

THE SIMPLIFICATION OF SIMPLE POLYGON ORIENTATION DETERMINATION

ANTHONY NGUYEN

1. INTRODUCTION

1.1. **Abstract.** Explain the purpose of our algorithm and why it's important/relevant.

Before we can go in-depth into our algorithm, we must define a polygon, specifically a simple polygon.

- A polygon encloses a region, which has a measurable area.
- The edges of a polygon only meet at their respective vertices and do not intersect anywhere else.
- Exactly two edges meet at a given vertex.
- The number of edges always equal the number of vertices.

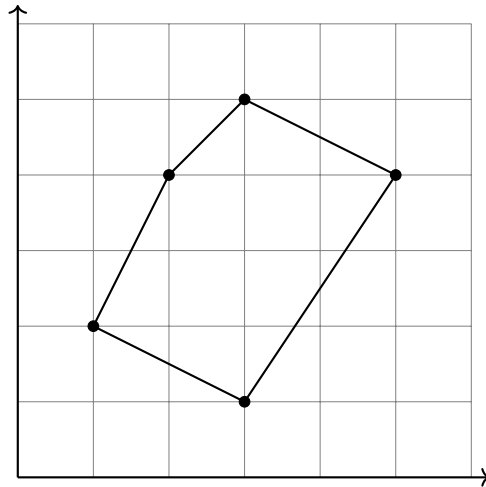


FIGURE 1. Example of Simple Polygon

For the polygon above, these vertices are provided without any form of ordering. When we do assign ordering to these points, we see two diverging possibilities: Clockwise (CW) and Counterclockwise (CCW).

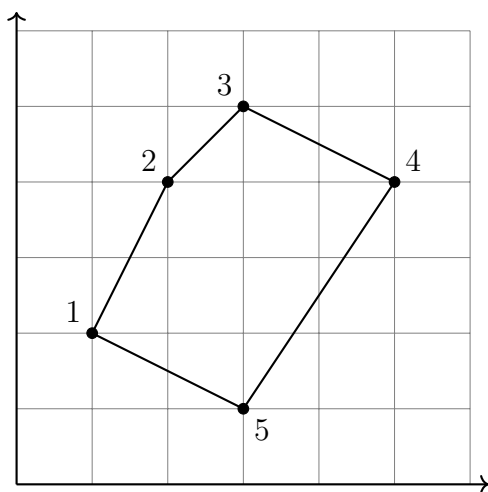


FIGURE 2. Simple Polygon with CW Orientation

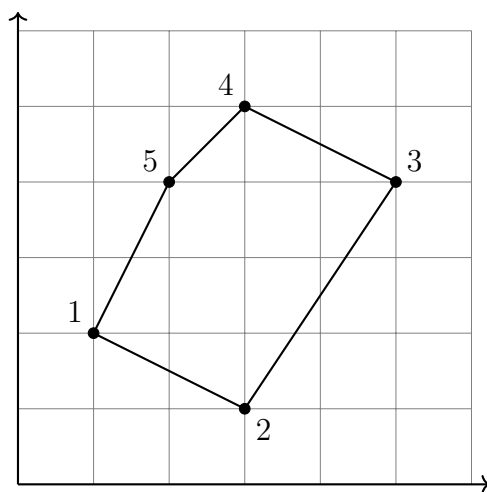


FIGURE 3. Simple Polygon with CCW Orientation

At first glance, we can easily identify the orientation of an ordered polygon if the polygon is arranged properly. However, how would we automate this process, while still maintaining speed and portability?

2. ALGORITHM ANALYSIS

By utilizing the bottom-most vertex and the cross product of its adjacent extending vectors, we can determine the orientation of a given simple polygon.

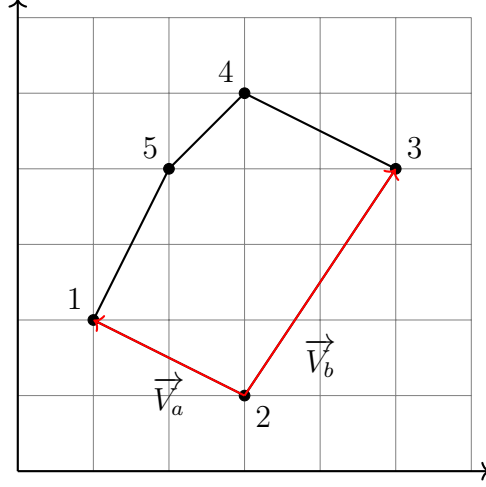
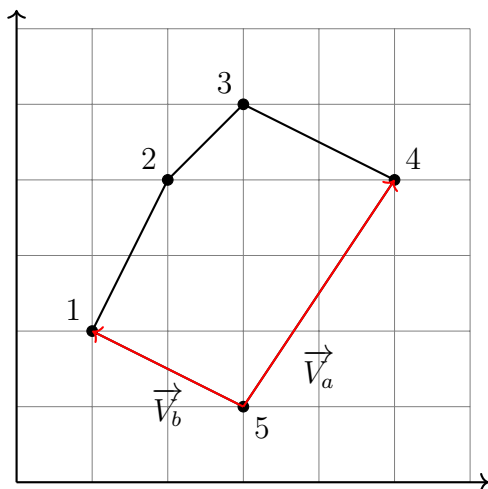


FIGURE 4. CCW Polygon, with Emphasized Extending Vectors

2.1. **Algorithm Breakdown.** The algorithm is as follows:

- Determine the vertex/vertices at the global minimum.
 - In the case of multiple vertices, select the vertex with the lowest x-value
- Using the selected vertex, determine the vectors going toward the adjacent vertices (\vec{V}_a to the previous vertex, \vec{V}_b to the next vertex)
- Determine the sign of $\vec{V}_a \times \vec{V}_b$
 - [Case A] If positive, the polygon is clockwise.
 - [Case B] If negative, the polygon is counter-clockwise.
 - [Case C] If zero, the orientation is indeterminable (to be discussed later).

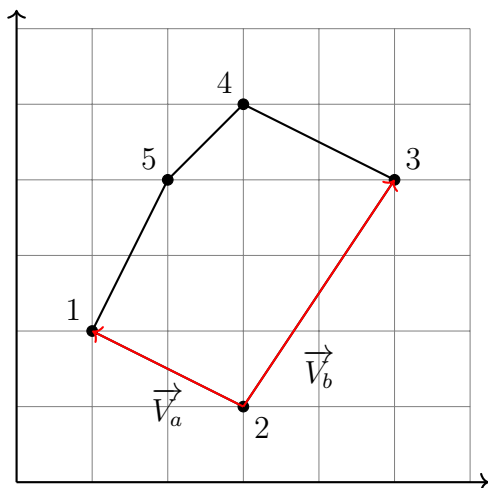
Case A:



$$\begin{aligned}\vec{V}_a \times \vec{V}_b &= \begin{vmatrix} X & Y & Z \\ 2 & 3 & 0 \\ -1 & 1 & 0 \end{vmatrix} \\ &= (0, 0, 5) \\ &\rightarrow 5 \hat{z} \text{ [CLOCKWISE]}\end{aligned}$$

FIGURE 5. Simple Polygon with CW Orientation

Case B:



$$\begin{aligned}\vec{V}_a \times \vec{V}_b &= \begin{vmatrix} X & Y & Z \\ -1 & 1 & 0 \\ 2 & 3 & 0 \end{vmatrix} \\ &= (0, 0, -5) \\ &\rightarrow -5 \hat{z} \text{ [COUNTER-CLOCKWISE]}\end{aligned}$$

FIGURE 6. Simple Polygon with CCW Orientation

Case C:

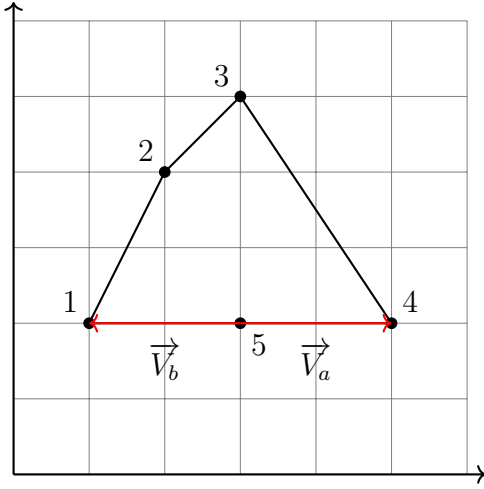


FIGURE 7. Simple Polygon with CW Orientation

$$\begin{aligned}\vec{V}_a \times \vec{V}_b &= \begin{vmatrix} X & Y & Z \\ 2 & 0 & 0 \\ -1 & 0 & 0 \end{vmatrix} \\ &= (0, 0, 0) \\ &\rightarrow 0 \hat{z} [\text{UNKNOWN}] \end{aligned}$$

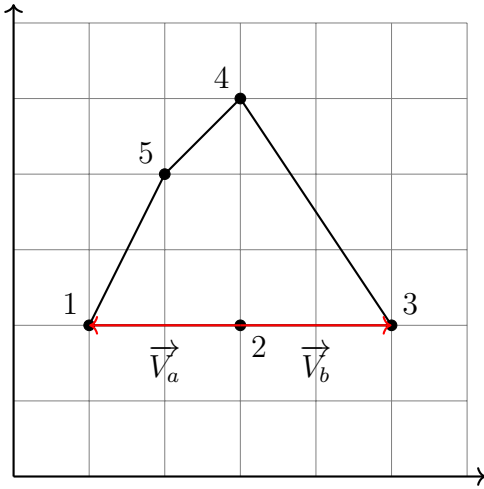


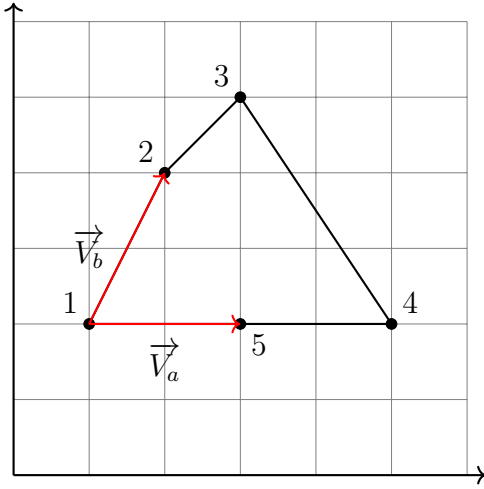
FIGURE 8. Simple Polygon with CCW Orientation

$$\begin{aligned}\vec{V}_a \times \vec{V}_b &= \begin{vmatrix} X & Y & Z \\ -1 & 0 & 0 \\ 2 & 0 & 0 \end{vmatrix} \\ &= (0, 0, 0) \\ &\rightarrow 0 \hat{z} [\text{UNKNOWN}] \end{aligned}$$

Because both polygons (with differing orientations) give us Z values with varying signs, we are unable to algorithmically identify these polygons' orientation in its current state.

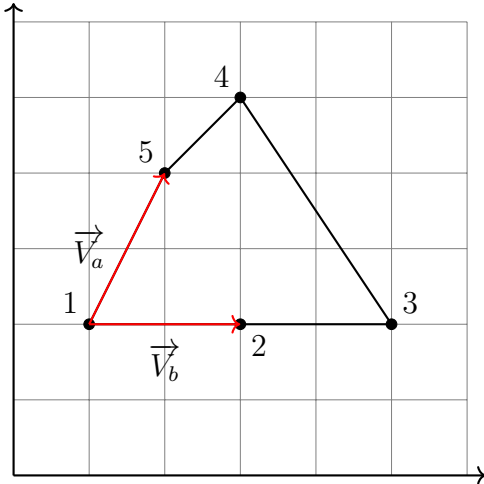
In order to prevent this edge case from causing errors, we decide to select a vertex (within our previous subset of vertices) with the lowest x-value. By doing this, we ensure that one of its adjacent vertices will have a differing y-value and will not cause an error of collinearity.

Case C (MODIFIED):



$$\begin{aligned}\vec{V}_a \times \vec{V}_b &= \begin{vmatrix} X & Y & Z \\ 2 & 0 & 0 \\ 1 & 2 & 0 \end{vmatrix} \\ &= (0, 0, 4) \\ &\rightarrow 4 \hat{z} \text{ [CLOCKWISE]}\end{aligned}$$

FIGURE 9. Simple Polygon with CW Orientation



$$\begin{aligned}\vec{V}_a \times \vec{V}_b &= \begin{vmatrix} X & Y & Z \\ 1 & 2 & 0 \\ 2 & 0 & 0 \end{vmatrix} \\ &= (0, 0, -4) \\ &\rightarrow -4 \hat{z} \text{ [COUNTER-CLOCKWISE]}\end{aligned}$$

FIGURE 10. Simple Polygon with CCW Orientation

3. MATHEMATICAL ANALYSIS

In order to determine the orientation of the entire simple polygon, we sample the vertex with the lowest y-value. From the sampled vertex, we calculate the sign of the cross-product of its preceding vector and its succeeding vector.

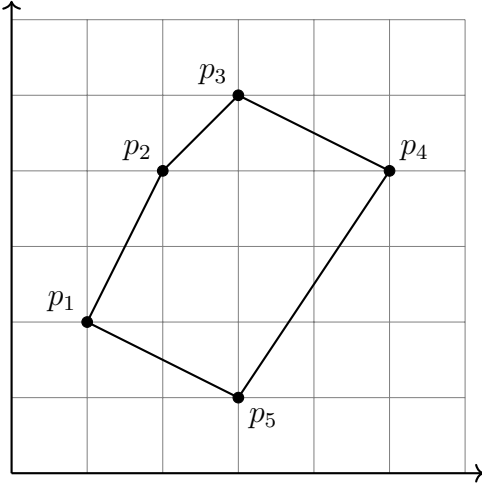


FIGURE 11. Simple Polygon with CW Orientation

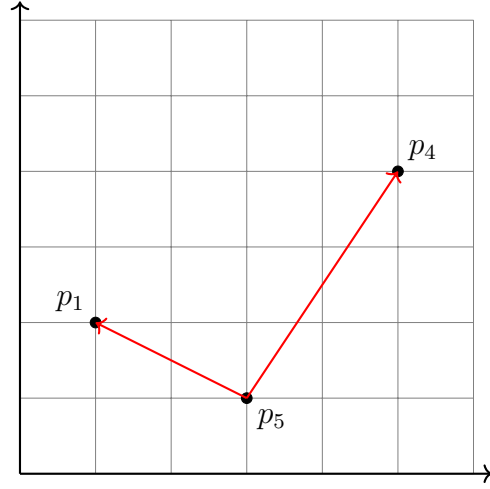


FIGURE 12. Sampled Vertex with its Extending Vectors

In regards to vertex p_5 , all other vertices either must have an equal or higher y-value because p_5 is the bottom-most vertex by definition. As a result, we know that all other vertices will have an equal or greater y-value than our chosen vertex. As a result, the polygon will be in the same orientation as the cycle entering p_5 , because this particular cycle cannot "loop around" the chosen vertex to change the orientation.

4. COMPARISON TO PRE-EXISTING ALGORITHMS

The traditional approach for orientation determination would be the Shoelace Algorithm.

Given a planar simple polygon with a Counter-Clockwise sequence of points $p_i = (x_i, y_i)$ $i = [1..n]$ in a Cartesian coordinate system:

$$(4.1) \quad A = \frac{1}{2} \sum_{i=1}^n \begin{vmatrix} x_i & y_i \\ x_{i+1} & y_{i+1} \end{vmatrix} \quad \begin{cases} A > 0 & \text{CCW} \\ A < 0 & \text{CW} \\ A = 0 & \text{N/A} \end{cases}$$

FIGURE 13. Utilizing Shoelace Algorithm for Orientation Determination

This algorithm operates with $O(n)$ multiplications.

In comparison, our algorithm operates with $O(1)$ multiplications. Due to the computational cost of multiplication, our algorithm's sampling technique allows for us to significantly decrease run time.

5. TESTING

Although we have previously proven that our algorithm works in a theoretical context, we must also rigorously test this algorithm in practical cases.

5.1. Image Testing.

One method used for testing was scraping a contour from a random pre-existing image.

To do so, we utilized a U2-Net Model¹ to differentiate between the foreground and the background of a given image. With this salient object detection, we can filter out the main focus of these images for testing purposes.



FIGURE 14. Original Image

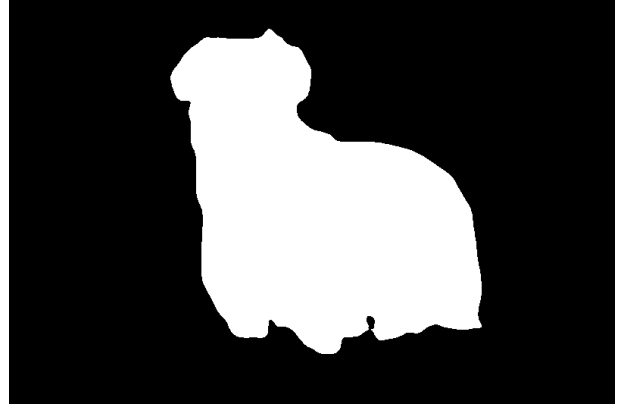


FIGURE 15. Masked Image

By performing contour detection upon this masked image, we are able to scrape an ordered polygon. From here, we able to choose which orientation that we want to have this polygon (either clockwise or counter-clockwise).

First, we utilized the Imgur API to generate a randomized set of images for us to test on. Then, after generating these contours, we tested our algorithm upon these images to determine a clockwise or counter-clockwise orientation.

¹<https://arxiv.org/pdf/2005.09007.pdf>

5.2. Polygon Testing.

The second method used for testing was randomly generating polygons, which we tested our algorithm upon.

Using Victor Hua's and Hanson Kang's Simple Polygon Generation algorithm ², we are able to efficiently generate rudimentary simple polygons. In order to make these irregular polygons into simple polygons that appeared more "natural," we utilized Chaikin's Algorithm ³ to smooth out these jagged edges.

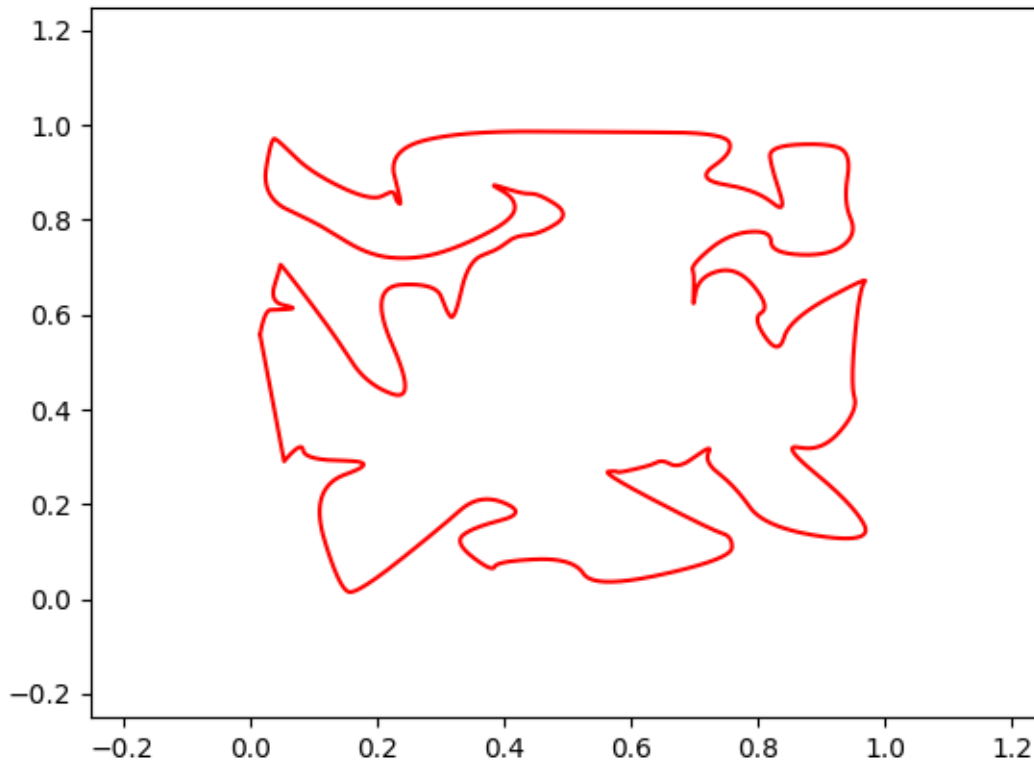


FIGURE 16. Smoothed Randomly-Generated Simple Polygon

Afterwards, we assigned an orientation to these newly-formed polygons and independently tested our algorithm.

²<https://github.com/spoiledhua/random-simple-polygon-generation>

³<https://www.cs.unc.edu/dm/UNC/COMP258/LECTURES/Chaikins-Algorithm.pdf>

5.3. Real-time Testing.

Our last method of testing was to use polygons that are drawn in real-time.

TKinter to draw polygons

Bentley-Ottmann algorithm to check for intersections (which breaks simple polygon clause)

6. CONCLUSION