Project Documentation

Andrei Suba, Silaghi-Fartan Stefan
West University of Timișoara,
Timișoara, Romania

Abstract

In this paper, we present a small application written in Python that allows a user to interactively create points in two-dimensional space using their mouse. The application then computes the convex layers of the resulting point set and displays the resulting figure. The goal of this project was to develop a simple and intuitive tool for visualizing and understanding the concept of convex layers in a geometric context. The application is implemented using a combination of Python's built-in graphical user interface (GUI) library, Tkinter, and the Jarvis's Algorithm (or Gift Wrapping Algorithm) for computing the convex hull of points. We discuss the design and implementation of the application, as well as its potential applications in educational settings and beyond.

Introduction

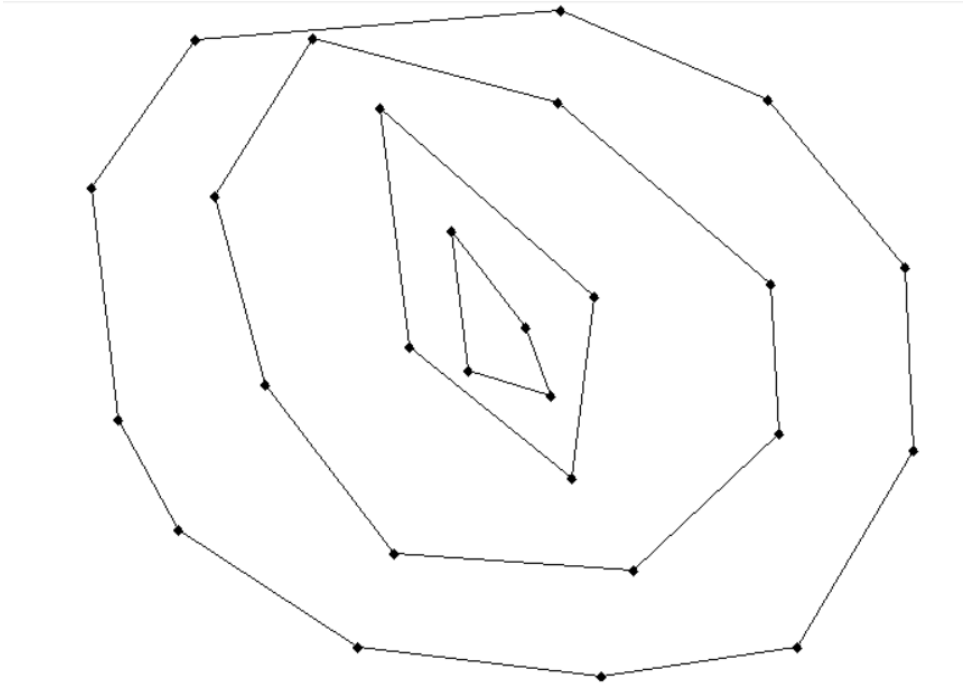
Computational geometry is a crucial field in computer science that deals with the efficient representation and manipulation of geometric objects. One important concept in this field is the convex layer, which refers to the smallest convex polygon that encloses a given set of points in two-dimensional space. In this paper, we present a small application that allows users to interactively create points using their mouse and visualize the resulting convex layers in real-time. The application was developed using Python and makes use of the Tkinter library for the graphical user interface and the Jarvis's Algorithm for computing convex hulls. Our goal in creating this application was to provide a simple and intuitive tool for visualizing and understanding convex layers in a geometric context. In the following sections, we will describe the design and implementation of the application in detail and discuss its potential applications and future directions.

Motivation

As part of our university course on computational geometry, we were required to develop a project that demonstrated our understanding of the subject. We decided to focus on convex layers, as we felt that this was an important concept in the field. Our goal was to develop a simple and intuitive tool that would make the concept of convex layers more accessible and engaging for students and other learners. We believe that our small application has the potential to be a useful resource in educational settings where visualization makes for a better understanding. We hope that our project will provide a valuable learning experience for both us and our users.

Problem Description

The convex layers of a set of points in a plane are a sequence of nested convex polygons having the points as their vertices. In order to achieve the convex layers of a set of points, first you must compute the convex hull of the given points. The next step is to ignore the points that are already connected and compute another convex hull of the remaining points. The process continues until there are no more points left to compute a convex hull for. In the end you will have a multitude of convex hulls named convex layers. The problem of constructing convex layers has also been called onion peeling or onion decomposition.



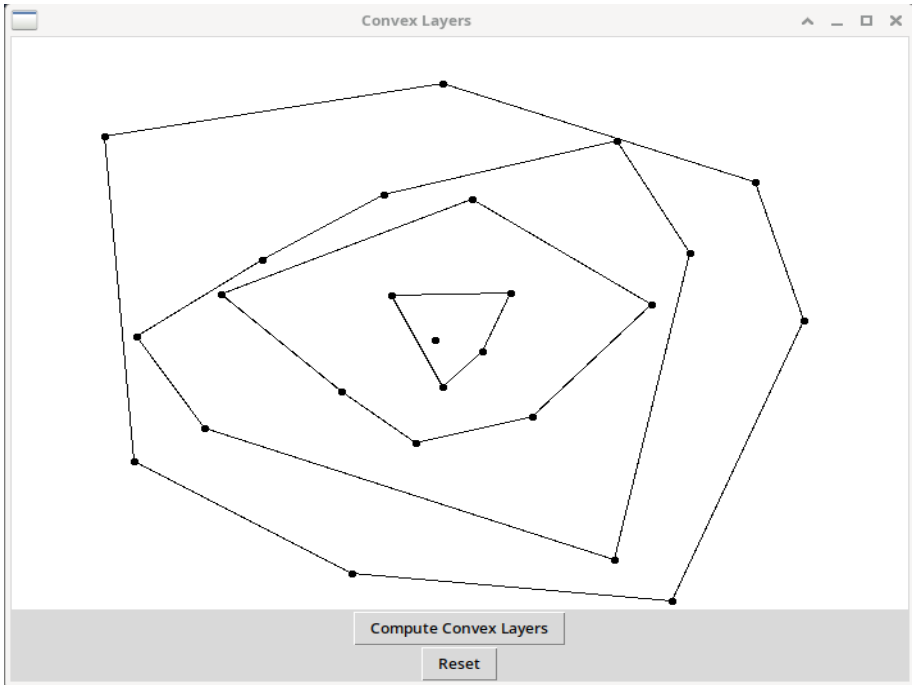
Design and Implementation

Introduction

In this chapter, we will discuss the technical details of the program that constructs the convex layers of a set of points. The user interface design, algorithm implementation and data structures will be explained. The goal of this chapter is to provide a clear understanding of the program's design and implementation and how they contribute to the program's overall functionality.

User Interface

The user interface of the Convex Layers program is designed to be simple. The main component of the interface is a canvas on which the user can draw points by clicking. The program also includes two buttons: one for calculating the convex hull of the points (`Compute Convex Layers`), and the other for clearing the canvas and starting over (`Reset`). The layout of the interface is designed to minimize clutter and maximize the visibility of the points. The canvas takes up most of the window, with the buttons located at the bottom. The color scheme is designed to be easy on the eyes, with black points and lines on a white background. To use the program, the user simply uses his mouse/touchpad and clicks on the canvas to add points. Once they are satisfied with the number of points, they can press the "`Compute Convex Layers`" button to see the convex layers of those points. If they make a mistake or want to start over, they can press the "`Reset`" button to delete all points and lines from the canvas.



Algorithm Implementation

The Convex Layers program implements the Jarvis March algorithm, also known as the Gift-wrapping algorithm, to calculate the convex hull of a set of points. The algorithm was first proposed by R. A. Jarvis in his article "On the identification of the convex hull of a finite set of points in the plane" (Jarvis, 1973). The basic idea behind the algorithm is to start with the leftmost point and then repeatedly move counterclockwise to the next point that is on the convex hull. This is done by considering all other points and choosing the one that is most counterclockwise from the current point. The algorithm then proceeds with this new point and continues until it reaches the starting point again. In our implementation, the algorithm starts by finding the leftmost point of the set of points. Then, it uses the orientation function to determine the orientation of three points, which is the current

point, the next point, and another point in the set. The function returns 0 for collinear points, 1 for clockwise orientation and 2 for counterclockwise orientation. Using this function, the algorithm finds the next point that is most counterclockwise from the current point. The algorithm continues this process until it reaches the starting point again, and in this way, it constructs the convex hull of the points.

Data Structures

The program utilizes several data structures to store and manipulate the points. The primary data structure is a list of `Point objects`, which store the x and y coordinates of each point. This list is used to store all the points that the user draws on the canvas.

```
# Point class with x, y as point
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

# Create a list to hold the points drawn by the user
listPoints = []

# Function to display the point and add it to listPoints
def draw_line(event):
    x1=event.x
    y1=event.y
    x2=event.x
    y2=event.y
    listPoints.append(Point(x1, y1))
    # Draw an oval in the given co-ordinates
    canvas.create_oval(x1,y1,x2,y2,fill="black", width=5)
```

Additionally, the program uses a list of integers to store the indices of the points that make up the convex hull. This list is used to draw the lines that make up the convex hull once it has been calculated. The program also uses a copy of the `listPoints` named `holder_listPoints` to keep track of the points that are not in the convex hull and rules them out. This is used to draw the new convex hull once the program is run multiple times. This way the program keeps track of the points that are not in the current convex hull.

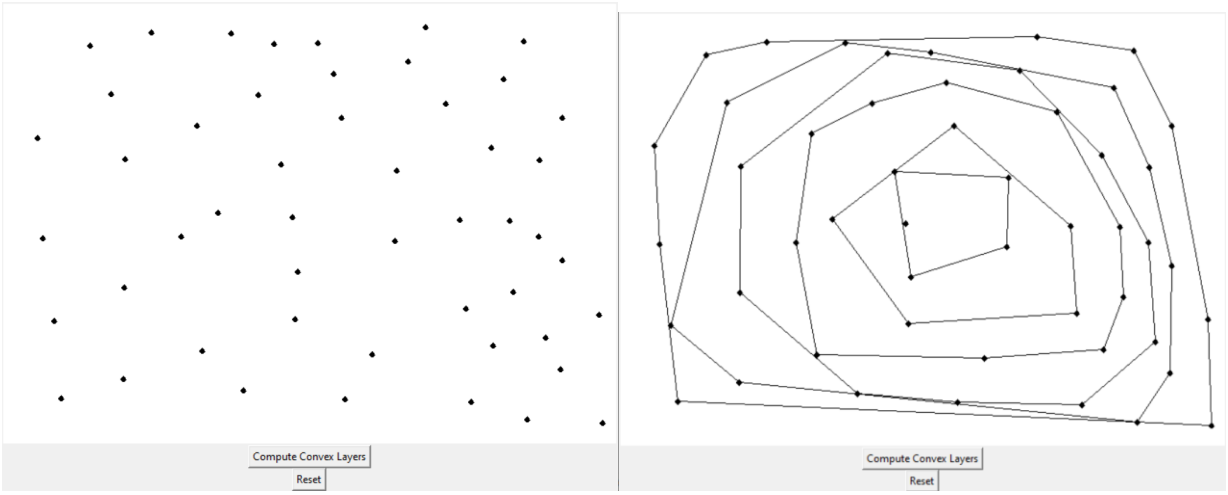
```
# Make the copy
holder_listPoints = []
holder_listPoints = listPoints.copy()
for index in range(len(hull)):
    if index != (len(hull) - 1):
        canvas.create_line(listPoints[hull[index]].x,\
        listPoints[hull[index]].y,\
        listPoints[hull[index + 1]].x,\
        listPoints[hull[index + 1]].y)
        holder_listPoints.remove(listPoints[hull[index]])
    else:
        canvas.create_line(listPoints[hull[index]].x,\
        listPoints[hull[index]].y,\
        listPoints[hull[0]].x,\
        listPoints[hull[0]].y)
        holder_listPoints.remove(listPoints[hull[index]])
listPoints = holder_listPoints.copy()
convexHull_2(listPoints, len(listPoints))
```

The data structures are implemented in such a way that they are easy to update and manipulate as the user adds and removes points. The use of the `Point class` and the list of indices also allows for easy integration with the Jarvis March algorithm for convex hull calculation, as described in Jarvis (1973) .

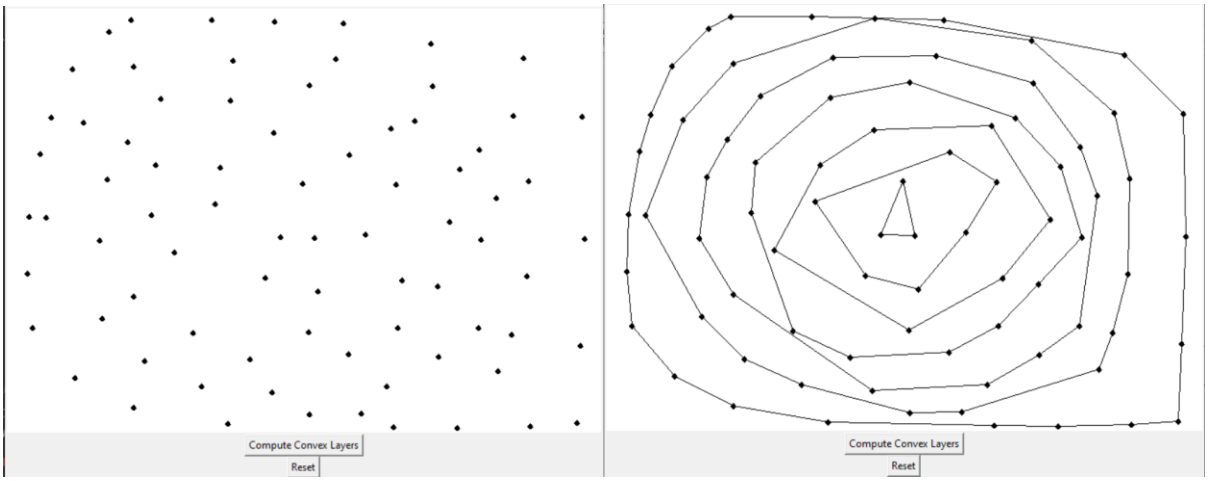
Results

If we run the program by using any Phyton compiler and input any number of points and then hit the “Compute Convex Layers” button, we see that the program automatically computes the convex layers of the points.

Example 1: Convex layers for 50 points



Example 2: Convex layers for 75 points



A bad case of our algorithm is in the case that a convex hull needs to be computed for only 2 points. In this case our algorithm skips the points.

Example 3: Convex layers error

Applications

An early application of the convex layers was in robust statistics (statistics with good performance for data drawn from a wide range of probability distributions), as a way of identifying outliers and measuring the central tendency of a set of sample points. In this context, the number of convex layers surrounding a given point is called its convex hull peeling depth, and the convex layers themselves are the depth contours for this notion of data depth.

Conclusion

Our job was to develop a project in which we implement an algorithm for a computational geometry topic of our choice. The application we made is computing the convex layers of a set of points chosen by the user, by computing multiple convex hulls of the points. We are thrilled by our accomplishment of realizing a code like this and we hope for many more successes in our programming future.