Lecture 2
GEOMETRIC COMPUTATION (I)

What distinguishes geometric computation from purely numerical computation is its use of "geometry". But this seems to be circular. We give some prime examples of geometric computation, and tries to delineate those that are not.

§1. Convex Hull of Planar Point Set

The problem is this: given a set S of points in the plane, we want to compute its "convex hull". See figure 1 for the convex hull of 8 points.

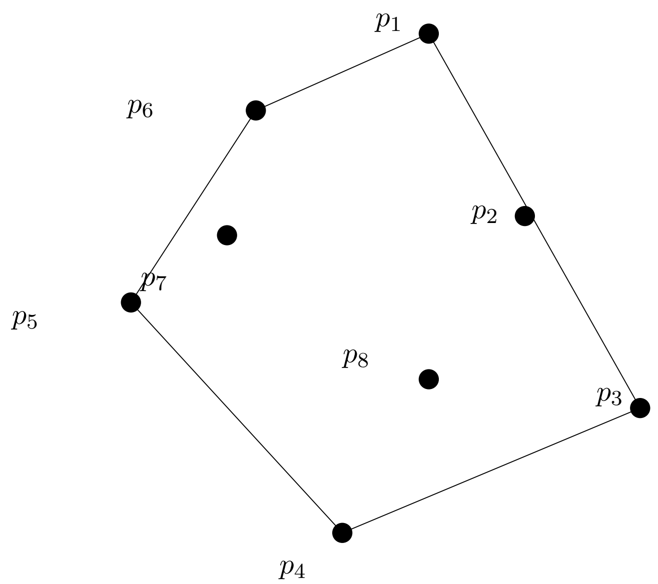


Figure 1: Convex hull of a set of points

The form of the output needs to be specified with some care: we not only want to determine the points in S which are in extremal position, we also want to list them in the correct circular order. Here, a possible ambiguity arises. Are points on the relative interior of a convex hull edge considered "extremal"? According to our definition of extremal below, such points are not extremal. But we may still want to list these points as convex hull points. As we will see, our solutions can handle whichever version we like.

Convexity Concepts. It is useful to take this opportunity to introduce some more general concepts of convexity in an Euclidean space $\mathbb{E}$ (of any dimensions). If $p, q \in \mathbb{E}$ are two points, then the closed line segment joining them is the set of points of the form

$$\lambda p + (1 - \lambda) q$$

for $0 \le \lambda \le 1$. This segment is denoted $[p, q]$. A set $R \subseteq \mathbb{E}$ of points is convex if for all $p, q \in R$, we $[p, q] \in R$. A point p is **extremal** in R if there is a line (hyperplane) through p that intersects R in exactly p. A set R is a **convex polyhedron** if R is a closed convex set with a finite number of extremal points. If R is also bounded, it is a **convex polytope**. A **polyhedron** (resp. **polytope**) is a finite union of convex polyhedrons (resp. polytopes). In 2-dimensions, a polytope is call a **polygon**.

Reduction to Circular Sorting. We can solve this in $O(n^2 \log n)$ time: for each points $p \in S$, we can decide if p is extremal in $O(n \log n)$ time, by circular sorting about p. Then p is extremal iff one of the angles spanned at p is at least (strictly more than) 180 deg.

Once we have the set of extremal points, we can order them in another $O(n^2)$ time. Actually, we can do this in linear time if we just retain a little information from the circular sorting phase above. Namely, when we determined a point p to be extremal, we have also found its two neighboring points p', p'' such that the angle $\angle(p', p, p'')$ is at least π (or strictly greater than π, depending on how we define the convex hull points). This information will give us the circular ordering of the convex hull points in a straightforward way.

Signed Area of a Triangle. Let us attend to the details of doing circular sorting. Can we avoid (transcendental) computation of angles in circular sorting? We first need to define the signed area of an oriented triangle.

Let A, B, C be three points in the plane. The area of ABC is half of the determinant

$$\Delta := \det \begin{bmatrix} a & a' & 1 \\ b & b' & 1 \\ c & c' & 1 \end{bmatrix} = \det \begin{bmatrix} b-a & b'-a' \\ c-a & c'-a' \end{bmatrix} \tag{1}$$

where $A :: (a, a')$, etc. To justify of this formula, we assume without loss of generality that $a = a' = 0$. First assume that both vectors (b, b') and (c, c') are in the first quadrant, with vector (b, b') making a larger angle with the positive x-axis than vector (c, c'). Referring to figure 2, the area of the parallelogram P is equal to the area of the enclosing rectangle $(b+b')(c+c')$ minus the area of the two smaller rectangles $2R = 2bc'$, minus the area of the 4 triangles $2(T_1 + T_2) = bb' + cc'$. This yields

$$\Delta = (b+b')(c+c') - (2bc' + bb' + cc') = b'c - bc'.$$

In case the vector $(b.b')$ makes a larger angle with the positive x-axis than vector (c, c'), we obtain the same area with an opposite sign. We leave it as an exercise to verify this formula even when the 2 vectors are not both in the first quadrant. Furthermore, if A, B, C are non-collinear, *the signed area of triangle ABC is positive iff A, B, C represents a counter-clockwise rotation.* There is also a generalization of this formula generalizes to volume in 3 and higer dimensions.

The Left-turn Predicate is defined as follows: $LeftTurn(A, B, C) = TRUE$ if the point C is to the left of the line from A to B. Thus

$$LeftTurn(A, B, C) \equiv AREA(A, B, C) > 0$$

How can we do circular sorting with the $LeftTurn$ predicate? First, let us suppose A has the smallest y-coordinate among the points in S. Then the angular sorting of S about A is easily done: Let C be the sorted list of the points in S, where each point B is sorted by the angle $\angle(XAB)$ where X is any point to the right of A. For points $p, q \in S$, p appears before q iff $LeftTurn(A, p, q)$. If A does not have the smallest y-coordinate, we partition S into $S = S_0 \uplus S_1$ where with S_0 comprising those points whose y-coordinate are at least the y-coordinate of A. We can now sort S_0 and S_1 separately, and join them into one circular list.

Gift Wrapping This algorithm is very intuitive. Begin with any point p_0 that on the convex hull of the set S, together with a known line through p_0 that intersects S at only p_0. Take a ray ρ along this line, and imagine it is a elastic string anchored at p_0. Start rotating ρ about p_0 (say, in the counter clockwise direction). In general, suppose we have found a sequence $p_0, p_1, \ldots, p_i$ of extremal points. The elastic string now stretches from p_0 to p_i, making a left-turn at each of the points $p_1, \ldots, p_i$, and finally stretches to infinity.

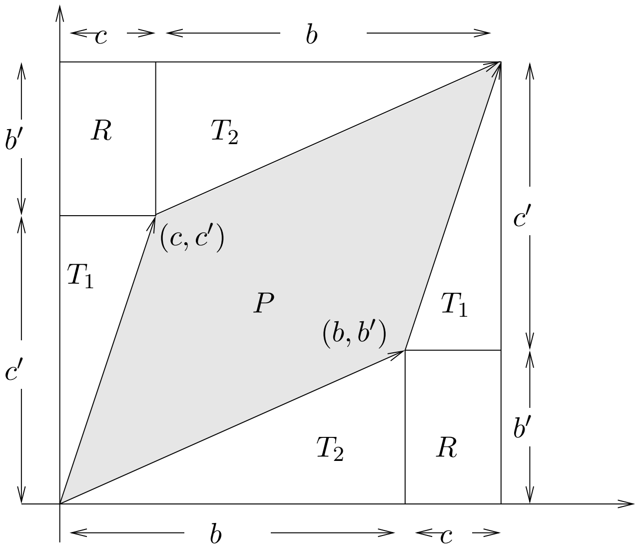


Figure 2: Area of parallelogram determined by vectors $(b, b'), (c, c')$.

We rotate ρ about p_i until it touches another extremal point p_{i+1}. Continue in this way, we eventually we return to p_0. The list of points found in this way constitute the convex hull.

How can we effect this physical action of rotating until we find the next point p_{i+1}? Using the direction from p_{i-1} to p_i as the initial direction, we need to find the point p such that the direction from p_i to p is smallest. Using the ideas above, this is easy to do in linear time. Hence the complexity of this algorithm is $O(nH)$ where $H \le n$ is the size of the hull. If $H = o(\log n)$, this can be faster than an $O(n \log n)$ algorithm.

Graham Scan Graham gave one of the first $O(n \log n)$ algorithm for this problem.

1. The first step is to find the $P :: (x, y)$ which is lexicographically minimum (minimize x and then minimize y). Let $P_0 :: (x_0, y_0)$ be this point. Clearly, P_0 is extremal be any definition of this word.
2. Sort the other points angularly about P_0. Angles are measured counter-clockwise from the negative y-axis. If two points have the same angle, drop the one with smaller x-coordinate. Let
$$P_0, P_1, \ldots, P_{n-1}$$
be the list of sorted points. Note that P_{n-1} is on the convex hull.
3. Create a stack S of points. Initials S with (P_{n-1}, P_0). In general, if $S = (Q_0, Q_1, \ldots, Q_k)$ were $k \ge 2$, then let *top* refer to Q_k and *next* refer to Q_{k-1}. To push an element Q onto S would make $S = (Q_1, \ldots, Q_k, Q)$. Similarly, for popping an element.
4. We process the points P_i in the i-th stage. If P_i is to the left of the line from *next* to *top* then push(P_i). Else, pop the stack.

Correctness of Graham Scan: we leave this as an exercise.

Divide and Conquer This is quite intuitive: we choose a vertical line L such that at most half of the points lies strictly to the left, and at most half of the points lies strictly to the right. Recursively, we construct the convex hull of the left and right subsets. We then construct the convex hull of the whole set by constructing the two "bridges" (these are the lines that are the common tangents of both convex hulls, and that has the 2 convex hulls on one side of the line). Once we have these lines, we can easily construct the final convex hull. This gives rise to the classic recurrence

$$T(n) \leq n + 2T(n/2)$$

which has solution $T(n) = O(n \log n)$.

Lower Bounds It can be shown that within a reasonable model of computation, every convex hull algorithm requires $\Omega(n \log n)$ steps in the worst case. In this sense, we usually call an algorithm with running time $O(n \log n)$ an "optimal algorithm". But witness the next result.

Ultimate Convex Hull ALgorithm A remarkable algorithm was discovered by Kirkpatrick and Seidel: their algorithm takes time $O(n \log H)$. Thus, it is simultaneously better than gift-wrapping and $O(n \log n)$. Notice that we manage to beat other "optimal algorithms". Of course, this is no contradiction because we have changed the rules of the analysis: we took advantage of another complexity parameter, h. This parameter h is a function of some **output size parameter**. Algorithms that can take advantage of the output size parameters are said to be **output-sensitive**. This is a very important idea of computational geometry. In terms of this h parameter, a lower bound of $\Omega(n \log h)$ can be shown.

OTHER ALGORITHMS There is another very general approach called incremental convex hull. We will return to this later, to introduce two other important themes in computational geometry: randomization and incremental algorithms. Another direction is to generalize the problem to the "dynamic setting". Here the point set S is considered a dynamic point set which can change over time. We can insert or delete from S. Incremental algorithms can be regarded as the solution for the "semi-dynamic" case in which we can only insert, but not delete points. Yet another direction is to consider the convex hull computation in higher (or arbitrary) dimensions.

Exercises

Exercise 1.1: Verify the area formula for all possible disposition of the vectors (b, b') and (c, c'). □

Exercise 1.2: Give a formula for the signed area of a simple polygon whose vertices are $A_1, A_2, \ldots, A_n$ (for $n \geq 3$). Let $A_i :: (x_i, y_i)$ for each i. □

Exercise 1.3: Can you derive the signed area of a tetrahedron in 3 dimensions? What is the volume formula for a d-dimensional simplex for $d > 3$? (In d dimensions, there is a constant c_d analogous to $c_2 = 1/2$ in 2 dimensions.) □

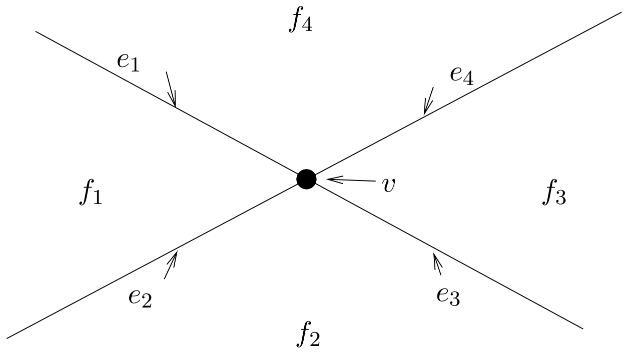
End Exercises

§2. The Nature of Geometric Computation

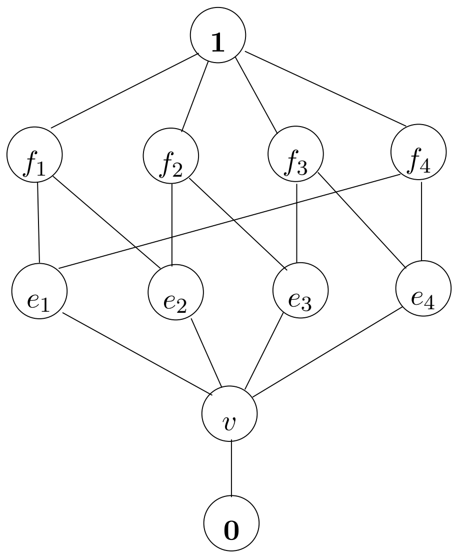
When is a computation said to be "geometric"? The above example of the convex hull problem would qualify in most opinions. Although we can examine more and more examples, we want to jump right into a general characterization. Let us begin with the uncontroversial remark that notions such as points, lines, hyperplanes, polytopes, triangulated surfaces are "geometric objects" *par excellence*. These objects can be in in "spatial relationship" with each other. Such relationships include inside/outside, disjoint/intersecting, touching, sidedness, co-incidence or other syzyzetic relations:

- A point can be inside a polytope.
- A line can lie on a hyperplane or be parallel to it, or intersect it transversally.
- Three points can be collinear.
- In dimensions 3 or more, four points may co-coplanar; and when co-planar, they may also be co-circular.

Such relationships are numerically defined, but more importantly, they are[1] **discrete**. A collection of geometric objects will define an implicit collection of such relationships, and these can be represented by combinatorial structures. For instance, a collection of points define a convex polytope (i.e., its convex hull). This polytope has facets of various dimensions which are in incidence (or adjacency) relationships. The **incidence graph** [2] is then used to represent this relationship among the facets. Let us define this concept for a collection H of n hyperplanes in d-dimensions: this collection partitions space into pairwise disjoint regions called **faces**, where each face f has a dimension $\dim(f)$ between 0 and d. Two faces f, g are **incident** (on each other) provided (1) $|\dim(f) - \dim(g)| = 1$ and (2) either $f \subseteq clg$ or $g \subseteq clf$. Here, clg denotes closure of a set g under the usual topology. The incidence graph $I(H)$ of H comprises a node $n(f)$ for each face f, and an edge from $n(f)$ to $n(g)$ provided $\dim(f) = \dim(g) + 1$ and f, g are incident. It is convenient to introduce two **improper faces** denoted $\mathbf{0}$ and $\mathbf{1}$ where $\dim(\mathbf{0}) = -1$ and $\dim(\mathbf{1}) = d + 1$. Moreover, every 0-dimensional face is incident on $\mathbf{0}$ and every d-dimensioan face is incident on $\mathbf{1}$.



(a) Arrangement of 2 lines



(b) Incidence Graph

Figure 3: Incidence Graphs

[1] Hoffmann [3] calls them "symbolic".

Thus, in general, we are dealing with collection of geometric objects in some special relations, which we might term a "geometric complex" for lack of a better term. Most geometric computation can be classified into the following categories: (1) constructing geometric complexes, (2) deriving geometric relationship among geometric complexes, and (3) searching in geometric complexes. As geometric complexes straddle the continuous world of numbers and the discrete world of relations, we suggest the mnemonic,

$$GEOMETRIC = NUMERIC + COMBINATORIAL \tag{2}$$

However, this could be misleading.

Example 1 (Non-Geometric Computations)
(a) Computing the determinant of a matrix M is non-geometric, even though the matrix structure is a combinatorial structure (albeit very regular one) and there are numerical entries.

(b) Computing the shortest path in a graph with edge weights (say, using Djikstra's algorithm) is likewise non-geometric, even though it has a combinatorial structure (the graph) and numerical values (the weights). ■

What is implicit in the equation (2) is that there are certain **consistency constraints** that must hold between the numeric and the combinatorial parts. In the extreme case, the numeric part completely determines the combinatorial part. In these two non-examples of geometric computation, there is no such relationship between the combinatorial and its numerical parts.

Remark. We should say that whenever we speak of "geometry" we have Euclidean geometry as our prime example. Extensions to other geometries studied in mathematics such as hyperbolic geometry, projective geometry, etc, are usually straightforward.

§3. Approaches to Robust Computation

Here are the main strategies that can be taken to solve nonrobustness:

Arithmetic Approaches. Since numerical errors is the source of non-robustness, we might try to fix the basic numerical system we use. One obvious direction is to abandon fixed-precision. It turns out that there are more issues that must be addressed, and this in essense is our preferred approach.

But within the fixed-precision paradigm, there are several proposals to generalize the usual notion of floating point numbers. These generalized floating point systems can greatly increase the range of the representable numbers (for any fixed numbers of bits). While these can decrease the incidence of non-robustness, by themselves, they do not eliminate nonrobustness. For instance, in conventional floating point systems, the number of bits allocated to the mantissa and the exponent are fixed. Matsui and Iri [7] proposed a system where these can vary. Another proposal is **level-index arithmetic** proposed by Clenshaw, Olver and Turner [1]. The price to be paid in these generalizations is the increased complexity of the algorithms for arithmetic operations.

One attempt to improve fixed precision arithmetic is to introduce some specialized low level operations that is widely used. In recent years, there has appeared in hardware the operation called FMA (fused multiply and add). This is the ternary operation of $FMA(a, b, c) = ab + c$. It is clearly very useful for implementing a scalar product $\sum_{i=1}^{n} a_i b_i$ using only n FMA's. Moreover, the rounding error of FMA is only one bit, which is an improvement over a stage rounding implied by the add and multiply operations.

Another major approach is to use **interval arithmetic** [8]. There are major research efforts aimed producing software libraries to support interval arithmetic. Note that interval arithmetic is independent of the FP approach: it is just as useful in arbitrary precision computation. One problem is that the interval bounds can rapidly increase, unless one is willing to abandon the FP paradigm.

Linguistic Approaches. Such approaches focuses on introducing linguistic constructs into the programming language which facilitate the construction of robust programs. A prime example is the language **Numerical Turing** [4, 5, 6] in which there is a concept of "precision block". The computation within such a block can be repeated with increasing precision until some objective is satisfied. Moreover, the proper treatment of arithmetic exceptions is recognized as an important part of the language (as noted earlier, most current languages do not support this properly, despite the IEEE standard for exceptions).

Geometric Approaches. Since "qualitative errors" is often a geometric concept, we can attempt to change the underlying geometry. In the introduction, we already explored this interpretation of epsilon-tweaking. There are many potential candidates for this kind of unusual geometries, even for the simple concept of lines. This will be illustrated below.

To illustrate the kinds of geometric approaches, we begin by asking the question: what is a "finite precision line"?

- Interval Geometry: This is the generic name for the approach of "fat geometric objects" in lecture 1. It is the geometric analogue of interval arithmetic.
- Discretized Geometry: This is well-known in computer graphics: a line is a suitable set of pixels on a finite screen.
- Topological Distortion: A line is a polyline. The distortion depends on the environment in which the line is placed.
- Rounded Geometry:

In the next lecture, we will consider these ideas in more detail.

References

[1] C. Clenshaw, F. Olver, and P. Turner. Level-index arithmetic: an introductory survey. In P. Turner, editor, *Numerical Analysis and Parallel Processing*, pages 95–168. Springer-Verlag, 1987. Lecture Notes in Mathematics, No.1397.

[2] H. Edelsbrunner. *Algorithms in Combinatorial Geometry.* Springer-Verlag, 1987.

[3] C. M. Hoffmann. *Geometric and Solid Modeling: an Introduction.* Morgan Kaufmann Publishers, Inc., San Mateo, California 94403, 1989.

[4] Holt, Matthews, Rosselet, and Cordy. *The Turing Programming Language.* Prentice-Hall, Englewood Cliffs, NJ, 1988.

[5] T. Hull, A. Abraham, M. Cohen, A. Curley, C. Hall, D. Penny, and J. Sawchuk. Numerical Turing. *ACM SIGNUM newsletter*, 20(3):26–34, July, 1985.

[6] T. Hull, M. Cohen, J. Sawchuk, and D. Wortman. Exception handling in scientific computing. *ACM Trans. on Math. Software*, 14(3):201–217, 1988.

[7] S. Matsui and M. Iri. An overflow/underflow-free floating-point representation of numbers. *J. Inform. Process*, 4(3):123–133, 1981.

[8] R. E. Moore. *Interval Analysis.* Series in Automatic Computation. Prentice-Hall, Englewood Cliffs, N.J., 1966.