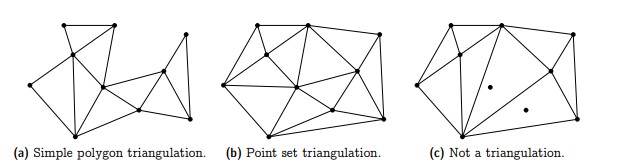
**Delaunay Triangulation + Voronoi Diagram**

**Background**

The Delaunay Triangulation is a special form of triangulation that directly relates to a Voronoi Diagram. Before discussing the Delaunay triangulation, we will define a normal triangulation. A triangulation within a polygon or a given set of points is where triangles are ‘formed’ by adding edges between vertices or points until every vertex or point is part of a triangle. For example, if you were to triangulate a square, you would simply have a diagonal line through two opposite vertices. Triangulation is useful in a multitude of ways. It can be used to easily determine the area of a shape, the distances between points, and the angles between edges of a shape. Below we will look at some pictures illustrating what we have just described above for a better understanding.



**Figure 1: Examples of what is and what is not a triangulation**

As you can see, simply triangulating a set of points can seem very simple, this may be true for small examples with a few points but can get very messy if we were to say have one hundred. This is why triangulation is typically done using a computer and in fact we have written a python script to test a triangulation later.

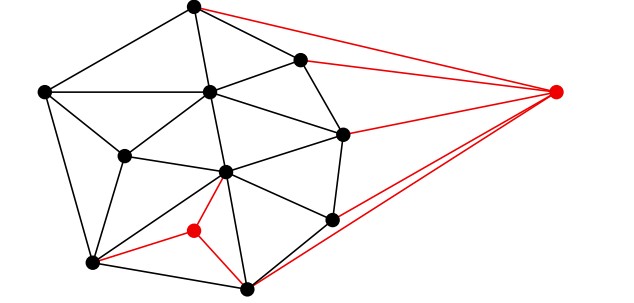
But, before we just dive into the fun part of visualizing the triangulation via a wacky python script, we must understand some proofs as well as define our triangulation in a mathematical way.

Definition of Triangulation: A **triangulation** of a finite point set P ⊂ R\*R is a collectionof triangles (*T*) that satisfy the following 3 properties:

1. P = UT∈T V(T);
2. conv(P) = UT∈T T; where conv(p) is the convolution of the point set
3. For every different pair of both U or T ∈ T, the intersection of them is either a shared vertex,edge,or nothing.

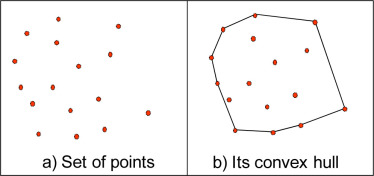
Now that we have defined a triangulation with the following properties we can explore proofs

Proof: For every set P ⊂ R\*R where P has >= 3 points has a triangulation, unless all points in P are on the same line as then it is a line not a triangle.

In order to prove this, we must determine if the added point “q” lies inside or outside the current triangulation (T). If q is on the exterior of T, then we connect q to all of the vertices of Cq-1:= conv({p1, . . . , pq−1}) that q “sees''. Meaning, for every vertex v in Cq-1 for which Pqv∩Ci−1 = {v}. Connecting the points in this way will always produce triangles as there are two points of tangency from Pq to Cq-1. If q lies within T, determine the specific triangle we are located within T and partition it into 3 separate smaller triangles. This is done doing the same step above, but explaining it this way may be more beneficial to just learn the basics. I believe that the above definition can be seen as confusing, but simply viewing the below figure we can see that adding a point isn’t too bad!

**Figure 2: Adding two points to a triangulation will always create more triangles thus another triangulation**

Proof: Any triangulation of a set P ⊂ R\*R with P having x amount of points, has exactly 3x−y−3 edges and 2x-y-2 triangles, where x is the number of vertices from P’s convex hull.

To understand this proof we must understand the definition of a convex hull which is rather simple. A convex hull is considered to be the smallest polygon that contains all of our vertices within it. 

**Figure 3: simple example showing convex hull to understand the above proof**

We know that each triangle has 3 edges, and that each internal edge in our triangulation is shared between 2 triangles and external edges belong to only one triangle. Then we have: 3t = 2(e − y) + y = 2e − y.

Adapting Euler's formula we get x + (t + 1) = v + f = e + 2.

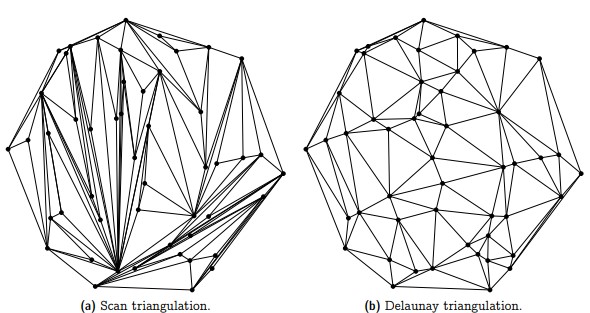
Finally after simplifying we can see the proof fall into place

e = x + t − 1 ⇒ 3e = 3x + 3t − 3 = 3x + 2e − y − 3 ⇒ e = 3x − y − 3

3t = 2e − y = 6x − 2y − 6 − y = 6x − 3y − 6 ⇒ t = 2x − y − 2

**Thus proving our statement!**

The above information has been describing scan triangulations in general, which are very useful in their own right. However, there exists a triangulation with all the properties as above, plus another special property we will discuss, that makes the triangulation not only look better, but give us useful properties. This difference will be shown to the reader in the following figure to show a very quick reason why we use this new triangulation method.

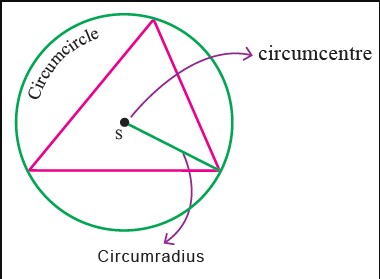
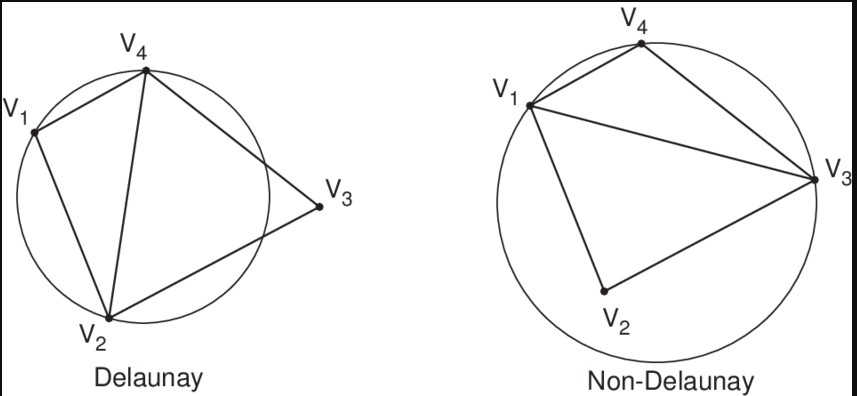


**Figure 4: scan vs Delaunay Triangulation**

The reader can hopefully very easily see that the Delaunay triangulation looks a lot nicer than just the scan one. The reasoning behind this is explained in the next section.

**Delaunay Triangulation**

The Delaunay triangulation was named after Boris Delaunay for the work he did with these types of triangulations in 1934. This type of triangulation is special in that it has the properties mentioned above, but also adds another property that makes it unique. The property the Delaunay triangulation adds is that the circumcircle - a circle that passes through all three of a triangle's vertices - of any triangle in the triangulation does not contain any other points that are also within the triangulation. This is known as the “Empty Circle Property”. To better understand what this means please view below figures.

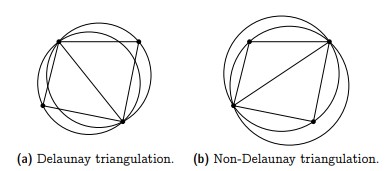


**Figure 5: Circumcircle**

**of a triangle and**

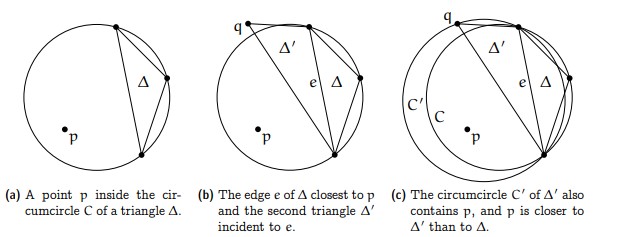
**example of delaunay**

**Triangulation vs not**

While the criteria for a Delaunay triangulation sounds restrictive, this type of triangulation is actually always possible for a given set of points as long as they all don’t lie on the same line. This is particularly useful when dealing with a set of four convex points meaning they are the polygon containing all vertices. There are only two possible triangulations for a set of four convex points, and only one of them is Delaunay. This can be clearly seen in the following figure.

**Figure 6: Delaunay Triangulation of 4 convex points**

It is seen here that if one triangulation is not Delaunay you can simply switch the edges dividing the triangles and you will have a triangulation that is Delaunay. This is an important property of Delaunay triangulations known as ‘flipping’ and is known as the Lawson flip algorithm. This is important as it makes finding a Delaunay triangulation for a large set of points easier. You’d simply start with all the convex sets of four points and triangulate them first, thus making the triangulation for the remaining points quicker and easier as you would repeat flipping until the triangulation is Delaunay. The correctness of this algorithm can be seen with the following figure.



**Figure7: Showing how flipping edges can have great results**

**It is imperative that the reader remembers this works, so long as the points are in the convex position. If the points are not contained within the convex hull, this will not work invalidating the triangulation entirely so keep this in mind!**

Another algorithm for the reader to chew on, is known as the Bowyer-Watson algorithm. This was devised by both Adrian Bowyer and David Watson in1981. However, they were not working together, they both independently discovered and published a paper describing their algorithms at the same time. What a coincidence! This is known as an incremental algorithm where we start with an already created Delaunay triangulation. We add points one at a time checking every time triangle to ensure that the new point is not located within its circumcircle. If the point is in the circle, remove the triangle. This will leave a star shaped polygon hole which we then re-triangulate using the newly added point. By doing the above mentioned adding of points seen in figure 2. **Remember adding a new point to a triangulation can always produce a new one!**

This algorithm can take O(NlogN) with N points being added. Below is some pseudocode that pretty well explains the algorithm and can be found implemented in our included python script!

**Bowyer-Watson Pseudocode**

function BowyerWatson (pointList)

// pointList is a set of coordinates defining the points to be triangulated

triangulation := empty triangle mesh data structure

add super-triangle to triangulation // generates a triangle around all others essentially convex hull

for each point in pointList do // add all the points one at a time to the triangulation

badTriangles := empty set //added to to remove a bad triangle

for each triangle in triangulation do // first find all the triangles that are no longer valid due to the insertion

if point is inside circumcircle of triangle

add triangle to badTriangles

polygon := empty set //object used to retriangulate make new triangle

for each triangle in badTriangles do // find the boundary of the polygonal hole

for each edge in triangle do

if edge is not shared by any other triangles in badTriangles

add edge to polygon

for each triangle in badTriangles do // remove them from the data structure

remove triangle from triangulation

for each edge in polygon do // re-triangulate the polygonal hole

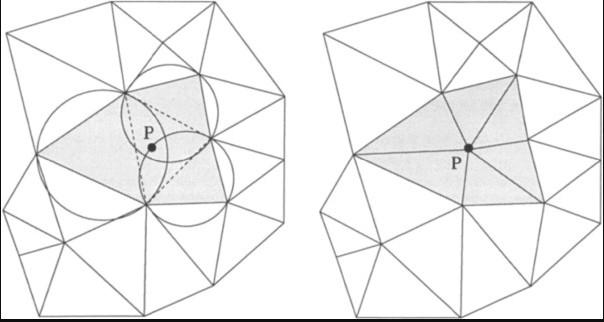
newTri := form a triangle from edge to point

add newTri to triangulation

for each triangle in triangulation // done inserting points, now clean up

if triangle contains a vertex from original super-triangle

remove triangle from triangulation

return triangulation

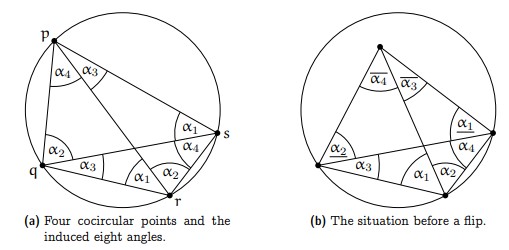
**Figure 8: Bowyer-Watson**

**Implemented adding point P**

Because of this empty circle property, as well as this flipping, the Delaunay triangulation gains a very interesting property. While regular triangulations could have many ‘skinny’ triangles (as seen in figure 4a), the Delaunay triangulation essentially makes these skinny triangles very difficult to produce, as these triangles would have very large circumcircles, and they would almost certainly enclose some other point. **This does not however imply that Delaunay triangulation will never produce small triangles so keep this in mind!** Due to this avoidance of skinny triangles, the Delaunay Triangulation maximizes the smallest angle of every triangle within the triangulation. We will now look at a proof of this to ensure the reader this is a fact.

Proof: Let P ⊆ R\*R be a finite set of points such that they are not on a line or a circle. Let D be the Delaunay triangulation of the point set P, and let T be any triangulation of P. Then the angles of triangulation T <= angles in D.

We know that we can always turn the triangulation T into a Delaunay triangulation by implementing our Lawson flip algorithm. We know that flipping an interior triangle will change a total of 6 interior angles. This is clear as remember triangles have 3 distinct angles, switching 3 twice is 6. Visualize this swapping in the below figure.



**Figure 9: Visualizing the maximum**

**angle being created with flips**

We will define angles smaller than α as α\_ and larger as α^.

These are our angles prior to our flipping:

α1 + α2, α3, α4, α1\_ , α2\_ , α3^ + α4^

After flipping we get the following representation:

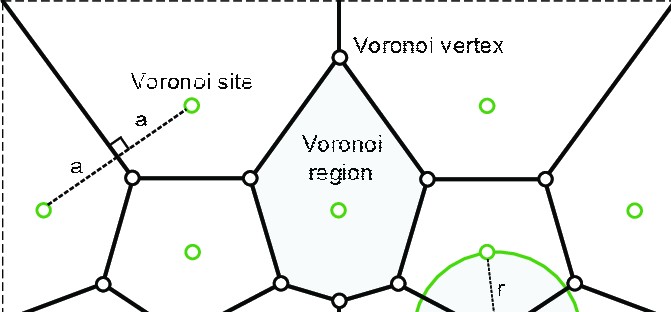
α1, α2, α3^, α4^, α1\_ + α4, α2\_ + α3.

Hopefully after looking at this the reader can recognize that for every single angle after the flip, there exists a smaller angle from before the flip. Examples include α1 > α1\_ ,

α1\_ + α4 > α4 and so on. Please take time to look at this and recognize it as truth as it is one of the main reasons why Delaunay triangulations are useful other than looking “nice”. This maximizing of the smallest angle makes calculations of edges lengths’ and distances involving those angles easier and more accurate.

If the reader is convinced that this relates to no other section in the textbook this is false.Delaunay triangulations do relate to some of the material learned in class! The Euclidean minimum spanning tree for a set of points is actually a subset of the Delaunay triangulation for the same set of points. Thus, being able to find a minimum spanning tree would already give you a head start in being able to find the Delaunay triangulation!

**Voronoi Diagram**

Last but not least, what the heck is a voronoi diagram!?The Delaunay triangulation for a set of points is closely related to the Voronoi diagram for that same set of points as they share a dual relationship. A voronoi diagram is a diagram that places each point of a graph into a specific region. The point which lies within a region is the closest point to everywhere within that shaded region. As you can imagine, Voronoi diagrams are extremely useful when it comes to real life maps. For example, a voronoi diagram of the airports in the United States would be very useful to pilots in case an emergency happens and they need to make a quick landing as this would tell them the immediate closest airport in their region. A voronoi diagram between three points can be found by dividing each edge between the points in half, and then drawing a perpendicular line stemming from that edge(aka perpendicular bisector). These lines will intersect somewhere, and then partition the plane into three separate regions, and these regions will form the voronoi diagram. 

**Figure 10: voronoi Construction**

Since a voronoi diagram for more than three points is essentially just the same process of finding perpendicular bisectors of triangles repeatedly, it is immediately evident that it will be related to triangulation in some form. In fact, since any triangle can be circumscribed, its perpendicular bisectors will always intersect at the center of the circumcircle. If we combine this with the fact that in a Delaunay triangulation, there are no other points inside the circumcircle, it is evident that a Voronoi diagram can be created from the circumcenters of a Delaunay triangulation. In fact connecting the circumcenters of the Delaunay Diagram will create the regions of the Voronoi diagram. The way to determine which circumcenters to connect is also reliant on the Delaunay diagram. If two triangles share an edge in the Delaunay triangulation, then the circumcenters of their circumcircles will be connected to form the Voronoi diagram.

Please visit the final figure on the last page to really emphasize the dual relationship between both Delaunay triangulation and Voronoi diagrams.

