Progress Report for Implementing Three Voronoi Diagram Computation Algorithms and Comparing Their Performance

CS478

Students

Burak Öztürk 21901841

Alp Tuğrul Ağçalı 21801799

Contents

In this report, we will inform you about our progress on the project in terms of;

- Research on the problem as the problem domain and structure
- Approaches used to solve the problem
- Algorithms that come from the approaches
- Pseudocodes

Problem Definition

A Voronoi diagram is defined by three sets; a set of sites (points) S, a set of edges E and a set of vertices V.

Set S is the input parameter of the function and the goal is to divide the plane into regions with edges such that every region only contains one site *s*[1] and every arbitrary point *p* in that region is closer to *s* than to any other site.[2] And the vertices are the points that 3 or more edges intersect like *v* in the Figure 1.

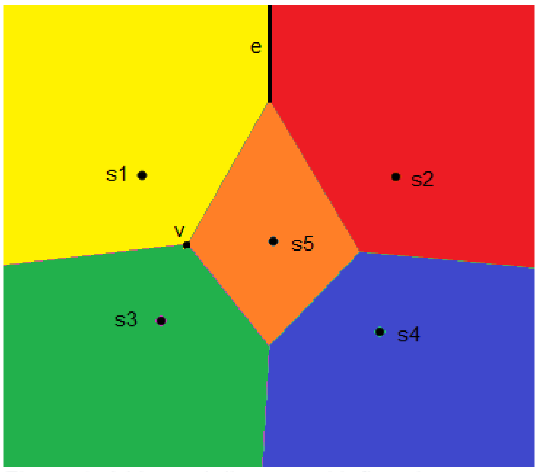


Figure 1: A Voronoi diagram with five sites.

This trait of Voronoi diagrams implicitly means an edge *e* is a line segment that two regions r_1 and r_2 share is the set of points that are the same distance from sites s_1 and s_2 (from r_1 and r_2 respectively).

Similarly, vertices are the points that n ≥ 3 regions $R_n = \{r_1, r_2, \dots, r_n\}$ share is the set of points that are the same distance from sites $S_n = \{s_1, s_2, \dots, s_n\}$ (from s_1 and s_2 respectively) but in this case the set only contains one point so these are vertices.[3]

Approaches

We studied three approaches that can be used to generate a Voronoi diagram from a set of sites. In this part, those approaches are discussed.

Randomized Incremental Algorithm

An incremental algorithm is an iterative algorithm that inserts a new element at each step while maintaining some properties over the elements[4]. In the Voronoi diagram, The incremental algorithm adds sites one by one, constantly updating the diagram for each added site.

Updating the diagram can include adding and deleting vertices, adding, resizing, or deleting edges. As an example, suppose we want to construct a diagram V' by adding a point p to V, a Voronoi diagram of k points. Assuming that the point p we added falls within one or more of the circumcircles

---

[1] For example, in Figure 1, there is only *s1* in the yellow region.

[2] Every point that is left of *e* is closer to *s1* than to *s2* in Figure 1 since the line that contains *e* is perpendicular to the line that contains *s1* and *s2*.

[3] The distances from the sites *s1*, *s2*, *s3* to the vertex *v* are all equal. Hence *v* is a vertex. (Note that the opposite of this reasoning is not always true.)

[4] Blelloch, Guy E., Yan Gu, Julian Shun, and Yihan Sun. “Parallelism in Randomized Incremental Algorithms.” Accessed March 25, 2023. https://people.csail.mit.edu/jshun/incremental.pdf.

to which the vertices of V are the center, the vertices that are centers to these circles cannot be part of the new diagram. This is because, by definition, there are no sites within the Voronoi corner circles.

As a result, the diagram is updated each time a new point is added. And in this way, a Voronoi diagram is created by adding all the points one by one.

In the following example we see that, when the newly added site is added to the inside of the circumcircle, whose center is the vertex of the Voronoi diagram, the center vertex disappears and new vertices are added. If the site is added outside of the circumcircle the vertex which is the center is not affected and new vertices are added.

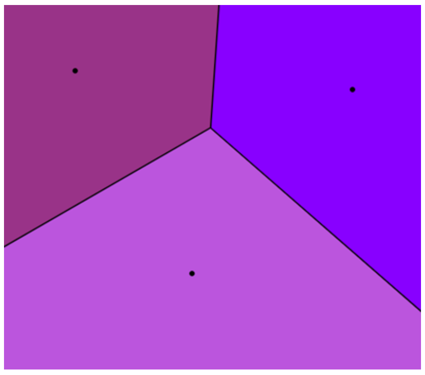


Figure 2: Before inserting a new site[5].

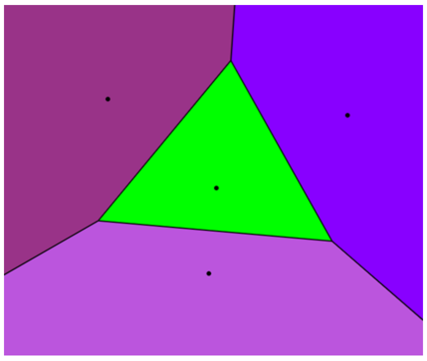


Figure 3: Inserting the new site inside of the circumcircle.

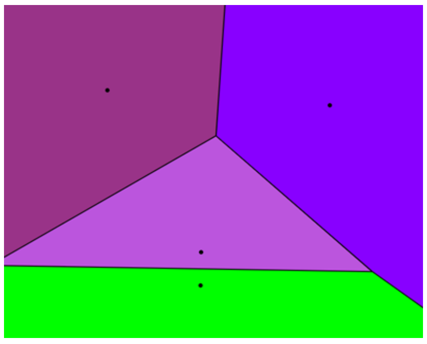


Figure 4: Inserting the new site outside of the circumcircle.

Since the Voronoi Diagram is dual of the Delaunay Triangulation and it is easier to check circumcircle when computing the Delaunay Triangulation with the Randomized Incremental Algorithm, we will compute Voronoi Diagram using Delaunay Triangulation.

[5] Figures created using https://cfbrasz.github.io/Voronoi.html

Summarized Algorithm:

1. Create the bounding triangle and add it to the triangulation T.

2. For every point p

a. find the triangle t in T that contains point p
b. If p is on the edge of t find t’ the other side of edge
c. Remove t and t’ from T
d. create new triangles with using p and vertices of t and t’ and add them T
e. for every new triangle
    - draw the circumcircle and check if there is any point inside it
    - if there is find the common edge of two triangle
    - connect the third vertices and remove the common edge

3. Remove the Bounding Triangle

4. Compute Voronoi Diagram From the T.

PseudoCode

```
Procedure VORONOI(S):
Input: Set of sites S
Output: Voronoi Diagram of S

triangulation = []
boundingTriangle = createBoundingTriangle(S)
triangulation.add(boundingTriangle ,0)

for site in S:
begin

     exTriangles = []
     newTriangles = []
     for t in triangulation
     begin
          if t.contains(site)
          begin
               exTriangles .add(t)
          end
     end
     for t in exTriangles
     begin
          vertices = t.getVertices()
          triangulation.remove(t)
          //createTriangles function return 2 triangles with 4
          //point
          newTriangles = createTriangles(vertices, site)
     end
```

```
        triangulation.add.newTriangles,0)
        for t in newTriangles
        begin
            commonVertices = []
            circle = drawCircumCircle(t)
            for triangle in triangulation
            begin
                for every vertice in triangle
                begin
                    if circle.contains vertice
                    begin
                        //otherVertices function returns other
                        //two vertices of the triangle except
                        //input parameter
                        commonVertices =
                        triangle.otherVertices(vertice)
                        changeThirdVertices(t, triangle,
                                                otherVertices)
                    end
                end
            end

        end
    end

    triangulation.remove(boundingTriangle)
    return computeVoronoiDiagram(triangulation)
```

Fortune's Algorithm

Fortune's Algorithm consists of sweep and beach lines, both of which move in the plane. Sweep line is a straight line perpendicular to the x-axis. It is assumed to go from left to right. As the sweep line moves to the right, the sites to the left of the line will be included in the Voronoi diagram, while those on the right will not be taken into account. Beach line, on the other hand, is not a straight line. It is a line to the left of the sweep line and consists of parabola segments. Beach Line parabolas represent points midway between the Sweep Line and the point, for each point to the left of the Sweep Line. As the Sweep Line progresses, the intersections of these parabolas define the edges of the Voronoi diagram.

The algorithm contains a Binary Search Tree to record the Beach Line and a priority queue to check for upcoming events. All sites are put into the priority queue and the algorithm progresses (from left to right) until the queue is finished[6].

The sites are sorted according to their x values when they are sent to the priority queue. The smallest X will be the first item to leave the queue. If the Xs are identical, the element with the lower Y value is sorted to come out first in the queue.

In the queue there are also circle events.Circle events are events that happen when the two parabola of the beach line collapses. The point where the collapse occurs is a Voronoi Vertex.

[6] https://en.wikipedia.org/wiki/Fortune%27s_algorithm

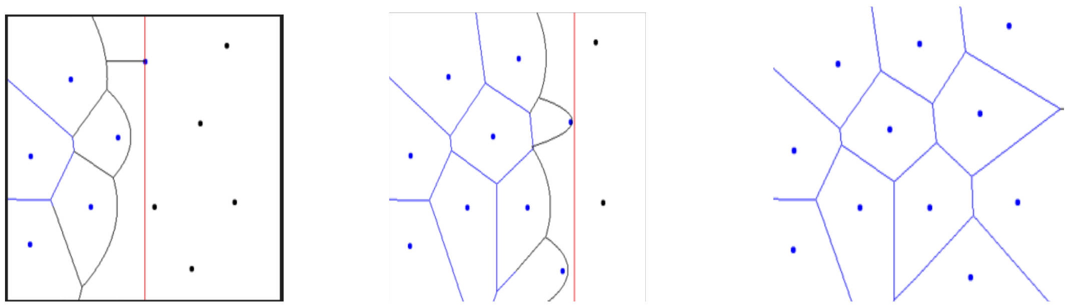


Figure 5: Visual Example of Fortune's Algorithm [7]

Summarized Algorithm:

1. Create an empty BST and empty priority queue for Beach and Sweep lines.

2. Insert sites to the priority queue

3. while priority queue is not empty

- a. Pop next event
- b. if event is SiteEvent
  - i. Insert point into BST
  - ii. Check circle events created by new site and add them to priority queue
- c. if event is circle Event
  - i. Remove the arc associated with the circle event from BST
  - ii. Add Voronoi Vertex
  - iii. Update edges
  - iv. Check circle events created by the removal of arc

4. Process remaining infinite edges.

PseudoCode

```
Procedure VORONOI(S):
Input: Set of sites S
Output: Voronoi Diagram of S

beachLine = new BST()
events = new priorityQueue()


for site in S
begin
     sites.insert(site)
end

while !events.isEmpty();
begin
     event = events.remove();
```

[7] https://en.wikipedia.org/wiki/Fortune%27s_algorithm

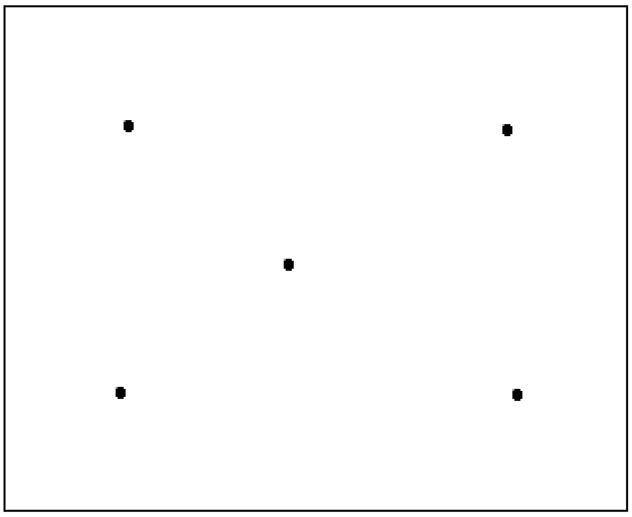
```
          if event.type == “site”
          begin
                arc = beachLine.insert(event.point)

                //checheckCircleEvents function checks potential
                //circle events and add them in to events if any
                checkCircleEvents(arc, events)
          end

          else
          begin
                arc = beachLine.remove(event.point)
                addVertice(event.point)
                updateEdges(event.point)
                check_circle_events(arc, events)
          end
    end
```

The Flipping Algorithm

The last algorithm we will consider takes its name from the method it uses to achieve the Delauney triangulation. It flips diagonals of quadrilaterals to achieve minimum inner angles. This is the key for the Voronoi-Delaunay duality.

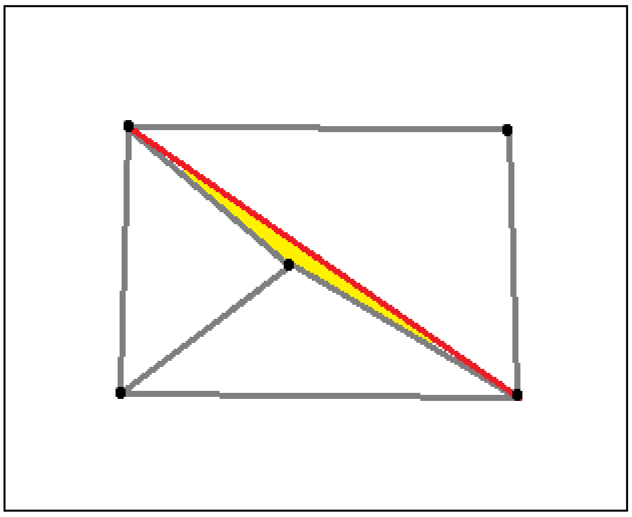
The duality states that any Voronoi diagram can be converted to its Delauney triangulation counterpart by only a trivial operation.[8] Thus, the main idea is using the sites of the diagram as the input point set of the Delauney triangulation and then flipping edges of the triangles to form the Voronoi diagram.

Consider start point for the algorithm below:

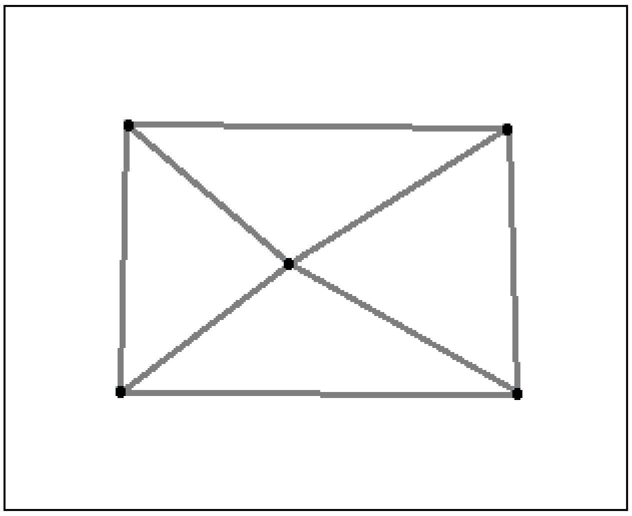


[8] https://en.wikipedia.org/wiki/Voronoi_diagram

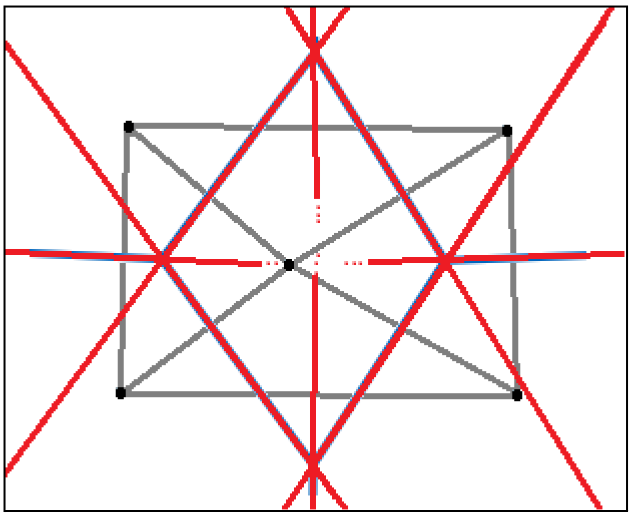
First step of the algorithm is Delauney triangulation. For Delauney, we start with an arbitrary triangulation and maximize inner angles by diagonal flipping. One arbitrary triangulation:



In the above triangulation, the yellow triangle defies Delaunay triangulation since the circumdisk of the yellow triangle contains the upper-right site. This problem is solved by changing the red edge for the other possible diagonal. Below is the result:

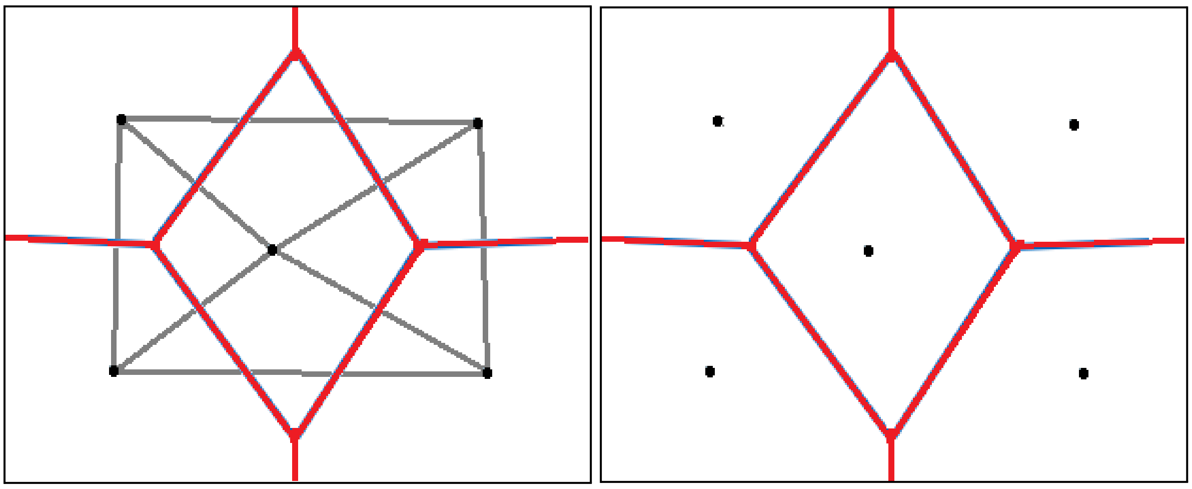


For the next step, perpendicular bisectors for every edge are calculated for every triangle:



After the last step, the Voronoi diagram is nearly finished. But lines need to be converted to line segments. To do that, remember every line segment needs to be the set of points that is equally distant to exactly two sites in a Voronoi diagram. Similarly, intersections of the line segments are the points that are equally distant to three or more sites.

Keeping that in mind, the lines that are passing through two intersections are reduced to the line segment between the intersections. Lines with one intersection are reduced to rays. If a line does not intersect with any other line it is left as a line. Below is the result, and without triangulation it is the Voronoi diagram for the points:



Summarized algorithm:

1. Sort sites by their x-values, increasing. (Use y-values, if x-values are equal.)
2. Select the first triangle to the first site.
3. ith triangle is the union of the (i-1)th triangle, ith site and the convex hull of the ith site and the points of the (i-1)th triangle. Last triangle is the starting triangulation.
4. Detect the illegal triangles by checking if circumdisks of triangles contain any sites in them.
5. Convert illegal triangles by flipping their longest edge with their quadrilateral's other possible diagonal.
6. Repeat 2 and 3 until no illegal triangles are left.
7. Calculate perpendicular bisectors from every edge of every triangle.

8. If a PB line from Step 2 has two points where it intersects other lines, take the line segment between the intersections. If there is one intersection, the line is reduced to a ray. (Technically every line and ray are limited by the bounding box, thus they are all line segments.)
9. Set of the line segments are the Voronoi diagram.

Pseudocode:

```
Procedure VORONOI(S):
Input: Set of sites S
Output: Voronoi triangulation of S

// Sorting S by x values for lexicographic triangulation
Sort(S, S.xValues)
T = S[1]   // First arbitrary triangle
// Lex. triangulation
foreach s in S[2:]
begin
     T = T ∪ s ∪ convexHull(T)
end

// Flipping until no illegal (thin) triangles left
illegalFlag = true
while illegal
begin
     illegal = false
     foreach triangle in T
     begin
          if getCircumdisk(triangle).includes(S)
          begin
               flip(max(triangle.edges))
               illegal = true
          end
     end
end

// Forming Voronoi diagram from triangulation
V = empty Voronoi set
foreach triangle in T
begin
     cc = circumcenter(triangle)
     V.vertices.add(cc)
     foreach neighbor triangle nt
     begin
          if (cc, circumcenter(nt)) in V.edges
               continue
     end
     V.edges.add(cc, circumcenter(nt))
end

return V
```