Report

Petar Covic, PLC70

October 2023

To run it, it will initially ask for if you are using a saved file, if you are, type "File" and then the path to the file. This takes text files. The scaling is 400:1m so any points should be scaled by 400 if reading it. The first line of the text file will take the number of polygons, the minimum number of vertices, the maximum number of vertices, the minimum radius, and the maximum radius. And every line after that will be the centerX,centerY:Then each vertex of the polygon.

If you are not using a file, it will ask you for those values to generate a map of random polygons.

1 Part 1

In this assignment, I created a scene of polygons in java. I had one standard scene with average sized polygons, one scene with a lot of very small polygons, one scene with a few big polygons, and one scene where there are only a few polygons of average size, to show a more spreadout layout. Figure 1 also shows a visualization of the initials as polygons

To generate each polygon, i ran a method that took the number of vertices, which was generated as a random between the user given minimum and maximum vertices, the maximum radius, and the minimum radius.

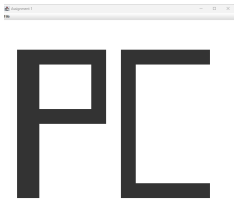

Figure 1: A visualization of creating initials out of polygons

2 Part 2

Figures 2-5 contain the examples that were generated in part 1 after running the collision checking algorithm.

The algorithm I chose to implement was the separating axis theorem because it was a mix of efficient, accurate, and simple.

I stored each of my polygons in an obstacle object, creating an array of Obstacles, that kept track of the points of each polygon, among other potentially important variables. So to implement the SAT algorithm, I created a nested for loop that would compare each Obstacle object with every that was created after that obstacle. This improved efficiency because if I checked each obstacle with past obstacles, id get the same results as with checking the past obstacles against the current one.

Then inside the nested for loop, I called a method that was given the two obstacles as parameters, and this methods runs the separating axis theorem on the obstacles to check for collision.

The first thing I do inside the method is get the points of the obstacles and assign the two sets of points to different arrays. Then for each point in the first obstacle, it creates a new array of size 2, where the index 0 is equal to the negative difference of the Y component of the next vertex and of the current vertex. And index 1 is the difference of the next vertex's x value and the current vertices x value.

In another loop that iterates through all the points of the first obstacle, i get the dot product of the first obstacle, and the array that I had just created. Then I get the minimum between the current minimum and the dot product, and I do the same thing for the maximum.

I then loop through the points of the second obstacle and do the same as before, getting the dot product, minimum, and maximum.

Then I check that the maximum of the second obstacle is larger or equal to the minimum of the first obstacle and the maximum of the first obstacle is larger or equal to the minimum of the second obstacle. If both are not true for any point, i return that there has been no collision, but if they are true for every point, it returns that a collision occurred.

This gave me a time complexity of $\mathrm{O}(\mathrm{n}^2), instead of the O(n^3) that was given by the naive approach, so I found th$

3 Part 3

In this part, I create a rectangular body meant to act as a vehicle. This rigid body is controlled by using the arrow keys. Up and down arrow are used to go forward and backward respectively, and Left and right are used to rotate counterclockwise and clockwise respectively.

4 Part 4

In the final part, I created a planar arm that can be controlled. It uses 1, 2, 4, 5 to rotate each joint. 1 and 4 rotate the first joint counterclockwise and clockwise respectively. 2, and 5 rotate the second joint counterclockwise and clockwise respectively.