CS633 Project Final Report

Author: Jeff Scanlon, Date: 12/18/2019

Abstract

A tool is developed and detailed to provide an approach to visualization and analysis of computational geometry algorithms. The intent of this tool is to provide a way for professors and students to better explore complex algorithms. Additionally, a tool such as this can help explore degenerate cases and why certain configurations are difficult or impossible for certain algorithms to process.

The code for this is available at: https://github.com/jsgmu/cs633

Motivation and Introduction

This project focuses on development of a tool to assist in the visualization and understanding of computational geometry algorithms. An example image of this tool in action:

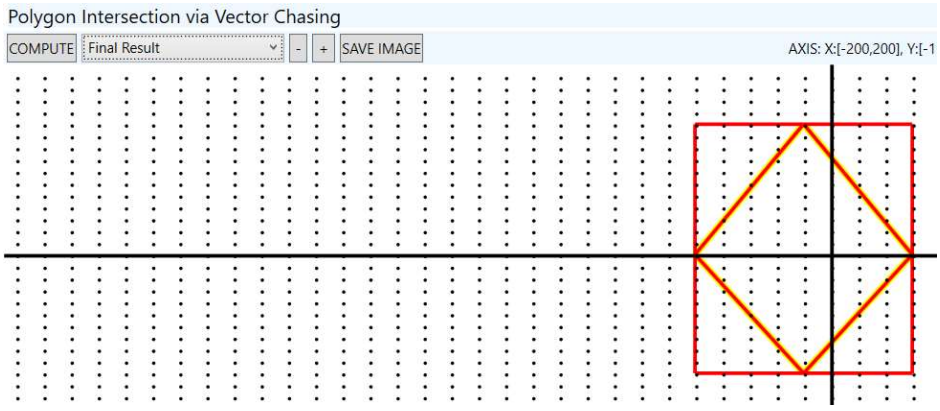

Fig 1: Final Result of Convex Polygon Intersection Algorithm

The key objectives of this project were to build a way to construct arbitrary demonstrations of computational geometry algorithms and to provide a way to explore these algorithms in more detail. The most significant challenge of this project was balancing development of the core tool itself, which is a structure upon which the algorithm analysis rests, and implementing, exploring and instrumenting (to work with the tool) various computational geometry algorithms.

This project features the development of a tool to visualize computational geometry algorithms. This project has two main objectives. First, it provides a framework for exploring multiple computational geometry algorithms in closer detail. Second, it attempts to bring a practical solution for studying these algorithms in a generalized way. Ultimately, planned features for the tool had to be cut, as well as cutting several computational geometry algorithms that had been planned, in order to bring the rest of the project to a reasonable completion. While not as complete as hoped, what is implemented does a strong job of showing what a tool like this can do to help professors build demonstrations for class, and to help students peel back the layers of an algorithm and understand it in more detail.

Methods Used and Studied

The presentation of this tool featured a smaller number of algorithms than were explored during its implementation. The featured algorithms are:

- Graham Scan
- Bowyer-Watson Delaunay Triangulation
- Convex Polygon Intersection via Vector Chasing

Algorithms explored but not fully implemented:

- Bounding Box via Rotating Calipers
- Delaunay Triangulation via Delaunator

Graham Scan

The implementation of Graham Scan was straightforward and assisted greatly in developing the rest of the tool. This algorithm has efficiency O(n log n), constrained by the initial requirement to sort the set of points by the angle they make with a determined lowest point and the x-axis.

As an example, the set of points can be chosen in the tool:

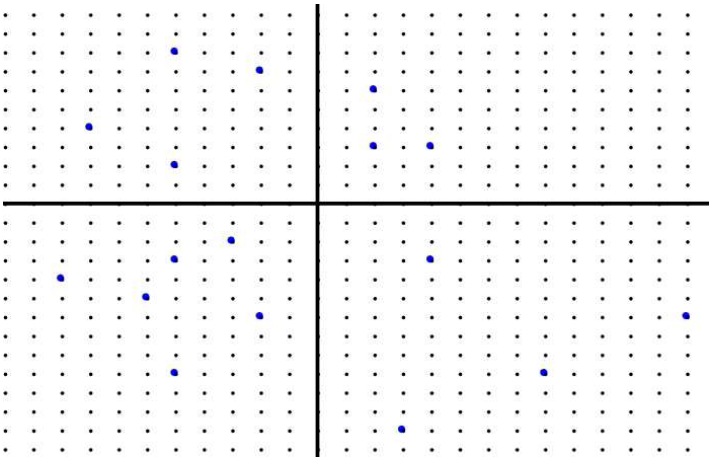

Fig 2: Initial set of points for Graham Scan demonstration

The algorithm then starts at the point with the lowest Y coordinate, in this case the point is in quadrant 4 and does not require a tie-breaker with X coordinate.

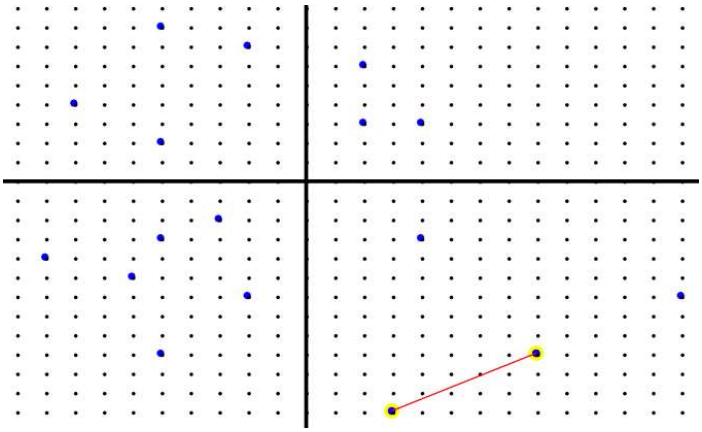

Figure 3: Beginning of Graham Scan demonstration execution

The tool also displays a snapshot of the code actually used to implement the algorithm, although completely stripped of noise like instrumentation, commented code, etc.

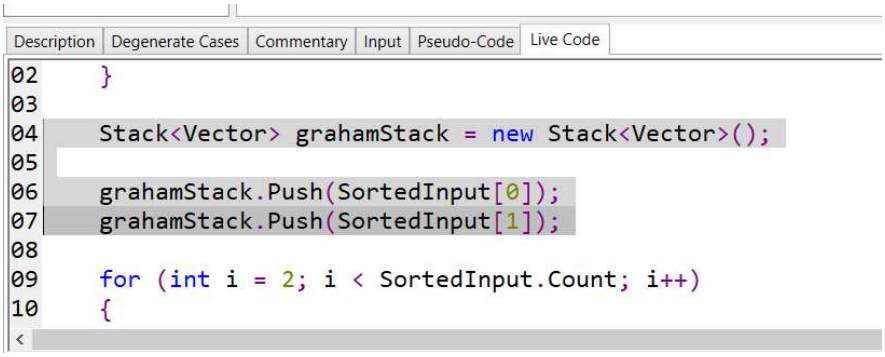

Figure 4: Highlighted code corresponds to status of algorithm in Figure 3

The main execution of this algorithm is captured in Figures 5 and 6, showing how the Graham Scan considers a point as part of the convex hull and then makes the determination the point actually isn't part of the hull and remedies this before continuing.

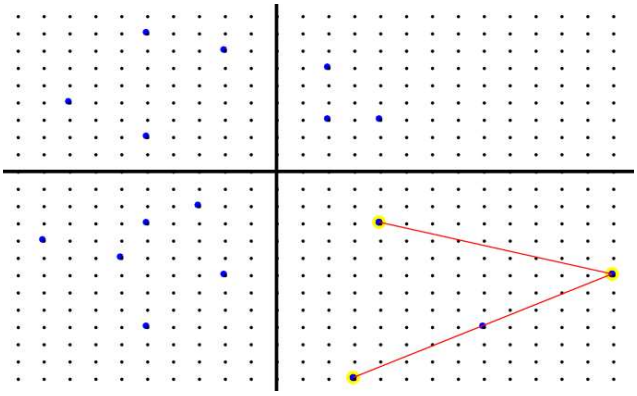

Figure 5: Graham Scan considering the next point

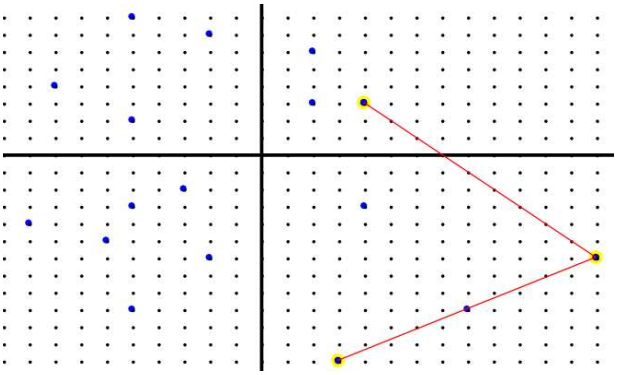

Figure 6: Graham Scan rejecting the point in Figure 5 and evaluating next point

Since the points are sorted by the angle they make with the lowest point and the x-axis, the evaluation of points is approximately that of a counter-clockwise sweep. When this particular algorithm is complete, the convex hull is formed, as shown in Figure 7.

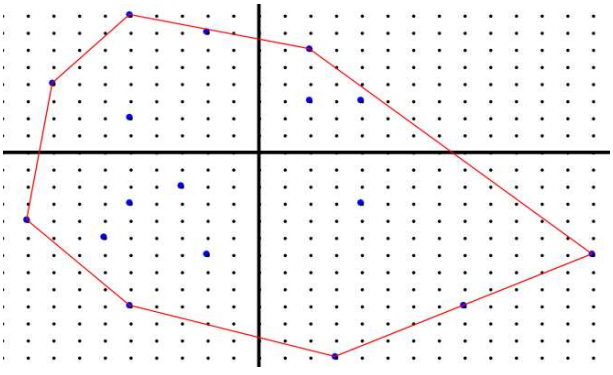

Figure 7: Completion of the Graham Scan demonstration

Bowyer-Watson Delaunay Triangulation

The second algorithm implemented in this tool is the Bowyer-Watson Delaunay triangulation algorithm. This is an incremental algorithm that adds points to a pending triangulation. Any time a point is part of a circumcircle of the points of an existing triangle, a re-triangulation must be performed. This can be seen in Figures 8 and 9.

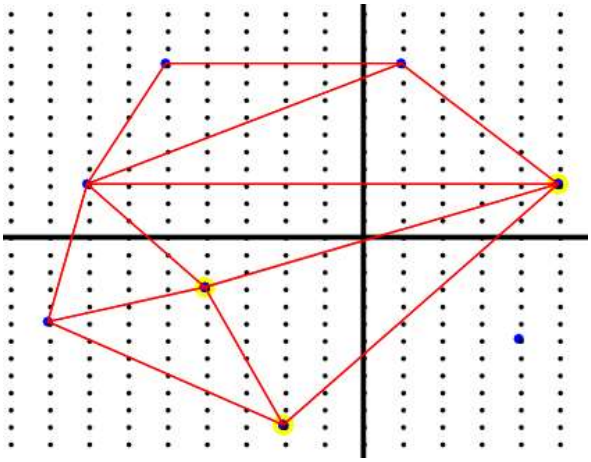

Figure 8: Points with yellow highlights show a circumcircle violation with next input point

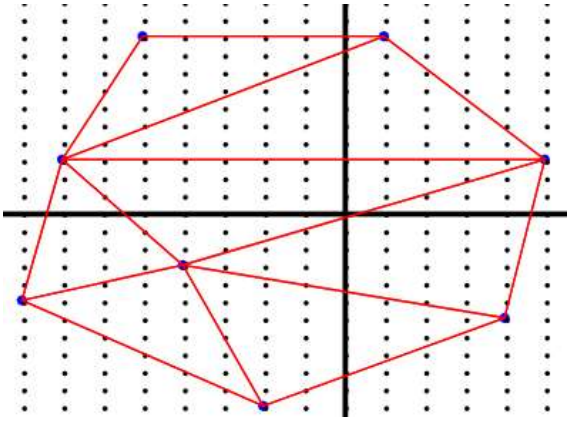

Figure 9: Algorithm breaks the triangle into two new ones, connecting to final input point

While not currently implemented, there are two aspects of this algorithm that could be added to this tool to improve its visualization. First, the point set is enclosed by a special, temporary super-triangle that serves to make algorithm easier to implement with handling of points on the hull. This can lead to situations where there is one or two points highlighted, instead of three, which can be confusing. Second, the circumcircles themselves could be drawn, in order to visualize this aspect of the algorithm. Despite these deficiencies, the operation of the algorithm demonstrates the Delaunay triangulation step-by-step.

"Vector Chasing" Convex Polygon Intersection

This is the most intricate and most interesting algorithm implemented as part of this tool. The "vector chasing" algorithm was developed by Joseph O'Rourke and his undergraduate students in 1982. The impressive aspect of this algorithm is that it can be done in linear time, specifically O(n + m) where n and m are the number of vertices of each of the polygons. The main operation of the algorithm involves taking one edge of each of the two polygons, giving it a direction so it becomes a direction vector, and then the vectors trace the outsides of the polygons. Deciding which vector to advance and inferring the polygon intersection from these two vectors provides the rest of the core operation of the algorithm.

The most important part of the algorithm is determining which vector to advance. The cross product is used to calculate a vector at right angles to both of the vectors on the polygons, and whether this vector is pointing out or in (or has no direction) gives us half of the information needed to determine which vector to advance. The other part of the required information is whether the head of one of these vectors is in the open halfplane of the other polygon's vector. Specifically, a vector is advanced in the cases of: (1) the cross product of the two vectors is greater than zero and the head of the vector is in the other's halfplane; or (2) the cross product is less than zero and the head of the vector is not in the other's open halfplane.

The special cases for this algorithm involve overlap of the vectors when they are in opposite directions; when the vectors are parallel with the head point located to the right of each other; and when the vectors are collinear. For the last collinear case, the point is not part of the intersection.

The polygons shown in Figure 10 serve as input for a demonstration so we can take a look at this algorithm in action. The intersection expected for this configuration is the entirety of the inner polygon.

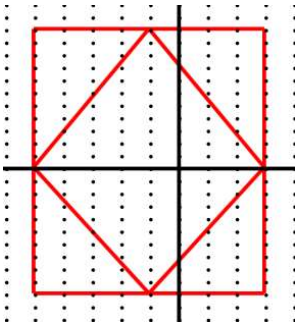

Figure 10: Two convex polygons, input to the "vector chasing" algorithm

In Figure 11, the blue lines indicate the direction vectors that trace the outsides of their polygons in a counter-clockwise direction.

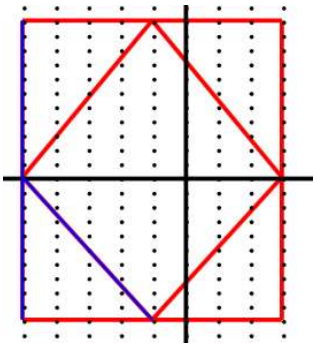

Figure 11: Blue lines indicating the direction vectors of this algorithm

One improvement to this tool here would be adding arrowheads, or at least a highlighted point, indicating the direction of the vector. Lacking this part of the visualization definitely leaves out a major part of this algorithm, but nonetheless stepping through the algorithm demonstrates how the vectors trace the edges.

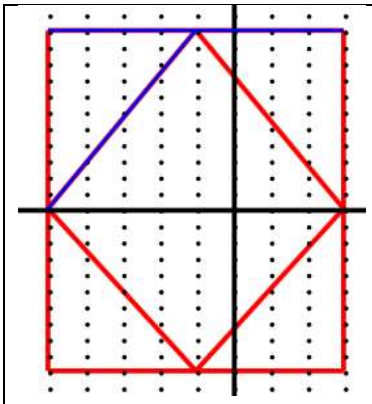

Figure 12

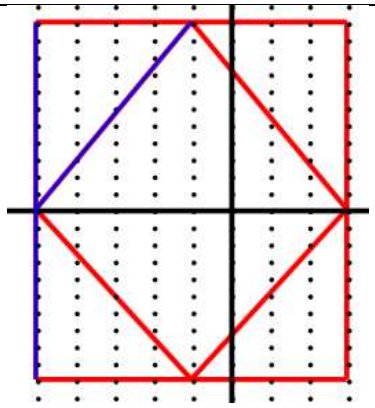

Figure 13

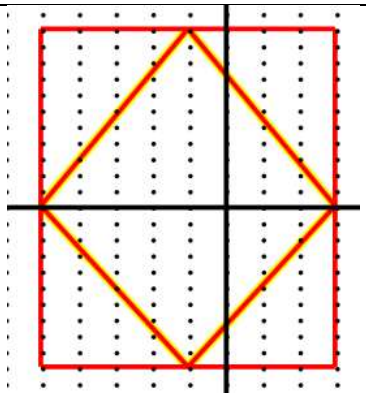

Figure 14

Figures 12 through 14 show the final frames of this algorithm as it processes the polygons. Figure 14 denotes the intersection arrived at by the algorithm via a yellow highlight and the blue direction vectors disappear.

Results / Findings

This project provided an interesting approach to breaking down several computational geometry algorithms and visualizing their operation. While not all planned features were implemented, nor all planned algorithms, what was implemented is a promising start to what could be an invaluable tool for communicating and improving the understanding of these algorithms for students.

Nothing novel was discovered about any of the implemented algorithms, though being able to visualize their operation, such as seeing the vectors chasing each other for polygon intersection, brought the algorithms to life, illustrating their core operation as they accomplished their work.

Conclusion

This project forms a strong starting point for what could be a useful pedagogical tool. Other than bug fixes, the improvements that could be made to the tool involve improving visualizations (adding circle drawing, for example, which could be quite useful for triangulation and Voronoi related algorithms), add text annotations on the drawing surface (to label points, or to bring attention to specific features), and copying demonstrations (or their input) to save time in creating multiple demonstrations that differ in tiny ways. Generating randomized input or importing data would also be useful ideas, as large data sets are difficult to input in any efficient manner.

The other major improvement would be addition of more computational geometry algorithms. If this were becoming a legitimate product/tool, the addition of more algorithms would be informed by a specific class syllabus, perhaps for an introductory computational geometry course, and then expanded from there for more advanced courses.

References

O'Rourke, J. (1998). *Computational geometry in C.* New York: Cambridge University Press.

Bowyer-Watson algorithm, https://en.wikipedia.org/wiki/Bowyer%E2%80%93Watson_algorithm

Bowyer-Watson algorithm, https://github.com/Bl4ckb0ne/delaunay-triangulation/blob/master/dt/delaunay.cpp

Delaunator algorithm, https://github.com/delfrrr/delaunator-cpp

https://www.ti.inf.ethz.ch/ew/Lehre/CG13/lecture/Chapter%206.pdf

http://www.personal.psu.edu/cxc11/AERSP560/DELAUNEY/13_Two_algorithms_Delauney.pdf

https://raw.githubusercontent.com/bkiers/GrahamScan/master/src/main/cg/GrahamScan.java

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

https://en.wikibooks.org/wiki/Algorithm_Implementation/Geometry/Convex_hull/Monotone_chain