Delaunay triangulation,

Theory vs practice

Olivier Devillers
Delaunay triangulation,

Theory vs practice


Bibliographical notes

[References, given in the abstract]

Benchmarks (src code)
Everybody in this room know what Delaunay is!
empty circle property
empty sphere property
A (partial) history of Delaunay algorithms
(and of computational geometry)
A (partial) history of Delaunay algorithms
(and of computational geometry)

Earlier algorithms [1970... ]
A (partial) history of Delaunay algorithms
(and of computational geometry)

Earlier algorithms [1970...] 

Incremental 2D/3D
Gift wrapping 2D
A (partial) history of Delaunay algorithms
(and of computational geometry)

Earlier algorithms [1970...]

Incremental 2D/3D

Gift wrapping 2D

Non optimal
"simple"
Actually coded
Linked to applications: meshes, reconstruction
A (partial) history of Delaunay algorithms
(and of computational geometry)

Earlier algorithms [1970...]

Worst case algorithms [1980...]

A (partial) history of Delaunay algorithms
(and of computational geometry)

Earlier algorithms [1970...]

Worst case algorithms [1980...]

Divide & conquer 2D

Plane sweep 2D
A (partial) history of Delaunay algorithms
(and of computational geometry)

Earlier algorithms [1970...]

Worst case algorithms [1980...]

Divide & conquer 2D

Plane sweep 2D

Non optimal
’simple’
actually coded
linked to applications: meshes, reconstruction
A (partial) history of Delaunay algorithms
(and of computational geometry)

Earlier algorithms [1970...]  
Worst case algorithms [1980...]  
Randomized algorithms [1990...]
A (partial) history of Delaunay algorithms
(and of computational geometry)

Earlier algorithms [1970...]  
Worst case algorithms [1980...]  
Randomized algorithms [1990...]  

Delaunay tree
Clarkson & Shor
Dynamic updates (history graph)
Delaunay hierarchy
Spatial sorting (BRIO)
A (partial) history of Delaunay algorithms
(and of computational geometry)

Earlier algorithms [1970. . . ]

Worst case algorithms [1980. . . ]

Randomized algorithms [1990. . . ]

Robustness issues [1995. . . ]
A (partial) history of Delaunay algorithms
(and of computational geometry)

Earlier algorithms [1970...]

Worst case algorithms [1980...]

Randomized algorithms [1990...]

Robustness issues [1995...]

Properties checking VRONI

Exact computation paradigm TRIANGLE
Talk outline
Talk outline

One word on robustness issues

Basic incremental algorithm

Locate by walk

Locate using randomized data structures

Vertex removal in 2D

Remarks on CGAL programming

Conclusion
One word on robustness issues

Basic incremental algorithm

Locate by walk

Locate using randomized data structures

Vertex removal in 2D

Remarks on CGAL programming

Conclusion
One word on robustness issues
One word on robustness issues

The exact computation paradigm
One word on robustness issues

Compute predicates approximately
One word on robustness issues

Compute predicates approximately

Certify result
One word on robustness issues

Compute predicates approximately

Certify result
One word on robustness issues

Compute predicates approximately

Certify result

Run exact computation
One word on robustness issues

- Compute predicates approximately
- Certify result
- Run exact computation
- Extra cost
One word on robustness issues

Compute predicates approximately

Certify result

Extra cost

Delaunay 2D 10M points

 Exact_predicates_inexact_constructions_kernel

10.6 seconds

Cartesian<double>

9.7 seconds
One word on robustness issues

Compute predicates approximately

Certify result

Extra cost

Delaunay 2D 10M points

Exact predicates inexact constructions kernel

Cartesian<double>

10.6 seconds  82 seconds

9.7 seconds  75 seconds
One word on robustness issues

Compute predicates approximately

Certify result

Extra cost

Delaunay 2D 10M points

Exact predicates inexact constructions kernel

3D

Cartesian<double>

10.6 seconds 82 seconds

may loop (or crash)
All benchmarks

2.3 GHz, 16 GByte workstation

CGAL 3.9 (Release mode)

3D

Exact predicates inexact constructions kernel

Exact_predicates_inexact_constructions_kernel

Cartesian<double>

10.6 seconds 82 seconds

9.7 seconds 75 seconds
Basic incremental algorithm

Locate by walk

Locate using randomized data structures

Vertex removal in 2D

Remarks on CGAL programming

Conclusion

One word on robustness issues
Basic incremental algorithm
Basic incremental algorithm

Locate
Basic incremental algorithm

Find conflicts
Basic incremental algorithm

Remove triangles
Basic incremental algorithm

Fill the hole
Locate by walk

Straight walk

Locate using randomized data structures

Vertex removal in 2D

Remarks on CGAL programming

Conclusion

One word on robustness issues

Basic incremental algorithm
Locate by walk - straight walk
Locate by walk - straight walk
Locate by walk - straight walk
Locate by walk - straight walk
Locate by walk - straight walk
Locate by walk - straight walk
Exit edge?
One orientation predicate
Locate by walk - straight walk

Exit edge? One orientation predicate
End of walk?
A second orientation predicate
Locate by walk - straight walk

End of walk?
A second orientation predicate
Locate by walk

- Straight walk
- Visibility walk

Locate using randomized data structures

Vertex removal in 2D

Remarks on CGAL programming

Conclusion

One word on robustness issues

Basic incremental algorithm
Locate by walk - visibility walk
Locate by walk - visibility walk
Locate by walk - visibility walk
Locate by walk - visibility walk
Locate by walk - visibility walk
Locate by walk - visibility walk
Locate by walk - visibility walk

Triangle with two exits
One orientation predicate
Locate by walk - visibility walk

Triangle with one exit

1.5 orientation predicate

One predicate if this neighbor tried first

Two predicates if this neighbor tried first
Locate by walk - visibility walk

1.25 orientation predicate?
Locate by walk

Visibility vs straight walk
<table>
<thead>
<tr>
<th>Visibility vs straight walk</th>
<th>2D and 3D</th>
</tr>
</thead>
<tbody>
<tr>
<td>less predicates per crossed edge</td>
<td></td>
</tr>
<tr>
<td>similar number of crossed edges</td>
<td></td>
</tr>
<tr>
<td>experimental / theoretical</td>
<td></td>
</tr>
</tbody>
</table>
Locate by walk

Visibility vs straight walk

Speed improvement?
Locate by walk

Visibility vs straight walk

Speed improvement?

Walk in Delaunay 1 Mpoints

Straight: 324 µs
Visibility: 285 µs
3D: 97 µs
Locate by walk

Visibility vs straight walk

Speed improvement?

Walk in Delaunay 1 Mpoints
- Straight: 324 µs
- Visibility: 285 µs
- 3D: 97 µs

Much easier to code
One word on robustness issues
Basic incremental algorithm

**Locate by walk**

- Straight walk
- Visibility walk
- Structural filtering

Locate using randomized data structures
Vertex removal in 2D
Remarks on CGAL programming
Conclusion
Locate by visibility walk - structural filtering
Locate by visibility walk - structural filtering
Locate by visibility walk - structural filtering

Walk may loop (not in Delaunay)
Locate by visibility walk - structural filtering

- Walk may loop (not in Delaunay)
- Robustness issue:
  - Non certified arithmetic
  - Rounding errors
  - Wrong decisions during walk
Locate by visibility walk - structural filtering

Walk may loop (not in Delaunay)

Robustness issue:
may loop
even in Delaunay
Locate by visibility walk - structural filtering

Walk may loop (not in Delaunay)

Robustness issue:
may loop
even in Delaunay
But only in very special configurations
Locate by visibility walk - structural filtering

Orientation predicates

Certify all along the walk

285 $\mu$seconds

Certify after a while, just in case

220 $\mu$seconds
Locate by visibility walk - structural filtering

Orientation predicates

Certify all along the walk

285 \( \mu \)seconds \quad 97 \( \mu \)seconds

Certify after a while, just in case

220 \( \mu \)seconds \quad 81 \( \mu \)seconds

2D \quad 3D

Walk in Delaunay 1 Mpoints
Locate by walk

- Straight walk
- Visibility walk
- Structural filtering
- Walk shape

Locate using randomized data structures

Vertex removal in 2D

Remarks on CGAL programming

Conclusion

One word on robustness issues

Basic incremental algorithm
Locate by visibility walk - walk shape
Locate by visibility walk - walk shape

Rightmost
Locate by visibility walk - walk shape

Leftmost
Locate by visibility walk - walk shape

In between
Locate by visibility walk - walk shape
Locate by visibility walk - walk shape

Turn counterclockwise from previous

Rightmost
Locate by visibility walk - walk shape

Turn clockwise from previous

Leftmost
Locate by visibility walk - walk shape

Balance left and right turns

first with proba $\frac{1}{3}$

first with proba $\frac{2}{3}$
Locate by visibility walk - walk shape

Walk in Delaunay 1 Mpoints

Leftmost  220 µseconds
Balanced  188 µseconds
## Locate by visibility walk

### Walk in Delaunay 1 Mpoints

<table>
<thead>
<tr>
<th>Walk Type</th>
<th>Time (µseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Straight walk</td>
<td>324</td>
</tr>
<tr>
<td>Visibility walk</td>
<td>285</td>
</tr>
<tr>
<td>Structural filtering</td>
<td>220</td>
</tr>
<tr>
<td>Balanced walk</td>
<td>188</td>
</tr>
</tbody>
</table>
One word on robustness issues
Basic incremental algorithm
Locate by walk

**Locate using randomized data structures**

Delaunay tree

Vertex removal in 2D
Remarks on CGAL programming
Conclusion
Data structures to locate - the Delaunay tree
Data structures to locate - the Delaunay tree
Data structures to locate - the Delaunay tree
Data structures to locate - the Delaunay tree
Data structures to locate - the Delaunay tree
Data structures to locate - the Delaunay tree
Data structures to locate - the Delaunay tree
Data structures to locate - the Delaunay tree
Data structures to locate - the Delaunay tree
Data structures to locate - the Delaunay tree
Data structures to locate - the Delaunay tree
Data structures to locate - the Delaunay tree
Data structures to locate - the Delaunay tree

locate based on incircle predicate

# triangles in the Delaunay tree
Data structures to locate - the Delaunay tree

How many triangles created by the last point?
Data structures to locate - the Delaunay tree

How many triangles created by the last point?
Data structures to locate - the Delaunay tree

How many triangles created by the last point?
Data structures to locate - the Delaunay tree

locate based on incircle predicate

\# triangles in the Delaunay tree

= 6n \ (\text{randomized})
Vertex removal in 2D
Remarks on CGAL programming
Conclusion

One word on robustness issues
Basic incremental algorithm
Locate by walk

**Locate using randomized data structures**

- Delaunay tree
- Delaunay hierarchy

Vertex removal in 2D
Remarks on CGAL programming
Conclusion
Data structures to locate - the Delaunay hierarchy
Data structures to locate - the Delaunay hierarchy
Data structures to locate - the Delaunay hierarchy
Data structures to locate - the Delaunay hierarchy
Data structures to locate - the Delaunay hierarchy
Data structures to locate - the Delaunay hierarchy
Data structures to locate - the Delaunay hierarchy
Data structures to locate - the Delaunay hierarchy
Data structures to locate - the Delaunay hierarchy
Data structures to locate - the Delaunay hierarchy

The Delaunay tree

locate based on incircle predicate

\# triangles in the Delaunay tree

= 6n (randomized)
Data structures to locate - the Delaunay hierarchy

The Delaunay hierarchy

based on orientation predicate

\# triangles in the hierarchy can be chosen

= 1.03 \times 2n \text{ (expected)}

The Delaunay tree

locate based on incircle predicate

\# triangles in the Delaunay tree

= 6n \text{ (randomized)}
Data structures to locate - the Delaunay hierarchy

The Delaunay hierarchy

based on orientation predicate

\# triangles in the hierarchy
can be chosen

= 1.03 \times 2n \ (expected)

The Delaunay tree

locate based on incircle predicate

\# triangles in the Delaunay tree

= 6n \ (randomized)

\[ O(n \log n) \]
Data structures to locate - the Delaunay hierarchy

The Delaunay hierarchy

Based on orientation predicate

\# triangles in the hierarchy

Can be chosen

\[ = 1.03 \times 2n \, (\text{expected}) \]

2.3 seconds

50,000 random points (original benchmarks in 2000).

The Delaunay tree

Locate based on incircle predicate

\# triangles in the Delaunay tree

\[ = 6n \, (\text{randomized}) \]

17 seconds
Locate using randomized data structures

- Delaunay tree
- Delaunay hierarchy
- Biased randomized insertion order
Data structures to locate - biased random insertion order
Data structures to locate - biased random insertion order

Locate is easy if you know a vertex nearby
Data structures to locate - biased random insertion order

Locate is easy if you know a vertex nearby

Natural idea: sort the points, locate from previous
Data structures to locate - biased random insertion order

\( x \)-order
Data structures to locate - biased random insertion order

Hilbert order
Data structures to locate - biased random insertion order

Biased order (Spatial sorting)
Data structures to locate - biased random insertion order

Biased order (Spatial sorting)
Data structures to locate - biased random insertion order
Data structures to locate - biased random insertion order

- Delaunay 2D 1M random points
- locate using Delaunay hierarchy: 6 seconds
- random order: 157 seconds
- $x$-order: 3 seconds
- Hilbert order: 0.8 seconds
- Biased order (Spatial sorting): 0.7 seconds
Data structures to locate - biased random insertion order

Delaunay 2D 100K parabola points

locate using Delaunay hierarchy 0.3 seconds
random order 128 seconds
$x$-order 632 seconds
Hilbert order 46 seconds
Biased order (Spatial sorting) 0.3 seconds
Data structures to locate

Construction of Delaunay 10 M random points

Delaunay tree $\sim 10$ mn (estimate)

Delaunay hierarchy 90 seconds

Biased random order 10.6 seconds
Remarks on CGAL programming

Conclusion

One word on robustness issues
Basic incremental algorithm
Locate by walk
Locate using randomized data structures

Vertex removal in 2D
Vertex removal
Vertex removal
Vertex removal
Vertex removal
Remarks on CGAL programming

One word on robustness issues
Basic incremental algorithm
Locate by walk
Locate using randomized data structures

Vertex removal in 2D
Various algorithms
Boundary expansion

Remarks on CGAL programming
Conclusion
Vertex removal - boundary expansion

release 3.5, 2D implementation
Vertex removal - boundary expansion

release 3.5, 2D implementation

hole boundary = queue
Vertex removal - boundary expansion

release 3.5, 2D implementation

find new incident triangle in linear time
Vertex removal - boundary expansion

release 3.5, 2D implementation
Vertex removal - boundary expansion

release 3.5, 2D implementation
One word on robustness issues
Basic incremental algorithm
Locate by walk
Locate using randomized data structures

**Vertex removal in 2D**

Various algorithms
Boundary expansion
Triangulate and sew

Remarks on CGAL programming
Conclusion
Vertex removal - triangulate and sew

current 3D implementation
Vertex removal - triangulate and sew

current 3D implementation

Delaunay of neighbors
Vertex removal - triangulate and sew

current 3D implementation

delete extra triangles and sew
Vertex removal - triangulate and sew

current 3D implementation

delete extra triangles and sew
Vertex removal in 2D

Various algorithms:
- Boundary expansion
- Triangulate and sew
- Flip the hole

Remarks on CGAL programming

Conclusion

One word on robustness issues

Basic incremental algorithm

Locate by walk

Locate using randomized data structures
Vertex removal - flip the hole
Vertex removal - flip the hole

triangulate from any vertex
Vertex removal - flip the hole

queue of edges to be checked
Vertex removal - flip the hole
Vertex removal - flip the hole
Vertex removal - flip the hole
Vertex removal - flip the hole
Vertex removal - flip the hole
Vertex removal - flip the hole
Vertex removal - flip the hole

a little bit faster
One word on robustness issues
Basic incremental algorithm
Locate by walk
Locate using randomized data structures

**Vertex removal in 2D**
Various algorithms
Low degree optimization

Remarks on CGAL programming
Conclusion
Vertex removal - low degree optimization

degree 3
Vertex removal - low degree optimization

degree 3

almost nothing to do
Vertex removal - low degree optimization

degree 4
Vertex removal - low degree optimization

just one incircle test to decide
Vertex removal - low degree optimization

degree 5
Vertex removal - low degree optimization

"star" the pentagon from the right vertex

degree 5
Vertex removal - low degree optimization

”star” the pentagon from the right vertex
Vertex removal - low degree optimization

"star" the pentagon from the right vertex

degree 5
Vertex removal - low degree optimization

"star" the pentagon from the right vertex

degree 5
Vertex removal - low degree optimization

“star” the pentagon from the right vertex
Vertex removal - low degree optimization

“star” the pentagon from the right vertex

degree 5
Vertex removal - low degree optimization

degree 5

Decision tree
Vertex removal - low degree optimization

degree 5

Decision tree
Vertex removal - low degree optimization

degree 6
Vertex removal - low degree optimization

degree 6
Vertex removal - low degree optimization

degree 6
Vertex removal - low degree optimization

degree 6
Vertex removal - low degree optimization

degree 6
Vertex removal - low degree optimization

degree 6
Vertex removal - low degree optimization

degree 6

14 results
Vertex removal - low degree optimization

degree 6
Decision tree
Vertex removal - low degree optimization

degree 6
Decision tree
Vertex removal - low degree optimization

degree 6
Decision tree

6 incircle predicates
Vertex removal - low degree optimization

degree 7
Decision tree
Vertex removal - low degree optimization

Degree 7
Decision tree

Symmetric tree

10 incircle predicates
Vertex removal - low degree optimization

<table>
<thead>
<tr>
<th>degree</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8*</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>† results</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>14</td>
<td>42</td>
<td>132</td>
<td>429</td>
</tr>
<tr>
<td>† leaves</td>
<td>1</td>
<td>2</td>
<td>6</td>
<td>24</td>
<td>130</td>
<td>≃500</td>
<td></td>
</tr>
<tr>
<td>⌈log₂ † results⌉</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>tree height</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>6</td>
<td>10</td>
<td>≃14</td>
<td></td>
</tr>
<tr>
<td>† lines of code</td>
<td>30</td>
<td>40</td>
<td>90</td>
<td>280</td>
<td>700</td>
<td>≃2500</td>
<td></td>
</tr>
</tbody>
</table>

* not implemented. The sizes of the tree and the code are estimated.
Vertex removal - low degree optimization

Remarks on implementation

limited memory allocation, use old faces "in place"
re-use as many neighbor links as possible
Vertex removal - low degree optimization

Remarks on implementation

limited memory allocation, use old faces "in place"
re-use as many neighbor links as possible

tree implementation

```python
if incircle(...)
    if incircle(...)
        if incircle(...)
            use_this_shape(face0, face1, face2...)
        else
            use_other_shape(face2, face3, face4...)
    else
        use_other_shape(face2, face3, face4...)
......
```
Vertex removal

deletion time per vertex

degree

10\mu s

Boundary expansion

small degrees

Flip the hole

3 4 5 6 7 8 9 10 11 \geq 12
Vertex removal

deletion time per vertex

10 μs

Boundary expansion

Flip the hole

small degrees

degree

3.5

3.6
Vertex removal

deletion time per vertex

10 µs

Boundary expansion

Flip the hole

small degrees

init (load memory)

degree

3 4 5 6 7 8 9 10 11 ≥ 12

3.5

3.6
Vertex removal

- Deletion time per vertex: 10 μs
- Boundary expansion
- Flip the hole
- Degree distribution
- Small degrees
- Init (load memory)

Degree distribution:
- Degree: 3, 4, 5, 6, 7, 8, 9, 10, 11, ≥ 12

Points:
- 30%
One word on robustness issues
Basic incremental algorithm
Locate by walk
Locate using randomized data structures
Vertex removal in 2D
Remarks on CGAL programming
Conclusion
Algorithmic choices
Theoretical efficiency

Algorithmic choices
Algorithmic choices

Theoretical efficiency

Practical efficiency
Algorithmic choices

Theoretical efficiency

Practical efficiency

Robustness issues
Algorithmic choices

Theoretical efficiency

Practical efficiency

Modularity
traits classes
data structures
geometry

Robustness issues

Minimal requirements
e.g. do not use strange predicates
Geometric traits

Triangulation data structure

Delaunay triangulation
$\mathbb{R}^2$

Geometric traits \quad Triangulation data structure

Delaunay triangulation
$\mathbb{R}^3$ projection

Geometric traits

Triangulation data structure

Delaunay triangulation
$\mathbb{R}^2$ other metric

Geometric traits  Triangulation data structure

Delaunay triangulation
Periodic triangulation

Geometric traits

Triangulation data structure

Delaunay triangulation
\approx 1 \mu s \text{ per point}
\[ \approx 8 \mu s \text{ per point} \]
One word on robustness issues
Basic incremental algorithm
Locate by walk
Locate using randomized data structures
Vertex removal in 2D
Remarks on CGAL programming

Conclusion
Delaunay challenges
Delaunay challenges

Practical vs worst case size of Delaunay 3D
Delaunay challenges

Practical vs worst case size of Delaunay 3D

Known results

\( \Theta(n^2) \) worst case

\( \Theta(n) \) random in ball

\( \Omega(n)O(n \log n) \) random on polyhedron

\( O(n \log n) \) good sample of smooth generic surface

\( \Theta(n \log n) \) random on cylinder
Delaunay challenges

Practical vs worst case size of Delaunay 3D

Known results

- \( \Theta(n^2) \) worst case
- \( \Theta(n) \) random in ball
- \( \Omega(n)O(n \log n) \) random on polyhedron
- \( O(n \log n) \) good sample of smooth generic surface
- \( \Theta(n \log n) \) random on cylinder

Find good models of practical data

(Smooth analysis)
Delaunay challenges

Practical vs worst case size of Delaunay 3D

Better algorithm for 3D deletion

10 \( \mu s \) to insert

100 \( \mu s \) to delete
Delaunay challenges

- Practical vs worst case size of Delaunay 3D
- Better algorithm for 3D deletion
- One billion points
  - Needs memory efficient algorithms
  - Cache effects are already important
Questions