Computational Geometry: Lecture 5

Don Sheehy

January 29, 2010

1 Degeneracy

In many of the algorithms that we have discussed so far, we have run into problems when that input is somehow troublesome. For example, we saw that our convex hull algorithm had to be more careful if any three points are collinear. Also, our sweepline algorithm had to special case the situation where an input line segment is horizontal.

For the sake of getting through the high level algorithms, we simply asserted that "bad stuff doesn't happen", but really, we'd like to give a more formal definition of what we mean by that. In the Computational Geometry literature, the term *general position* is used to describe an such arrangements of points. Sometimes, it means simply that no three points are collinear. Other times, it might mean something stronger such as no four points are cocircular.

An input point set is *g*eneric of *in general position* if it avoids some measure zero set in the space of inputs. Another way to think of this is to assume that the input has been infinitesimally perturbed.

As we saw last time, the operations that we will be doing on point sets usually take the form of a determinant of some matrix. We called these linear predicates. The calculation of the determinant is the evaluation of some polynomial. Recall that the zero set of a nonzero polynomial has measure zero in the parameter space. You may be more familiar with the case of univariate polynomials in which there are only a finite number of zeros. Any random perturbation to the coordinates of a matrix with determinant zero will cause the determinant to change.

For the most part we will limit our attention to input in general position. This is also why we often talk of them as linear *predicates* despite the fact that can take three values. We simply assume that they will not evaluate to zero on the inputs.

2 Delaunay Triangulations

For now, we will define a triangulation of a point set P to be a decomposition of $CC(P)$ into triangles with vertices in P.

Triangulations are useful for many applications. As a simple example, consider a set of points P in the plane corresponding to gps coordinates. Now imagine that associated with each point I have an altitude measurement. If I want to take my data and draw a 3D terrain, I can start with a triangulation in the plane and then lift the triangles into 3D by lifting their corners.

I started with an altitude function $f : P \to \mathbb{R}$. With a triangulation T, I get a natural extension of this function to all of $CC(P)$, i.e. $f_T : CC(P) \to \mathbb{R}$, by looking at the height of the terrain at any point inside the triangulation.

Today, we'll be looking at the most important triangulation of all, the **Delaunay Triangulation**. A triangulation T of a set of points P is the Delaunay triangulation of P (denoted $\text{Del}(P)$) if for all triangles $t \in T$, the circumcircle of t (denoted $cc(t)$) is empty of points from P.

From this definition, there are several obvious questions to ask:

1. Does it always exist?
2. Is is unique?
3. How do we compute it?
4. How quickly can we test if a triangulation is in fact the $\text{Del}(P)$?

The answers to the first two questions are both "Yes (assuming general position)". We will see why this is the case next time, but for now I'll ask you to assume it so that we can move on to some other interesting properties of the Delaunay triangulation and get a look at an algorithm.

The Delaunay triangulation has two very important properties that we will be dealing with today.

- Among all triangulations of P, $\text{Del}(P)$ maximizes the minimum angle.
- $\text{Del}(P)$ is locally checkable.

We'll clarify what the second one means in a second. First, let's consider a warmup problem. Given a triangulation T how do you test if $T = \text{Del}(P)$. The naïve algorithm would, for each triangle, check if any of the points encroach on its circumcircle. There are at most $O(n)$ triangles[1] and each requires $O(n)$ work to check encroachment. So the total running time of this algorithm is $O(n^2)$. If we have a data structure that can tell us in constant time what triangles lie on either side of an edge, we can actually do this computation in $O(n)$ time using the following ideas.

3 Local Delaunay Characterization

Say that an edge $\overline{ab}$ is locally Delaunayif either

- $\overline{ab}$ is on $CH(P)$, or

[1] This follows from Euler's Formula.

- $\overline{ab}$ is adjacent to triangles $\triangle abc$ and $\triangle abd$ and $d \ni cc(abc)$.

Let's do a quick sanity check. Suppose $\triangle abc \in \text{Del}(P)$. Then clearly the edge $\overline{ab}$ is locally Delaunay. How about a converse question: if $\overline{ab}$ is locally Delaunayin some triangulation T, is $\overline{ab}$ necessarily an edge of a Delaunay triangle? Here, the answer is "No". If $\triangle abc$ and $\triangle abd$ are the two triangles adjacent to $\overline{ab}$ then the locally Delaunaycondition for $\overline{ab}$ only guarantees that c and d do not encroach the triangles $\triangle abd$ and $\triangle abc$ respectively, but other vertices can also encroach these triangles. You are encouraged to draw a picture for this counterexample.

The fact that we can check the Delaunay property by local checks is encapsulated in the following Lemma.

Lemma 3.1. *Given a triangulation T of P, if every edge e in T is locally Delaunaythen $T = \text{Del}(P)$.*

Proof. Pick an arbitrary point $p \in P$. Choose $x \in CC(P)$. Say that the triangle containing x is $\triangle abc$. Consider the line segment $\overline{px}$. We will prove that for any choice of x, if $\overline{px}$ only crosses locally Delaunayedges, then $p \ni cc(abc)$. The proof will be by induction on the number of triangulation edges crossed by $\overline{px}$. By induction, assume that if $\overline{px}$ crosses $\leq k$ edges then $p \ni cc(abc)$. The base of the induction ($k = 1$) follows exactly from the definition of the local Delaunay property. Without loss of generality, assume that $\overline{px}$ crosses the edge $\overline{ab}$ to enter $\triangle abc$. Let $d \in P$ be such that $\triangle abd$ is the other triangle sharing the edge $\overline{ab}$. Consider a point x' on $\overline{px}$ that lies in $\triangle abd$. The line segment $\overline{px'}$ crosses one fewer edge than $\overline{px}$ and so by induction, $p \ni cc(abd)$. However, the fact that $\overline{ab}$ is locally Delaunayimplies that $cc(abd)$ contains the intersection of $cc(abc)$ and the halfspace containing d bounded by the line through $\overline{ab}$. Since p lies in this same halfspace and is not contained in $cc(abd)$, it follows that $p \ni cc(abc)$. So, we have proven that p does not encroach on any circumcircle. Since the choice of p was arbitrary, we find that no point in P encroaches on the circumcircle of any of the triangles in T and thus, $T = \text{Del}(P)$. □

The preceding lemma implies a natural algorithm for testing if a triangulation is the Delaunay triangulation. Verify that each edge is locally Delaunay. Testing the locally Delaunaycondition requires at most one InCircle test. You might want to prove for yourself why we don't need two tests, one for the triangle on either side.

4 Edge Flips

Given 4 points in convex position, there are two possible triangulations corresponding to the two choices of a diagonal. We call the swap of one edge for the other, an *edge flip* (or sometimes just a *flip*). The key fact to observe is that when we do the flip, one of the triangulations is Delaunay and the other is not. That is, if the one interior edge is not locally Delaunay, flipping it will make it locally Delaunay.

The notion of a flip, leads to an idea for a simple algorithm for computing the Delaunay triangulation from an arbitrary triangulation.

5 A very simple algorithm

Suppose we have been given some triangulation T of P. At a high level, we can run the following algorithm:

Algorithm 1 FLIPTODELAUNAY

Input: A triangulation T of a set $P \in \mathbb{R}^2 : |P| = n$
while some edge e is not locally Delaunay **do**
  FLIP(e)
end while

At first sight, this algorithm perhaps doesn't seem like it should work. It depends on many other facts that we will have to prove. For example, although a flip does make one new edge locally Delaunay, it may cause other edges to longer be locally Delaunay, so it is not even immediately clear that the algorithm terminates. Also, if an edge is not locally Delaunay, does that necessarily mean that we can flip it?

We won't have time to prove everything today, but we can at least get a sense for why this algorithm might terminate.

Lemma 5.1. *Flipping an edge increases the minimum angle of triangles adjacent to the flipped edge.*

Proof. We will use the following basic fact from geometry illustrated in Figure 1. For any chord $\overline{ab}$ through a circle S and any point c on S forming a CCW turn with a and b, the angle $\angle acb$ is the same regardless of which point c on S is chosen. If c is chosen outside of S, then that angle is strictly larger than it would have been for c on S.

Let a, b, c, and d be four points in convex position as shown in Figure 2. The edge $\overline{ab}$ is locally Delaunay. We want to show that the smallest angle in the triangulation that uses $\overline{ab}$ is larger than the smallest angle in the triangulation that uses $\overline{cd}$. It will suffice to show that for each angle in the Delaunay triangulation, there is some smaller angle in the triangulation before the flip.

We use the fact about angles opposite circle chords to yield the following inequalities.

$$\theta_1 > \theta_4 \tag{1}$$
$$\theta_2 > \theta_7 \tag{2}$$
$$\theta_6 > \theta_3 \tag{3}$$
$$\theta_5 > \theta_8 \tag{4}$$

We also have the trivial inequalities, $\theta_7 + \theta_8 > \theta_7$ and $\theta_3 + \theta_4 > \theta_3$. This complete the proof.

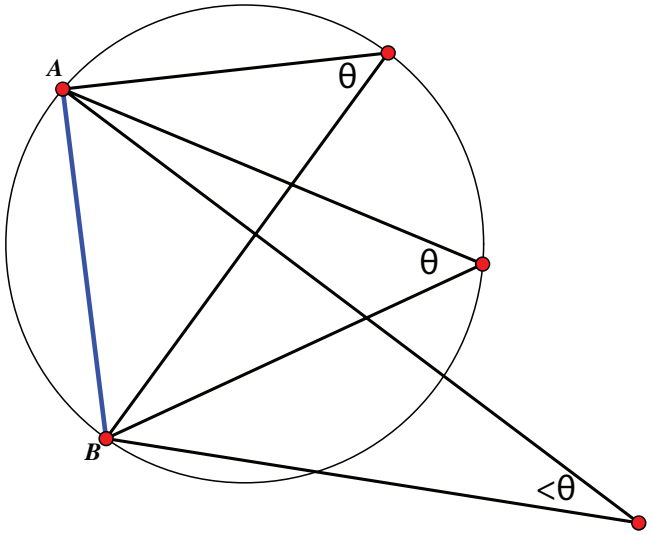

Figure 1:

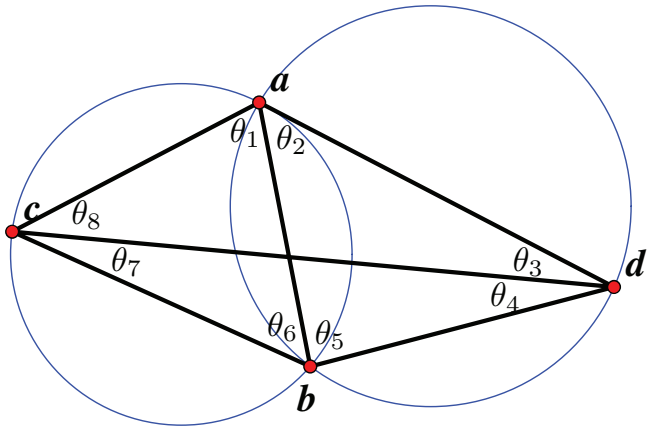

Figure 2:

□

So, we see that angles are improved locally with every flip. This hints at three interesting facts that we'll see in more rigor next time.

1. The flip algorithm terminates.
2. The Delaunay triangulation has the largest minimum angle among all triangulations of P.
3. Any two triangulations of P can be connected by a finite sequence of flip operations. This is also known as the connectivity of the flip graph of planar triangulations.

It may not be immediately obvious why we will be able to derive these three facts, but I claim that we have all the pieces in place. We'll see how to put them together next time.