Computational Geometry: Lecture 2

Don Sheehy

January 25, 2010

1 Where this course lives on THE TRIANGLE

There is a triangle that software engineers like to draw. The three vertices are labeled “good”, “fast”, and “cheap”. It is usually drawn on the chalkboard and followed with some statement about how you only get to pick two. Interestingly, filmmakers use the same triangle with the same caveat. So, next time someone asks you for a similarity between software developers and filmmakers, you'll have a better answer than, “They have a higher than average incidence of facial hair.”

Mathematicians also have a triangle that they draw, but in their version, the vertices are labeled “novel”, “useful”, and “beautiful” and the goal is to get at least one. This triangle is used to gauge the relative importance of a work. It can also be applied to classes. There is a lot of computational geometry in the useful corner and there is a fair amount in the novel corner, but you should know at the outset that I have a strong bias towards the beautiful. If you think that sounds like BS, we have all semester to prove you wrong.

2 Convex Hulls

We will start with a rough idea of a convex hull for points in the plane and then proceed to formulate an algorithm for computing one. By looking at our first attempt at an algorithm, we will hopefully get a better idea of the relationship between the convex hull problem in $\mathbb{R}^2$ and the sorting problem.

Let's start with a set of n points $p_1, \ldots, p_n$ in the plane. Imagine that the plane is replaced with the surface of a table big enough to hold all of the points. Now, to mark the points, I could hammer a nail, halfway into the table. The convex hull of this point set is just the shape a rubber band would make if I stretched it around the nails.

The nails-in-a-table model of computation does not come up in computer science courses very often. Clearly, it can do at least one problem very well, it can compute 2D convex hulls in constant time. It is what is sometimes referred to as a *physical model of computation.*

So, how do we come up with an algorithm for computing a convex hull in our model of computation, the Real-RAM? That is, how do we output a list of the vertices $v_1, \ldots, v_k$ of the convex hull in some cyclic ordering.

Let's start with an easier problem, just find one vertex of the convex hull. You should be able to convince yourself that the leftmost point will have to touch the rubber band. So, we can find one vertex of the convex hull in $O(1)$ time. Can we find an edge? After a little thought, one idea is to just choose the edge

3 Points, Vectors, and Types

When we think of a point p as an element of Euclidean space, R^d, we usually think of it as represented by a set of coordinates $(p_1, \ldots, p_d)$. This is the same representation we might use to describe a vector in the real vector space $\mathbb{R}^d$. Is there a difference between points and vectors? Yes, recall that any vector space must have a unique zero element, that is, an origin. For most problems on point sets, at least in this class, where we place the origin doesn't matter. The relative scale also does not matter. This is more like the ruler-and-compass world, where we started by drawing two points, one for the origin and one for the scale.

We're going to want to do arithmetic on points. We could just treat the points as vectors and do vector arithmetic, and that is what we'll do, but we have to be careful. Given two points $a = (a_x, a_y)$ and $b = (b_x, b_y)$, the sum $a + b = (a_x + b_x, a_y + b_y)$ depends on the origin. If we move the origin by a vector $v = (v_x, v_y)$, then a and b move by v and we get that $a + b = (a_x + b_x - 2v_x, a_y + b_y - 2v_y)$. In order to keep things well defined, we will say that arithmetic on points will yield a vector. Only certain arithmetic combinations of points will yield a point.

4 Combinations

There are four basic kinds of vector combinations that we care about: linear, non-negative, affine, and convex.

Definition 4.1. *A* **linear combination** *of $v_1, \ldots, v_n \in \mathbb{R}^d$ is any sum of the form*

$$\sum_{i=1}^{n} \alpha_i v_i,$$

where $\alpha_i \in \mathbb{R}$ for all i.

Definition 4.2. *A* **non-negative combination** *is a linear combination in which $\alpha_i \geq 0$ for all i.*

Definition 4.3. *An* **affine combination** *is a linear combination in which $\sum_{i=1}^{n} \alpha_i = 1$.*

Definition 4.4. *A* **convex combination** *is a linear combination that is both* non-negative *and* affine*.*

A combination takes a collection of vectors and some coefficients (the α_i's) and produces a new point. We can also use combinations to produce infinite sets by taking the **closure** of all possible combinations. The closure of linear combinations is usually referred to as the *span*. The closure of non-negative combinations is often called a cone. You should convince yourself that the non-negative combinations of two points in the plane forms a an infinite wedge pointed at the origin. The affine closure of 2 points is the infinite line between them. The convex closure of two points is the line segment between them. The convex closure of a set of points in the plane is a convex polygon.

5 What is a convex hull?

Definition 5.1. *A set S is convex if it is closed under convex combinations. Equivalently, S is convex if $a, b \in S$ implies $\overline{ab} \subseteq S$.*

Definition 5.2. *The convex hull of a set S is the boundary of the convex closure.*

We will use CH to refer to the convex hull and CC to refer to the convex closure.

6 A simple algorithm

At this point in the class we came up with a very basic convex hull algorithm. Our approach was really simple. First, find one point v_0 on the convex hull (we chose the leftmost point). Next, we picked a second point by choosing the point that makes that smallest slope when connected to v_0. We then put this operation in a loop, adding the edge in each iteration that moves rightward and has the least slope. This gave us the bottom convex hull and we more or less convinced ourselves that, barring degeneracies, we could run the algorithm again while standing on our heads to get the upper hull.

A quick analysis showed that our naive little algorithm runs in time $O(n^2)$. Looking a little closer, it became clear that the algorithm we developed looks a heck of a lot like a class "bad" algorithm, Selection Sort.

Somehow, using a bad sorting algorithm with a slope comparison replacing the usual comparison gave us something like a convex hull algorithm. There are a couple obvious questions we should ask ourselves at this point. First, what else is different about this comparison operation? Second, what if we used a smarter sorting algorithm? We'll cover both of these questions in more depth in the coming lectures.