1 Introduction

Computational geometry encompasses a broad and rich area of problem domains. This week's assignment attempts to provide an overview of a few of the central themes and best-known results. While it's impossible to do justice to such a wide ranging area in a single assignment, I hope that you will get a feeling for the subject matter and some of its scope.

At it's heart, computational geometry is concerned with the design of efficient algorithms for answering questions involving finite collections of geometric objects. *Visibility problems* arise when one is interested in computing unobstructed lines of sight. *Proximity problems* involve determining which of a given set of geometric objects are closest to a query object. *Range searching*, common in database applications, deals with queries asking for *all* objects satisfying some geometric constraint—for example all, reporting all geometric objects contained within a rectangular window. A related topic, *point location*, given a partition of a portion of the plane into polygonal regions, attempts to determine which region contains a query point. *Triangulations* and other graph structures imposed on a set of objects seek to capture the internal structure (or shape) of the set of objects: for example, one might add edges to a set of points in the plane in a way that attempted to capture some aspect of its internal structure. *Motion planning problems* involve computing paths that a geometric object might take through a collection of geometric obstacles without ever making contact with any of them. The moving object need not be rigid.

The above list is far from exhaustive. In order to avoid exhaustion, let's jump in....

2 Visibility

Some of the best known and most heavily studied problems in this area are visible/hidden surface determination problems in computer graphics. Another well-studied area is sensor placement. As an example of the latter, consider the (famous) result known as *Chvátal's Art Gallery Theorem*. We are given a *simple*[1] polygon. A pair of points $\{u, v\}$ on the interior or boundary of the polygon are *mutually visible* if the line segment connecting them is completely contained within[2] the polygon. Colloquially we often say that u and v *see* one another (or that u sees v, v sees u, etc.). A set X of points in P *guards* P if *every* point in P can be seen (is visible) from some point in X. Chvátal's Theorem states that regardless of the shape of the simple polygon P, one can always find a set X of at most $\lfloor n/3 \rfloor$ points in P such that X guards P, where n is the number of vertices of P. In fact, the set X of guards can all be assumed to be vertices of P.

Chvátal's original proof [1975], while lovely, was succeeded by a shorter, very elegant, proof by Fisk [1978]. Both proofs depend on the following (obvious-sounding) property:

Lemma 1. *Every polygon P contains a* diagonal[3].

Problem 1. *Prove Lemma 1. Note: Be careful to avoid making plausible but false (or unproven) claims!*

Question 1. *Did you find it difficult to achieve suitable precision of argument in your proof? Why?*

Problem 2. *Argue that Lemma 1 implies that any simple polygon can triangulated (divided into triangular regions) by diagonals and that every triangulation uses exactly $n - 3$ diagonals.*

Problem 3. *Prove Chvátal's Theorem. Hint: Use the result of Problem 2 to triangulate P, then color the vertices of P with a small number of colors; finally, exploit the coloring to identify an appropriate set of guards.*
How efficiently (in the $O()$ sense) could you find such a guard set?

This is the earliest and simplest of a large collection of problems related to visibility. For further reading see Joe O'Rourke's great text *Art Gallery Theorems and Applications*, which is on reserve at Schow.

There is an obvious optimization problem associated with Chvátal's Theorem, namely, that of computing a set of guards of minimum size. It took just over a decade before this problem was shown to be NP-hard by Lee and Lin.

[1] A (not necessarily convex) polygon P in which no edges intersect.

[2] That is, inside or on the boundaary of the polygon.

[3] A diagonal of a polygon P is a pair of non-consecutive vertices of P that are mutually visible.

Reading 1. *Read Lee and Lin's 1986 paper,* Computational Complexity of Art Gallery Problems*, which is available from the course webpage. Be prepared to sketch the proof of the NP-hardness of the original (vertex guards) art gallery problem.*

The triangulation theorem above provokes a further question: Given a simple polygon on n vertices, how efficiently can one find a triangulation of it? Usually one assumes that the polygon is described by an array of its vertices in the order that they would be visited by walking along the boundary of the polygon. This question proved much more challenging than expected: $O(n \log n)$ time algorithms were developed in the late '70s and early '80s. In the late '80s, the first $O(n \log \log n)$ algorithm was published, followed shortly thereafter by a $O(n \log^* n)$ algorithm. During the same period, fast (and simpler) randomized algorithms were developed. Finally, in 1990, Bernard Chazelle at Princeton published a (quite complicated) $O(n)$ time algorithm.

3 Range Searching & Point Location

Reading 2. *Read Bentley's 1975 (award-winning) paper,* Multidimensional Binary SearchTrees Used for Associative Searching. *The k-dimensional presentation can get a bit abstract. Work through the details of the construction of a 2-d-tree and be prepared to describe how the 2-d-tree can be constructed in $O(n \log n)$ time and $O(n)$ space as well as how a rectangular range query (i.e., report all points of the tree contained in some rectangular region) can be answered in $O(\sqrt{n} + k)$ time, where k is the number of points contained in the rectangle.*

A few years later, others developed data structures that reduced the time needed to answer range queries at the expense of (a modest amount of) additional storage. One structure was the *d-dimensional range tree*. In one dimension, where each point is a single number, the range tree is just a balanced binary search tree. Each number is stored as the key of an internal node as well as being stored at a leaf. Given an interval $[x_l, x_r]$, all leaves x in the tree with $x_l \leq x \leq x_r$ are reported. This reporting can be done in time $O(\log n + k)$ where k is the number of values reported.

Problem 4. *Explain how this time bound is achieved.*

Higher dimensional range trees can be recursively defined.

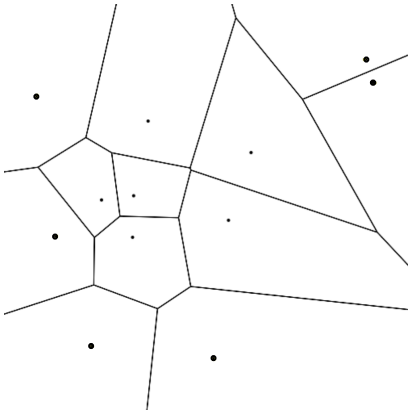
Reading 3. *Read Lueker's 1978 paper,* A Data Structure for Orthogonal Range Queries*. Be prepared to present an overview of the structure and how it achieves the stated average and worst-case running times.*

4 Proximity

One of the most fundamental problems in Computation Geometry is that of proximity. In its simplest form, one is given a finite set S of points in the plane (or higher dimensional space) and the goal is to process the points of S so that one can efficiently answer multiple queries of the form "Given point p, which point in S is closest to p? Given such a set $S = \{s_1, \ldots, s_n\}$—which for now we will assume are in *general position* [4]—we can partition of the plane into n regions, where region V_i consists of all points strictly closer to s_i than to any other point in S. This partition is called the *Voronoi diagram* of S; let's denote it by $V(S)$.

What does $V(S)$ look like? For any points $s_i, s_j \in S$, let $\ell_{i,j}$ be the perpendicular bisector of the line segment $\overline{s_i s_j}$, and let $\ell^i_{i,j}$ and $\ell^j_{i,j}$ be the half-planes determined by $\ell_{i,j}$ containing s_i and s_j respectively. $V_i = \bigcap_j \ell^i_{i,j}$, that is, V_i is the intersection of all of the half-planes containing s_i determined by the lines $\ell_{i,j}$. Here's an example, with the points of S having infinite Voronoi regions rendered larger:

[4] No 3 points on a common line and no 4 on a common circle.



Note that the boundaries of the regions, which are portions of the lines $\ell_{i,j}$ form a collection of edges and rays; we will use the notation $V(S)$ to refer to this 1-dimensional structure. We call the points at which multiple edges of the Voronoi diagram meet the *vertices* of the diagram.

Problem 5. *Prove that, if the points of S are in general position, as defined above, then each vertex of $V(S)$ has degree 3.*

We would like to be able to efficiently construct a data structure that captures the Voronoi diagram of S and allows for efficient answering of proximity questions for arbitrary points. There are a number of algorithms for constructing such a structure; one of the most well-known was designed by Steve Fortune.

Reading 4. *Read Fortune's 1986 paper,* A Sweepline Algorithm for Voronoi Diagrams. *Be prepared to explain, with pictures, how and why the algorithm works.*

There's a very useful dual structure hiding behind the Voronoi diagram of a set S. Create a graph having vertex set S as follows: For each pair $s_i, s_j \in S$ of points whose Voronoi regions share an edge of the diagram, create an edge in the graph. It turns out that if the points in S are in general position, then this graph is a triangulation—that is, a planar graph in which every internal face is a triangle. It is called the *Delaunay triangulation* of S.

Problem 6. *Prove that the graph described above is in fact a planar triangulation. Hint: See the previous problem.*

The Delaunay triangulation has many interesting features. Here is one.

Problem 7. *Let S be a set of points in general position. Define a graph $C(S)$ on S as follows: for each triple of points $x, y, z \in S$, the three edges $\{x, y\}, \{y, z\}, \{z, x\}$ are in $C(S)$ exactly when the (unique) circle passing through points x, y, z contains no other point of S either inside or on its boundary. Prove that $C(S)$ is the Delaunay triangulation of S.*

Harder (but not too much) to show is that, over all triangulations of S, $D(S)$ is the triangulation the maximizes the minimum angle over all internal angles of the triangulation. For you graphics fans, this makes the Delaunay triangulation a nice choice for creating a triangular mesh for the purposes of creating a 3-D terrain from a sample of 2-D points, since it avoids the kind of long, skinny triangles that make for poor images.

5 Further Reading

If you find this material interesting, you might enjoy looking at the draft chapter of a text by Joe Mitchell and Subhash Suri linked to on the course website. It provides a nice overview of several key areas in computational geometry. Also, I have put two well-known texts on reserve at Schow: *Computational Geometry*, by Preparata and Shamos, and *Computational Geometry*, by de Berg, van Kreveld, Overmars, and SchwartsKop. You might find them of use as you work through this week's assignment or as you consider project topics.