CMSC 425: Lecture 9
Geometric Data Structures for Games: Geometric Graphs
Thursday, Feb 21, 2013

Reading: Today's materials is presented in part in *Computational Geometry: Algorithms and Applications* (3rd Edition) by M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars.

Geometric Graphs and Subdivisions: We continue our discussion of fundamental geometric data structures. Today we will discuss geometric graphs and subdivisions. In computer games, it is often desirable to construct a framework to help plan the movement of various non-player entities. There are two particular structures of interest that we will consider.

Geometric graphs: This is a graph structure consisting of nodes and edges, where the nodes are points in space (for us, typically lying on a 2-dimensional surface) and the edges are straight lines (see Fig. 1(a)). Such graphs can be used for planning navigation and motion within an otherwise unstructured environment.

Subdivisions: A subdivision is a partitioning of some region of space into simple regions. A cell complex that subdivides a surface into a collection of cells is an example, but there are other types of subdivisions (see Fig. 1(b)). Subdivisions have many uses. They provide a framework onto which to anchor textures. They provide a method for describing points on a complex surface (by specifying the cell containing the point and location of the point within the cell). They can be used for interpolation, by specifying values at the vertices of the complex and interpolating values in between.

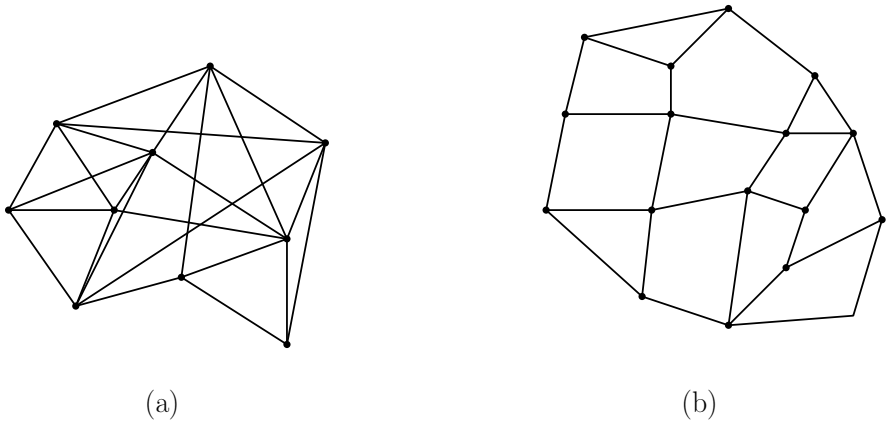

Fig. 1: A geometric graph (a), and a planar subdivision (b).

Although geometric graphs and subdivisions can be defined in obstacle-free environments, when used in games they are often constrained by the presence of constraining obstacles. In our discussions below we will often introduce each structure in an unconstrained setting, and then we will present strategies for restricting them depending on environmental constraints.

Visibility Graphs: Our first structure is motivated by the following simple navigation problem. Suppose that you want to compute the shortest path from one point s to another point t in the plane while

avoiding a collection of polygonal obstacles (see Fig. 2(a)). The shortest path between two points is a straight line, but such a line is only of use to us if it does not intersect the interior of any obstacles. What we would like to do is to compute a graph such that the shortest path in this graph is the shortest path between s and t.

Observation: The shortest path between any two points that avoids a set of polygonal obstacles in the plane is a polygonal curve (a sequence of line segments joined end to end), whose vertices are either vertices of the obstacles or the points s and t, such that no segment intersects the interior of any of the obstacles.

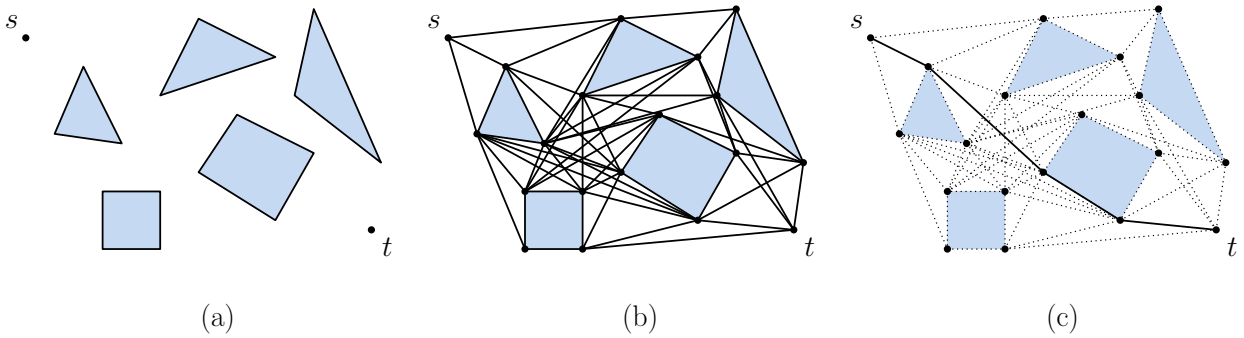

Fig. 2: A set of polygonal objects (a), the associated visibility graph (b), and a shortest path (c).

We say that two points u and v are *visible* to each other if the line segment $\overline{uv}$ does not intersect the interior of any obstacle. (Note that it is allowed to look right along the "wall" of any obstacle.) Basically, this observation tells us that that we should follow only visible segments, and it is never in our interest to "bend" the path unless we are at a corner point of some obstacle. This motivates the following definition:

Definition: The *visibility graph* of s and t and the obstacle set is a graph whose vertices are s and t the obstacle vertices, and vertices v and w are joined by an edge if v and w are either mutually visible or if (v, w) is an edge of some obstacle (see Fig. 2(b)).

It follows directly from the above observation that we can compute the shortest path between any two points by computing the visibility graph (including these points) and then running any shortest path algorithm, such as Dijkstra's algorithm on the result (see Fig. 2(c)).

If n denotes the total number vertices in our obstacles, then clearly the visibility graph can have as many as $O(n^2)$ edges. This is unfortunately rather large, but we will discuss a more efficient approach, which saves on space but sacrifices accuracy.

Can we compute the visibility graph efficiently? A naive approach would be to generate all $O(n^2)$ pairs of vertices (u, v), and then test whether the line segment $\overline{uv}$ intersects any obstacles. Testing a segment against the obstacles will take $O(n)$ time, which would lead to an $O(n^3)$ time algorithm. A more efficient approach is to perform an *angular sweep* around each vertex. Imagine that you want to simulate a rotating spot light centered at a vertex p (see Fig. 3). As you swing the spot-light beam around, you maintain a sorted list of edges that are intersected by the current spot-light beam. (The beam is actually an x-ray beam, since it penetrates through the obstacles.) Whenever the beam

encounters a new vertex v in the sweep, it updates the edge list by either inserting or deleting edges that are incident to this vertex. If the change affects the *closest edge*, then we have discovered a new visibility edge from p to v, which we add to our visibility graph.

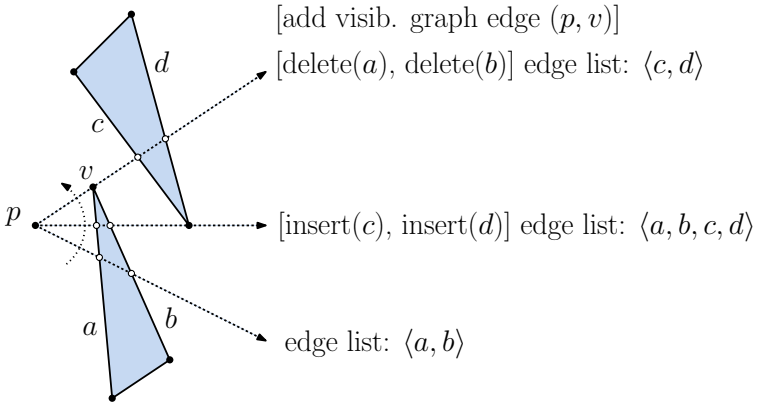

Fig. 3: Computing visibility edges by angular sweep.

How long does this angular-sweep algorithm take? There are n object vertices, and visiting them in angular sorted order can be done in $O(n \log n)$ time. (I am omitting lots of details, but trust me.) Since this sweep needs to be done for all n vertices, the total running time is $O(n^2 \log n)$, which is much faster than the $O(n^3)$ naive solution.

Yao Graphs, Θ-Graphs, and Spanners: The principal shortcoming of the visibility graph is its size. Can we avoid the quadratic size complexity? Here is an idea. Rather than storing an edge to *every* visible vertex, how about if we store only a subset of the "most relevant" edges. But, how do we know which edges are most relevant? If a vertex has has a very high degree in the visibility graph, we know that there must be many edges that are traveling in essentially the same direction. Do we need all of these edges? What if we were to store just a single edge to the closest of these vertices.

This motivates the following idea. First, fix an integer $m \geq 6$. For each vertex u of your obstacle set (including s and t), subdivide the angular space surrounding u into a collection m infinite cones, each subtending an angle of $\theta_m = 2\pi/m$. For each such cone, add an undirected edge between u and the closest visible vertex (if one exists) within this cone (see Fig. 4(a)). In contrast to the visibility graph, which might have $O(n^2)$ edges, this graph has only $O(nm)$ edges. If m is a small constant, this is much more space efficient. This is sometimes called the *Yao graph* (named after Andrew Yao, a famous computer scientist, who first used this construction).

For technical reasons, this definition is not the one that is usually preferred. (This is for two reasons, one practical and one theoretical. Unfortunately, it will take us too far afield to explain why.) The alternate definition that is preferred is the following. Again, consider any point u and all the visible vertices that lie within one of its cones. Project these vertices orthogonally onto the central axes of the cone. Add an undirected edge between u and the vertex whose projection is closest to u (see Fig. 4(a)). The resulting graph is called the *θ_m-graph*.

You might wonder whether the Yao graph (with m cones) and θ_m-graph are at all different. It would seem that the closest vertex to u will be the same as the vertex in the cone that has the closest projection onto the central axis. With a bit of work, it can be shown that they are indeed different, but you need

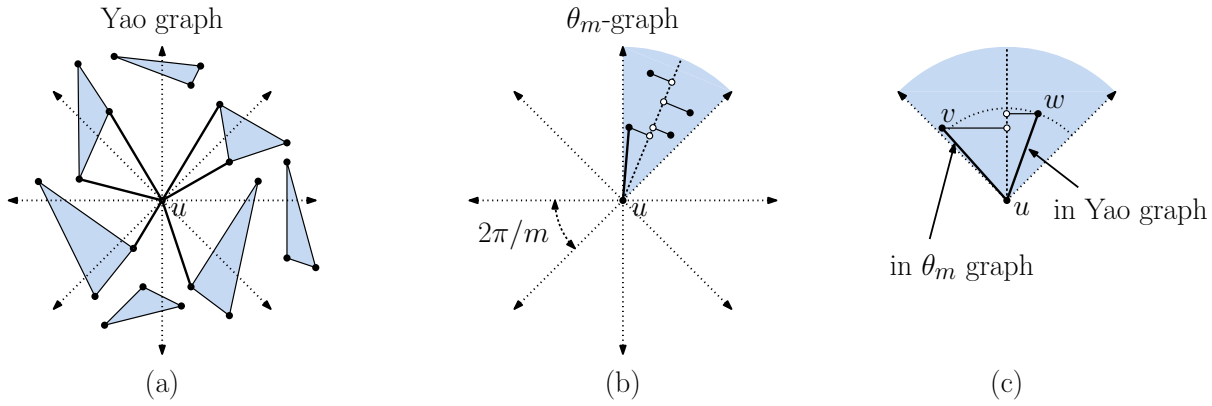

Fig. 4: The Yao graph (a), the θ_8-graph (b), and the difference between them (c).

a pretty large cone angle or you need to be quite careful in placing the points to generate a difference between the two (see Fig. 4(c)).

Both the Yao graph and the θ_m-graph are subgraphs of the visibility graph. Therefore, the shortest path from s to t in either of these graphs cannot be any shorter that the true shortest path. The interesting question is just how bad might the resulting path be. We will not prove it here, but the following theorem shows that, as m grows, the θ_m-graph produces ever more accurate approximations to the shortest path length.

Theorem: Consider any obstacle set and any two points s and t. Let ℓ denote the length of the shortest obstacle avoiding path from s to t (that is, the shortest path in the visibility graph) and ℓ_m is the length of the shortest path in the θ_m-graph, for $m \geq 7$. Then

$$\frac{\ell_m}{\ell} \leq \frac{1}{1 - 2\sin(\theta_m/2)},$$

where $\theta_m = 2\pi/m$ (the cone angle).

There is a similar bound for Yao graphs, where the ratio is $1/(\cos\theta - \sin\theta)$ assuming you have at least 9 cones. You might ask, then how large do I need to set m to get a reasonably good approximation? To guarantee that ratio is at most 2 (a 100% error), you would need to set $m = 13$. To get the ratio down to 1.1 (a 10% error) you would need to set $m = 70$. These bounds are terribly pessimistic, however. In practice, I would conjecture that setting $m = 12$, would probably yield paths that are nearly indistinguishable from the optimum.

By the way, a graph that has the property that the shortest path in the graph is an approximation to the true shortest path is called a *spanner*. The theorem above states that the θ_m-graph is a ρ-spanner, where $\rho = 1/(1 - 2\sin(\theta_m/2)$ spanner.

In case you are interested, θ_m-graphs can be computed quite efficiently, in roughly $O(nm \log n)$ time. Since the graph has $O(nm)$ edges, this is pretty close to optimal. Indeed, the principal reason that people prefer the θ_m-graph over the Yao graph is that θ_m graphs can be computed more efficiently. In practice, the two data structures are virtually identical in all other respects.

Voronoi Diagrams and Delaunay Triangulations: Next, let us consider some examples of important geometric subdivisions. Suppose you are give a collection of points P in the plane, and you would like to define the notion of a *sphere of influence* for each point. For example, each point might represent a guard tower. Each guard tower is responsible for guarding the points that are closer to it than some other guard tower. This implicitly subdivides the plane into regions according to which of the guard towers is closest to it. The resulting subdivision of the plane is called the *Voronoi diagram*, and it is geometric structure of fundamental importance. An example of such a diagram is shown in Fig. 5.

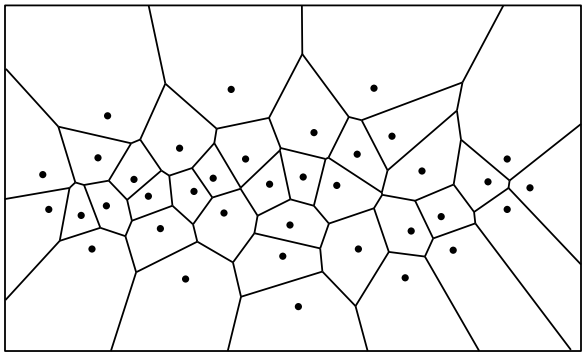

Fig. 5: The Voronoi diagram.

We will say much about the Voronoi diagram here. (If you want to learn more, take a course on Computational Geometry.) Given a set of n points, the Voronoi diagram can be computed in $O(n \log n)$ time. It can be represented as a DCEL (doubly-connected edge list).

The Voronoi diagram of a set of points in the plane is a planar subdivision, that is, a cell complex. The *dual* of such subdivision is another subdivision that is defined as follows. For each face of the Voronoi diagram, we create a vertex (corresponding to the point). For each edge of the Voronoi diagram lying between two sites p_i and p_j, we create an edge in the dual connecting these two vertices. Finally, each vertex of the Voronoi diagram corresponds to a face of the dual (see Fig. 6).

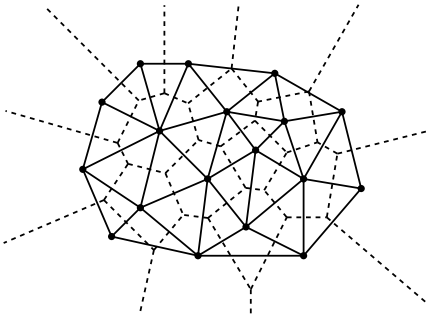

Fig. 6: The Delaunay triangulation (solid) and Voronoi diagram (dashed).

The resulting dual graph is also planar subdivision. Observe that (assuming that the points are not in some highly improbable configuration) the vertices of the Voronoi diagram all have degree three, that is, they are each incident to the three edges. It follows that the faces of the resulting dual graph (excluding the exterior face) are all triangles. Thus, the resulting dual graph is a triangulation of the sites, called the *Delaunay triangulation*.

Why are Delaunay triangulations of interest? If you are given a set of points and you want to determine the "best" way to connect them to form a cell complex, it is hard to beat the Delaunay triangulation. It has a number of interesting properties. First, it tends to produce nice "fat" triangles. For example, among all possible ways of triangulating a point set, the Delaunay triangulation maximizes the minimum angle appearing in any of its triangles. (It hates skinny triangles.)

Another interesting property is that, if you take any triangle, and consider the circumcircle passing through these three points, this circle cannot contain any of the other points inside it. This is called the *empty circumcircle property*. This might seem to be a property that many triangulations might possess, but this not the case. Indeed, if every triangle of a triangulation of a set of points satisfies the empty circumcircle property, then this is the Delaunay triangulation.

The Relative Neighborhood Graph and Gabriel Graph: We said that the Delaunay triangulation is a cell complex, but it can be thought of as a geometric graph. It is but one example of a straight-line planar graph defined on a set P of n points in the plane. Two other popular examples include the *Gabriel graph* and the *relative neighborhood graph* (or RNG). Let P be any set of points in the plane. As with the Delaunay triangulation, the points of P will be the vertices of graph.

In the Gabriel graph, denoted $\mathrm{GG}(P)$, two points p_i and p_j are joined by an edge if the disk whose diameter is p_ip_j contains no other points of P (see Fig. 7(b)).

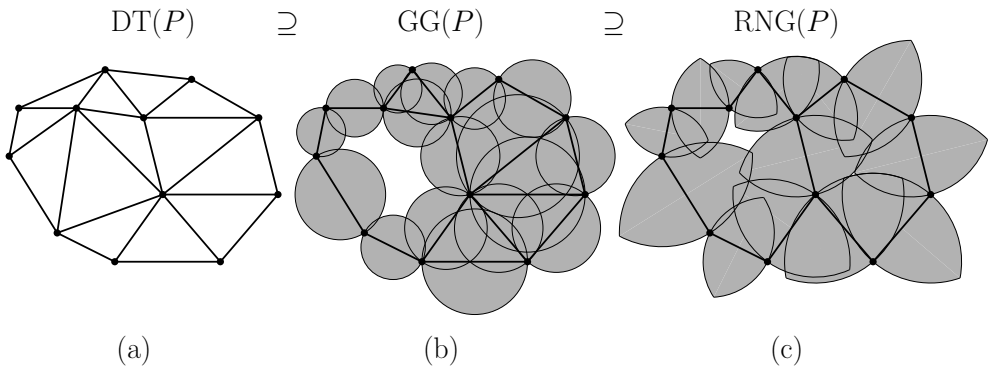

Fig. 7: The Delaunay triangulation (a), Gabriel graph (b), and the relative neighborhood graph (c).

In the RNG, denoted $\mathrm{RNG}(P)$, there is an edge joining p_i and p_j if there is no point $p_k \in P$ that is simultaneously closer to p_i and p_j than they are to each other. That is, there is no p_k such that $\max(\mathrm{dist}(p_i, p_k), \mathrm{dist}(p_j, p_k)) < \mathrm{dist}(p_i, p_j)$. Given two points p_i and p_j, define their *lune* to be the "football shaped' region where the circle centered at p_i and passing through p_j intersects the circle centered at p_j and passing through p_i. The RNG edge condition is equivalent to saying that p_i and p_j are adjacent if and only if $\mathrm{lune}(p_i, p_j)$ contains no other points of P (see Fig. 7(c)).

We will leave it as an exercise to prove that these three geometric graphs form a nested hierarchy, that is, for any point set P, we have $\mathrm{RNG}(P) \subseteq \mathrm{GG}(P) \subseteq \mathrm{DT}(P)$.