Lecture 08 - Delaunay Triangulations

Philip Caplan
CSCI 0422 - Geometric Modeling (Spring 2022)
Learning objectives

By the end of this lecture you will be able to:

- determine whether a point is inside some triangle’s circumcircle,
- compute the Delaunay triangulation of a set of points using the Bowyer-Watson algorithm,
- define a few properties of Delaunay triangulations,
- identify issues with Delaunay triangulations.
Getting started...

Switch Host & Client today!

$ git pull
$ make update
$ cmake .
$ make template_class08_delaunay
$ cmake .

Compiling and running the exercise:

$ make class08_delaunay

Compiling and running the solution (after class):

$ make class08_delaunay_sol
So far: we’ve prescribed the connectivity between points to define triangles.

but what if you’re just given points with no assumed structure?

how will you generate a mesh?
Delaunay triangulations: no vertex is contained in the circumcircle of a triangle.
Delaunay triangulations: no vertex is contained in the circumcircle of a triangle.
How to determine if a point is inside a triangle circumcircle?

Calculate the sign of the determinant of the following matrix.

\[
\begin{vmatrix}
    p_{0,x} - q_x & p_{0,y} - q_y & (p_{0,x} - q_x)^2 + (p_{0,y} - q_y)^2 \\
    p_{1,x} - q_x & p_{1,y} - q_y & (p_{1,x} - q_x)^2 + (p_{1,y} - q_y)^2 \\
    p_{2,x} - q_x & p_{2,y} - q_y & (p_{2,x} - q_x)^2 + (p_{2,y} - q_y)^2 \\
\end{vmatrix}
\] (1)

we’ll call this the "incircle" predicate.
Another interpretation of the "incircle" property.

Each point $p$ lifted to $\mathbb{R}^3$: $(p, ||p||^2)$

→ can compute the Delaunay triangulation from the underside of the convex hull of the lifted points.
A note on convex hulls.
Exercises: complete the `delaunay_det2d` function.

First, investigate `src/flux-base/src/core/linear_algebra.h` and look for a function to calculate determinants.

```cpp
double delaunay_det2d( const double* p0 , const double* p1 , const double* p2 , const double* q ) {
    // matrix to build up, provides entry read/write access like m(i,j)
    mats<3,3,double> m;

    flux_implement;
}
```

Then test on the right triangle defined in `delaunay.cpp` using the following values for `q`:

- `q = (0.25, 0.25),`
- `q = (1, 1),`
- `q = (1, 0).`

You can call `delaunay2d` to print the result.

Where is the circumcenter and what is the radius?
What about this?

![Diagram with points and lines labeled p₀, p₁, p₂, q, δ, and r.]
Use exact geometric predicates!

In flux:

- include predicates.h
- call initialize_predicates()
- call incircle, orient2d, etc.
These types of triangles do come up in practice!
The Bowyer-Watson algorithm.
The Bowyer-Watson algorithm can be used to construct DT’s incrementally.

Main idea: keep inserting a vertex into an existing Delaunay triangulation.
The Bowyer-Watson algorithm can be used to construct DT’s incrementally.

Main idea: keep inserting a vertex into an existing Delaunay triangulation.
The Bowyer-Watson algorithm can be used to construct DT’s incrementally.

Main idea: keep inserting a vertex into an existing Delaunay triangulation.
The Bowyer-Watson algorithm can be used to construct DT’s incrementally.

Main idea: keep inserting a vertex into an existing Delaunay triangulation.
How to initialize the Delaunay triangulation?

create "supertriangles" to enclose points
Bowyer-Watson algorithm.

1. Compute the bounding square and create corner (ghost) vertices.
Bowyer-Watson algorithm.

1. Compute the bounding square and create corner (ghost) vertices.
2. Create initial triangulation \( \mathcal{T} \) with two "super triangles" using the vertices from step 1.
Bowyer-Watson algorithm.

1. Compute the bounding square and create corner (ghost) vertices.
2. Create initial triangulation $\mathcal{T}$ with two "supertriangles" using the vertices from step 1.
3. Add each point $p$ from the input set $\mathcal{P}$, one at a time:
   (a) Search current $\mathcal{T}$ for all triangles whose circumcircle encloses $p$, call these "broken" triangles $\mathcal{C}$. 

(a) Search current $\mathcal{T}$ for all triangles whose circumcircle encloses $p$, call these "broken" triangles $\mathcal{C}$. 

(b) Compute the boundary of $\mathcal{C}$, i.e. the set of edges that define the polygon boundary of the triangles in $\mathcal{C}$.
(c) Remove all the triangles in $\mathcal{C}$ from $\mathcal{T}$.
(d) Insert new triangles into $\mathcal{T}$, which are defined by connecting the edges in Step 3b to $p$. 


Bowyer-Watson algorithm.

1. Compute the bounding square and create corner (ghost) vertices.
2. Create initial triangulation $\mathcal{T}$ with two "supertriangles" using the vertices from step 1.
3. Add each point $p$ from the input set $\mathcal{P}$, one at a time:
   (a) Search current $\mathcal{T}$ for all triangles whose circumcircle encloses $p$, call these "broken" triangles $\mathcal{C}$.
   (b) Compute the boundary of $\mathcal{C}$, i.e. the set of edges that define the polygon boundary of the triangles in $\mathcal{C}$. 
Bowyer-Watson algorithm.

1. Compute the bounding square and create corner (ghost) vertices.
2. Create initial triangulation $\mathcal{T}$ with two "supertriangles" using the vertices from step 1.
3. Add each point $p$ from the input set $\mathcal{P}$, one at a time:
   (a) Search current $\mathcal{T}$ for all triangles whose circumcircle encloses $p$, call these "broken" triangles $\mathcal{C}$.
   (b) Compute the boundary of $\mathcal{C}$, i.e. the set of edges that define the polygon boundary of the triangles in $\mathcal{C}$.
   (c) Remove all the triangles in $\mathcal{C}$ from $\mathcal{T}$.
Bowyer-Watson algorithm.

1. Compute the bounding square and create corner (ghost) vertices.

2. Create initial triangulation $\mathcal{T}$ with two "supertriangles" using the vertices from step 1.

3. Add each point $p$ from the input set $\mathcal{P}$, one at a time:
   
   (a) Search current $\mathcal{T}$ for all triangles whose circumcircle encloses $p$, call these "broken" triangles $\mathcal{C}$.
   
   (b) Compute the boundary of $\mathcal{C}$, i.e. the set of edges that define the polygon boundary of the triangles in $\mathcal{C}$.
   
   (c) Remove all the triangles in $\mathcal{C}$ from $\mathcal{T}$.
   
   (d) Insert new triangles into $\mathcal{T}$, which are defined by connecting the edges in Step 3b to $p$. 
Improving the performance of the algorithm.

- Pick one of the corners. Sort remaining vertices by distance from this point.
- Insert points in order of increasing distance from starting vertex.
- Keep track of one of the elements that was just created.
- Start your search for broken triangles from an element that was just created.
- Keep track of neighbors and update them as the triangulation is modified.

**Timing:** (seconds)

<table>
<thead>
<tr>
<th># points</th>
<th>unoptimized</th>
<th>optimized</th>
</tr>
</thead>
<tbody>
<tr>
<td>1k</td>
<td>0.033</td>
<td>0.0083</td>
</tr>
<tr>
<td>10k</td>
<td>2.98</td>
<td>0.090</td>
</tr>
<tr>
<td>100k</td>
<td>397.65</td>
<td>1.12</td>
</tr>
<tr>
<td>1M</td>
<td>n/a</td>
<td>17.26</td>
</tr>
</tbody>
</table>
Issues with geometry conformity.

possible solution: constrained Delaunay triangulation (recover edge constraints, no new points)
Issues with geometry conformity.

possible solution: constrained Delaunay triangulation (recover edge constraints, no new points)
Issues with geometry conformity.

possible solution: constrained Delaunay triangulation (recover edge constraints, no new points)
Issues with geometry conformity.

possible solution: conforming Delaunay triangulation (insert Steiner vertices)
Issues with geometry conformity.

possible solution: conforming Delaunay triangulation (insert Steiner vertices)
Your TODO list . . .

- nothing added to flux-base that requires testing today,
- work on Project 2!

next week: Voronoi diagrams!!

- office hours on Fridays are now from 1pm-2pm.
Don’t forget to commit and push your changes!

Host: (assuming you are in top-level flux directory)

```bash
$ git add exercises/class08
$ git commit -a -m "added exercises from lecture 8"
$ git push
```

If you are in a build directory, the first command would be: $ git add ../exercises/class08

Client:

```bash
$ git pull
```