Geometric transformations

Scribe: Carl Burch; Lecturer: Gary Miller; Thursday, November 5, 1998

Note about exercises: *What are labeled as "exercises" in these notes are* not *assigned problems. They are just useful things to verify on your own if you want to understand the material better.*

We will discuss a few fundamental transforms from geometry. Using these transforms we will see that the convex hull problem has many unexpected applications, including **nearest-neighbor queries**, **mesh generation** (in scientific computation), **graphics**, and **operations research**.

Throughout, we use d as the dimension of our space. We think of our coordinates as real numbers; thus we are in $\mathbb{R}^d$. Most of the time $d = 2$ will be interesting enough. But unless we say otherwise, the claims will be true for any d.

Think of each point $p \in \mathbb{R}^d$ as a column vector of d reals

$$p = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_d \end{bmatrix} .$$

Recall that the length of vector p is

$$\sqrt{p_1^2 + p_2^2 + \cdots + p_d^2} .$$

We can write the square of this as $p^T p$, the dot product of p with itself. The notation $p^T p$ comes from regarding p as a $d \times 1$ matrix and multiplying (as matrices) p's transpose with p.

$$p^T p = \begin{bmatrix} p_1 & p_2 & \cdots & p_d \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_d \end{bmatrix} = p_1^2 + p_2^2 + \cdots + p_d^2$$

1 Linear programming and geometry

Geometric definitions A *hyperplane* in d dimensions is a flat $(d-1)$-dimensional body with any orientation. In 2 dimensions, a hyperplane is simply a line; in 3 dimensions, it is a plane. Algebraically, we can specify a hyperplane with a d-dimensional vector $p = [p_1, \ldots, p_d]^T$ and a scalar a as the set of points

$$\{x \in \mathbb{R}^d : p_1 x_1 + p_2 x_2 + \cdots + p_d x_d = a\} = \{x \in \mathbb{R}^d : p^T x = a\} .$$

Figure 1 illustrates the hyperplane $p^T p = 1$.

Claim 1 *A hyperplane defined by point $p \in \mathbb{R}^d$ and scalar a is normal (ie, perpendicular) to the line between the origin and p, and it intersects this line at*

$$p' = \frac{a}{p^T p} p .$$

Exercise 1: Prove this claim.

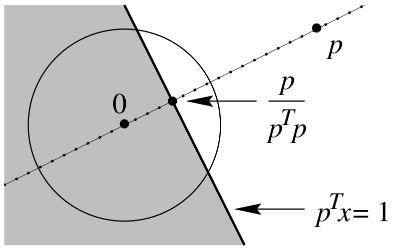

Figure 1: The hyperplane $p^T x = 1$ and the halfspace $p^T x \leq 1$.

A *halfspace* is the set of all points lying on one side of a hyperplane, plus the hyperplane itself. We can algebraically specify a hyperplane with a d-dimensional vector p and a scalar a as the set of points $\{x \in \mathbb{R}^d : p^T x \leq a\}$. Note that the halfspace contains the origin exactly when $a \geq 0$.

A *polytope* is an intersection of a set of halfspaces. In 2 dimensions, a polytope is a polygon with its interior; in 3 dimensions, it is a solid convex body with flat faces. Note that some polytopes are unbounded (as is the case with the polytope described by a single halfspace).

> **Exercise 2:** Argue that each vertex of a d-dimensional polytope is at the intersection of at least d halfspaces' boundaries. (In 2 dimensions, each vertex of the polygon is at the intersection of at least 2 lines.)

Linear programs A *linear program* is a problem asking us to find a d-dimensional vector $x = [x_1, \ldots, x_d]^T$ so that $c_1 x_1 + \cdots + c_d x_d = c^T x$ is minimum, where x must also satisfy all of a set of m linear *constraints*

$$\begin{array}{ccccccccc} a_{1,1}x_1 & + & a_{1,2}x_2 & + & \cdots & + & a_{1,d}x_d & \leq & b_1 \\ a_{2,1}x_1 & + & a_{2,2}x_2 & + & \cdots & + & a_{2,d}x_d & \leq & b_2 \\ & & & & \vdots & & & & \vdots \\ a_{m,1}x_1 & + & a_{m,2}x_2 & + & \cdots & + & a_{m,d}x_d & \leq & b_m \end{array}$$

Note that each constraint i can be written as $A_i x \leq b_i$, where A_i is the ith row of an $(m \times d)$-dimensional matrix A.

Do not confuse linear programming with computer programming. Like the term *dynamic programming*, the term predates computer programming. A linear program is just a particular type of problem where we want to minimize a linear function over a set of linear constraints.

Linear programs often turn up in **operations research** (OR), where people want to optimize industrial processes like airplane routing, inventory selection, and sports season schedules. (Major league baseball seriously contracts a CMU OR researcher to work on this!) The programs tend to involve thousands of dimensions and thousands of constraints. Thus efficient ways of solving them are crucial... and the geometric view can be helpful in understanding how to do it.

Each constraint of a linear program, you can see, is a halfspace, described by the d-dimensional vector $A_i{}^T$ and scalar b_i. Call this halfspace HS_i. The constraint requires the point x to lie inside HS_i for each i. Thus x must lie in the intersection of all the halfspaces, $\bigcap_{i=1}^m \mathrm{HS}_i$—which is a polytope. Thus we can regard a linear program geometrically as the problem of finding a d-dimensional point x in a polytope so that $c^T x$ is minimized.

> **Exercise 3:** Show that if a linear program has an optimal solution (that is, there is a point satisfying all the constraints but the objective function $c^T x$ is bounded for points within the polytope), then one of the vertices of the linear program's polytope is an optimal solution.

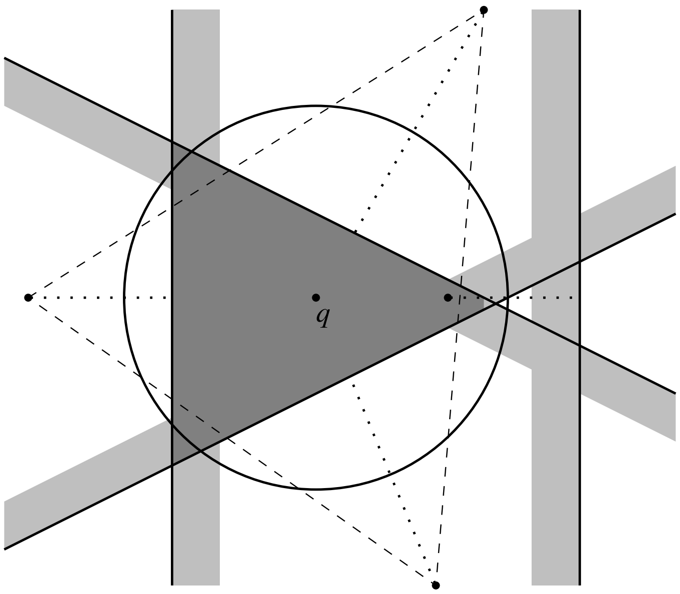

Figure 2: Computing the faces of a polytope.

Exercises 2 and 3 imply a simple but inefficient way to solve a linear program: Consider each set of d constraints (there are $\binom{m}{d}$ of them), and compute where they intersect, if anywhere. If this point satisfies all the constraints, then it is a vertex of the polytope. By computing $c^T x$ at each vertex x, we can find the optimal value.

2 Transformations

The dual transform One of the simplest—and possibly the most amazing—transforms is the *dual transform*. This transformation maps each point of $\mathbb{R}^d$ to a corresponding halfspace: The point p is mapped to the set of points

$$\mathrm{HS}_p = \{x \in \mathbb{R}^d : p^T x \leq 1\}\ .$$

Given a set of halfspaces, we might want to know which halfspaces lie on their polytope's boundary. For example, we could use such an algorithm to remove redundant constraints from a linear program. The following claim gives us a simple technique for doing this.

Claim 2 *Let P be a set of n points in $\mathbb{R}^d$ and HS_p be the halfspace determined from the point p using the dual transform. A point p is on $\mathrm{ConvexHull}(P)$ iff HS_p is on the boundary of the polytope $\bigcap_{p\in P} \mathrm{HS}_p$.*

Thus if we want to compute the faces of a polytope given a by a set of halfspaces and we know some point q in the polytope we can transform the problem to a convex hull problem, as illustrated in Figure 2: First translate everything so that q is at the origin; then take the inverse dual transform to map the halfspaces to points in $\mathbb{R}^d$ (illustrated by the dotted lines). We then compute the convex hull of the points (the dashed lines) and transform the points on the hull back to halfspaces. (In our illustration, one of the points is not on the hull and so its corresponding halfspace must not lie on the polytope, which we can visually verify in the picture.) Finally we translate everything back so that q is where it began.

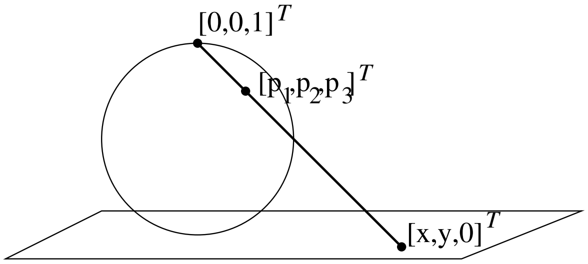

Figure 3: The stereographic map.

Reflection about a sphere The map from p to the intersection p' of the boundary of HS_p with the line connecting the origin and p is called *the reflection about the unit sphere* and is important on its own. In this transformation, each point p is mapped to $p/(p^T p)$. Every point outside the unit sphere goes to a point inside and every point inside goes to a point outside.

> **Exercise 4:** Show that the reflection about the unit sphere inverts itself (ie, $p'' = p$).

An interesting fact about the reflection about the unit sphere is that it maps spheres to spheres. By a *sphere*, we mean the set of points x lying at distance r from some center c; algebraically, this is specified as $\{x \in \mathbb{R}^d : (x-c)^T(x-c) = r^2\}$. By the *unit sphere* we mean the sphere centered at the origin with radius 1, $\{x \in \mathbb{R}^d : x^T x = 1\}$.

Claim 3 *If we take each point of a sphere centered at c with radius r and reflect it about the unit sphere, we end up with another sphere (centered at $c/(c^T c - r^2)$).*

> **Exercise 5:** Prove this claim.

In particular, if we think of a hyperplane as a sphere of infinite radius, then a hyperplane will be mapped to a sphere (passing through the origin), and a sphere passing through the origin will be mapped to a hyperplane. Also, if two curves intersect with an angle θ, then if we map the curves about the unit sphere they will still intersect with angle θ.

The stereographic map The stereographic map is a map which sends points on the 3-dimensional sphere of diameter 1 (radius $1/2$) onto points in the 2-dimensional plane.

Suppose we think of our plane in $\mathbb{R}^3$ at $z = 0$, and we place the unit diameter sphere with the south pole at the origin and the north pole at the point $[0, 0, 1]^T$. For each point p on the sphere except the north pole map it to the plane as follows: Draw a ray starting at the north pole and passing through p; extend the ray until it intersects the plane; and map p to this point on the plane. This is illustrated in Figure 3.

> **Exercise 6:** Compute the x and y coordinates of a point when it is stereographically mapped to the plane from a point $[p_1, p_2, p_3]^T$ on the diameter-1 sphere.

The parabolic map Another useful transformation is the *parabolic map*. This takes points in $\mathbb{R}^d$ and maps them to points in $\mathbb{R}^{d+1}$: The transform maps the point $p = [p_1, \ldots, p_d]^T$ to the point $p' = [p_1, \ldots, p_d, p^T p]^T$. That is, given p we keep the same d coordinates and append a new coordinate corresponding to the square of p's length. We will see an application of the parabolic map in the context of Delaunay triangulation (Claim 4).

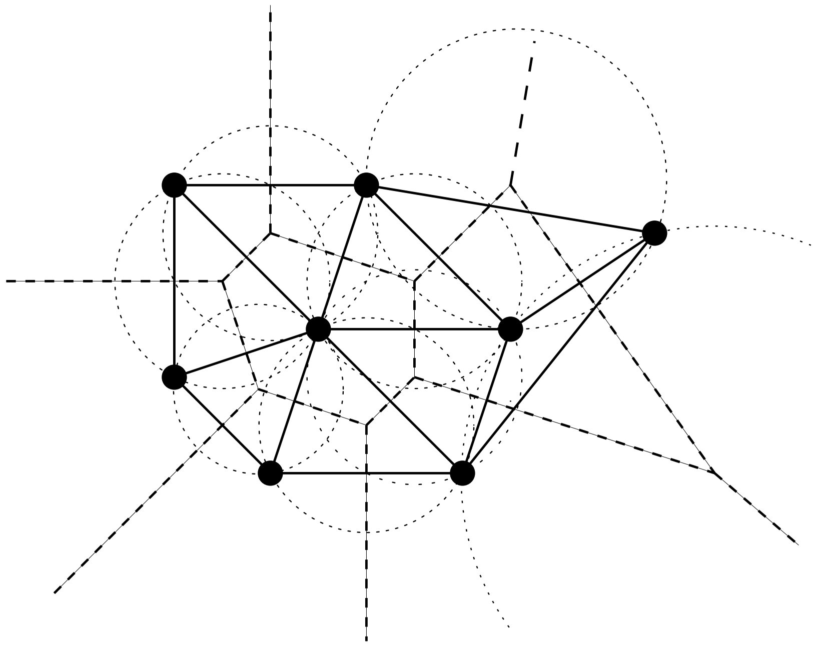

Figure 4: Voronoi diagram (dashed lines), Delaunay triangulation (solid lines), and circumcircles (dotted curves).

3 Voronoi diagrams and Delaunay triangulations

The Voronoi diagram For a set of points P, the *Voronoi diagram* is a partition of the plane into regions. Each region corresponds to a point $p \in P$ and is defined as the set of points in $\mathbb{R}^d$ that are closer to p than to any other point of P:

$$\mathrm{Vor}_p = \{x \in \mathbb{R}^d : \forall q \in P. dist(x, p) \leq dist(x, q)\} .$$

An example Voronoi diagram is illustrated by dashed lines in Figure 4.

Voronoi diagrams are a natural object to look at if we are interested in **nearest-neighbor queries**. We might have a set of points P which we can preprocess, where we want to be able to quickly determine for query points q which point in P is closest to q. (For example, in text categorization, we might represent a category as a vector where each dimension specifies a particular word's frequency—the computational geometry category would have an unusually high frequency of the word *vector*, for example. Given a document to be classified, we might compute want to its word distribution and hence its vector, and then categorize the document by using a nearest-neighbor query to find which category point is closest to the document's vector.) The Voronoi regions can help determine the answer to this type of query quickly.

One simple way of computing the Voronoi region for a point $p \in P$ is to consider $|P| - 1$ halfspaces: For each $q \in P$ (except p), construct the hyperplane bisecting the segment connecting p and q, and include the halfspace lying on the same side of the hyperplane as p. Using our observations about the dual transform (Claim 2), we can prune off any redundant edges. Figure 5 illustrates this; the thin dashed line is the only one that can be pruned.

Exercise 7: Why does this construction algorithm produce a valid solution?

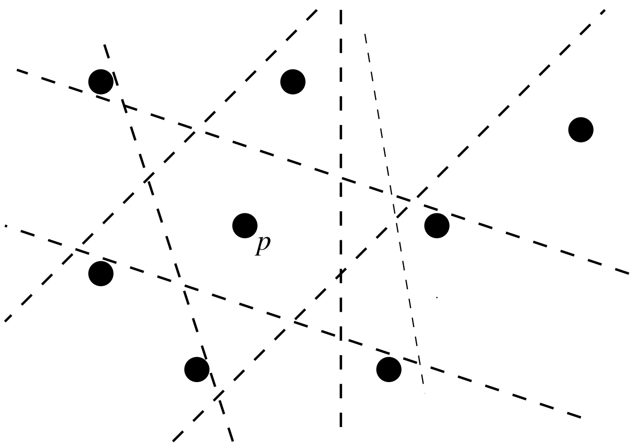

Figure 5: Constructing a Voronoi region for p.

The Delaunay triangulation A related idea is the *Delaunay triangulation*: Given a set of points P, their Delaunay triangulation is the set of all triangles $\{p_1, p_2, p_3\}$ of points in P so that if I draw a circle passing through the three points then no other points of P are contained within the circle. Figure 4 illustrates an example Delaunay triangulation with solid lines; the corresponding circles are the dotted curves.

A use of Delaunay triangulation pops up in **mesh generation**. Scientists and engineers often want to solve complicated differential equations that cannot be computed analytically. For example, an aerospace engineer may be interested in determining the flow of air about an airplane wing, but the equations specifying airflow are too complicated to be solved by hand. One common technique is to sample lots of points and iteratively compute the influence of points on their neighbors. The graph given by the Delaunay triangulation is helpful in determining which points influence which others.

A Delaunay triangulation is in some sense the dual of a Voronoi diagram drawn on the same set of points. Each intersection of the Voronoi diagram is the center of one of the circles described by a triangle of points in the Delaunay triangulation. And, if there is an edge between p_1 and p_2 in the Delaunay triangulation, then some subsegment of the perpendicular bisector appears in the Voronoi diagram.

> **Exercise 8:** Say we have the Delaunay triangulation of a set of 2-dimensional points given to us as a graph with an adjacency-list representation, where each adjacency list is sorted in counterclockwise order. This graph has $n = |P|$ vertices and $m = 3n - 6$ edges. Find an $O(n)$-time algorithm to construct the graph corresponding to the Voronoi diagram.

A simple way to compute a Delaunay triangulation is to consider each set of 3 points and determine whether the circle going through them contains any other points. A naïve implementation of this would consider each of the $\binom{n}{3}$ triples of points, construct a circle, and then check each of the remaining $n - 3$ points to see if they lie in the circle, for a total of $O(n^4)$ time.

The following fact, illustrated in Figure 6, gives a much more efficient way to compute the Delaunay triangulation in $\mathbb{R}^d$ given a convex hull program for $\mathbb{R}^{d+1}$.

Claim 4 *The Delaunay triangulation of a set of points P in $\mathbb{R}^d$ is equivalent to the result of projecting the points onto the paraboloid in $\mathbb{R}^{d+1}$ using the parabolic map, computing the convex hull of these points, and projecting the lower envelope (that is, the faces of the hull facing the base plane) back to $\mathbb{R}^d$.*

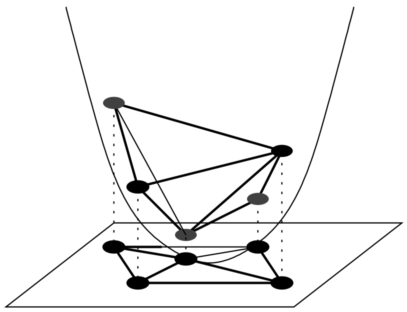

Figure 6: Constructing a Delaunay triangulation (on five points here) by projecting to a paraboloid.

4 Summary

We have seen a variety of geometric transformations. Coupling these with convex hulls leads to useful, efficient algorithms for removing redundant constraints from a linear program, for computing a Voronoi diagram (useful in answering nearest-neighbor queries), and for computing a Delaunay triangulation (giving us a mesh suitable for scientific computation). Thus geometric transformations and convex hulls are powerful tools useful in a variety of contexts, besides being interesting objects themselves.