WEST-EAST HIGH SPEED FLOW FIELD CONFERENCE
19-22, November 2007
Moscow, Russia

PARALLEL MESH GENERATION

Sergey Polyakov*, Igor Popov*, Igor Sedykh*

** Institute for Mathematical Modeling, Russian Academy of Science, (IMM RAS)*
4-A Miusskaya Square, 125047 Moscow, Russia
Email:isedykh@gmail.com, web page: http://www.imamod.ru

Key words: Parallel algorithm, mesh generation, Delaunay triangulation.

Abstract. In this work we consider a problem of constructing an effective parallel algorithm for constructing 2D-meshes in areas with piecewise linear boundary. Major applications of this algorithm could be found in problems of mathematical physics with real geometry. Mesh must satisfy Delaunay criterion, should be able to do refinement and movement of nodes.

1. INTRODUCTION

Many problems of mathematical physics deal with complex geometry of calculation areas. This means, that we construct effective 2D-meshes (and 3D-meshes) in such areas. To do so, we build a boundary for this area (convex hull, for example). After that, we will build a rough triangulation that could be modified afterwards. There is a number of methods that can improve our mesh: Delaunay criterion, nodes movement, refinement. Usually these approaches are combined. There are many algorithms for Delaunay criterion, such as known front method, chain algorithm, Dirichlet cell complexes etc. It appears that these methods are not so good for problems with real geometry, for the huge meshes and complex areas. Also, thay are not so good for constructing their parallel modifications, especially for the refinement algorithm, because it is hard to effectively combine parts of the entire grid from different processes.

In this work we will consider two algorithms: the first for building rough “starting” triangulation in an area with piecewise linear boundary and the second for design Delaunay triangulation and its parallel modification. Also, we will show examples for both algorithms.

2. SIMPLE ALGORITHM DESCRIPTION

Simple algorithm for constructing Delaunay triangulation starts with rouh triangulation and than, using some basic modifications, makes it a Delaunay triangulation. We will call this rouh triangulation a *starting* one. First, we will propose an algorithm for this mesh.

Let we have a non-convex area with a piecewise linear boundary without self-intersections. Note, that case when a boundary has self-intersections is not special, because we consider such area as a complex of sub-domains. The algorithm:

- we count oriented squares, obtained by adjacent edges of the boundary;
- list of squares need to be sorted in descending order, using the quickest sorting method;
- negative areas correspond to pieces, where domain is non-convex, so we do not consider them, as well as zero areas – we won’t be able to construct a triangle using them;

- we take minimal square in the list and build a triangle. This triangle will be the first in resulting triangulation;
- After we have "cut" corresponding vertex and two edges from the boundary, we'll have to replace them with a new edge from constructed triangle. After that, we recount only changed squares;
- iterations continue, as we construct new triangles.

It is important, that we can not cut an edge, if the other points of boundary occur in the resulting triangle (letter "M", for example). Our algorithm is not stopped, because of this remark. Indeed on each iteration, there always will be vertex that can be cut otherwise the boundary will not be closed.

Thus, we have constructed starting triangulation. It is time to move on and construct Delaunay triangulation, make refinement of the mesh etc.

By definition, triangle that has adjacent edge to another is called a *neighbor* for it. We describe a simple algorithm for Delaunay triangulation. We will use the following version of this criterion – triangle satisfy Delaunay criterion if there are no opposite vertexes of his neighbors in circumscribed circle to a given triangle. If triangle doesn't satisfy Delaunay criterion, than we will have to *flip* adjacent edge to another one (see fig. 1). This procedure is very simple – we delete edge *a* from these triangles and draw an edge *b*. Of course, we will have to rebuild their neighbor numbers.

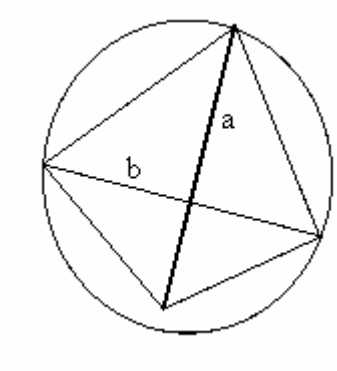


Figure 1. Flip procedure.

Finally, let's describe algorithm:

- we check if first triangle in triangulation satisfy Delonay criterion;
- we flip edges, if necessary;
- iterations continue, on the other triangles in the list;
- algorithm stops when all triangles have being rebuild.

This is important for the practical realization of the algorithm to take a special constant ε, for calculations, if a vertex lies inside of a circle. This is vital for the algorithm to be finite. Also, if a vertex lies on a circle, we do not flip corresponding vertex.

3. PARALLEL ALGORITHM

Parallel algorithm will be based on a following statement, taken without proof.

Statement. *Procedure of adding a point to the Delonay triangulation implies only local rebuild of triangles around this point.*

Note, that in this statement term “local” means that number of triangles rebuild after addition of a point constrained by a constant. This constant depends only on area, and on how fine is the mesh. If the mesh becomes finer, than this constant will be smaller.

Consider an area, where we will apply our algorithm:

- we need to split area into sub areas, so we split the boundary of it. Main idea is to split area so, that each sub area have not more, than two neighbors – on the “top” and on the “bottom”;
- each process builds a starting triangulation and makes it a Delaunay triangulation in corresponding sub area. If it is needed, we make refinement of the mesh, up to the moment, when all radiuses of the circumscribed circles will be smaller than $R_{\max}$;
- each process builds an *interface lines*, that consist of vertexes of triangles, that regard to the boundary, where we have cut our area. We take only vertexes that do not lie on the boundary itself. We do not consider in the following interface triangles;
- areas among interface lines of two sub areas need to be triangulated to make a single mesh in the whole area. We make this processes also parallel;
- each interface area need to be rebuild and refinement should be made. This process goes fast, because of the statement, so we have real benefit in time;
- sub-areas are connected.

It finishes our algorithm. We do not have to rebuild all triangles on the final steps of algorithm, because of the **Statement**. If we have to construct a very fine mesh, than the process of rejoining sub-domains take less time, because of smaller constant, mentioned above. So our algorithm is faster on fine meshes.

We have our algorithm that can build fine meshes in non-convex areas and can be used for solving practical problems of mathematical physics. We can make very fine meshes that can allow obtain more accurate results.

4. EXAMPLES

There is number of results for our algorithm. They are obtained by a program that realizes described algorithms, using C++.

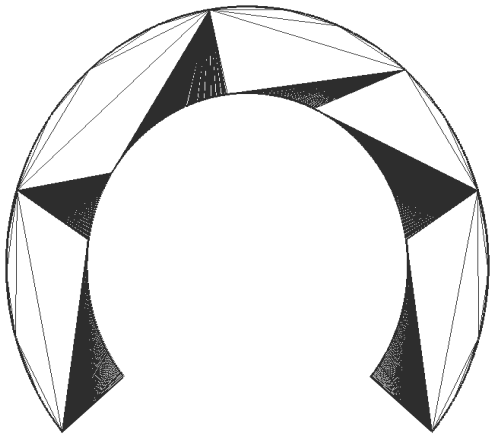


Figure 2. Rough triangulation

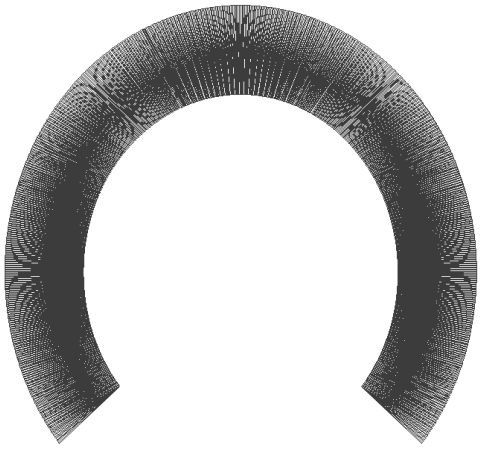


Figure 3. Delaunay triangulation

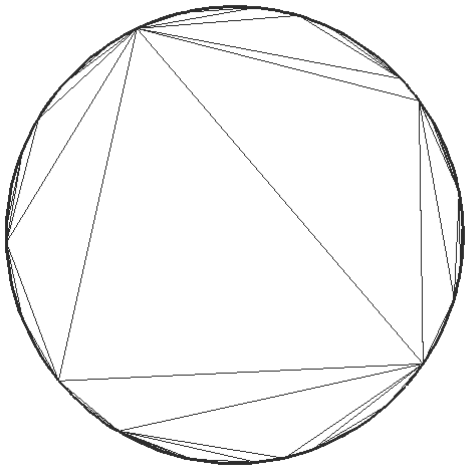


Figure 4. Delaunay triangulation on a model of a ciracle – 200 points on boundary

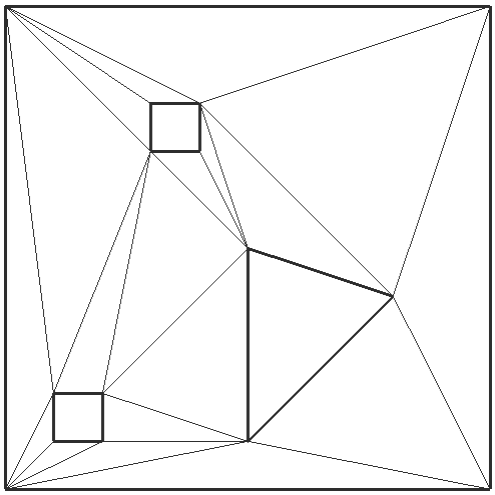


Figure 5. Delaunay triangulation in an area with "holoes"

5. CONCLUSION

Further development of these algorithms is: speed up of mesh generation, increasing efficiency of methods, modernization of algorithm for building starting triangulation, refinement algorithm. These methods will be adapted to 3D geometry in complex areas with refinement and parallel computations.

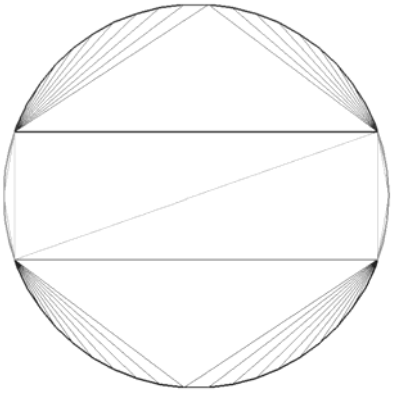
6. ACKNOWLEDGMENTS

This work was supported by Russian Foundation for Basic Research (grants No. 06-01-00233, 05-07-90230).

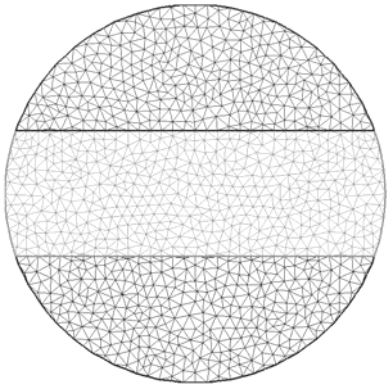
7. REFERENCES

1. F. Preparata and M. Sheimos, Calculational Geometry. [in Russian], Moscow, 1989.
2. Skvortsov A.V. Review of Delonay triangulation contruction algorithms. [in Russian], Numerical methods and programming. 2002, **3**, 14-39.
3. Solov'ev A.V., Slov'eva E. V., Tishkin I. F. and others. About one algorithm for constructing Dirichlet cell complexes. [in Russian], Moscow, 1985. preprint of IAM RAS USS, 68.
4. Franklin & Marshall College, Delaunay Triangulation: incremental algorithm. Internet materials from http://www.fandm.edu.
5. Guy E. Blelloch, Gary L. Miller, Dafna Talmor. Developing a practical projection-based parallel Delaunay algorithm. Proceedings of the twelfth annual symposium on Computational geometry, 1996, pp. 186 – 195.
6. D.T. Lee and B.J. Schachter. Two Algorithms for Constructing a Delonay Triangulation. International Journal of Computer and Information Sciences, 1980, 9(3), pp. 219-242.

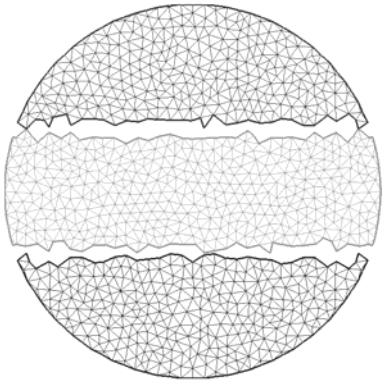
1)



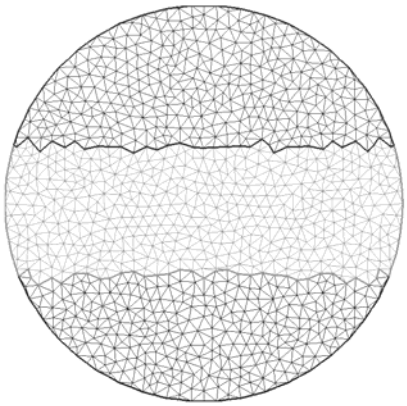
2)



3)



4)



5)

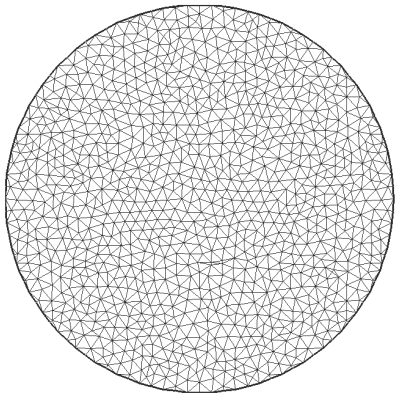


Figure 6. Parallel algorithm steps: