Mesh generation with FreeFem++ and Triangle

Alessandra Agostini
Chiara Piazzola

Scientific Computing
Master's Degree in Mathematics and Applications
Computer Science Department
University of Verona

July 22, 2014

Aim of our project

We want to build a mesh using Triangle, a software written specifically for triangulations. Moreover, we want to compare the mesh generated by Triangle with the one generated by FreeFem++ on the same domain and with a similar number of points.

Structure of our work:

- Part 1: how Triangle works
- Part 2: how to triangulate a given domain with Triangle and FreeFem++
- Part 3: comparison between meshes

Part 1
HOW TRIANGLE WORKS

Triangle

- is a Two-Dimensional Quality Mesh Generator and Delaunay Triangulator
- was created at Carnegie Mellon University (Pittsburgh, Pennsylvania) as part of the Quake project (tools for large-scale earthquake simulation) by Jonathan Shewchuk (now Professor in Computer Science at the University of California at Berkeley)
- can be downloaded from http://www.cs.cmu.edu/ quake/triangle.html
- won the 2003 James Hardy Wilkinson Prize in Numerical Software

Some definitions

Delaunay triangulation

Triangulation of the vertex set with the property that no vertex in the vertex set falls in the interior of the circumcircle (circle that passes through all three vertices) of any triangle in the triangulation.

Voronoi diagram

Subdivision of the plane into polygonal regions, where each region is the set of points in the plane that are closer to some input vertex than to any other input vertex. The Voronoi diagram is the geometric dual of the Delaunay triangulation.

Examples

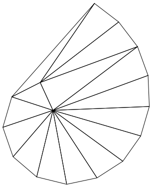


Figure: Delaunay triangulation

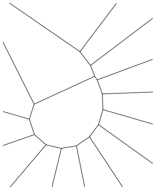


Figure: Voronoi diagram

Definitions

Planar Straight Line Graph (PSLG)

Collection of vertices and segments. Segments are edges whose endpoints are vertices in the PSLG, and whose presence in any mesh generated from the PSLG is enforced.

Constrained Delaunay triangulation of a PSLG

A triangulation where each PSLG segment is present as a single edge. A constrained Delaunay triangulation is not truly a Delaunay triangulation.

Definitions

Conforming Delaunay triangulation of a PSLG

A true Delaunay triangulation in which each PSLG segment may have been subdivided into several edges by the insertion of additional vertices, called Steiner points. Steiner points are also inserted to meet constraints on the minimum angle and maximum triangle area.

Constrained conforming Delaunay triangulation of a PSLG

A constrained Delaunay triangulation that includes Steiner points. It usually takes fewer vertices to make a good-quality CCDT than a good-quality CDT, because the triangles do not need to be Delaunay.

Examples

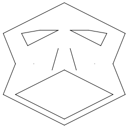


Figure: Face graph

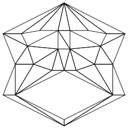


Figure: Constrained Delaunay triangulation

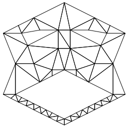


Figure: Conforming Delaunay triangulation

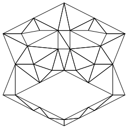


Figure: Constrained conforming Delaunay triangulation

Running Triangle

The default mesh is the Delaunay triangulation if the input is a set of vertices.
Otherwise, if the input is a planar straight line graph, the default is a constrained Delaunay triangulation.
To run Triangle we have to write

```
triangle [options] namefile
```

The output will be one or more files where we there will be stored informations about the new mesh (list of vertices, of edges, ...).
To visualize the mesh, we have to use the software Show Me:

```
showme namefile.1
```

File formats

- .node: a list of vertices

  First line:<# of vertices><dimension(must be 2)><# of attributes>
  <# of boundary markers (0 or 1)>
  Remaining lines:<vertex #><x><y>[attributes][boundary marker]

- .poly: a set of points where we can add informations about edges and possible holes in the domain

  First line: <#of vertices><dimension(must be 2)><#of attributes>
  <# of boundary markers (0 or 1)>
  Following lines: <vertex #><x><y>[attributes][boundary marker]
  One line: <# of segments> <# of boundary markers (0 or 1)>
  Following lines: <segment#><endpoint><endpoint>[boundary marker]
  One line: <# of holes>
  Following lines: <hole #><x><y>
  Optional line: <# of regional attributes and/or area constraints>
  Optional following lines:<region #><x><y><attribute><maximum area>

File formats

- .ele

```
First line:<# of triangles><nodes per triangle><# of attributes>
Following lines: <triangle #><node><node><node> ... [attributes]
```

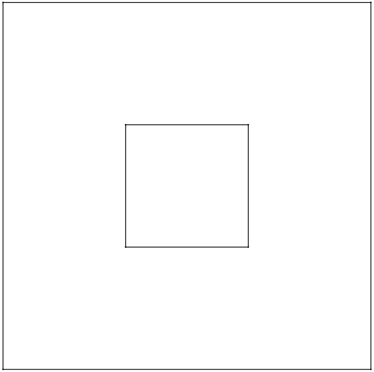
- .area
  to give each triangle a maximum area that is used for mesh refinement;
- .edge (-e switch)
  a list of edges of the triangulation;
- .neigh (-n switch)
  a list of triangles neighboring each triangle;

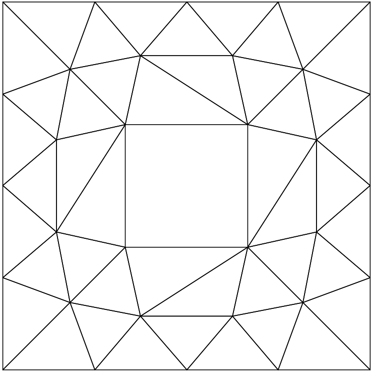
Example: box.poly

```
8 2 0 1 # A box with 8 points in 2D, no attributes, 1 boundary marker.
 1    0 0    0 # Outer box has these vertices
 2    0 3    0
 3    3 0    0
 4    3 3    0
 5    1 1    0 # Inner square has these vertices:
 6    1 2    0
 7    2 1    0
 8    2 2    0
8 1 # Eight segments with boundary markers.
 1    1 2    0
 2    2 4    0
 3    4 3    0
 4    3 1    0
 5    5 7    0     # These four segments enclose the hole.
 6    7 8    0
 7    8 6    0
 8    6 5    0
1 # One hole in the middle of the inner square.
 1    1.5 1.5
```

Example: box.poly

```
triangle -q30a0.3 box.poly
```





Options: angle constraint

We can force the minimun angle of the triangles to be greater than a specific degree using the option -q

```
triangle -q[] namefile
```

where -q may be followed by the value of the minimum angle. If no number is specified, the default degree is 20. Here is an example:

```
triangle -q30 box.poly
```

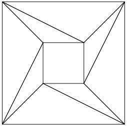


Figure: Default Delaunay triangulation

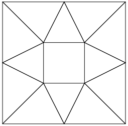


Figure: Triangulation with angle constraint

Options: area constraint

We can force also the maximum area of the triangles using the option -a.

```
triangle -a[] namefile
```

We recover the picture of the spiral. Here follows what we get with the line

```
triangle -a0.1 spiral
```

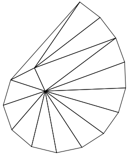


Figure: Default Delaunay triangulation

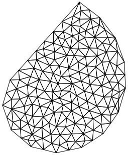


Figure: Triangulation with area constraint

Other usefull options

-c	Encloses the convex hull with segments;
-v	Outputs the Voronoi diagram associated with the triangulation;
-O	Suppresses holes: ignores the holes in the .poly file;
-V	Verbose: Gives detailed information about what Triangle is doing;
-e	Outputs a list of edges of the triangulation;
-n	Outputs a list of triangles neighboring each triangle;
-D	Conforming Delaunay: use this switch if you want all triangles in the mesh to be Delaunay, and not just constrained Delaunay; or if you want to ensure that all Voronoi vertices lie within the triangulation;
-r	Refines a previously generated mesh.

Part 2

HOW TO TRIANGULATE A GIVEN DOMAIN WITH TRIANGLE AND FREEFEM++

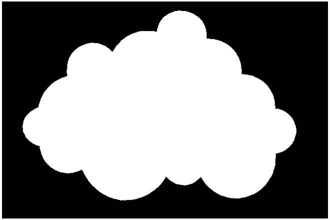
How to triangulate a given domain

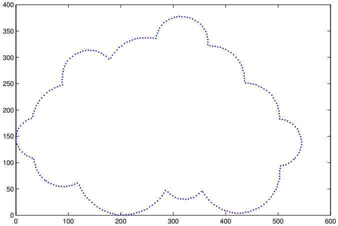
In order to compare the meshes generated by Triangle and FreeFrem++, we need the same initial set of points which defines the border of our domain. In following we show how to write input files compatible with Triangle and FreeFem++, given a database of boundary points.

How to get the boundary of an image

We try with a shape extracted from an image. We obtain the boundary of a black-white image using MatLab.

```
Bw = imread('cloud.jpg');
% traces the exterior boundary of objects
B = bwboundaries(BW);
b = B{1};
save('cloud.mat','b');
```





How to write a .poly file

```
load('cloud.mat');
fileID = fopen('cloud.poly', 'w');
% border points
fprintf(fileID, '# border points \n');
fprintf(fileID,'%i 2 0 1 \n',size(b,1));
for i=1:size(b,1)
    fprintf(fileID,'%i %i %i 2 \n',i, b(i,1),b(i,2));
end
% edges
fprintf(fileID, '# edges \n');
fprintf(fileID,'%i 1 \n',size(b,1));
for i=1:(size(b,1)-1)
    fprintf(fileID,'%i %i %i 2 \n',i,i,i+1);
end
fprintf(fileID,'%i %i %i 2 \n',size(b,1),size(b,1),1);
% holes
fprintf(fileID, '# holes \n');
fprintf(fileID,'0');
```

How to write a .edp file

We have to:

- open an output .edp file;
- write in a parametric way every single segment connecting each pair of border points;
- build the mesh on the given border.

In this way, we obtain a too thick triangulation and so we have to remesh:

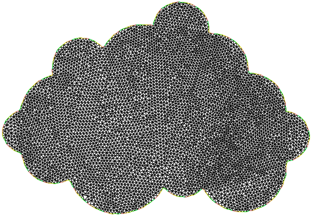
```
Th = adaptmesh (Th,iso=1);
```

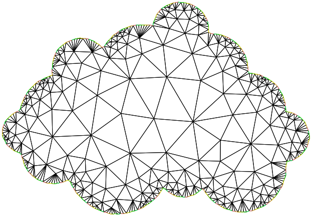
with the option $iso = 1$ we force the mesh to be isotropic. For our purpose it's important to notice that the mesh generated in this way has minimum corner angle of 10 degrees.

Cloud in Freefem++

`mesh Th = buildmesh (C);`

`Th = adaptmesh (Th,iso=1);`





$n = 283$ boundary points
nb of vertices $= 5479$
nb of triangles $= 10673$

$n = 283$ boundary points
nb of vertices $= 474$
nb of triangles $= 660$

Part 3
COMPARISON BETWEEN MESHES

Mesh comparison: Triangle vs FreeFem++

Our final goal is to understand if there's a way to decide which is the best mesh.

Definition

A family of triangulation $\mathcal{T}_h$ is said *regular* if there exist a constant $\sigma > 0$, independent of h, such that:

$$\frac{h_K}{\rho_K} \leq \sigma, \quad \forall K \in \mathcal{T}_h$$

where h_K is the diameter and ρ_K the radius of the inscribed circle of the triangle K.

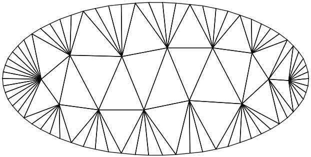
Mesh comparison: Triangle vs FreeFem++

So, the idea is to compare the triangulations computing the ratio $\delta = \frac{h_K}{\rho_K}$ for all the triangles in the meshes.
For all the triangles we extract the x and y component of its three vertices and we compute the longest edge and the radius of the inscribed circle.

First, we compare the meshes on a simple shape: an ellipse.
We generate 70 border points with MatLab:

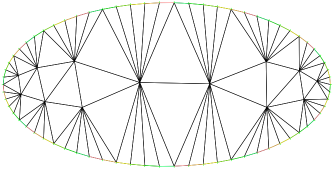
```
A = 2; B = 1; n = 70;
t = linspace(0,2*pi,n+1)';
b = [A*cos(t), B*sin(t)];
save('PuntiEllisse.mat', 'b');
```

Using Triangle we get a triangulation with 83 vertices and 94 triangles:



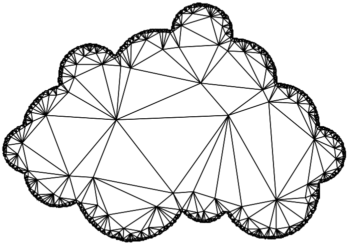
$\delta_{max} = 12.2378$
$\delta_{min} = 3.5781$
$mean(\delta) = 8.1088$

Using FreeFem++ we obtain a triangulation with 83 vertices and 94 triangles:

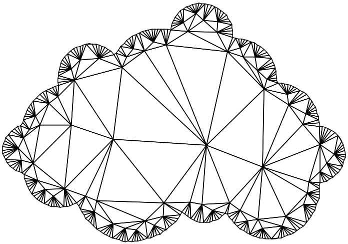


$\delta_{max} = 14.3814$
$\delta_{min} = 3.71904$
$mean(\delta) = 8.15213$

Mesh comparison: a cloud with Triangle

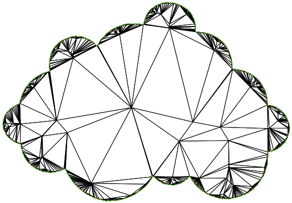


$n = 1415$ border points
nb of vertices $= 1810$
nb of triangles $= 2203$
$\delta_{max} = 19.8435$
$\delta_{min} = 3.4732$
$mean(\delta) = 7.8816$

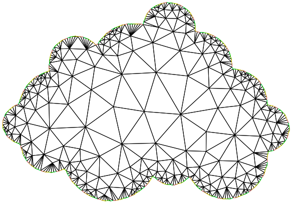


$n = 283$ border points
nb of vertices $= 354$
nb of triangles $= 423$
$\delta_{max} = 20.9908$
$\delta_{min} = 3.5613$
$mean(\delta) = 8.4976$

Mesh comparison: a cloud with FreeFem++



$n = 1415$ border points
nb of vertices $= 2802$
nb of triangles $= 4187$
$\delta_{max} = 340.042$
$\delta_{min} = 3.61803$
$mean(\delta) = 17.7454$



$n = 283$ border points
nb of vertices $= 474$
nb of triangles $= 660$
$\delta_{max} = 16.2344$
$\delta_{min} = 3.53984$
$mean(\delta) = 5.83016$

Mesh comparison: summary and conclusions

Now we resume the results obtained for the ellipse:

	Triangle	FreeFem++
Nb of vertices	83	83
Nb of triangles	94	94
δ_{max}	12.2378	14.3814
δ_{min}	3.5781	3.71904
$mean(\delta)$	8.1088	8.15213

and for the cloud:

	Triangle ($n = 283$)	FF++ ($n = 283$)	Triangle ($n = 1415$)	FF++ ($n = 1415$)
Nb of vertices	354	474	1810	2802
Nb of triangles	423	660	2203	4187
δ_{max}	20.9908	16.2344	19.8435	340.042
δ_{min}	3.5613	3.53984	3.4732	3.61803
$mean(\delta)$	8.4976	5.83016	7.8816	17.7454

Thank you for your attention!