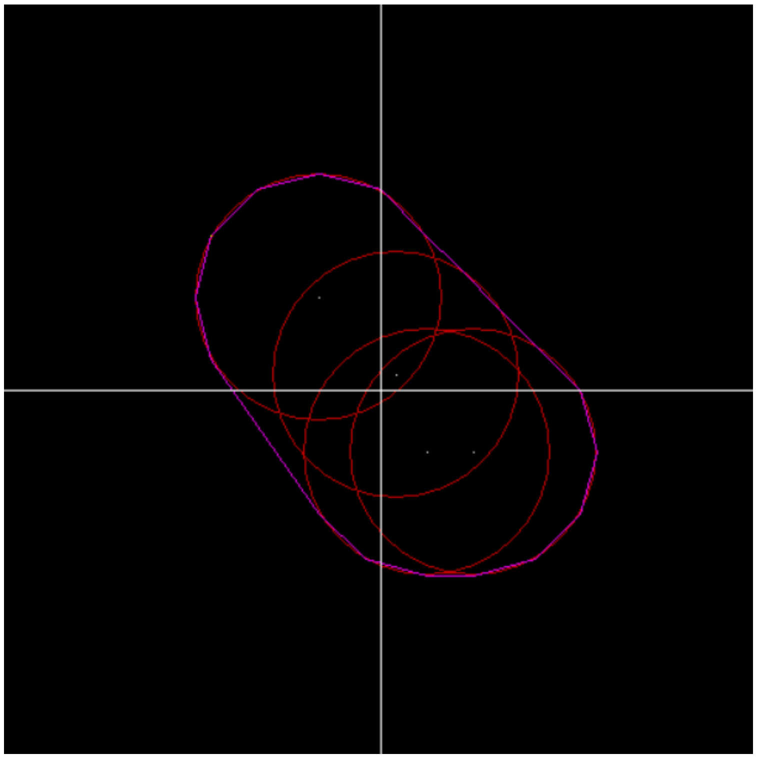
```
C:\Users\bgoel\Desktop\New folder\Project1.exe
****************************************************************
1. Jarvis March Algorithm
2. Graham Scan
3. Exit
****************************************************************

Enter your choice :
```

Choice 1 : Jarvis March

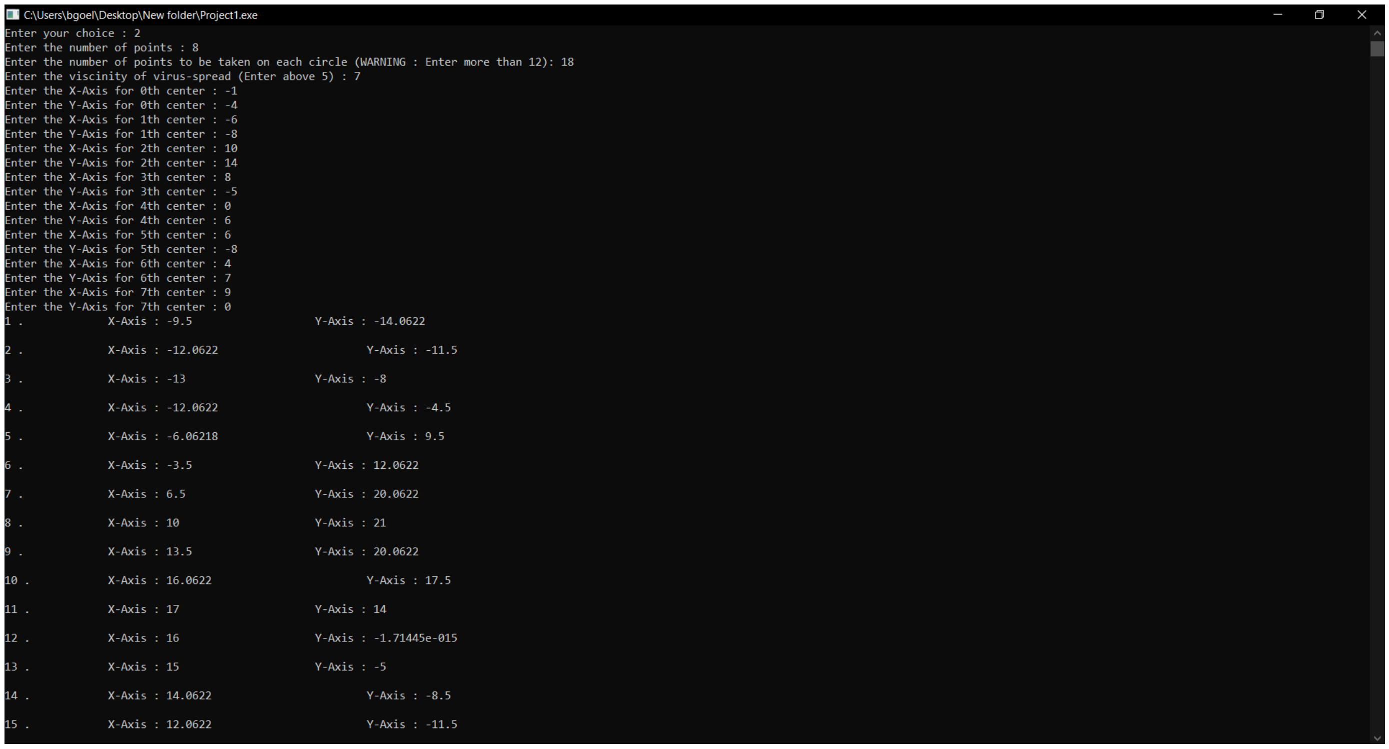


```
C:\Users\bgoel\Desktop\New folder\Project1.exe
Enter your choice : 1
Enter the number of points : 4
Enter the number of points to be taken on each circle (WARNING : Enter more than 12):
13
Enter the viscinity of virus-spread (Enter above 5) : 8
Enter the X-Axis for 0th center : 1
Enter the Y-Axis for 0th center : 1
Enter the X-Axis for 1th center : 6
Enter the Y-Axis for 1th center : -4
Enter the X-Axis for 2th center : 3
Enter the Y-Axis for 2th center : -4
Enter the X-Axis for 3th center : -4
Enter the Y-Axis for 3th center : 6
1 .             X-Axis : -12                    Y-Axis : 6

2 .             X-Axis : -10.9282                       Y-Axis : 2

3 .             X-Axis : -3.9282                        Y-Axis : -8

4 .             X-Axis : -1                     Y-Axis : -10.9282

5 .             X-Axis : 3                      Y-Axis : -12

6 .             X-Axis : 6                      Y-Axis : -12

7 .             X-Axis : 10                     Y-Axis : -10.9282

8 .             X-Axis : 12.9282                        Y-Axis : -8

9 .             X-Axis : 14                     Y-Axis : -4

10 .            X-Axis : 12.9282                        Y-Axis : 0

11 .            X-Axis : 7.9282                 Y-Axis : 5

12 .            X-Axis : 5                      Y-Axis : 7.9282

13 .            X-Axis : 2.9282                 Y-Axis : 10

14 .            X-Axis : 8.88178e-016                   Y-Axis : 12.9282

15 .            X-Axis : -4                     Y-Axis : 14

16 .            X-Axis : -8                     Y-Axis : 12.9282

17 .            X-Axis : -10.9282                       Y-Axis : 10

Total Area Coverd by these points is : 437.564
```

Plotted Graph (Jarvis March)



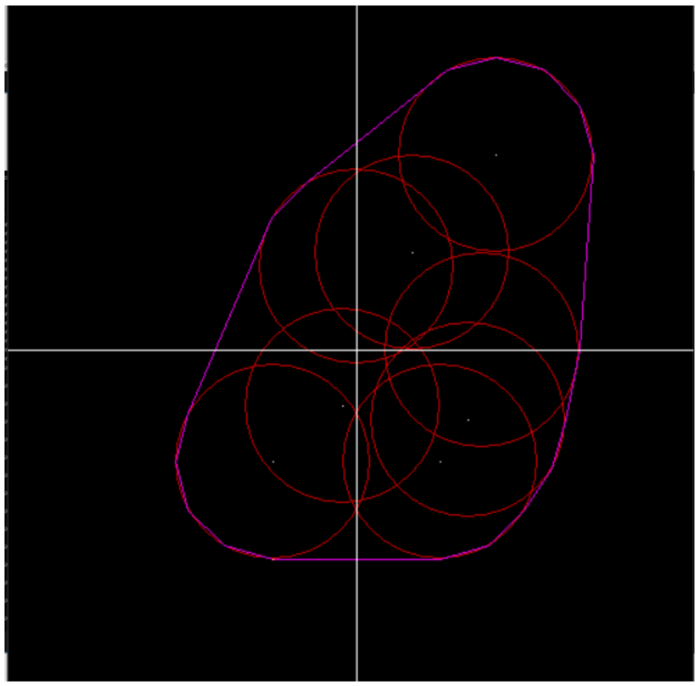
Choice 2 (Graham Scan)



C:\Users\bgoel\Desktop\New folder\Project1.exe

```
Enter your choice : 2
Enter the number of points : 8
Enter the number of points to be taken on each circle (WARNING : Enter more than 12): 18
Enter the viscinity of virus-spread (Enter above 5) : 7
Enter the X-Axis for 0th center : -1
Enter the Y-Axis for 0th center : -4
Enter the X-Axis for 1th center : -6
Enter the Y-Axis for 1th center : -8
Enter the X-Axis for 2th center : 10
Enter the Y-Axis for 2th center : 14
Enter the X-Axis for 3th center : 8
Enter the Y-Axis for 3th center : -5
Enter the X-Axis for 4th center : 0
Enter the Y-Axis for 4th center : 6
Enter the X-Axis for 5th center : 6
Enter the Y-Axis for 5th center : -8
Enter the X-Axis for 6th center : 4
Enter the Y-Axis for 6th center : 7
Enter the X-Axis for 7th center : 9
Enter the Y-Axis for 7th center : 0
1 .             X-Axis : -9.5                   Y-Axis : -14.0622

2 .             X-Axis : -12.0622                       Y-Axis : -11.5

3 .             X-Axis : -13                    Y-Axis : -8

4 .             X-Axis : -12.0622                       Y-Axis : -4.5

5 .             X-Axis : -6.06218                       Y-Axis : 9.5

6 .             X-Axis : -3.5                   Y-Axis : 12.0622

7 .             X-Axis : 6.5                    Y-Axis : 20.0622

8 .             X-Axis : 10                     Y-Axis : 21

9 .             X-Axis : 13.5                   Y-Axis : 20.0622

10 .            X-Axis : 16.0622                        Y-Axis : 17.5

11 .            X-Axis : 17                     Y-Axis : 14

12 .            X-Axis : 16                     Y-Axis : -1.71445e-015

13 .            X-Axis : 15                     Y-Axis : -5

14 .            X-Axis : 14.0622                        Y-Axis : -8.5

15 .            X-Axis : 12.0622                        Y-Axis : -11.5
```

Plotted Graph (Graham Scan)



To access the code for future reference:

https://codeshare.io/5zAnZ4

https://www.youtube.com/watch?v=TEMhWt9WwTA //This might be helpful to run the code successfully