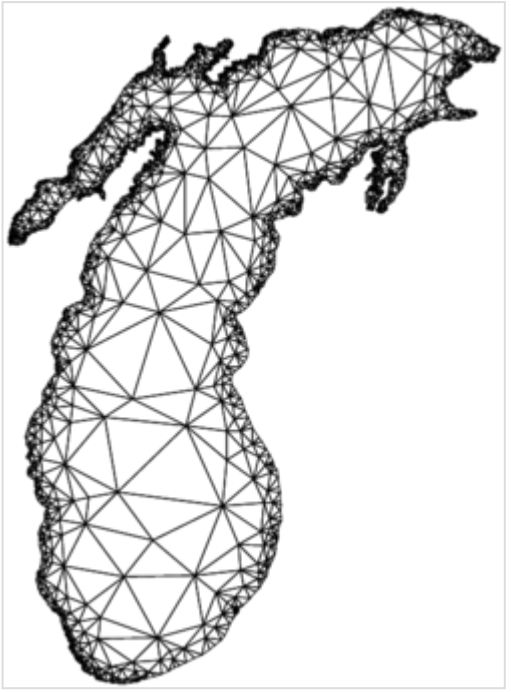
Delaunay refinement

In mesh generation, **Delaunay refinements** are algorithms for mesh generation based on the principle of adding Steiner points to the geometry of an input to be meshed, in a way that causes the Delaunay triangulation or constrained Delaunay triangulation of the augmented input to meet the quality requirements of the meshing application. Delaunay refinement methods include methods by Chew and by Ruppert.

Chew's second algorithm

Chew's second algorithm takes a piecewise linear system (PLS) and returns a constrained Delaunay triangulation of only quality triangles where quality is defined by the minimum angle in a triangle. Developed by L. Paul Chew for meshing surfaces embedded in three-dimensional space,[1] Chew's second algorithm has been adopted as a two-dimensional mesh generator due to practical advantages over Ruppert's algorithm in certain cases and is the default quality mesh generator implemented in the freely available Triangle package.[2] Chew's second algorithm is guaranteed to terminate and produce a local feature size-graded meshes with minimum angle up to about 28.6 degrees.[3]



Mesh of Lake Michigan using Chew's second algorithm implemented in the Triangle package.

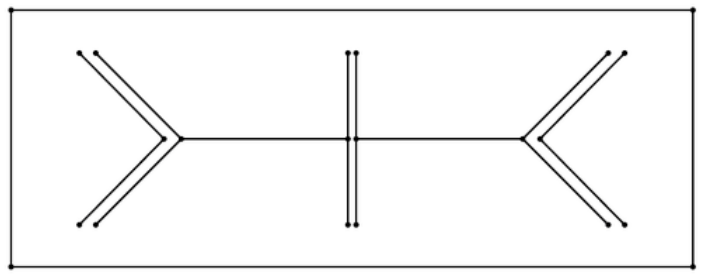
The algorithm begins with a constrained Delaunay triangulation of the input vertices. At each step, the circumcenter of a poor-quality triangle is inserted into the triangulation with one exception: If the circumcenter lies on the opposite side of an input segment as the poor quality triangle, the midpoint of the segment is inserted. Moreover, any previously inserted circumcenters inside the diametral ball of the original segment (before it is split) are removed from the triangulation. Circumcenter insertion is repeated until no poor-quality triangles exist.

Ruppert's algorithm

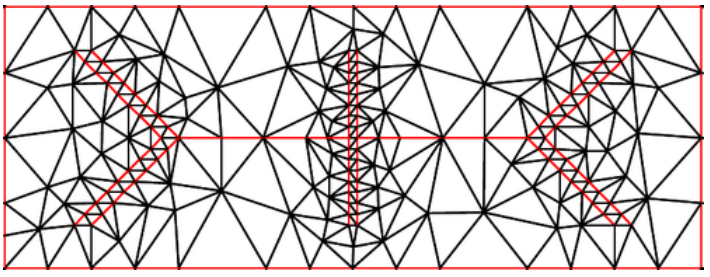
Ruppert's algorithm takes a planar straight-line graph (or in dimension higher than two a piecewise linear system) and returns a conforming Delaunay triangulation of only quality triangles. A triangle is considered poor-quality if it has a circumradius to shortest edge ratio larger than some prescribed threshold. Discovered by Jim Ruppert in the early 1990s,[4] "Ruppert's algorithm for two-dimensional quality mesh generation is perhaps the first theoretically guaranteed meshing algorithm to be truly satisfactory in practice."[5]

Motivation

When doing computer simulations such as computational fluid dynamics, one starts with a model such as a 2D outline of a wing section. The input to a 2D finite element method needs to be in the form of triangles that fill all space, and each triangle to be filled with one kind of material – in this example, either "air" or "wing". Long, skinny triangles cannot be simulated accurately. The simulation time is generally proportional to the number of triangles, and so one wants to minimize the number of triangles, while still using enough triangles to give reasonably accurate results – typically by using an unstructured grid. The computer uses Ruppert's algorithm (or some similar meshing algorithm) to convert the polygonal model into triangles suitable for the finite element method.



Input planar straight-line graph

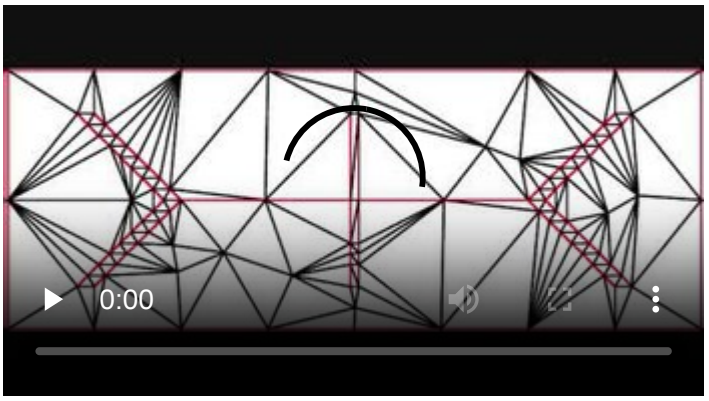


Output conforming Delaunay triangulation

Example of Ruppert's algorithm

Algorithm

The algorithm begins with a Delaunay triangulation of the input vertices and then consists of two main operations.



Intermediate triangulations of Ruppert's algorithm

- The midpoint of a segment with non-empty diametral circles is inserted into the triangulation.
- The circumcenter of a poor-quality triangle is inserted into the triangulation, unless this circumcenter lies in the diametral circle of some segment. In this case, the encroached segment is split instead.

These operations are repeated until no poor-quality triangles exist and all segments are not encroached.

Pseudocode

```
function Ruppert(points, segments, threshold) is
    T := DelaunayTriangulation(points)
    Q := the set of encroached segments and poor quality triangles

    while Q is not empty:                  // The main loop
        if Q contains a segment s:
            insert the midpoint of s into T
        else Q contains poor quality triangle t:
            if the circumcenter of t encroaches a segment s:
                add s to Q;
            else:
                insert the circumcenter of t into T
            end if
        end if
```

```
        update Q
    end while

    return T
end Ruppert.
```

Practical usage

Without modification Ruppert's algorithm is guaranteed to terminate and generate a quality mesh for non-acute input and any poor-quality threshold less than about 20.7 degrees. To relax these restrictions various small improvements have been made. By relaxing the quality requirement near small input angles, the algorithm can be extended to handle any straight-line input.[6] Curved input can also be meshed using similar techniques.[7] Ruppert's algorithm can be naturally extended to three dimensions, however its output guarantees are somewhat weaker due to the sliver type tetrahedron.

An extension of Ruppert's algorithm in two dimensions is implemented in the freely available Triangle package. Two variants of Ruppert's algorithm in this package are guaranteed to terminate for a poor-quality threshold of about 26.5 degrees.[8] In practice these algorithms are successful for poor-quality thresholds over 30 degrees. However, examples are known which cause the algorithm to fail with a threshold greater than 29.06 degrees.[9]

See also

- Local feature size
- Polygon mesh
- TetGen
- Voronoi diagram

References

1. Chew, L. Paul (1993). "Guaranteed-quality mesh generation for curved surfaces". *Proceedings of the Ninth Annual Symposium on Computational Geometry*. pp. 274–280.
2. Shewchuk, Jonathan (2002). "Delaunay refinement algorithms for triangular mesh generation" (https://doi.org/10.1016%2Fs0925-7721%2801%2900047-5). *Computational Geometry: Theory and Applications*. **22** (1–3): 21–74. doi:10.1016/s0925-7721(01)00047-5 (https://doi.org/10.1016%2Fs0925-7721%2801%2900047-5).
3. Rand, Alexander (2011). "Where and How Chew's Second Delaunay Refinement Algorithm Works" (http://2011.cccg.ca/PDFschedule/papers/paper91.pdf) (PDF). *Proceedings of the 23rd Canadian Conference on Computational Geometry*. pp. 157–162.
4. Ruppert, Jim (1995). "A Delaunay refinement algorithm for quality 2-dimensional mesh generation". *Journal of Algorithms*. **18** (3): 548–585. doi:10.1006/jagm.1995.1021 (https://doi.org/10.1006%2Fjagm.1995.1021).
5. Shewchuk, Jonathan (12 August 1996). "Ruppert's Delaunay Refinement Algorithm" (https://www.cs.cmu.edu/~quake/tripaper/triangle3.html). Retrieved 28 December 2018.
6. Miller, Gary; Pav, Steven; Walkington, Noel (2005). "When and why Delaunay refinement algorithms work". *International Journal of Computational Geometry and Applications*. **15** (1): 25–54. doi:10.1142/S0218195905001592 (https://doi.org/10.1142%2FS0218195905001592).

7. Pav, Steven; Walkington, Noel (2005). *Delaunay refinement by corner lopping*. Proceedings of the 14th International Meshing Roundtable. pp. 165–181.
8. Shewchuk, Jonathan (2002). "Delaunay refinement algorithms for triangular mesh generation" (https://doi.org/10.1016%2Fs0925-7721%2801%2900047-5). *Computational Geometry: Theory and Applications*. **22** (1–3): 21–74. doi:10.1016/s0925-7721(01)00047-5 (https://doi.org/10.1016%2Fs0925-7721%2801%2900047-5).
9. Rand, Alexander (2011). "Improved Examples of Non-Termination for Ruppert's Algorithm". arXiv:1103.3903 (https://arxiv.org/abs/1103.3903) [cs.CG (https://arxiv.org/archive/cs.CG)]..

Further reading

- Rineau, Laurent. "2D Conforming Triangulations and Meshes" (https://www.cgal.org/Manual/latest/doc_html/cgal_manual/Mesh_2/Chapter_main.html). Retrieved 28 December 2018.
- Shewchuk, Jonathan. "Triangle: A Two-Dimensional Quality Mesh Generator and Delaunay Triangulator" (https://www.cs.cmu.edu/~quake/triangle.html). Retrieved 28 December 2018.
- Si, Hang (2015). "TetGen: A Quality Tetrahedral Mesh Generator and a 3D Delaunay Triangulator" (https://web.archive.org/web/20181229075437/http://wias-berlin.de/software/index.jsp?id=TetGen&lang=1). Archived from the original on 29 December 2018. Retrieved 28 December 2018.