The University of Rhode Island

Convex Hull Data Structure: Implementation of Graham's Scan algorithm.

Group Members:
Alex Lupo, Michael Marsella, Jake Nicynski, Jackson Perry

Presentation Date:
12/6/23

Table of Contents

INTRODUCTION:

What is a Convex Hull?

Context & Purpose

The development of convex hull algorithms dates back to ancient times in accordance with mathematical and technological advancements. Specifically, in 1972, the Graham Scan algorithm was introduced by Ronald Graham, which initially sorts all points based upon their polar angles, and then constructs the convex hull point by point. The Graham Scan algorithm proved to be most effective in the efficiency and accuracy of the development of a convex hull based upon its computational time complexity.

Convex hulls serve as a fundamental concept in geometric and mathematical practices. A convex hull is the formation of the smallest possible shape that encompasses all given points. The convex hull includes the line segment connecting two separate points or vertices. Generally, the purpose of a convex hull is to provide a structural representation of a set of points with the inclusion of its outer boundary. Convex hulls' functionality extends to areas of encapsulation, analysis, outlier detection, and other applicable processes.

Spatial encapsulation refers to the processes behind the formation of a convex hull. The encapsulation of the points in each set is achieved by the nature of the bounding structure, ensuring all possible points are within the hull. Convex hulls' ability to capture any particular outliers is conducted through this encapsulation process. Additionally, convex hulls' computational efficiency promotes their ability to achieve desirable time complexities whilst handling large datasets. Parsing through datasets, the determinant of whether a point is a part of the hull can be incredibly useful in practical applications. For example, collision avoidance systems in a Tesla exemplifies a use case of convex hull through the detection of other roadway users, and objects. GPS systems highlight a fundamental feature of a convex hull, involved in deciphering the shortest line segment between points. This allows for path-decision making in computing the quickest path to the designated point or destination.

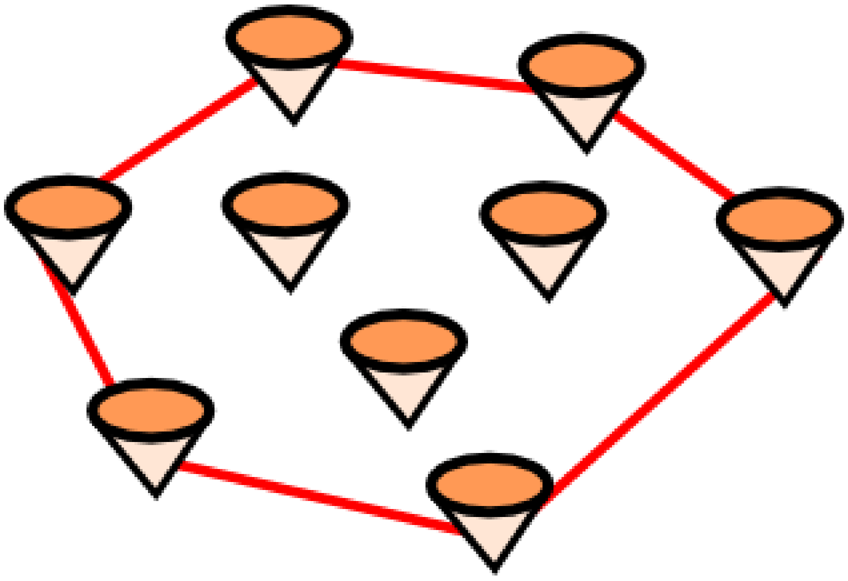
Why we chose Convex Hull?

We chose convex hull because we found the algorithm to be the most interesting and unique out of all of the possible choices for our project. We thought that convex hull would be a great test of our skills and a challenge to implement. Over the weeks of the project, we have brainstormed many different possible uses for convex hull and are very pleased to see our implementations work with our code. Some of our implemenations have been storm tracking and collision detection which have been a very interesting challenge to implement using SFML graphing library. With the project coming to an end we believe we have succesfully accomplished what we wanted to achieve with our convex hull project and are overall very pleased with the outcome of the code.

METHODS:

Convex Hull in Depth

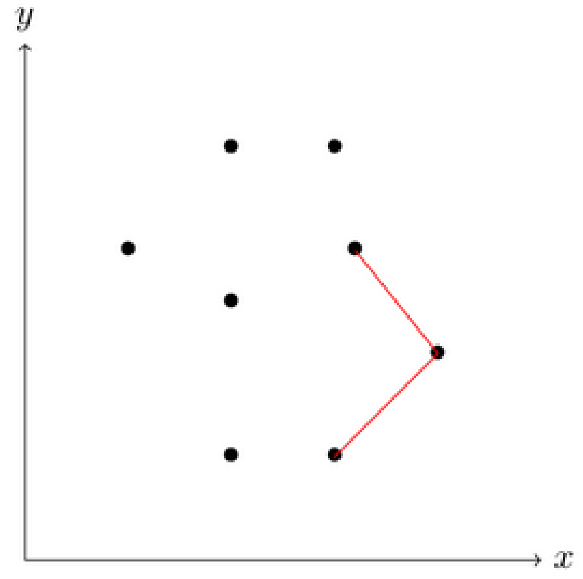
A convex hull is defined as: the set "Y is said to be *convex* if for any points *a*, *b* ∈ *Y*, every point on the straight-line segment joining them is also in *Y"* (Forcey, 10.2.1). From this definition, a convex hull of a certain number of points, is the smallest convex set that contains all of those points. For practicality, considering darts on a dartboard as points in the Euclidean space, if you were to stretch a rubber band out across all of the darts, the boundary formed by the rubber band would be the formation of the convex.



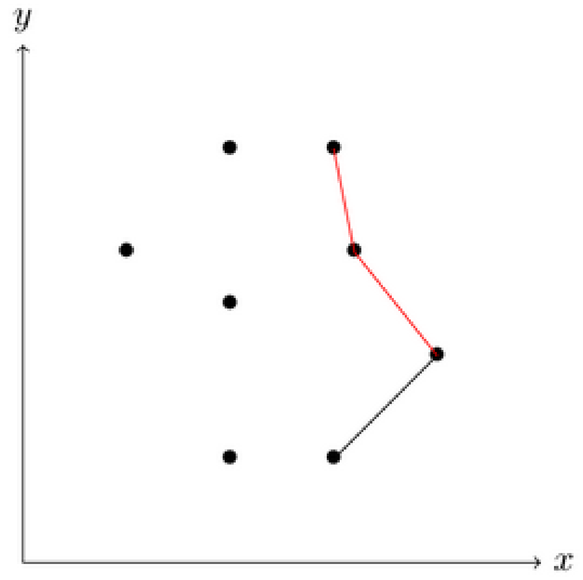
Algorithm in Depth

The Graham Scan algorithm is an effective solution to solving a convex hull. Using the Graham Scan algorithm allows one to easily decipher the points contained within the hull, and the order in which the points form the hull. First, the algorithm typically determines the lowest y-point coordinate as a starting point for the development of the hull. This starting point is critical as it serves as a part of the bounding region which contains the set of all the points. If multiple points share the same y-coordinate, looking to the lowest x-coordinate, for example, would be necessary to choose an initial point. This enables the boundary of the convex hull to start with the lowest and leftmost point. With the established starting point, other points are then sorted based upon the angle formed along the x-axis to this point. The sorting aspect involved in the

Graham Scan approach allows for each point to be accounted for as to whether it is a part of the formation of the hull. The sign of the cross product between the points is a key comparison to make in the determination of the formed angle. Next, the points in the set are determined to be a part of the convex based upon whether a concave corner is formed in relation to triplets of points. The middle point of a concave corner in relation to the polygon, cannot lie on the convex hull. Points that seem to be a part of the convex hull can be pushed onto a stack, and then later, if the point ends up forming a concave corner, the point can easily be removed from the stack.

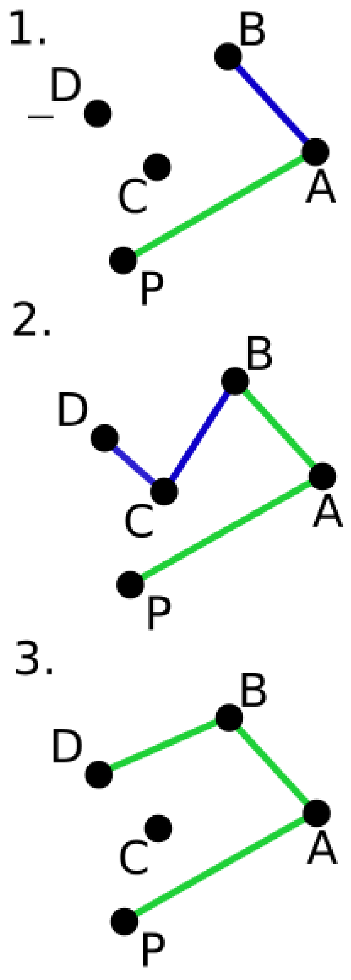


This figure illustrates an example of a convex corner which is a valid part of the convex hull, allowing the algorithm to continue to the remaining points.



However, in the figure shown above, the edge marked in red forms a concavity in Euclidean space, resulting in the middle point not being a part of the convex hull.

From the sorted array of points based upon angles, an alternative perspective to Graham's Scan algorithm proceeds in evaluating the path formed by the points in terms of a clockwise or counterclockwise notion. Again, the algorithm is evaluating triplets of points, but now determining whether they are directed in a counterclockwise direction. If the direction formed is clockwise, the middle point of the segment is discarded, or popped from the stack.



As illustrated above in the formation of a convex hull, the starting point is P as it's the lowest y-coordinate point. From point P, the following two points A, and B are observed as they form an angle with P. The line segment PAB is directed in the counterclockwise direction, in addition to line segment ABC. However, the segment BCD is directed in the clockwise direction, making the removal of point C necessary to ensure the integrity of the convex remains. The formation of the convex requires continuous left turns to be made for the development of the polygon structural boundary. If a right turn is encountered, which is visualized from point C to D, the middle point is removed as a concavity is formed. Thus, the removal of C allows for the convex structure to be achieved.

IMPLEMENTATION:

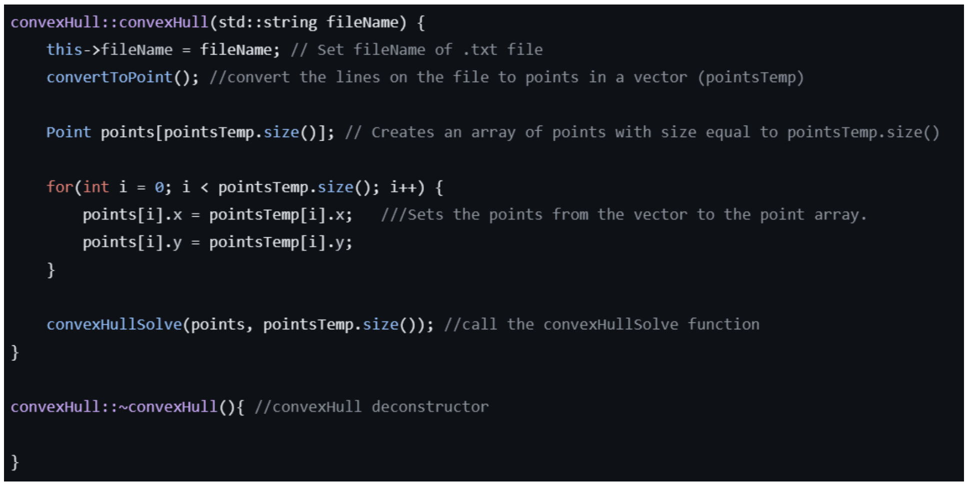
Our Code

For a more in-depth look click here

*Functions with only headers are too large to include in the report

In convexHull.cpp...

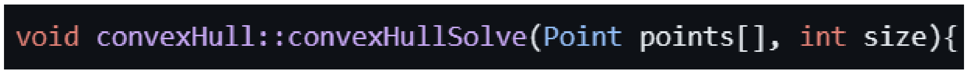
Constructor & Destructor:



```
convexHull::convexHull(std::string fileName) {
    this->fileName = fileName; // Set fileName of .txt file
    convertToPoint(); //convert the lines on the file to points in a vector (pointsTemp)

    Point points[pointsTemp.size()]; // Creates an array of points with size equal to pointsTemp.size()

    for(int i = 0; i < pointsTemp.size(); i++) {
        points[i].x = pointsTemp[i].x;   ///Sets the points from the vector to the point array.
        points[i].y = pointsTemp[i].y;
    }

    convexHullSolve(points, pointsTemp.size()); //call the convexHullSolve function
}

convexHull::~convexHull(){ //convexHull deconstructor

}
```

The constructor takes in the file name and converts the points in the file to the points in a vector. Then the constructor takes the points in the vector and converts it to an array. It then solves the convex hull within the constructor.

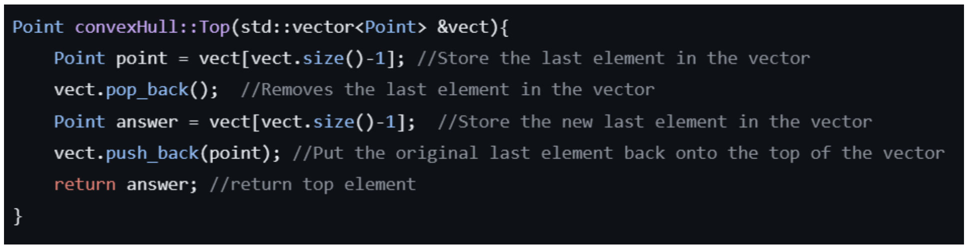
convexHullSolve:



```
void convexHull::convexHullSolve(Point points[], int size){
```

The convexHullSolve function is the main function that solves for the convex hull points. It is in its own function to prevent messy code and make it cleaner to read through for simplicity. It uses implementations of the Graham scan algorithm.

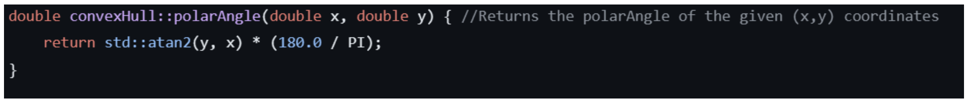
Top:



```
Point convexHull::Top(std::vector<Point> &vect){
    Point point = vect[vect.size()-1]; //Store the last element in the vector
    vect.pop_back();  //Removes the last element in the vector
    Point answer = vect[vect.size()-1];  //Store the new last element in the vector
    vect.push_back(point); //Put the original last element back onto the top of the vector
    return answer; //return top element
}
```

This function is used to return the second to last point in the vector.

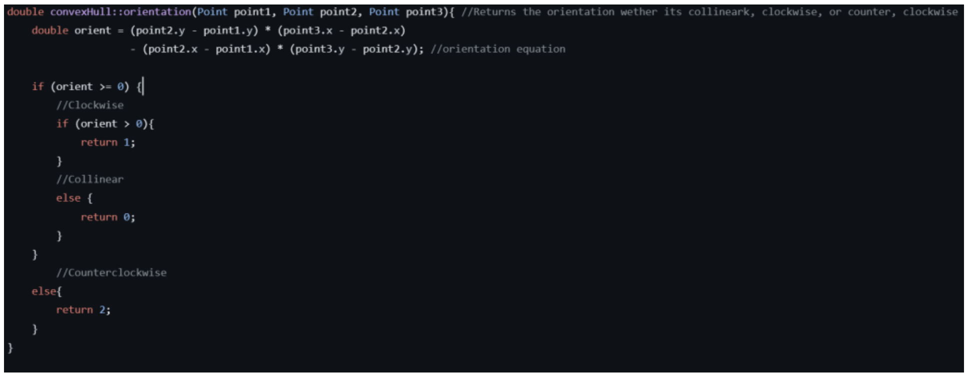
polarAngle:



```
double convexHull::polarAngle(double x, double y) { //Returns the polarAngle of the given (x,y) coordinates
    return std::atan2(y, x) * (180.0 / PI);
}
```

This function uses the polar angle equation and returns the polar angle of the given coordinates.

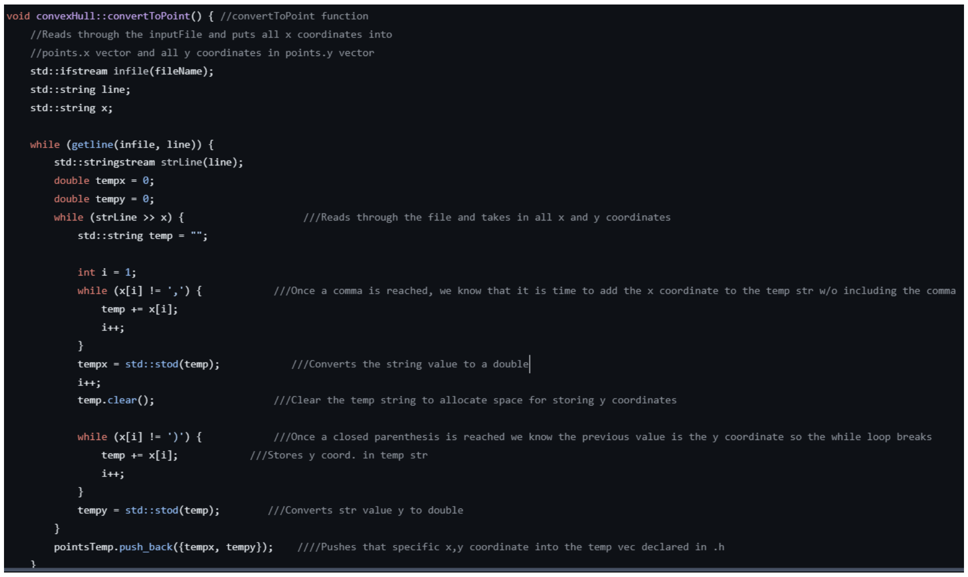
orientation:



```
double convexHull::orientation(Point point1, Point point2, Point point3){ //Returns the orientation wether its collineark, clockwise, or counter, clockwise
    double orient = (point2.y - point1.y) * (point3.x - point2.x)
                - (point2.x - point1.x) * (point3.y - point2.y); //orientation equation

    if (orient >= 0) {
        //Clockwise
        if (orient > 0){
            return 1;
        }
        //Collinear
        else {
            return 0;
        }
    }
        //Counterclockwise
    else{
        return 2;
    }
}
```

The orientation function finds whether the point is clockwise, counterclockwise, or collinear based on the orientation equation for points.

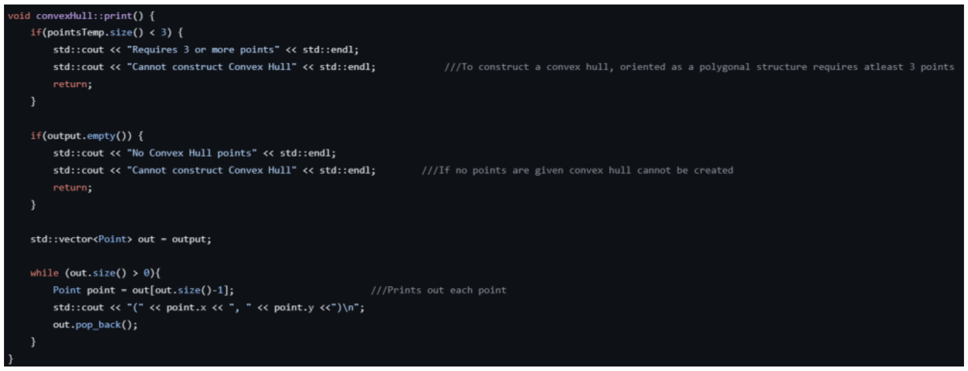
convertToPoint:



```cpp
void convexHull::convertToPoint() { //convertToPoint function
    //Reads through the inputFile and puts all x coordinates into
    //points.x vector and all y coordinates in points.y vector
    std::ifstream infile(fileName);
    std::string line;
    std::string x;

    while (getline(infile, line)) {
        std::stringstream strLine(line);
        double tempx = 0;
        double tempy = 0;
        while (strLine >> x) {                    ///Reads through the file and takes in all x and y coordinates
            std::string temp = "";

            int i = 1;
            while (x[i] != ',') {            ///Once a comma is reached, we know that it is time to add the x coordinate to the temp str w/o including the comma
                temp += x[i];
                i++;
            }
            tempx = std::stod(temp);            ///Converts the string value to a double
            i++;
            temp.clear();                    ///Clear the temp string to allocate space for storing y coordinates

            while (x[i] != ')') {            ///Once a closed parenthesis is reached we know the previous value is the y coordinate so the while loop breaks
                temp += x[i];            ///Stores y coord. in temp str
                i++;
            }
            tempy = std::stod(temp);        ///Converts str value y to double
        }
        pointsTemp.push_back({tempx, tempy});    ////Pushes that specific x,y coordinate into the temp vec declared in .h
    }
```

The convertToPoint function is used to take in a text file of points in the form (x,y) coordinates and store them into a vector for the code to use in the following lines after.

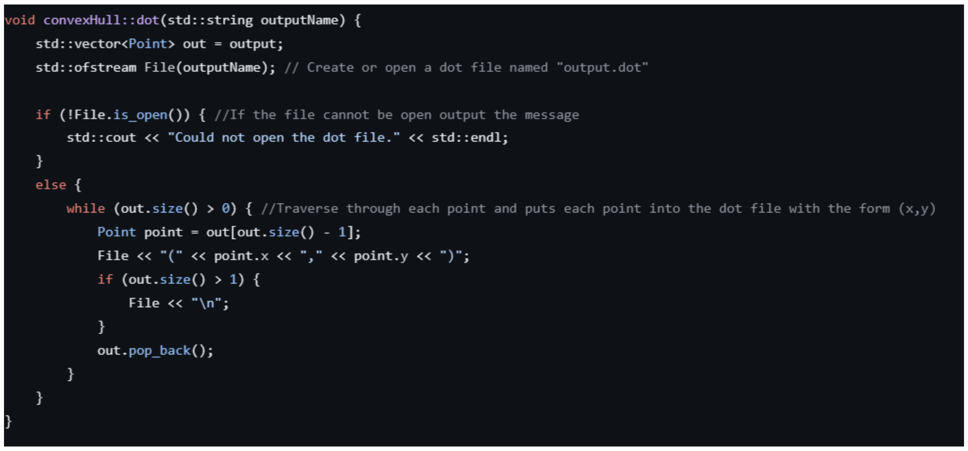
print:



```cpp
void convexHull::print() {
    if(pointsTemp.size() < 3) {
        std::cout << "Requires 3 or more points" << std::endl;
        std::cout << "Cannot construct Convex Hull" << std::endl;            ///To construct a convex hull, oriented as a polygonal structure requires atleast 3 points
        return;
    }

    if(output.empty()) {
        std::cout << "No Convex Hull points" << std::endl;
        std::cout << "Cannot construct Convex Hull" << std::endl;        ///If no points are given convex hull cannot be created
        return;
    }

    std::vector<Point> out = output;

    while (out.size() > 0){
        Point point = out[out.size()-1];                        ///Prints out each point
        std::cout << "(" << point.x << ", " << point.y <<")\n";
        out.pop_back();
    }
}
```

The print function outputs the convex hull points out onto the terminal. If there are less than three points it will output a message saying that there needs to be more points. If the vector is empty, it will output a message saying there are no convex hull points. If the above if statements are not used, then output the convex hull points

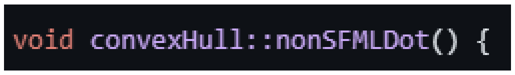
dot:



```
void convexHull::dot(std::string outputName) {
    std::vector<Point> out = output;
    std::ofstream File(outputName); // Create or open a dot file named "output.dot"

    if (!File.is_open()) { //If the file cannot be open output the message
        std::cout << "Could not open the dot file." << std::endl;
    }
    else {
        while (out.size() > 0) { //Traverse through each point and puts each point into the dot file with the form (x,y)
            Point point = out[out.size() - 1];
            File << "(" << point.x << "," << point.y << ")";
            if (out.size() > 1) {
                File << "\n";
            }
            out.pop_back();
        }
    }
}
```

The dot function converts the points in the output vector to a dot file. In the function a dot file is created, and the points are inputted into the dot file in the form (x,y). If the dot file cannot open a message will appear in the terminal and say that the file could not be open.

nonSFMLDot:

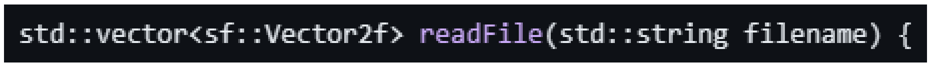


```
void convexHull::nonSFMLDot() {
```

The nonSFMLDot function converts the points in the output vector to a dot file. In the function a dot file is created, and the points are inputted into the dot file in the form 0 [label="(1,1)"];. If the file cannot open, then a message will appear in the terminal and say that the file could not be open. This dot file visualization is used for the online visualization tool Graphviz.

In graph.cpp...

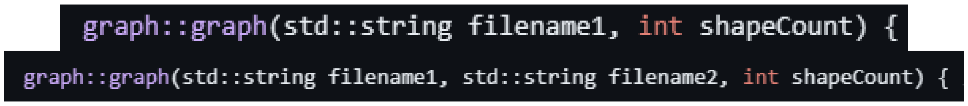
readFile:



```
std::vector<sf::Vector2f> readFile(std::string filename) {
```

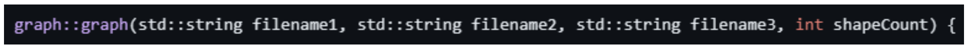
readFile reads through the .dot file that the main.cpp passes in provided by the convexHull.cpp. The function will read through and sort each x and y point into a Vector2f for the code to use in further parts of the code.

graph:



```
graph::graph(std::string filename1, int shapeCount) {
```

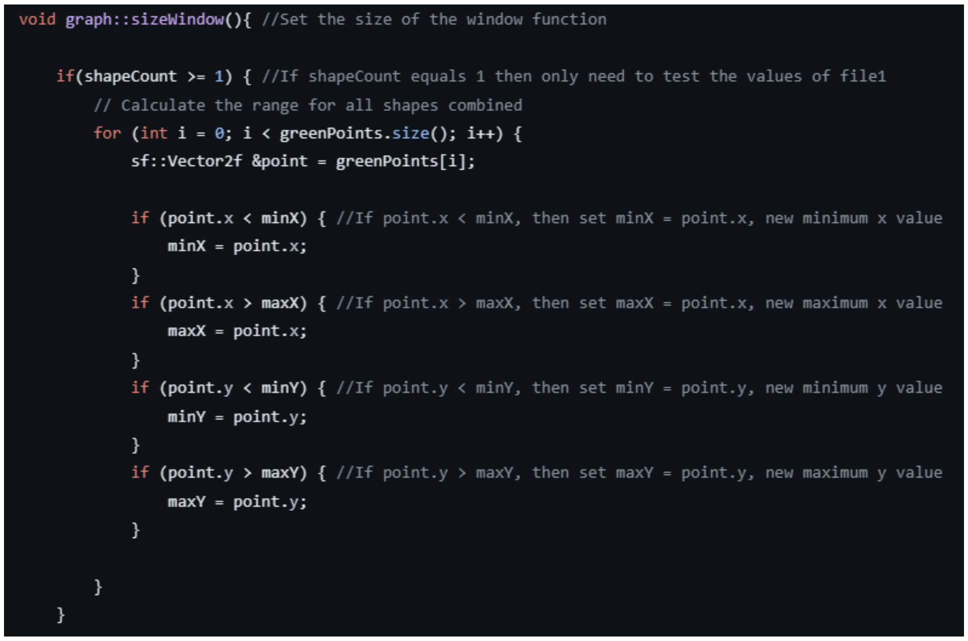
```
graph::graph(std::string filename1, std::string filename2, int shapeCount) {
```



```
graph::graph(std::string filename1, std::string filename2, std::string filename3, int shapeCount) {
```

These constructors are used to call the graph function. Depending on the size of the input to argv[] based off the amount of txt files imputed, the main.cpp will either call the graph constructor for one, two, or three shapes to graph.

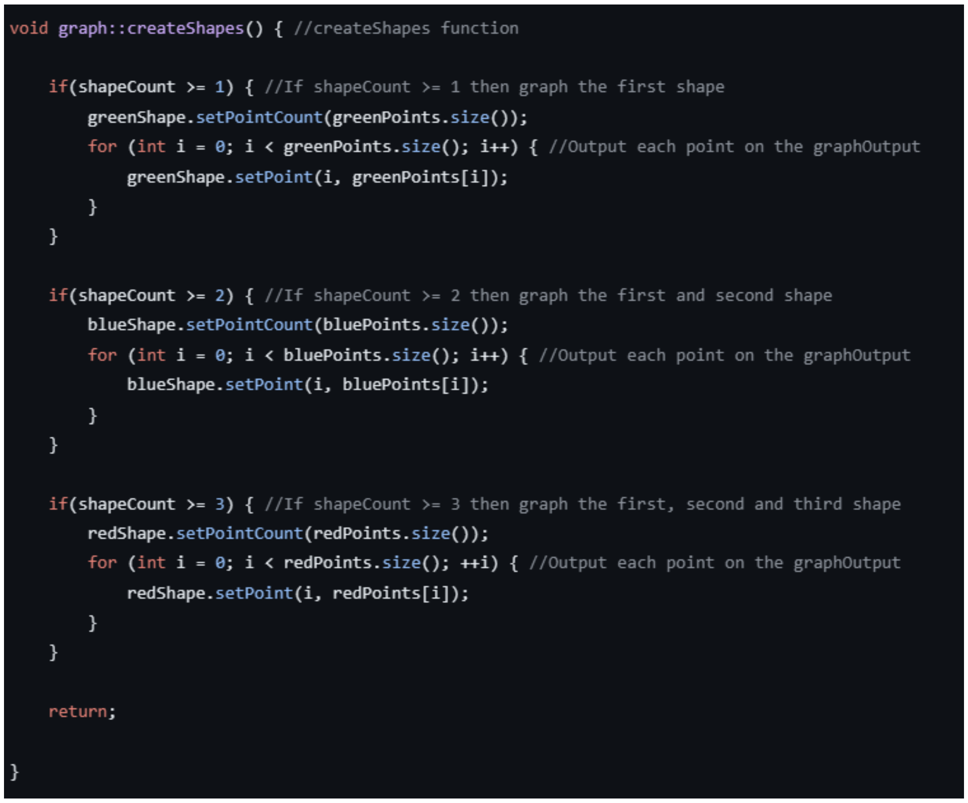
sizeWindow:



```
void graph::sizeWindow(){ //Set the size of the window function

    if(shapeCount >= 1) { //If shapeCount equals 1 then only need to test the values of file1
        // Calculate the range for all shapes combined
        for (int i = 0; i < greenPoints.size(); i++) {
            sf::Vector2f &point = greenPoints[i];

            if (point.x < minX) { //If point.x < minX, then set minX = point.x, new minimum x value
                minX = point.x;
            }
            if (point.x > maxX) { //If point.x > maxX, then set maxX = point.x, new maximum x value
                maxX = point.x;
            }
            if (point.y < minY) { //If point.y < minY, then set minY = point.y, new minimum y value
                minY = point.y;
            }
            if (point.y > maxY) { //If point.y > maxY, then set maxY = point.y, new maximum y value
                maxY = point.y;
            }

        }
    }
```

The sizeWindow function creates the size of the output window for the SFML visualization. The function uses sets of if statements to also resize the window based on the points.

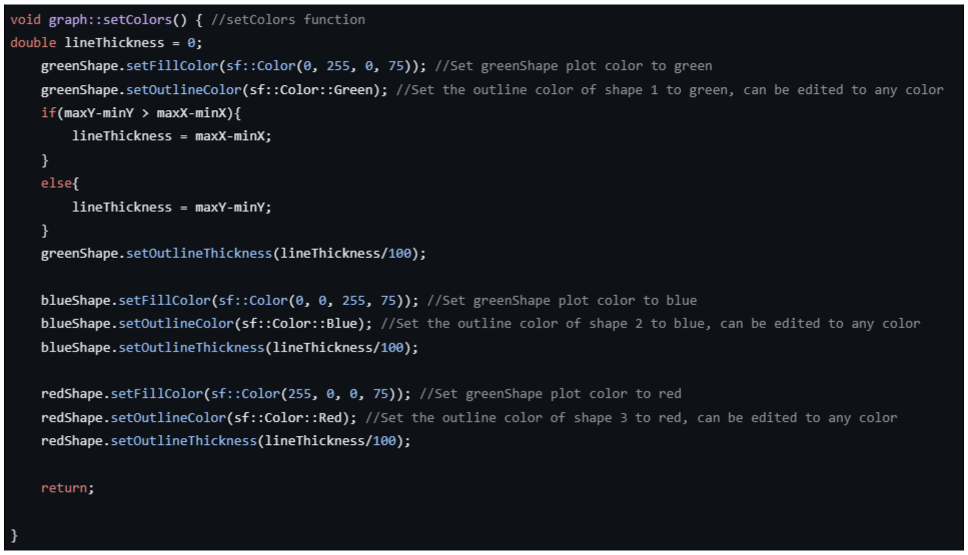
createShapes:



```cpp
void graph::createShapes() { //createShapes function

    if(shapeCount >= 1) { //If shapeCount >= 1 then graph the first shape
        greenShape.setPointCount(greenPoints.size());
        for (int i = 0; i < greenPoints.size(); i++) { //Output each point on the graphOutput
            greenShape.setPoint(i, greenPoints[i]);
        }
    }

    if(shapeCount >= 2) { //If shapeCount >= 2 then graph the first and second shape
        blueShape.setPointCount(bluePoints.size());
        for (int i = 0; i < bluePoints.size(); i++) { //Output each point on the graphOutput
            blueShape.setPoint(i, bluePoints[i]);
        }
    }

    if(shapeCount >= 3) { //If shapeCount >= 3 then graph the first, second and third shape
        redShape.setPointCount(redPoints.size());
        for (int i = 0; i < redPoints.size(); ++i) { //Output each point on the graphOutput
            redShape.setPoint(i, redPoints[i]);
        }
    }

    return;

}
```

The createShapes function creates the shape of the convex hull points to be graphed later.

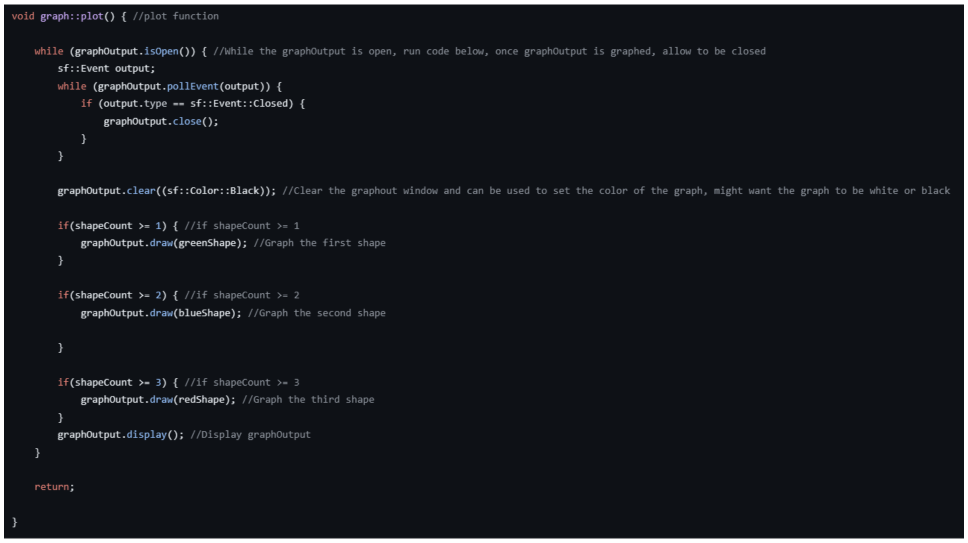
setColors:



```cpp
void graph::setColors() { //setColors function
double lineThickness = 0;
    greenShape.setFillColor(sf::Color(0, 255, 0, 75)); //Set greenShape plot color to green
    greenShape.setOutlineColor(sf::Color::Green); //Set the outline color of shape 1 to green, can be edited to any color
    if(maxY-minY > maxX-minX){
        lineThickness = maxX-minX;
    }
    else{
        lineThickness = maxY-minY;
    }
    greenShape.setOutlineThickness(lineThickness/100);

    blueShape.setFillColor(sf::Color(0, 0, 255, 75)); //Set greenShape plot color to blue
    blueShape.setOutlineColor(sf::Color::Blue); //Set the outline color of shape 2 to blue, can be edited to any color
    blueShape.setOutlineThickness(lineThickness/100);

    redShape.setFillColor(sf::Color(255, 0, 0, 75)); //Set greenShape plot color to red
    redShape.setOutlineColor(sf::Color::Red); //Set the outline color of shape 3 to red, can be edited to any color
    redShape.setOutlineThickness(lineThickness/100);

    return;

}
```

The setColors function is used to set the colors of the convex hull diagrams on the SFML visualization. The function outlines the convex hull points with a line using the colors red, blue, and green. It also fills the shape with one of the three colors.

plot:



```cpp
void graph::plot() { //plot function

    while (graphOutput.isOpen()) { //While the graphOutput is open, run code below, once graphOutput is graphed, allow to be closed
        sf::Event output;
        while (graphOutput.pollEvent(output)) {
            if (output.type == sf::Event::Closed) {
                graphOutput.close();
            }
        }

        graphOutput.clear((sf::Color::Black)); //Clear the graphout window and can be used to set the color of the graph, might want the graph to be white or black

        if(shapeCount >= 1) { //if shapeCount >= 1
            graphOutput.draw(greenShape); //Graph the first shape
        }

        if(shapeCount >= 2) { //if shapeCount >= 2
            graphOutput.draw(blueShape); //Graph the second shape

        }

        if(shapeCount >= 3) { //if shapeCount >= 3
            graphOutput.draw(redShape); //Graph the third shape
        }
        graphOutput.display(); //Display graphOutput
    }

    return;

}
```

The plot function is used to graph the shapes and call the display window function which will display the image for the user.

Test Cases

We designed the following 10 test cases to evaluate the accuracy of our Graham's scan convex hull algorithm implementation:

1. 100_points.txt
2. empty_test.txt
3. equal_points.txt
4. four_poles_with_duplicates.txt
5. horizontal_line.txt
6. one_point.txt
7. parabola_points.txt
8. random_square_points.txt
9. two_points.txt
10. vertical_line.txt

Each test case is in the format:

(x1,y1)

(x2,y2)

(x3,y3)...

The purpose of this is to generate the expected output from our test case files to compare with the results of our own convex hull code.

Imported ConvexHull structure from scipy to compare with ours ↓

```
from scipy.spatial import ConvexHull
import matplotlib.pyplot as plt
```

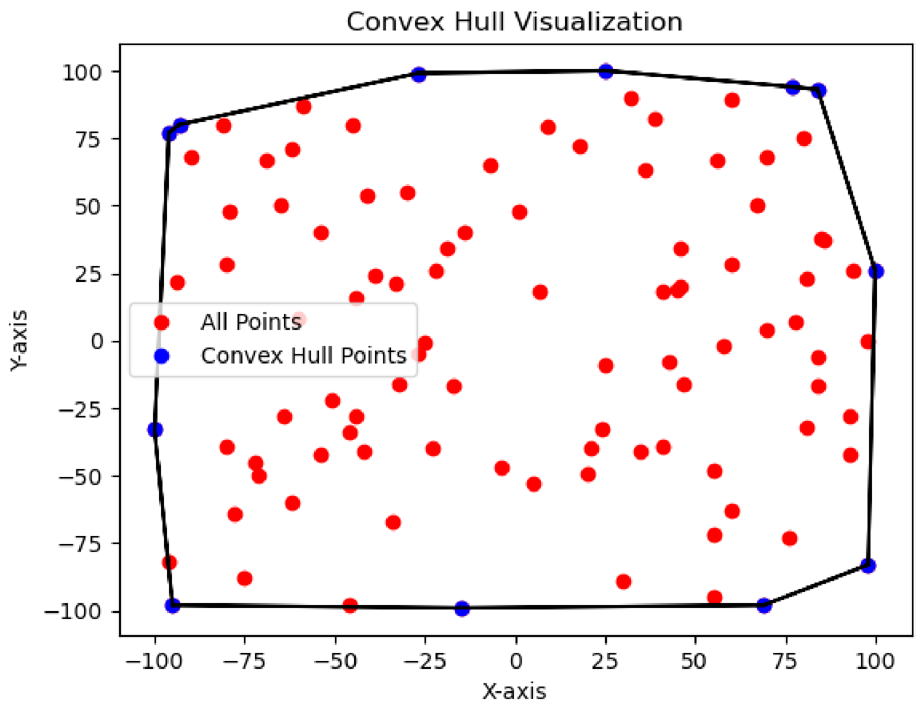
Python function to construct and plot convex hull given a file in the specified format:

```
def constructHull(file_name):
    with open(file_name, 'r') as file:
        lines = file.readlines()

    # Parse values
    x_values = []
    y_values = []

    for line in lines:
        x, y = map(int, line[line.index('(') + 1:line.index(')')].split(','))
        x_values.append(x)
        y_values.append(y)



    points = list(zip(x_values, y_values))

    try:
        hull = ConvexHull(points)

        convex_hull_points = [points[i] for i in hull.vertices]
        print("Convex Hull Points:", convex_hull_points)

        # Plot the points
        plt.plot(*zip(*points), 'ro', label='All Points')

        # Plot the convex hull points
        plt.plot(*zip(*[points[i] for i in hull.vertices]), 'bo', label='Convex Hull Points')

        # Plot the convex hull edges
        for simplex in hull.simplices:
            plt.plot([points[i][0] for i in simplex] + [points[simplex[0]][0]],
                     [points[i][1] for i in simplex] + [points[simplex[0]][1]], 'k-')
        plt.title('Convex Hull Visualization')
        plt.xlabel('X-axis')
        plt.ylabel('Y-axis')
        plt.legend()
        plt.show()

    except Exception as e:
        print("Cannot construct convex hull. Plotting original points.")

        # Plot the points without attempting to compute the convex hull
        plt.plot(*zip(*points), 'ro', label='All Points')
        plt.title('Point Visualization')
        plt.xlabel('X-axis')
        plt.ylabel('Y-axis')
        plt.legend()
        plt.show()
```

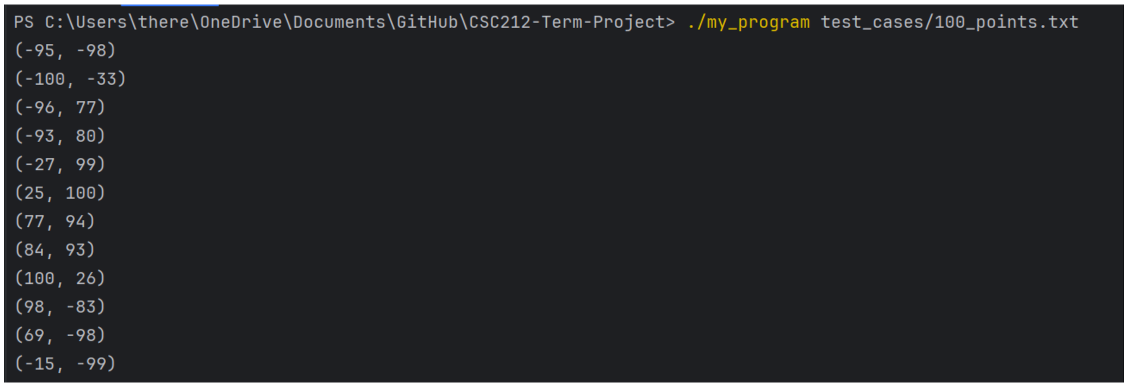
TEST CASE 1: 100 Points

Expected output:

```
constructHull('test_cases/100_points.txt')

Convex Hull Points: [(-96, 77), (-100, -33), (-95, -98), (-15, -99), (69, -
98), (98, -83), (100, 26), (84, 93), (77, 94), (25, 100), (-27, 99), (-93,
80)]
```

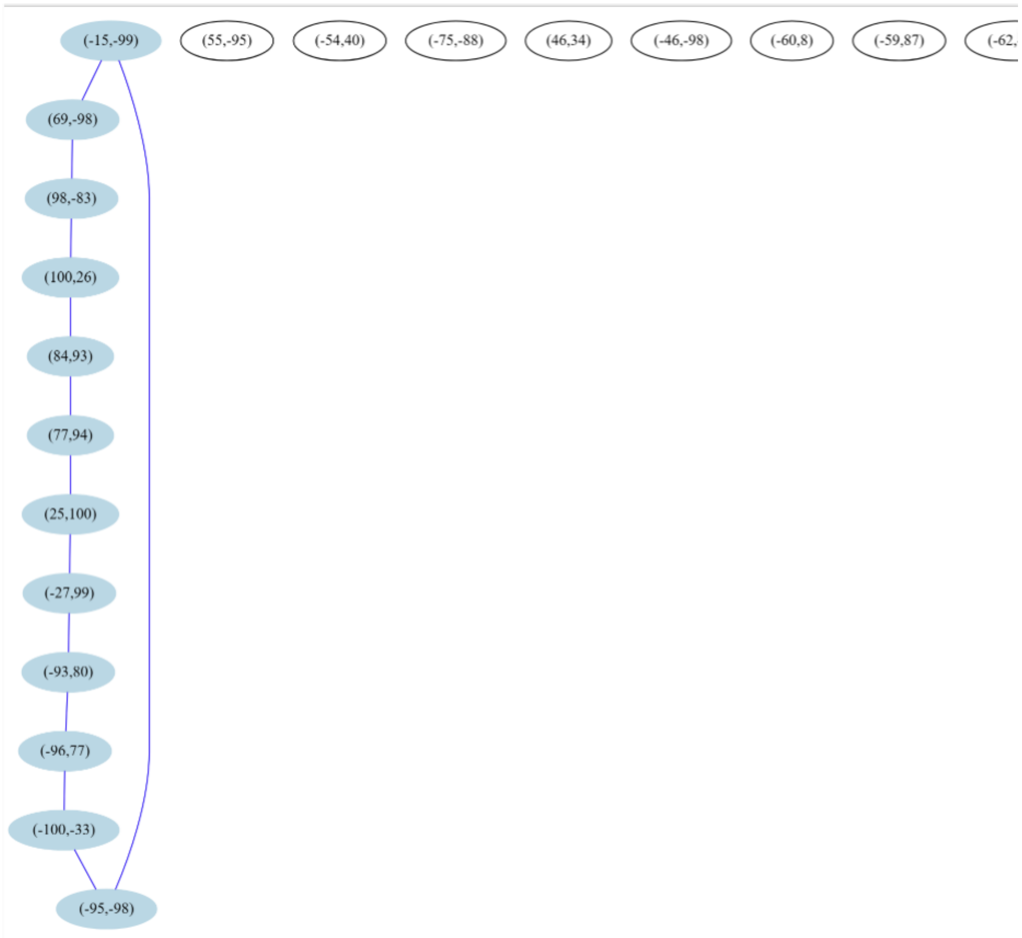


Our output:



```
PS C:\Users\there\OneDrive\Documents\GitHub\CSC212-Term-Project> ./my_program test_cases/100_points.txt
(-95, -98)
(-100, -33)
(-96, 77)
(-93, 80)
(-27, 99)
(25, 100)
(77, 94)
(84, 93)
(100, 26)
(98, -83)
(69, -98)
(-15, -99)
```

Our .dot file visualization(s):

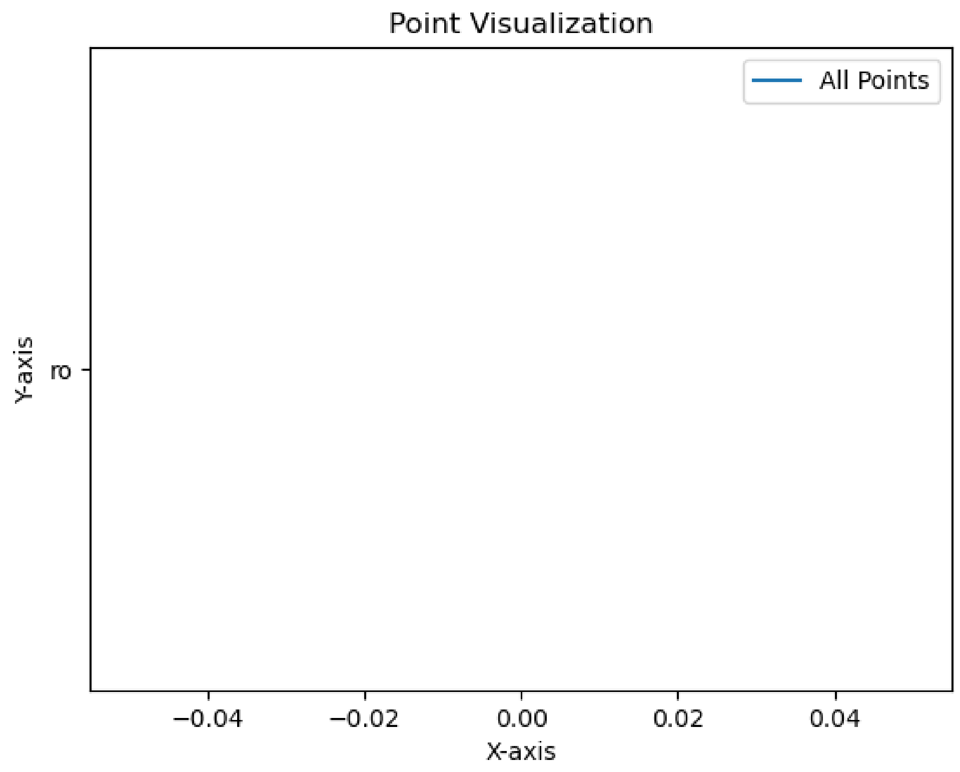


Our results seem to match!

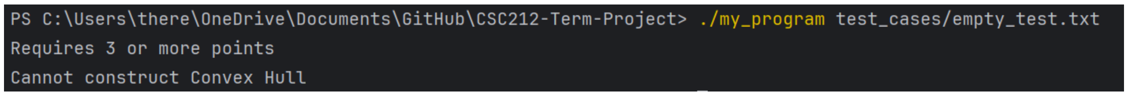
TEST CASE 2: Empty File

Expected output:

```
constructHull('test_cases/empty_test.txt')

Cannot construct convex hull. Plotting original points.
```



Our output:



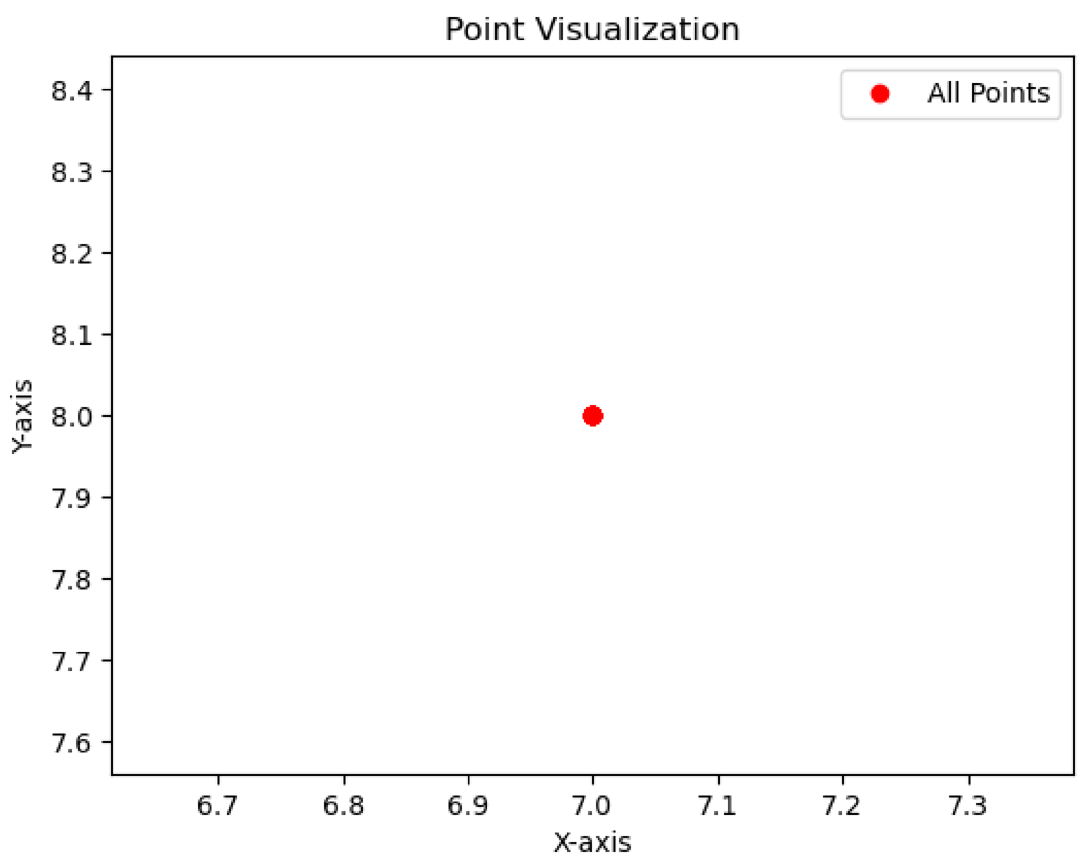
```
PS C:\Users\there\OneDrive\Documents\GitHub\CSC212-Term-Project> ./my_program test_cases/empty_test.txt
Requires 3 or more points
Cannot construct Convex Hull
```

Our results match!

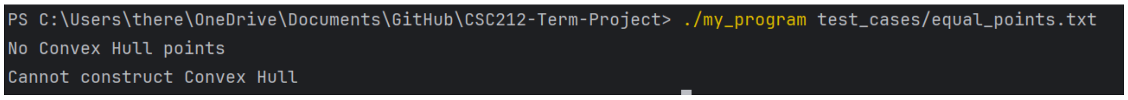
TEST CASE 3: Duplicate Points (more than 3 points)

Expected output:

```
constructHull('test_cases/equal_points.txt')

Cannot construct convex hull. Plotting original points.
```

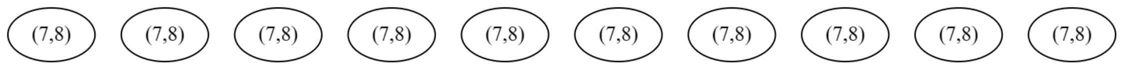


Our output:



```
PS C:\Users\there\OneDrive\Documents\GitHub\CSC212-Term-Project> ./my_program test_cases/equal_points.txt
No Convex Hull points
Cannot construct Convex Hull
```

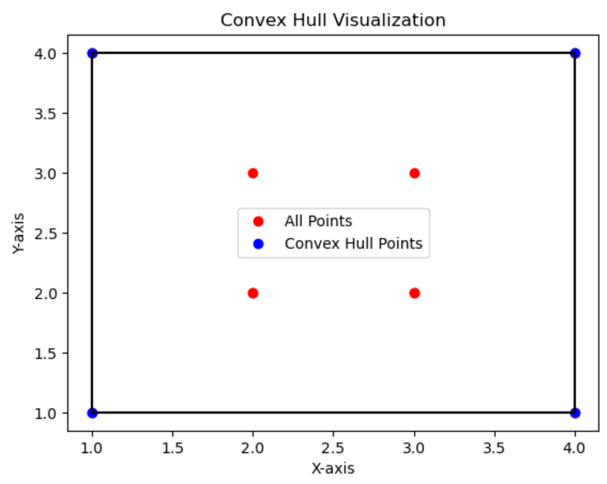
Our .dot file visualization(s):



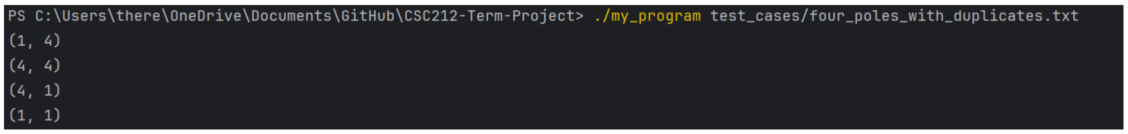
TEST CASE 4: Four Poles with Duplicate Points

Expected output:

```
constructHull('test_cases/four_poles_with_duplicates.txt')

Convex Hull Points: [(1, 4), (1, 1), (4, 1), (4, 4)]
```

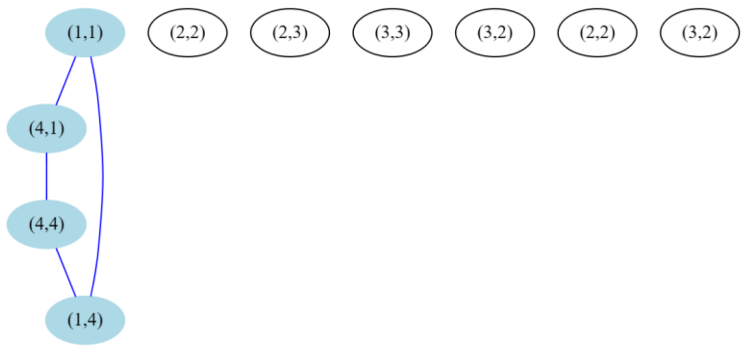


Our output:



```
PS C:\Users\there\OneDrive\Documents\GitHub\CSC212-Term-Project> ./my_program test_cases/four_poles_with_duplicates.txt
(1, 4)
(4, 4)
(4, 1)
(1, 1)
```

Our .dot file visualizations:

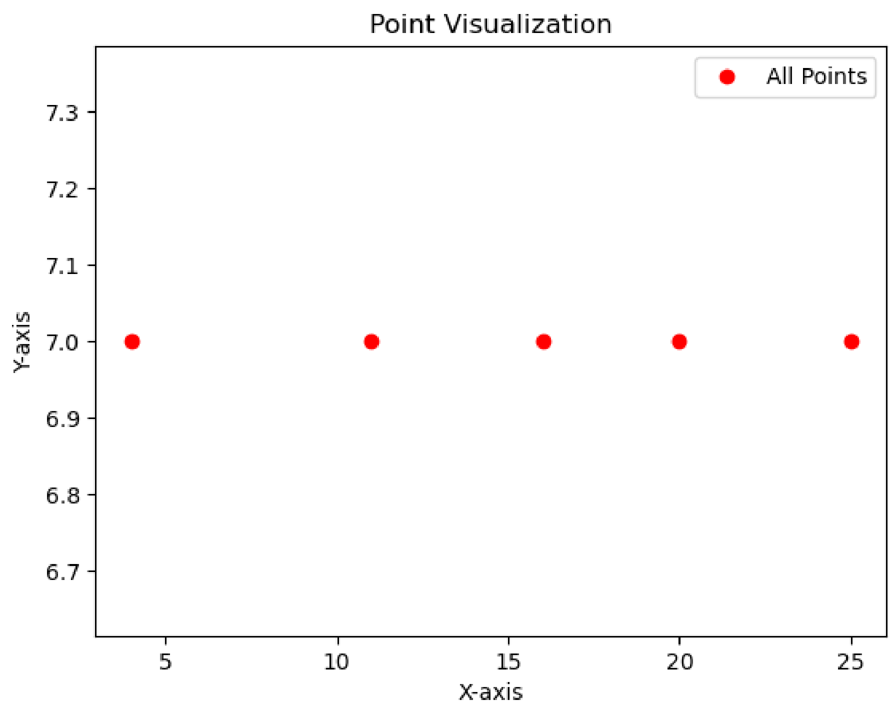


Our results match!

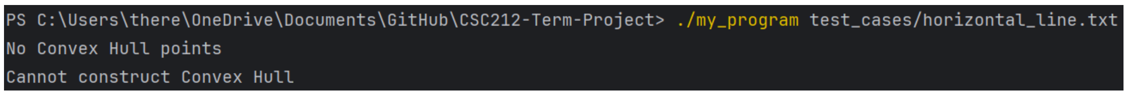
TEST CASE 5: Horizontal Line (same y values)

Expected output:

```
constructHull('test_cases/horizontal_line.txt')

Cannot construct convex hull. Plotting original points.
```

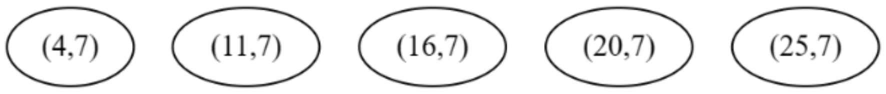


Our output:



```
PS C:\Users\there\OneDrive\Documents\GitHub\CSC212-Term-Project> ./my_program test_cases/horizontal_line.txt
No Convex Hull points
Cannot construct Convex Hull
```

Our .dot file visualization(s):

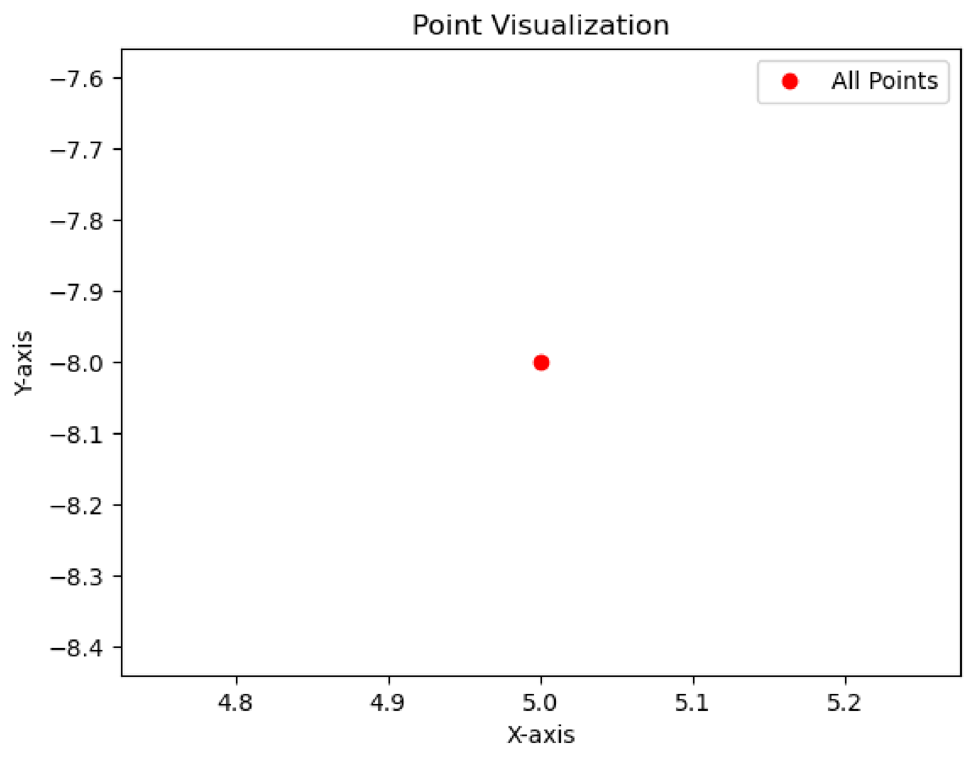


Our results match!

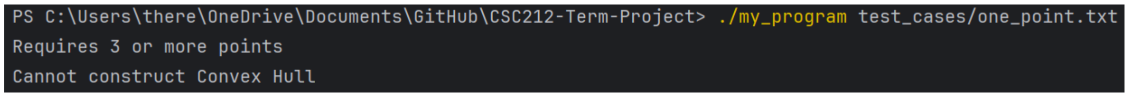
TEST CASE 6: One Point

Expected output:

```
constructHull('test_cases/one_point.txt')

Cannot construct convex hull. Plotting original points.
```

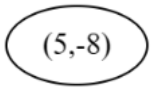


Our output:



```
PS C:\Users\there\OneDrive\Documents\GitHub\CSC212-Term-Project> ./my_program test_cases/one_point.txt
Requires 3 or more points
Cannot construct Convex Hull
```

Our .dot file visualization(s):

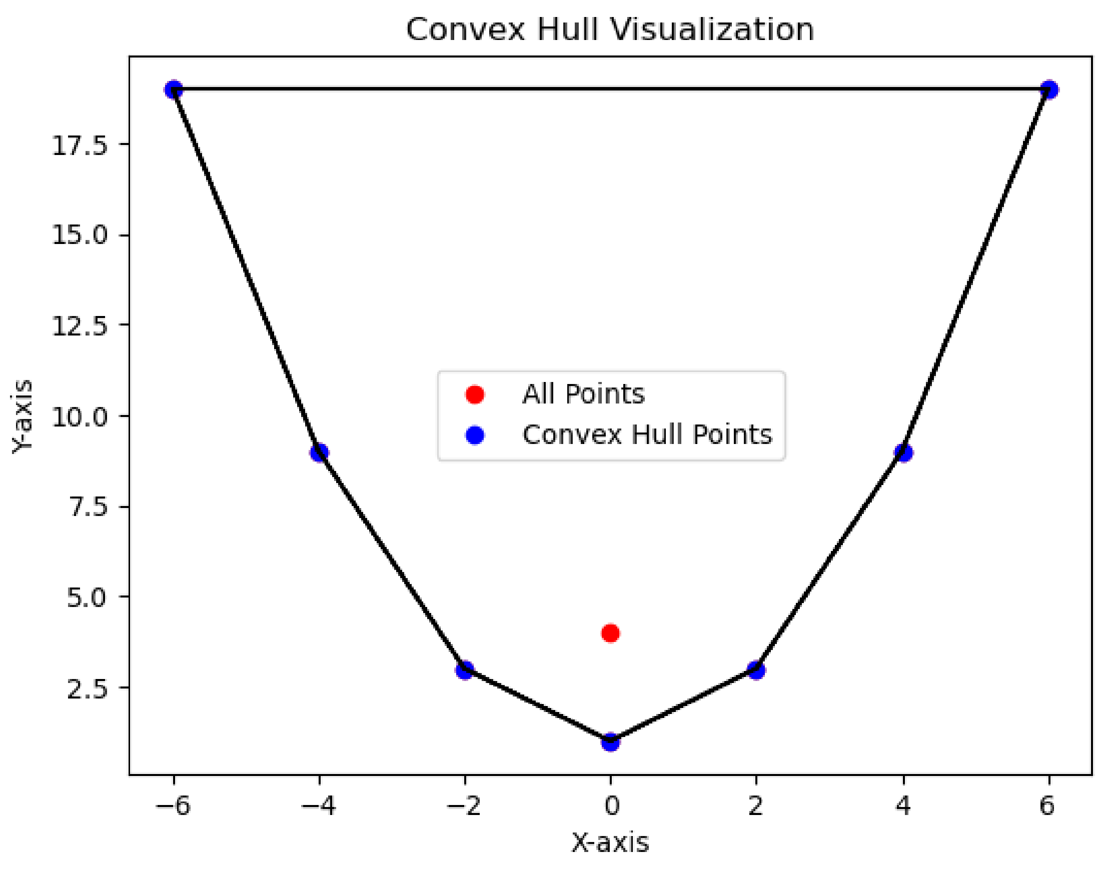


Our results match!

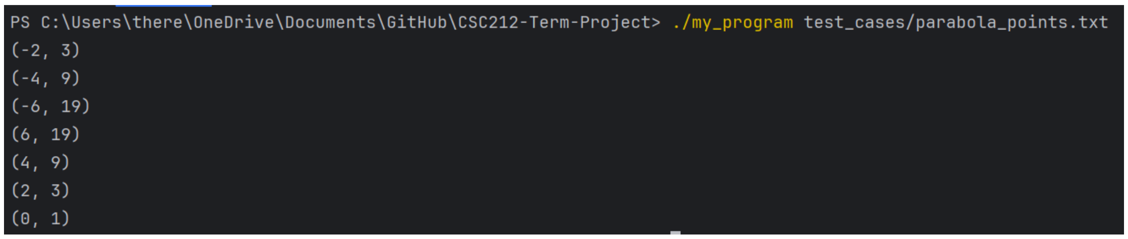
TEST CASE 7: Points On a Parabola

Expected output:

```
constructHull('test_cases/parabola_points.txt')

Convex Hull Points: [(6, 19), (-6, 19), (-4, 9), (-2, 3), (0, 1), (2, 3), (4, 9)]
```

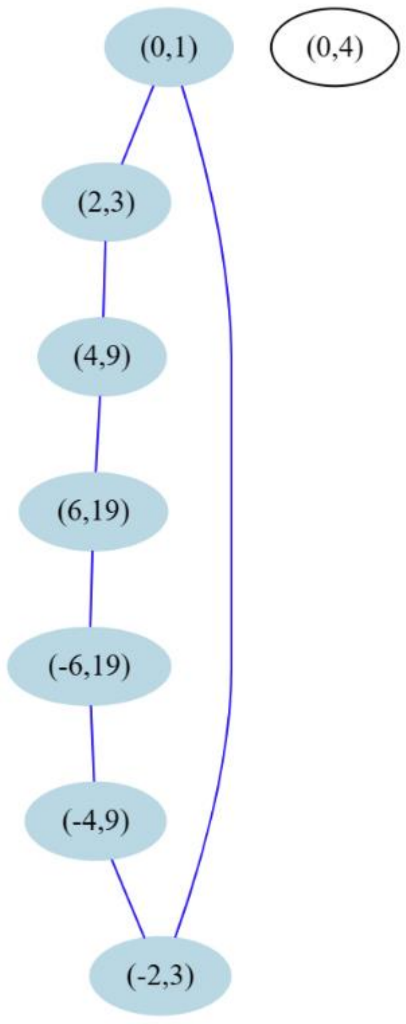


Our output:



```
PS C:\Users\there\OneDrive\Documents\GitHub\CSC212-Term-Project> ./my_program test_cases/parabola_points.txt
(-2, 3)
(-4, 9)
(-6, 19)
(6, 19)
(4, 9)
(2, 3)
(0, 1)
```

Our .dot file visualization(s):

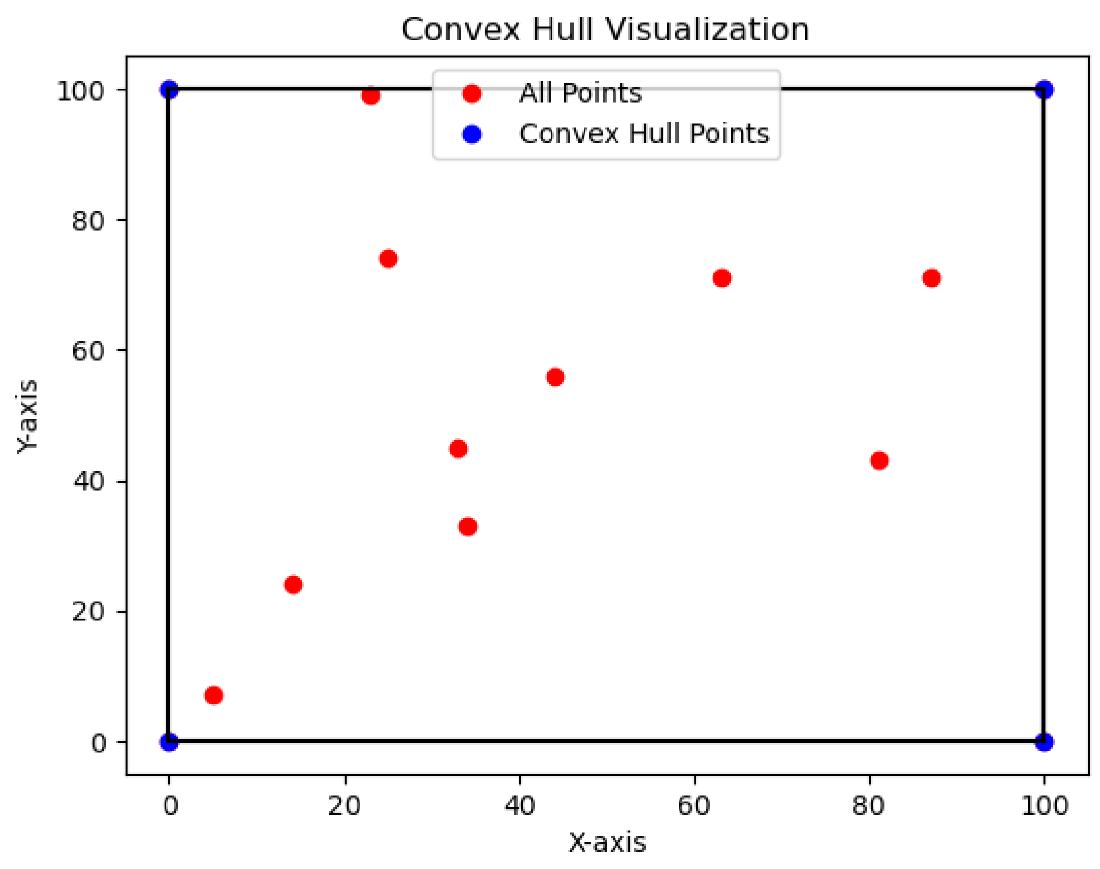


Our results match!

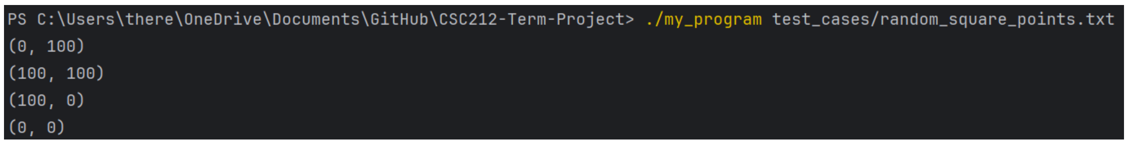
TEST CASE 8: Random Points In a Square

Expected output:

```
constructHull('test_cases/random_square_points.txt')

Convex Hull Points: [(0, 100), (0, 0), (100, 0), (100, 100)]
```

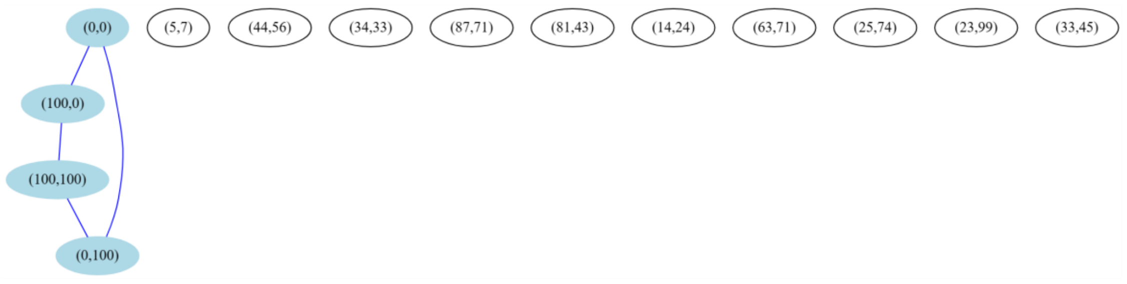


Our output:



```
PS C:\Users\there\OneDrive\Documents\GitHub\CSC212-Term-Project> ./my_program test_cases/random_square_points.txt
(0, 100)
(100, 100)
(100, 0)
(0, 0)
```

Our .dot file visualization(s):

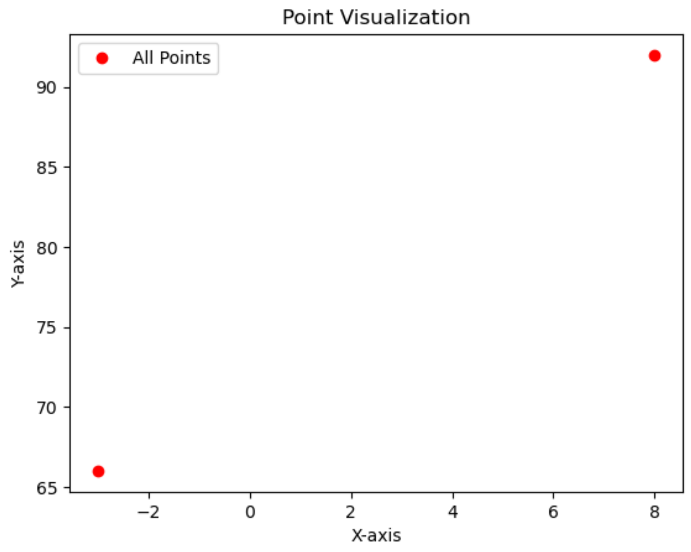


Our results match!

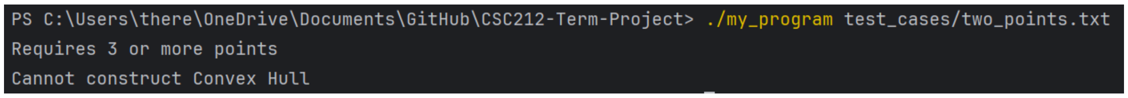
TEST CASE 9: Two Points

Expected output:

```
constructHull('test_cases/two_points.txt')

Cannot construct convex hull. Plotting original points.
```

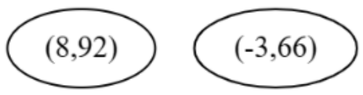


Our output:



```
PS C:\Users\there\OneDrive\Documents\GitHub\CSC212-Term-Project> ./my_program test_cases/two_points.txt
Requires 3 or more points
Cannot construct Convex Hull
```

Our .dot file visualization(s):

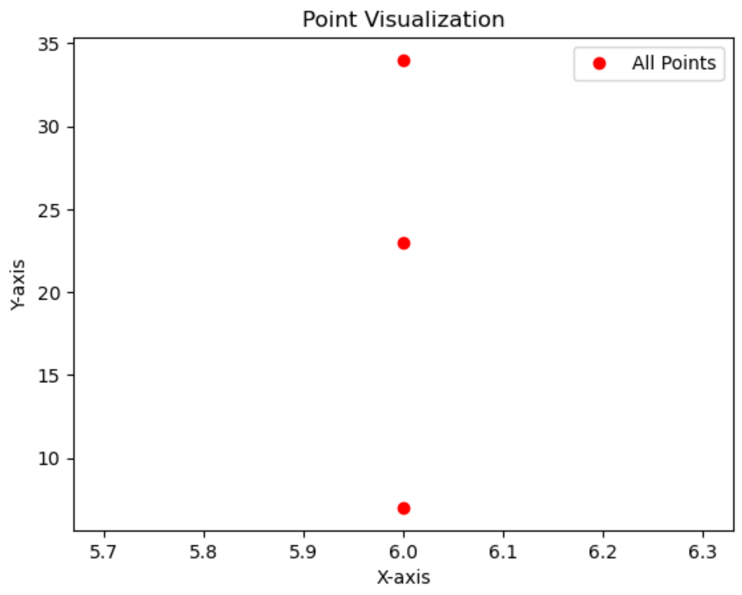


Our results match!

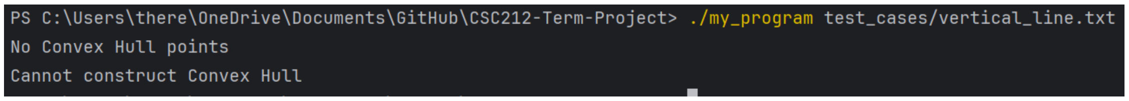
TEST CASE 10: Vertical Line (same x values)

Expected output:

```
constructHull('test_cases/vertical_line.txt')

Cannot construct convex hull. Plotting original points.
```

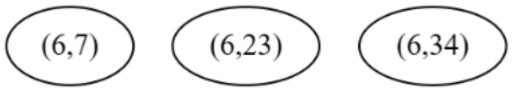


Our output:



```
PS C:\Users\there\OneDrive\Documents\GitHub\CSC212-Term-Project> ./my_program test_cases/vertical_line.txt
No Convex Hull points
Cannot construct Convex Hull
```

Our .dot file visualization(s):



Our results match!

We passed our test cases!

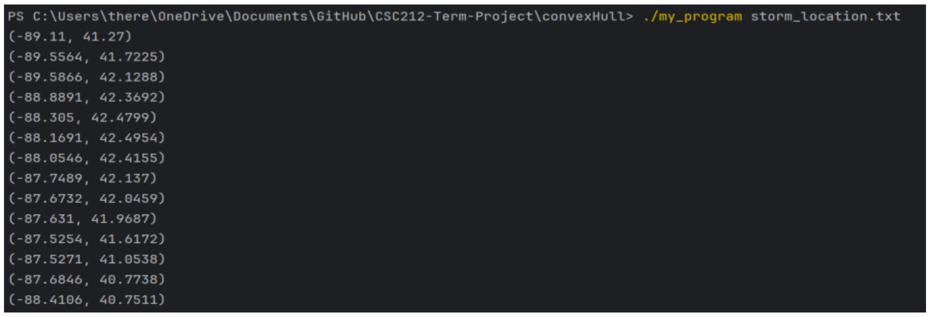
Applications

Using a convex hull to visualize the affected area of storms/natural disasters:

We have obtained a descriptive storm event dataset from the National Centers for Environmental Information [3]. Specifically, we then analyzed the impactful episode of a powerful derecho on August 10th, which traversed Iowa, northern Illinois, and northern Indiana, causing severe wind damage. In north-central and northeast Illinois, winds of 60 to 100 mph downed trees, damaged homes, and left nearly one million customers without power. The storm, fueled by high moisture and explosive instability, resulted in extensive damage and the formation of 14 tornadoes in the region.

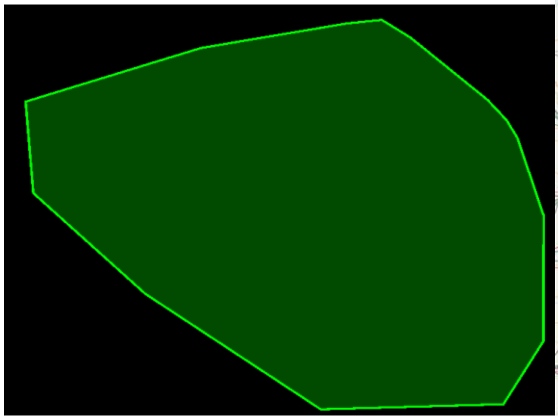
We then recorded the latitude and longitude values of each event during this episode in a text file, intending to use them to compute a convex hull and generate a corresponding visualization.

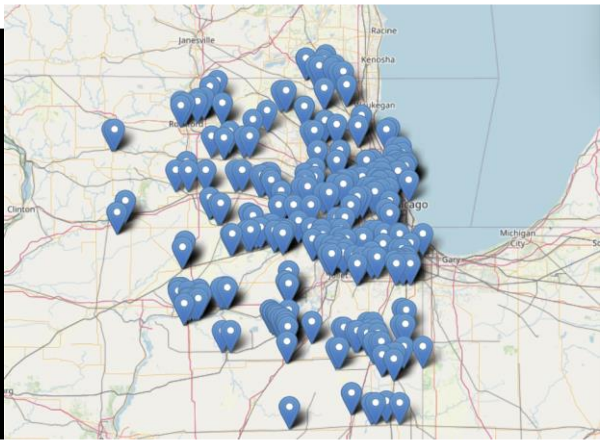
Our convex hull points:



```
PS C:\Users\there\OneDrive\Documents\GitHub\CSC212-Term-Project\convexHull> ./my_program storm_location.txt
(-89.11, 41.27)
(-89.5564, 41.7225)
(-89.5866, 42.1288)
(-88.8891, 42.3692)
(-88.305, 42.4799)
(-88.1691, 42.4954)
(-88.0546, 42.4155)
(-87.7489, 42.137)
(-87.6732, 42.0459)
(-87.631, 41.9687)
(-87.5254, 41.6172)
(-87.5271, 41.0538)
(-87.6846, 40.7738)
(-88.4106, 40.7511)
```

Our SFML visualization:



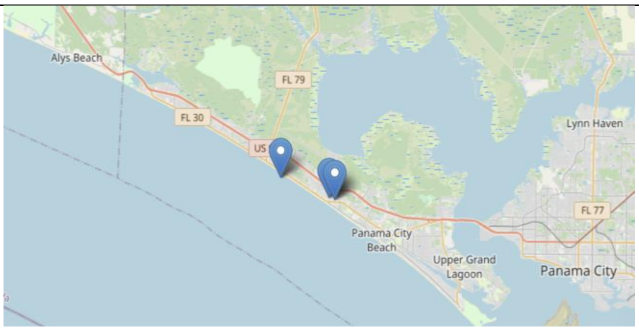
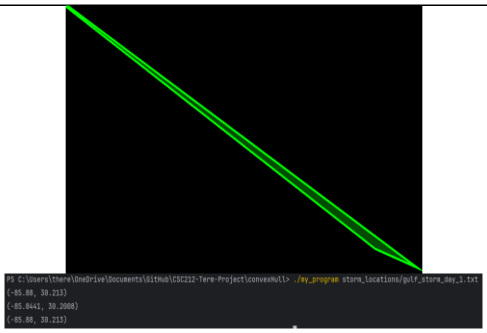


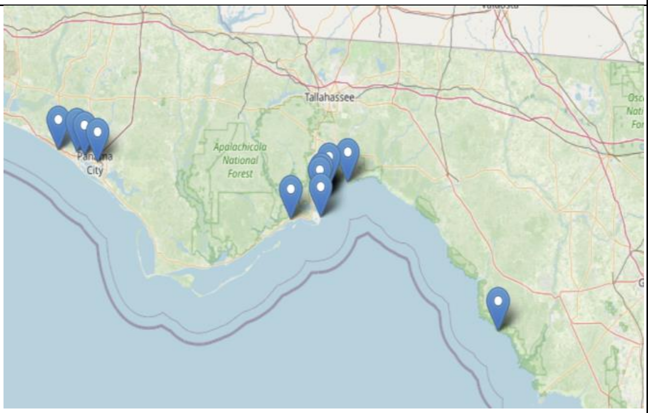
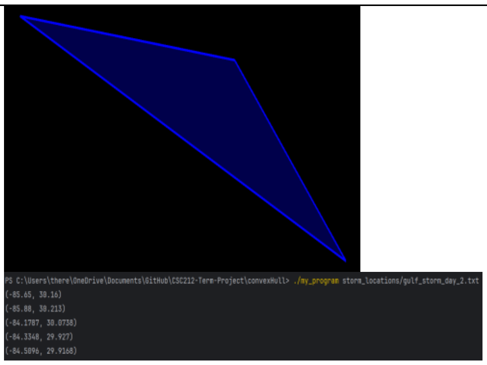
You can see how our convex hull visualization matches the location points marked on the map.

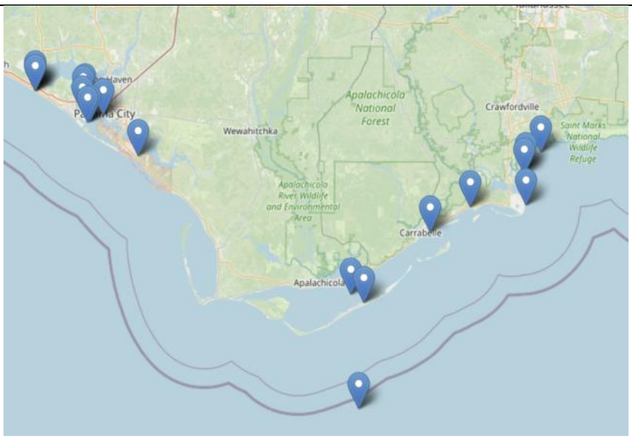
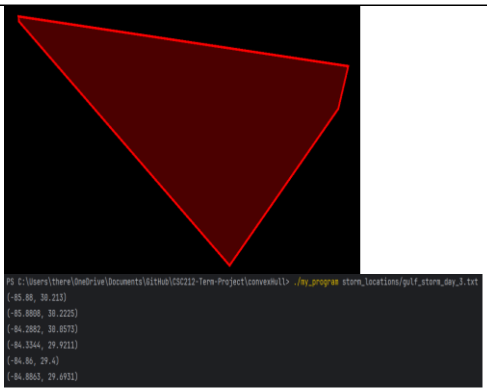
We can even visualize a storm's area of effect each day of its duration.

In July 2020, along the Gulf of Mexico were strong storms that moved across the coastal waters with wind gusts of 34 knots or higher measured.

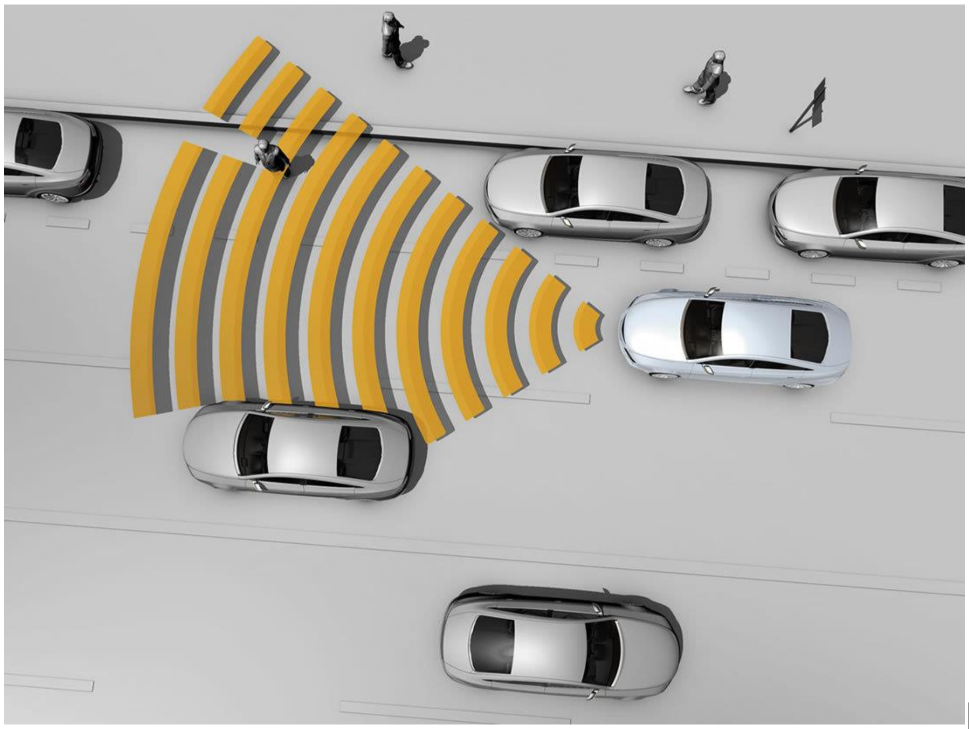
	Map	Our Output

Day 1	 Alys Beach
	FL 79
	FL 30
	US
	Lynn Haven
	FL 77
	Panama City Beach
	Upper Grand Lagoon
	Panama City
		`PS C:\Users\there\OneDrive\Documents\GitHub\CSC212-Term-Project\convexHull> ./my_program storm_locations/gulf_storm_day_1.txt`
		`(-85.88, 30.213)`
		`(-85.8441, 30.2008)`
		`(-85.88, 30.213)`

Day 2		`PS C:\Users\there\OneDrive\Documents\GitHub\CSC212-Term-Project\convexHull> ./my_program storm_locations/gulf_storm_day_2.txt`
		`(-85.65, 30.16)`
		`(-85.88, 30.213)`
		`(-84.1787, 30.0738)`
		`(-84.3348, 29.927)`
		`(-84.5096, 29.9168)`

Day 3	 Panama City
	Wewahitchka
	Apalachicola National Forest
	Crawfordville
	Saint Marks National Wildlife Refuge
	Apalachicola River Wildlife and Environmental Area
	Carrabelle
	Apalachicola
		`PS C:\Users\there\OneDrive\Documents\GitHub\CSC212-Term-Project\convexHull> ./my_program storm_locations/gulf_storm_day_3.txt`
		`(-85.88, 30.213)`
		`(-85.8808, 30.2225)`
		`(-84.2882, 30.0573)`
		`(-84.3344, 29.9211)`
		`(-84.86, 29.4)`
		`(-84.8863, 29.6931)`

Along with storm mapping, convex hull can also be useful in visualizing collision detections. Like in most cars with automated detection systems that detect whether or not something is in the radius of a car, convex hull can simulate that detection radius.

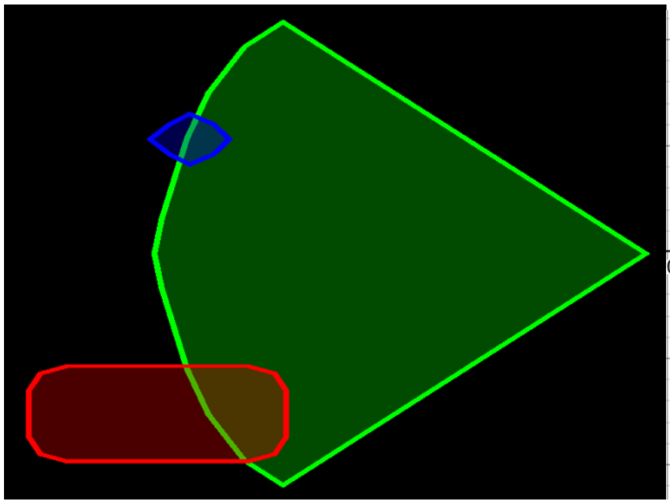

[21]

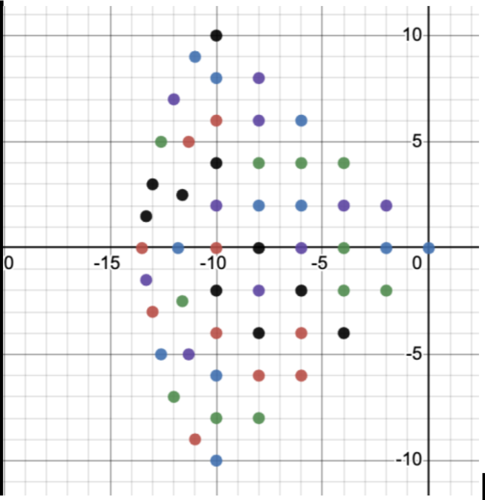
Our imputed points:

car_radius.txt	Person1.txt	car1.txt
(0,0)	(-16.6,-5)	(-10.3,-5.3)
(-2,-2)	(-18.6,-5)	(-10.3,-8.7)
(-2,2)	(-16.6,-5)	(-16,-5)
(-4,-4)	(-18.6,-5)	(-17,-6)
(-6,-6)	(-17.6,-4)	(-16,-9)
(-6,6)	(-17.6,-6)	(-17,-8)
(-8,-8)	(-17,-5.6)	(-16.7,-8.7)
(-8,8)	(-17,-4.4)	(-16.7,-5.3)
(-10,10)	(-18.1,-5.6)	(-10,-6)
(-11,-9)	(-18.1,-4.4)	(-11,-5)
(-11,9)	(-17.6,-5.6)	(-10,-8)
(-12,-7)	(-17.6,-4.4)	(-11,-9)
(-12,7)	(-18.1,-5)	(-12.3,-9)
(-13,3)	(-17,-5)	(-14.7,-9)
(-13.5,0)	(-17.6,-5)	(-13.5,-9)
(-2,0)		(-12.3,-5)
(-4,0)		(-14.7,-5)
(-6,0)		(-13.5,-6)
(-8,0)		(-13.5,-6)
(-10,0)		(-13.5,-7)
(-11.8,0)		(-13.5,-8)
(-4,-2)		(-12.3,-6)
(-4,2)		(-12.3,-7)
(-6,-2)		(-12.3,-8)
(-6,-4)		(-14.7,-6)
(-6,2)		(-14.7,-6)
(-6,4)		(-14.7,-7)
(-8,-2)		(-14.7,-8)
(-8,-4)		(-16,-7)
(-8,-6)		(-16,-8)

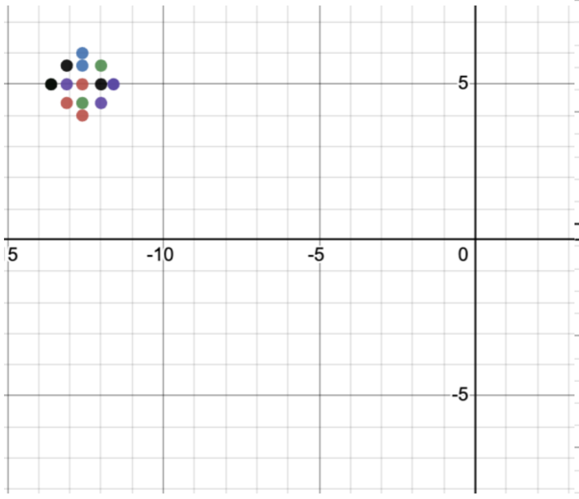
(-8,2)		(-11,-6)
(-8,4)		(-11,-7)
(-8,6)		(-11,-8)
(-10,-2)		(-17,-7)
(-10,-4)		(-10,-7)
(-10,-6)		(-13.5,-5)
(-10,-8)		(-16,-6)
(-10,2)		
(-10,4)		
(-10,6)		
(-10,8)		
(-11.6,-2.5)		
(-11.3,-5)		
(-11.6,2.5)		
(-11.3,5)		
(-12.6,-5)		
(-12.6,5)		
(-13.3,-1.5)		
(-13.3,1.5)		
(-13,-3)		
(-10,-10)		
(-4,4)		

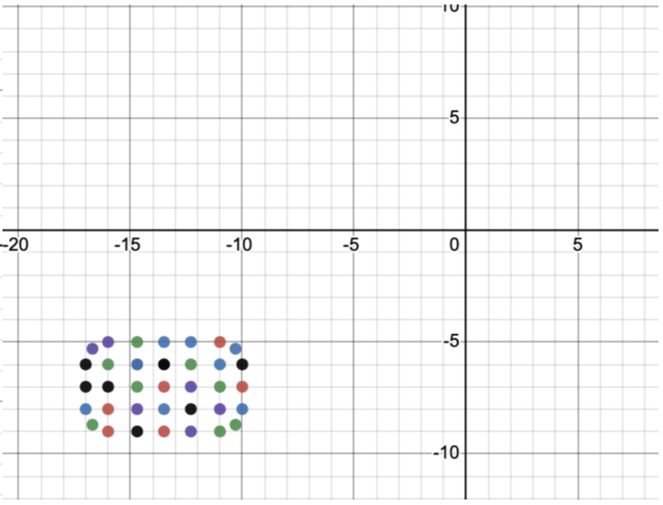
Our SFML visualization:





[22]





As you can see our convex hull visualization matches the location of the points marked on graphs using Desmos[22] and produces the combined desired output which can visualize the collision detection within the radius of the sensor.

CONTRIBUTIONS:

Group Member Contributions:

Alex Lupo	Jackson Perry	Jake Nicynski	Michael Marsella
convexHull.cpp (version 1) convexHull.cpp (version 1)	Dot file function for Graphviz visualizations	convexHull.cpp (version 2) convexHull.h (version 2)	ConvertToPointORIG.cpp
convexHull.cpp (version 3) convexHull.h (version 3)	Designed and performed test cases	convexHull.cpp (version 3) convexHull.h (version 3)	Assisted w/ development of convex hull.cpp & .h, slight revision to graph.cpp
convexHull.cpp (version 4) convexHull.h (version 4)	convexHull.cpp versions 3 & 4 revisions	convexHull.h (version 4) convexHull.h (version 4)	ConvexHullstart.cpp & Convexhull(noSMFL).cpp & .h
graph.cpp graph.h	Report, instructions,	graph.cpp graph.h	Report and slideshow

(SFML)	and slideshow	(SFML)	
convexHull.cpp convexHull.h main.cpp	convexHull.cpp convexHull.h main.cpp	convexHull.cpp convexHull.h main.cpp	
Report, instructions, and slideshow		Report, instructions, and slideshow	

CONCLUSION:

Through our research in convex hulls, we have determined the significant ability of convex hulls to solve realistic world problems. As conducted by the test cases, we have visualized a convex hull's ability to easily form a polygon given a certain number of points. This information can be incredibly useful. For example, as visualized, tracking the impact of a storm day by day can be done through convex hulls. Additionally, collision avoidance systems can be implemented through a convex hull. Convex hull's practicality can also be extended to establishment of borders for land plotting or involved with determining the area of natural aspects of mother nature such as waterfalls, and tornados. Overall, convex hull's ability to solve problems as a data structure can be enacted in daily life.

Bibliography

[1] Lennon, Michael. “An Overview to Graham Scan”. Medium, Medium, 5 April 2021,

https://medium.com/smucs/an-overview-of-the-graham-scan-e6ecc25c30da

[2] Upadhyay, Amit. “What is the logic behind the Graham Scan’s algorithm for convex hull?”

Quora, 2017. https://www.quora.com/What-is-the-logic-behind-Grahams-scan-algorithm-for-convex-hull

Storm dataset source: [3] https://www.ncdc.noaa.gov/stormevents/

[4] Sommer, Pascal. “A Gentle Introduction to the Convex Hull Problem.” Medium, Medium, 19 June 2020, medium.com/@pascal.sommer.ch/a-gentle-introduction-to-the-convex-hull-problem-62dfcabee90c

[5] “Convex Hull.” Convex Hull - an Overview | ScienceDirect Topics, www.sciencedirect.com/topics/mathematics/convex-hull Accessed 4 Dec. 2023.

[6] “Convex Hull.” Brilliant Math &amp; Science Wiki, brilliant.org/wiki/convex-hull/. Accessed 4 Dec. 2023.“Convex Hull.” From Wolfram MathWorld, mathworld.wolfram.com/ConvexHull.html. Accessed 4 Dec. 2023.

[7] “An Investigation of Graham’s Scan and Jarvis’ March.” Chris Harrison | ConvexHull, www.chrisharrison.net/index.php/Research/ConvexHull Dec. 2023.

[8] https://web.eecs.utk.edu/~roffutt/files/sp20ppts/Convex%20Hull%20Algorithms.pdf

[9] "Download Clion." *JetBrains*, JetBrains, 1 June 2021, www.jetbrains.com/clion/download/#section=mac.

[10] *Homebrew*, brew.sh/. Accessed 4 Dec. 2023.

[11] "Download." *Download (SFML)*, Laurent Gomila, www.sfml-dev.org/download.php. Accessed 4 Dec. 2023.

[12] *Graphviz*, The Graphviz Authors, graphviz.org/. Accessed 4 Dec. 2023.

[13] "Category:Animated GIF Files." Wikimedia Commons, commons.wikimedia.org/wiki/Category:Animated_GIF_files. Accessed 4 Dec. 2023.

[14] "Convex Hull Using Divide and Conquer Algorithm." GeeksforGeeks, GeeksforGeeks, 29 Oct. 2023, www.geeksforgeeks.org/convex-hull-using-divide-and-conquer-algorithm/

[15] Example of a Two-Tone (Aka 'Mooney') Image. on First ... - Researchgate www.researchgate.net/figure/Example-of-a-two-tone-aka-Mooney-image-On-first-viewing-this-image-appears-as-a_fig4_326466342 Accessed 4 Dec. 2023.

[16] Example Convex Hull Algorithm (Bradski &amp; Kaehler, 2008) B Pothole Model ..., www.researchgate.net/figure/Example-Convex-Hull-Algorithm-Bradski-Kaehler-2008-B-Pothole-model_fig2_279538022 Accessed 4 Dec. 2023.

[17] A Convex Hull with Elastic Band Analogy - Researchgate, www.researchgate.net/figure/A-convex-hull-with-elastic-band-analogy_fig3_319459132 Accessed 4 Dec. 2023.

[18] "Geoboard." Wikipedia, Wikimedia Foundation, 25 Apr. 2022, en.wikipedia.org/wiki/Geoboard.

[19] Olawanletjoel. "Big O Cheat Sheet – Time Complexity Chart." freeCodeCamp.Org, freeCodeCamp.org, 10 Apr. 2023, www.freecodecamp.org/news/big-o-cheat-sheet-time-complexity-chart/.Graham-Scan Algorithm Flow Diagram Algorithm 2:Graham-Scan(Q) 1 Let Be ...,

[20] www.researchgate.net/figure/GRAHAM-SCAN-algorithm-flow-diagram-Algorithm-2GRAHAM-SCANQ-1-Let-be-p0-be-the-point-in_fig1_338644159 Accessed 4 Dec. 2023.

[21] "Convex Hull." Brilliant Math &amp; Science Wiki, brilliant.org/wiki/convex-hull/. Accessed 4 Dec. 2023.

[22] "Graphing Calculator." Desmos, www.desmos.com/calculator. Accessed 4 Dec. 2023.