Geometric Basics: We assume that any geometric primitive involving a constant number of elements of constant complexity can be computed in $O(1)$ time, and we will not concern ourselves with how this computation is done. (For example, given three noncolinear points in the plane, compute the unique circle passing through these points.) Nonetheless, for a bit of completeness, let us begin with a quick review of the basic elements of affine and Euclidean geometry.

There are a number of different geometric systems that can be used to express geometric algorithms: affine geometry, Euclidean geometry, and projective geometry, for example. This semester we will be working almost exclusively with affine and Euclidean geometry. Before getting to Euclidean geometry we will first define a somewhat more basic geometry called *affine geometry*. Later we will add one operation, called an inner product, which extends affine geometry to Euclidean geometry.

Affine Geometry: An affine geometry consists of a set of *scalars* (the real numbers), a set of *points*, and a set of *free vectors* (or simply *vectors*). Points are used to specify position. Free vectors are used to specify direction and magnitude, but have no fixed position in space. (This is in contrast to linear algebra where there is no real distinction between points and vectors. However this distinction is useful, since the two are really quite different.)

The following are the operations that can be performed on scalars, points, and vectors. Vector operations are just the familiar ones from linear algebra. It is possible to subtract two points. The difference $p - q$ of two points results in a free vector directed from q to p. It is also possible to add a point to a vector. In point-vector addition $p + v$ results in the point which is translated by v from p. Letting S denote an generic scalar, V a generic vector and P a generic point, the following are the legal operations in affine geometry:

$$\begin{array}{rcll} S \cdot V & \rightarrow & V & \text{scalar-vector multiplication} \\ V + V & \rightarrow & V & \text{vector addition} \\ P - P & \rightarrow & V & \text{point subtraction} \\ P + V & \rightarrow & P & \text{point-vector addition} \end{array}$$

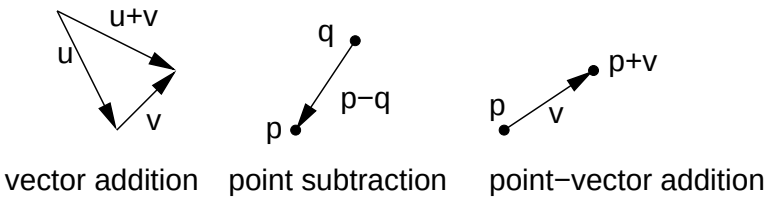

Figure 6: Affine operations.

A number of operations can be derived from these. For example, we can define the subtraction of two vectors $\vec{u} - \vec{v}$ as $\vec{u} + (-1) \cdot \vec{v}$ or scalar-vector division $\vec{v}/\alpha$ as $(1/\alpha) \cdot \vec{v}$ provided $\alpha \neq 0$. There is one special vector, called the *zero vector*, $\vec{0}$, which has no magnitude, such that $\vec{v} + \vec{0} = \vec{v}$.

Note that it is *not* possible to multiply a point times a scalar or to add two points together. However there is a special operation that combines these two elements, called an *affine combination*. Given two points p_0 and p_1 and two scalars α_0 and α_1, such that $\alpha_0 + \alpha_1 = 1$, we define the affine combination

$$\text{aff}(p_0, p_1; \alpha_0, \alpha_1) \;=\; \alpha_0 p_0 + \alpha_1 p_1 \;=\; p_0 + \alpha_1 (p_1 - p_0).$$

Note that the middle term of the above equation is not legal given our list of operations. But this is how the affine combination is typically expressed, namely as the weighted average of two points. The right-hand side (which is easily seen to be algebraically equivalent) is legal. An important observation is that, if $p_0 \neq p_1$, then

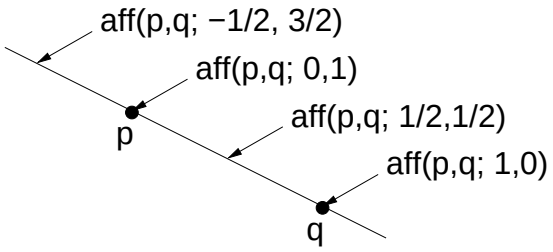

Figure 7: Affine combination.

the point $\mathrm{aff}(p_0, p_1; \alpha_0, \alpha_1)$ lies on the line joining p_0 and p_1. As α_1 varies from $-\infty$ to $+\infty$ it traces out all the points on this line.

In the special case where $0 \leq \alpha_0, \alpha_1 \leq 1$, $\mathrm{aff}(p_0, p_1; \alpha_0, \alpha_1)$ is a point that subdivides the line segment $\overline{p_0 p_1}$ into two subsegments of relative sizes α_1 to α_0. The resulting operation is called a *convex combination*, and the set of all convex combinations traces out the line segment $\overline{p_0 p_1}$.

It is easy to extend both types of combinations to more than two points, by adding the condition that the sum $\alpha_0 + \alpha_1 + \alpha_2 = 1$.

$$\mathrm{aff}(p_0, p_1, p_2; \alpha_0, \alpha_1, \alpha_2) \;=\; \alpha_0 p_0 + \alpha_1 p_1 + \alpha_2 p_2 \;=\; p_0 + \alpha_1(p_1 - p_0) + \alpha_2(p_2 - p_0).$$

The set of all affine combinations of three (noncolinear) points generates a plane. The set of all convex combinations of three points generates all the points of the triangle defined by the points. These shapes are called the *affine span* or *affine closure*, and *convex closure* of the points, respectively.

Euclidean Geometry: In affine geometry we have provided no way to talk about angles or distances. Euclidean geometry is an extension of affine geometry which includes one additional operation, called the *inner product*, which maps two real vectors (not points) into a nonnegative real. One important example of an inner product is the *dot product*, defined as follows. Suppose that the d-dimensional vector $\vec{u}$ is represented by the (nonhomogeneous) coordinate vector $(u_1, u_2, \ldots, u_d)$. Then define

$$\vec{u} \cdot \vec{v} = \sum_{i=0}^{d-1} u_i v_i,$$

The dot product is useful in computing the following entities.

Length: of a vector $\vec{v}$ is defined to be $|\vec{v}| = \sqrt{\vec{v} \cdot \vec{v}}$.

Normalization: Given any nonzero vector $\vec{v}$, define the *normalization* to be a vector of unit length that points in the same direction as $\vec{v}$. We will denote this by $\hat{v}$:

$$\hat{v} = \frac{\vec{v}}{|\vec{v}|}.$$

Distance between points: Denoted either $\mathrm{dist}(p, q)$ or $|pq|$ is the length of the vector between them, $|p - q|$.

Angle: between two nonzero vectors $\vec{u}$ and $\vec{v}$ (ranging from 0 to π) is

$$\mathrm{ang}(\vec{u}, \vec{v}) = \cos^{-1}\left(\frac{\vec{u} \cdot \vec{v}}{|\vec{u}||\vec{v}|}\right). = \cos^{-1}(\hat{u} \cdot \hat{v}).$$

This is easy to derive from the law of cosines.

Lecture 3: Orientations and Convex Hulls

Reading: Chapter 1 in the 4M's (de Berg, van Kreveld, Overmars, Schwarzkopf). The divide-and-conquer algorithm is given in Joseph O'Rourke's, "Computational Geometry in C." O'Rourke's book is also a good source for information about orientations and some other geometric primitives.

Orientation: In order to make discrete decisions, we would like a geometric operation that operates on points in a manner that is analogous to the relational operations $(<, =, >)$ with numbers. There does not seem to be any natural intrinsic way to compare two points in d-dimensional space, but there is a natural relation between ordered $(d+1)$-tuples of points in d-space, which extends the notion of binary relations in 1-space, called *orientation*.

Given an ordered triple of points $\langle p, q, r\rangle$ in the plane, we say that they have *positive orientation* if they define a counterclockwise oriented triangle, *negative orientation* if they define a clockwise oriented triangle, and *zero orientation* if they are collinear (which includes as well the case where two or more of the points are identical). Note that orientation depends on the order in which the points are given.

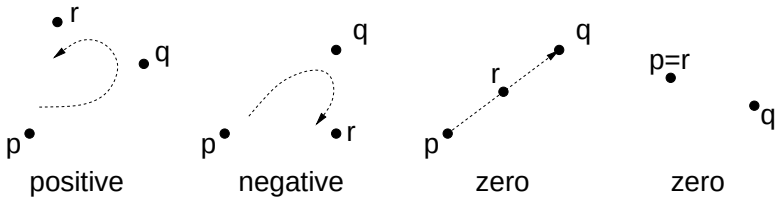

Figure 8: Orientations of the ordered triple (p, q, r).

Orientation is formally defined as the sign of the determinant of the points given in homogeneous coordinates, that is, by prepending a 1 to each coordinate. For example, in the plane, we define

$$\text{Orient}(p, q, r) = \det \begin{pmatrix} 1 & p_x & p_y \\ 1 & q_x & q_y \\ 1 & r_x & r_y \end{pmatrix}.$$

Observe that in the 1-dimensional case, Orient(p, q) is just $q - p$. Hence it is positive if $p < q$, zero if $p = q$, and negative if $p > q$. Thus orientation generalizes $<, =, >$ in 1-dimensional space. Also note that the sign of the orientation of an ordered triple is unchanged if the points are translated, rotated, or scaled (by a positive scale factor). A reflection transformation, e.g., $f(x, y) = (-x, y)$, reverses the sign of the orientation. In general, applying any affine transformation to the point alters the sign of the orientation according to the sign of the matrix used in the transformation.

In general, given an ordered 4-tuple points in 3-space, we can also define their orientation as being either positive (forming a right-handed screw), negative (a left-handed screw), or zero (coplanar). This can be generalized to any ordered $(d+1)$-tuple points in d-space.

Areas and Angles: The orientation determinant, together with the Euclidean norm can be used to compute angles in the plane. This determinant Orient(p, q, r) is equal to twice the signed area of the triangle $\triangle pqr$ (positive if CCW and negative otherwise). Thus the area of the triangle can be determined by dividing this quantity by 2. In general in dimension d the area of the simplex spanned by $d+1$ points can be determined by taking this determinant and dividing by $(d!)$. Once you know the area of a triangle, you can use this to compute the area of a polygon, by expressing it as the sum of triangle areas. (Although there are other methods that may be faster or easier to implement.)

Recall that the dot product returns the cosine of an angle. However, this is not helpful for distinguishing positive from negative angles. The sine of the angle $\theta = \angle pqr$ (the signed angled from vector $p - q$ to vector $r - q$) can

be computed as

$$\sin\theta = |p-q||r-q|\text{Orient}(q,p,r).$$

(Notice the order of the parameters.) By knowing both the sine and cosine of an angle we can unambiguously determine the angle.

Convexity: Now that we have discussed some of the basics, let us consider a fundamental structure in computational geometry, called the *convex hull*. We will give a more formal definition later, but the convex hull can be defined intuitively by surrounding a collection of points with a rubber band and letting the rubber band snap tightly around the points.

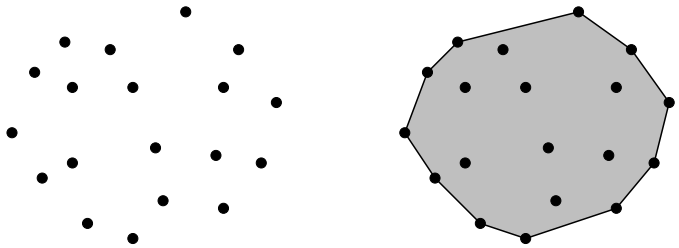

Figure 9: A point set and its convex hull.

There are a number of reasons that the convex hull of a point set is an important geometric structure. One is that it is one of the simplest shape approximations for a set of points. It can also be used for approximating more complex shapes. For example, the convex hull of a polygon in the plane or polyhedron in 3-space is the convex hull of its vertices. (Perhaps the most common shape approximation used in the minimum axis-parallel bounding box, but this is trivial to compute.)

Also many algorithms compute the convex hull as an initial stage in their execution or to filter out irrelevant points. For example, in order to find the smallest rectangle or triangle that encloses a set of points, it suffices to first compute the convex hull of the points, and then find the smallest rectangle or triangle enclosing the hull.

Convexity: A set S is *convex* if given any points $p, q \in S$ any convex combination of p and q is in S, or equivalently, the line segment $\overline{pq} \subseteq S$.

Convex hull: The *convex hull* of any set S is the intersection of all convex sets that contains S, or more intuitively, the smallest convex set that contains S. Following our book's notation, we will denote this $\mathcal{CH}(S)$.

An equivalent definition of convex hull is the set of points that can be expressed as convex combinations of the points in S. (A proof can be found in any book on convexity theory.) Recall that a convex combination of three or more points is an affine combination of the points in which the coefficients sum to 1 and all the coefficients are in the interval $[0, 1]$.

Some Terminology: Although we will not discuss topology with any degree of formalism, we will need to use some terminology from topology. These terms deserve formal definitions, but we are going to cheat and rely on intuitive definitions, which will suffice for the sort simple, well behaved geometry objects that we will be dealing with. Beware that these definitions are not fully general, and you are refered to a good text on topology for formal definitions. For our purposes, define a *neighborhood* of a point p to be the set of points whose distance to p is strictly less than some positive r, that is, it is the set of points lying within an open ball of radius r centered about p. Given a set S, a point p is an *interior point* of S if for some radius r the neighborhood about p of radius r is contained within S. A point is an *exterior point* if it an interior point for the complement of S.

Points that are neither interior or exterior are boundary points. A set is *open* if it contains none of its boundary points and *closed* if its complement is open. If p is in S but is not an interior point, we will call it a *boundary point*. We say that a geometric set is *bounded* if it can be enclosed in a ball of finite radius. A *compact set* is one that is both closed and bounded.

In general, convex sets may have either straight or curved boundaries and may be bounded or unbounded. Convex sets may be topologically open or closed. Some examples are shown in the figure below. The convex hull of a finite set of points in the plane is a bounded, closed, convex polygon.

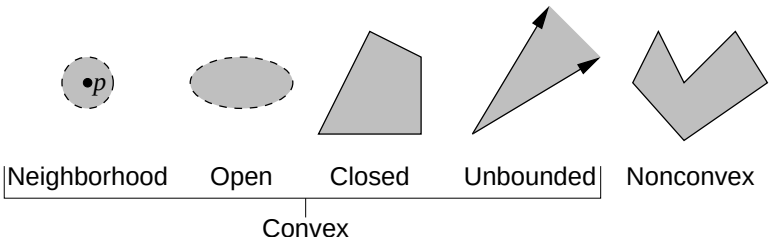

Figure 10: Terminology.

Convex hull problem: The (planar) *convex hull problem* is, given a set of n points P in the plane, output a representation of P's convex hull. The convex hull is a closed convex polygon, the simplest representation is a counterclockwise enumeration of the vertices of the convex hull. (A clockwise is also possible. We usually prefer counterclockwise enumerations, since they correspond to positive orientations, but obviously one representation is easily converted into the other.) Ideally, the hull should consist only of *extreme points*, in the sense that if three points lie on an edge of the boundary of the convex hull, then the middle point should not be output as part of the hull.

There is a simple $O(n^3)$ convex hull algorithm, which operates by considering each ordered pair of points (p, q), and the determining whether all the remaining points of the set lie within the half-plane lying to the right of the directed line from p to q. (Observe that this can be tested using the orientation test.) The question is, can we do better?

Graham's scan: We will present an $O(n \log n)$ algorithm for convex hulls. It is a simple variation of a famous algorithm for convex hulls, called *Graham's scan*. This algorithm dates back to the early 70's. The algorithm is based on an approach called *incremental construction*, in which points are added one at a time, and the hull is updated with each new insertion. If we were to add points in some arbitrary order, we would need some method of testing whether points are inside the existing hull or not. To avoid the need for this test, we will add points in increasing order of x-coordinate, thus guaranteeing that each newly added point is outside the current hull. (Note that Graham's original algorithm sorted points in a different way. It found the lowest point in the data set and then sorted points cyclically around this point.)

Since we are working from left to right, it would be convenient if the convex hull vertices were also ordered from left to right. The convex hull is a cyclically ordered sets. Cyclically ordered sets are somewhat messier to work with than simple linearly ordered sets, so we will break the hull into two hulls, an *upper hull* and *lower hull*. The break points common to both hulls will be the leftmost and rightmost vertices of the convex hull. After building both, the two hulls can be concatenated into a single cyclic counterclockwise list.

Here is a brief presentation of the algorithm for computing the upper hull. We will store the hull vertices in a stack U, where the top of the stack corresponds to the most recently added point. Let first(U) and second(U) denote the top and second element from the top of U, respectively. Observe that as we read the stack from top to bottom, the points should make a (strict) left-hand turn, that is, they should have a positive orientation. Thus, after adding the last point, if the previous two points fail to have a positive orientation, we pop them off the stack. Since the orientations of remaining points on the stack are unaffected, there is no need to check any points other than the most recent point and its top two neighbors on the stack.

Let us consider the upper hull, since the lower hull is symmetric. Let $\langle p_1, p_2, \ldots, p_n \rangle$ denote the set of points, sorted by increase x-coordinates. As we walk around the upper hull from left to right, observe that each consecutive triple along the hull makes a right-hand turn. That is, if p, q, r are consecutive points along the upper hull, then Orient$(p, q, r) < 0$. When a new point p_i is added to the current hull, this may violate the right-hand turn