Delaunay Triangulations

Presented by Glenn Eguchi
6.838 Computational Geometry
October 11, 2001
Motivation: Terrains

- Set of data points $A \subset \mathbb{R}^2$
- Height $f(p)$ defined at each point $p$ in $A$
- How can we most naturally approximate height of points not in $A$?
Option: Discretize

- Let \( f(p) = \) height of nearest point for points not in \( A \)
- Does not look natural
Better Option: Triangulation

- Determine a *triangulation* of A in R2, then raise points to desired height
- *triangulation*: planar subdivision whose bounded faces are triangles with vertices from A
Triangulation: Formal Definition

- **maximal planar subdivision**: a subdivision $S$ such that no edge connecting two vertices can be added to $S$ without destroying its planarity
- **triangulation** of set of points $P$: a maximal planar subdivision whose vertices are elements of $P
Triangulation is made of triangles

- Outer polygon must be convex hull
- Internal faces must be triangles, otherwise they could be triangulated further
Triangulation Details

For $P$ consisting of $n$ points, all triangulations contain $2n-2-k$ triangles, $3n-3-k$ edges

- $n =$ number of points in $P$
- $k =$ number of points on convex hull of $P$
Terrain Problem, Revisited

- Some triangulations are “better” than others
- Avoid skinny triangles, i.e. maximize minimum angle of triangulation
Angle Optimal Triangulations

- Create *angle vector* of the sorted angles of triangulation $T$, $(\alpha_1, \alpha_2, \alpha_3, \ldots \alpha_{3m}) = A(T)$ with $\alpha_1$ being the smallest angle.
- $A(T)$ is larger than $A(T')$ iff there exists an $i$ such that $\alpha_j = \alpha'_j$ for all $j < i$ and $\alpha_i > \alpha'_i$.
- Best triangulation is triangulation that is *angle optimal*, i.e. has the largest angle vector. Maximizes minimum angle.
Angle Optimal Triangulations

Consider two adjacent triangles of $T$:

- If the two triangles form a convex quadrilateral, we could have an alternative triangulation by performing an *edge flip* on their shared edge.
Illegal Edges

- Edge $e$ is illegal if:
  \[ \min_{1 \leq i \leq 6} \alpha_i < \min_{1 \leq i \leq 6} \alpha'_i. \]

- Only difference between $T$ containing $e$ and $T'$ with $e$ flipped are the six angles of the quadrilateral.
Illegal Triangulations

- If triangulation $T$ contains an illegal edge $e$, we can make $A(T)$ larger by flipping $e$.
- In this case, $T$ is an illegal triangulation.
Thale’s Theorem

- We can use Thale’s Theorem to test if an edge is legal without calculating angles.

Let $C$ be a circle, $l$ a line intersecting $C$ in points $a$ and $b$ and $p$, $q$, $r$, and $s$ points lying on the same side of $l$. Suppose that $p$ and $q$ lie on $C$, that $r$ lies inside $C$, and that $s$ lies outside $C$. Then:

$$\angle arb > \angle apb = \angle aqb > \angle asb.$$
Testing for Illegal Edges

- If $p_i, p_j, p_k, p_l$ form a convex quadrilateral and do not lie on a common circle, exactly one of $p_ip_j$ and $p_kp_l$ is an illegal edge.

- The edge $p_ip_j$ is illegal iff $p_l$ lies inside $C$. 
Computing Legal Triangulations

1. Compute a triangulation of input points $P$.
2. Flip illegal edges of this triangulation until all edges are legal.

- Algorithm terminates because there is a finite number of triangulations.
- Too slow to be interesting…
Sidetrack: Delaunay Graphs

• Before we can understand an interesting solution to the terrain problem, we need to understand Delaunay Graphs.

• Delaunay Graph of a set of points P is the dual graph of the Voronoi diagram of P
Delaunay Graphs

To obtain $DG(P)$:

- Calculate $Vor(P)$
- Place one vertex in each site of the $Vor(P)$
Constructing Delaunay Graphs

If two sites $s_i$ and $s_j$ share an edge ($s_i$ and $s_j$ are adjacent), create an arc between $v_i$ and $v_j$, the vertices located in sites $s_i$ and $s_j$. 
Constructing Delaunay Graphs

Finally, straighten the arcs into line segments. The resultant graph is $DG(P)$. 
Properties of Delaunay Graphs

No two edges cross; $\text{DG}(P)$ is a planar graph.

- Proved using Theorem 7.4(ii).
- Largest empty circle property
Delaunay Triangulations

- Some sets of more than 3 points of Delaunay graph may lie on the same circle.
- These points form empty convex polygons, which can be triangulated.
- *Delaunay Triangulation* is a triangulation obtained by adding 0 or more edges to the Delaunay Graph.
Properties of Delaunay Triangles

From the properties of Voronoi Diagrams…

- Three points $p_i, p_j, p_k \in P$ are vertices of the same face of the $\text{DG}(P)$ iff the circle through $p_i, p_j, p_k$ contains no point of $P$ on its interior.
Properties of Delaunay Triangles

From the properties of Voronoi Diagrams…

- Two points $p_i, p_j \in P$ form an edge of $\mathcal{DG}(P)$ iff there is a closed disc $C$ that contains $p_i$ and $p_j$ on its boundary and does not contain any other point of $P$. 
Properties of Delaunay Triangles

From the previous two properties...

- A triangulation $T$ of $P$ is a DT($P$) iff the circumcircle of any triangle of $T$ does not contain a point of $P$ in its interior.
Legal Triangulations, revisited

A triangulation $T$ of $P$ is legal iff $T$ is a $\text{DT}(P)$.

- $\text{DT} \rightarrow \text{Legal}$: Empty circle property and Thale’s Theorem implies that all $\text{DT}$ are legal

- $\text{Legal} \rightarrow \text{DT}$: Proved on p. 190 from the definitions and via contradiction.
DT and Angle Optimal

The angle optimal triangulation is a DT. Why?
• If $P$ is in general position, $\text{DT}(P)$ is unique and thus, is angle optimal.

What if multiple DT exist for P?
• Not all DT are angle optimal.
• By Thale’s Theorem, the minimum angle of each of the DT is the same.
• Thus, all the DT are equally “good” for the terrain problem. All DT maximize the minimum angle.
Terrain Problem, revisited

Therefore, the problem of finding a triangulation that maximizes the minimum angle is reduced to the problem of finding a Delaunay Triangulation.

So how do we find the Delaunay Triangulation?
How do we compute $\text{DT}(P)$?

- We could compute $\text{Vor}(P)$ then dualize into $\text{DT}(P)$.
- Instead, we will compute $\text{DT}(P)$ using a randomized incremental method.
Algorithm Overview

1. Initialize triangulation $T$ with a “big enough” helper bounding triangle that contains all points $P$.
2. Randomly choose a point $p_r$ from $P$.
3. Find the triangle $\Delta$ that $p_r$ lies in.
4. Subdivide $\Delta$ into smaller triangles that have $p_r$ as a vertex.
5. Flip edges until all edges are legal.
6. Repeat steps 2-5 until all points have been added to $T$.

Let’s skip steps 1, 2, and 3 for now…
Triangle Subdivision: Case 1 of 2

Assuming we have already found the triangle that $p_r$ lives in, subdivide $\Delta$ into smaller triangles that have $p_r$ as a vertex.

Two possible cases:
1) $p_r$ lies in the interior of $\Delta
Triangle Subdivision: Case 2 of 2

2) $p_r$ falls on an edge between two adjacent triangles
Which edges are illegal?

- Before we subdivided, all of our edges were legal.
- After we add our new edges, some of the edges of T may now be illegal, but which ones?
Outer Edges May Be Illegal

- An edge can become illegal only if one of its incident triangles changed.
- Outer edges of the incident triangles \( \{p_jp_k, p_ip_k, p_kp_j\} \) or \( \{p_ip_l, p_lp_j, p_jp_k, p_kp_i\} \) may have become illegal.
New Edges are Legal

Are the new edges (edges involving $p_r$) legal?

Consider any new edge $p_r p_l$.

Before adding $p_r p_l$,

- $p_l$ was part of some triangle $p_i p_j p_l$
- Circumcircle $C$ of $p_i$, $p_j$, and $p_l$ did not contain any other points of $P$ in its interior
New edges incident to $p_r$ are Legal

- If we shrink $C$, we can find a circle $C'$ that passes through $p_rp_1$
- $C'$ contains no points in its interior.
- Therefore, $p_rp_1$ is legal.

Any new edge incident $p_r$ is legal.
Flip Illegal Edges

• Now that we know which edges have become illegal, we flip them.

• However, after the edges have been flipped, the edges incident to the new triangles may now be illegal.

• So we need to recursively flip edges…
LegalizeEdge

$pr =$ point being inserted
$p_ip_j =$ edge that may need to be flipped

**LEGALIZEEDGE**($pr$, $p_ip_j$, $T$)
1. **if** $p_ip_j$ is illegal
2. **then** Let $p_ip_jp_1$ be the triangle adjacent to $prp_ip_j$ along $p_ip_j$
3. Replace $p_ip_j$ with $prp_1$
4. **LEGALIZEEDGE**($pr$, $p_ip_1$, $T$)
5. **LEGALIZEEDGE**($pr$, $p_1p_j$, $T$)
Flipped edges are incident to $p_r$

Notice that when LEGALIZEEDGE flips edges, these new edges are incident to $p_r$.

- By the same logic as earlier, we can shrink the circumcircle of $p_i p_j p_1$ to find a circle that passes through $p_r$ and $p_1$.
- Thus, the new edges are legal.
Bounding Triangle

Remember, we skipped step 1 of our algorithm.

1. Begin with a “big enough” helper bounding triangle that contains all points.

Let \{p_{-3}, p_{-2}, p_{-1}\} be the vertices of our bounding triangle.

“Big enough” means that the triangle:

- contains all points of P in its interior.
- will not destroy edges between points in P.
Considerations for Bounding Triangle

• We could choose large values for $p_{-1}$, $p_{-2}$ and $p_{-3}$, but that would require potentially huge coordinates.

• Instead, we’ll modify our test for illegal edges, to act as if we chose large values for bounding triangle.
Bounding Triangle

We’ll *pretend* the vertices of the bounding triangle are at:

\[ p_{-1} = (3M, 0) \]
\[ p_{-2} = (0, 3M) \]
\[ p_{-3} = (-3M, -3M) \]

\[ M = \text{maximum absolute value of any coordinate of a point in P} \]
Modified Illegal Edge Test

$p_ip_j$ is the edge being tested

$p_k$ and $p_l$ are the other two vertices of the triangles incident to $p_ip_j$

Our illegal edge test falls into one of 4 cases.
Illegal Edge Test, Case 1

Case 1) Indices i and j are both negative

- \( p_i p_j \) is an edge of the bounding triangle
- \( p_i p_j \) is legal, want to preserve edges of bounding triangle
Illegal Edge Test, Case 2

Case 2) Indices i, j, k, and l are all positive.

- This is the normal case.
- $p_ip_j$ is illegal iff $p_l$ lies inside the circumcircle of $p_ip_jp_k$
Illegal Edge Test, Case 3

Case 3) Exactly one of i, j, k, l is negative

- We don’t want our bounding triangle to destroy any Delaunay edges.
- If i or j is negative, \( p_i p_j \) is illegal.
- Otherwise, \( p_i p_j \) is legal.
Illegal Edge Test, Case 4

Case 4) Exactly two of i, j, k, l are negative.

- k and l cannot both be negative (either $p_k$ or $p_l$ must be $p_r$)
- i and j cannot both be negative
- One of i or j and one of k or l must be negative
- If negative index of i and j is smaller than negative index of k and l, $p_ip_j$ is legal.
- Otherwise $p_ip_j$ is illegal.
Triangle Location Step

Remember, we skipped step 3 of our algorithm.

3. Find the triangle $T$ that $p_r$ lies in.

- Take an approach similar to Point Location approach.
- Maintain a point location structure $D$, a directed acyclic graph.
Structure of $D$

- Leaves of $D$ correspond to the triangles of the current triangulation.
- Maintain cross pointers between leaves of $D$ and the triangulation.
- Begin with a single leaf, the bounding triangle $p_{-1}p_{-2}p_{-3}$
Subdivision and $D$

- Whenever we split a triangle $\Delta_1$ into smaller triangles $\Delta_a$ and $\Delta_b$ (and possibly $\Delta_c$), add the smaller triangles to $D$ as leaves of $\Delta_1$
Subdivision and $\mathbf{D}$

$\Delta_1 \quad \Delta_2 \quad \Delta_3$

$\downarrow$ split $\Delta_1$

$\Delta_A \quad \Delta_B \quad \Delta_C$

$\Delta_1 \quad \Delta_2 \quad \Delta_3$
Edge Flips and \( D \)

- Whenever we perform an edge flip, create leaves for the two new triangles.
- Attach the new triangles as leaves of the two triangles replaced during the edge flip.
Edge Flips and $\Delta$
Searching $D$

$p_r = \text{point we are searching with}$

1. Let the current node be the root node of $D$.
2. Look at child nodes of current node. Check which triangle $p_r$ lies in.
3. Let current node = child node that contains $p_r$
4. Repeat steps 2 and 3 until we reach a leaf node.
Searching $\mathbb{D}$

- Each node has at most 3 children.
- Each node in path represents a triangle in $\mathbb{D}$ that contains $p_r$
- Therefore, takes $O(\text{number of triangles in } \mathbb{D} \text{ that contain } p_r)$
Properties of $D$

Notice that the:

- Leaves of $D$ correspond to the triangles of the current triangulation.
- Internal nodes correspond to *destroyed triangles*, triangles that were in an earlier stage of the triangulation but are not present in the current triangulation.
Algorithm Overview

1. Initialize triangulation $T$ with helper bounding triangle. Initialize $D$.

2. Randomly choose a point $p_r$ from $P$.

3. Find the triangle $\triangle$ that $p_r$ lies in using $D$.

4. Subdivide $\triangle$ into smaller triangles that have $p_r$ as a vertex. Update $D$ accordingly.

5. Call LEGALIZEEDGE on all possibly illegal edges, using the modified test for illegal edges. Update $D$ accordingly.

6. Repeat steps 2-5 until all points have been added to $T$. 
Analysis Goals

- Expected running time of algorithm is: \(O(n \log n)\)
- Expected storage required is: \(O(n)\)
First, some notation...

- $P_r = \{p_1, p_2, \ldots, p_r\}$
  - Points added by iteration $r$
- $\Omega = \{p_{-3}, p_{-2}, p_{-1}\}$
  - Vertices of bounding triangle
- $DG_r = DG(\Omega \cup P_r)$
  - Delaunay graph as of iteration $r$
Sidetrack: Expected Number of $\Delta$s

It will be useful later to know the expected number of triangles created by our algorithm…

**Lemma 9.11** Expected number of triangles created by $\text{DELAUNAYTRIANGULATION}$ is $9n+1$.

- In initialization, we create 1 triangle (bounding triangle).
Expected Number of Triangles

In iteration $r$ where we add $p_r$,

- in the subdivision step, we create at most 4 new triangles. Each new triangle creates one new edge incident to $p_r$
- each edge flipped in $\text{LEGALIZEEDGE}$ creates two new triangles and one new edge incident to $p_r$
Expected Number of Triangles

Let \( k \) = number of edges incident to \( p_r \) after insertion of \( p_r \), the degree of \( p_r \)

- We have created at most \( 2(k-3)+3 \) triangles.
- -3 and +3 are to account for the triangles created in the subdivision step

The problem is now to find the expected degree of \( p_r \)
Expected Degree of $p_r$

Use backward analysis:

- Fix $P_r$, let $p_r$ be a random element of $P_r$
- $D G_r$ has $3(r+3)-6$ edges
- Total degree of $P_r \leq 2[3(r+3)-9] = 6r$

$E[\text{degree of random element of } P_r] \leq 6$
Triangles created at step $r$

Using the expected degree of $p_r$, we can find the expected number of triangles created in step $r$.

$$\text{deg}(p_r, DG_r) = \text{degree of } p_r \text{ in } DG_r$$

\[
E[\text{number of triangles created in step } r] \leq E[2\text{deg}(p_r, DG_r) - 3]
\]
\[
= 2E[\text{deg}(p_r, DG_r)] - 3
\]
\[
\leq 2 \cdot 6 - 3 = 9
\]
Expected Number of Triangles

Now we can bound the number of triangles:
\[ \leq 1 \text{ initial } \Delta + \Delta s \text{ created at step 1 } + \Delta s \text{ created at step 2 } + \ldots + \Delta s \text{ created at step } n \]
\[ \leq 1 + 9n \]

Expected number of triangles created is 9n+1.
Storage Requirement

- D has one node per triangle created
- 9n+1 triangles created
- O(n) expected storage
Expected Running Time

Let’s examine each step…

1. Begin with a “big enough” helper bounding triangle that contains all points.
   O(1) time, executed once = O(1)

2. Randomly choose a point $p_r$ from $P$.
   O(1) time, executed n times = O(n)

3. Find the triangle $\Delta$ that $p_r$ lies in.
   
   *Skip step 3 for now...*
Expected Running Time

4. *Subdivide Δ into smaller triangles that have pr as a vertex.*
   
   O(1) time executed n times = O(n)

5. *Flip edges until all edges are legal.*
   
   In total, expected to execute a total number of times proportional to number of triangles created = O(n)

Thus, total running time without point location step is O(n).
Point Location Step

- Time to locate point $p_r$ is
  $O(\text{number of nodes of } \mathcal{D} \text{ we visit})$
  $+ O(1)$ for current triangle
- Number of nodes of $\mathcal{D}$ we visit
  $= \text{number of destroyed triangles that contain } p_r$
- A triangle is destroyed by $p_r$ if its circumcircle contains $p_r$

We can charge each triangle visit to a Delaunay triangle whose circumcircle contains $p_r$
**Point Location Step**

\[ K(\Delta) = \text{subset of points in } P \text{ that lie in the circumcircle of } \Delta \]

- When \( p_r \in K(\Delta) \), charge to \( \Delta \).
- Since we are iterating through \( P \), each point in \( K(\Delta) \) can be charged at most once.

Total time for point location:

\[
O(n + \sum_{\Delta} \text{card}(K(\Delta))),
\]
Point Location Step

We want to have $O(n \log n)$ time, therefore we want to show that:

$$\sum_{\Delta} \text{card}(K(\Delta)) = O(n \log n),$$
Point Location Step

Introduce some notation…

\[ T_r = \text{set of triangles of } \mathbb{DG}(\Omega \cup P_r) \]
\[ T_r \setminus T_{r-1} \] triangles created in stage \( r \)

Rewrite our sum as:

\[
\sum_{r=1}^{n} \left( \sum_{\Delta \in T_r \setminus T_{r-1}} \text{card}(K(\Delta)) \right).
\]
Point Location Step

More notation…

\[ k(P_r, q) = \text{number of triangles } \Delta \in T_r \text{ such that } q \text{ is contained in } \Delta \]

\[ k(P_r, q, p_r) = \text{number of triangles } \Delta \in T_r \text{ such that } q \text{ is contained in } \Delta \text{ and } p_r \text{ is incident to } \Delta \]

Rewrite our sum as:

\[ \sum \limits_{\Delta \in \mathcal{T}_r \setminus \mathcal{T}_{r-1}} \text{card}(K(\Delta)) = \sum \limits_{q \in P \setminus P_r} k(P_r, q, p_r). \]
Point Location Step

Find the $E[k(P_r, q, p_r)]$ then sum later...

- Fix $P_r$, so $k(P_r, q, p_r)$ depends only on $p_r$.
- Probability that $p_r$ is incident to a triangle is $3/r$

Thus:

$$E[k(P_r, q, p_r)] \leq \frac{3k(P_r, q)}{r}.$$
Point Location Step

Using:

\[ E[k(P_r, q, p_r)] \leq \frac{3k(P_r, q)}{r}. \]

We can rewrite our sum as:

\[ E\left[ \sum_{\Delta \in T_r \setminus T_{r-1}} \text{card}(K(\Delta)) \right] \leq \frac{3}{r} \sum_{q \in P \setminus P_r} k(P_r, q). \]
Point Location Step

Now find $E[k(P_r, p_{r+1})] \ldots$

- Any of the remaining $n-r$ points is equally likely to appear as $p_{r+1}$

So:

$$E[k(P_r, p_{r+1})] = \frac{1}{n-r} \sum_{q \in P \setminus P_r} k(P_r, q).$$
Point Location Step

Using:

\[ E[k(P_r, p_{r+1})] = \frac{1}{n-r} \sum_{q \in P \setminus P_r} k(P_r, q). \]

We can rewrite our sum as:

\[ E\left[ \sum_{\Delta \in \mathcal{T}_r \setminus \mathcal{T}_{r-1}} \text{card}(K(\Delta)) \right] \leq 3 \left( \frac{n-r}{r} \right) E[k(P_r, p_{r+1})]. \]
Point Location Step

Find $k(P_r, p_{r+1})$

- number of triangles of $\mathcal{T}_r$ that contain $p_{r+1}$
- these are the triangles that will be destroyed when $p_{r+1}$ is inserted; $\mathcal{T}_r \setminus \mathcal{T}_{r+1}$
- Rewrite our sum as:

$$E\left[ \sum_{\Delta \in \mathcal{T}_r \setminus \mathcal{T}_{r-1}} \text{card}(K(\Delta)) \right] \leq 3 \left( \frac{n-r}{r} \right) E\left[ \text{card}(\mathcal{T}_r \setminus \mathcal{T}_{r+1}) \right].$$
Point Location Step

Remember, number of triangles in triangulation of n points with k points on convex hull is 2n-2-k

- $T_m$ has $2(m+3)-2-3=2m+1$
- $T_{m+1}$ has two more triangles than $T_m$

Thus, $\text{card}(T_r \setminus T_{r+1})$

= card(triangles destroyed by $p_r$)

= card(triangles created by $p_r$) – 2

= card($T_{r+1} \setminus T_r$) - 2

We can rewrite our sum as:

$$E\left[ \sum_{\Delta \in T_r \setminus T_{r-1}} \text{card}(K(\Delta)) \right] \leq 3\left( \frac{n-r}{r} \right) \left( E[\text{card}(T_{r+1} \setminus T_r)] - 2 \right).$$
Point Location Step

Remember we fixed $P_r$ earlier…

- Consider all $P_r$ by averaging over both sides of the inequality, but the inequality comes out identical.

\[
E[\text{number of triangles created by } p_r] = E[\text{number of edges incident to } p_{r+1} \text{ in } T_{r+1}] = 6
\]

Therefore:

\[
E[\sum_{\Delta \in T_r \setminus T_{r-1}} \text{card}(K(\Delta))] \leq 12\left(\frac{n-r}{r}\right).
\]
Analysis Complete

\[ E \left[ \sum_{\Delta \in T_r \setminus T_{r-1}} \text{card}(K(\Delta)) \right] \leq 12 \left( \frac{n-r}{r} \right). \]

If we sum this over all \( r \), we have shown that:

\[ \sum_{\Delta} \text{card}(K(\Delta)) = O(n \log n), \]

And thus, the algorithm runs in \( O(n \log n) \) time.