Unstructured Grid Generation in 2-D and 3-D

S. Fatemeh Razavi, Jeff Boisvert, Juliana Leung

Numerical modeling is used to predict flow behavior and simulate transport phenomena. One of the origins of complexity in a geological media is the existence of a large number of fractures. An efficient discritization of the fracture media is required. This Discritization or mesh generation requires significant preprocessing effort. The objective of this paper is meshing these complex fracture structures in 2D and 3D essential for future simulation purposes. A program was developed to generate unstructured triangular mesh for a randomly generated discrete fracture network in 2-D and TetGen program was used to generate mesh for a 3-D discrete fracture network. On 3-D grids, geostatistical modeling is done to populate the grids with permeability values.

1. Introduction

There are two approaches to model flow and mass transport in fractured porous media, dual continuum model proposed by warren & Root in 1963 and discrete fracture model (DFM). DFM is a reasonable method to model laboratory scale fractured porous media (Ito, 2003).

Considering reservoir heterogeneities mainly included discrete fracture networks is the main focus of this research. The goal is simulation on the unstructured grids, which has been populated with permeability and porosity values by Geostatistical methods, to obtain effective permeability tensor that can be used for flow simulations at a coarse scale. This is important because current computational capabilities do not permit flow simulation on a field-scale DFM model containing hundreds or thousands of fractures. Flow is considered along the fractures and that's why creating grids aligned fractures is of importance.

Unstructured grids are suitable choice to grid design that have the required flexibility to handle domains with complicated geometries and boundaries. One of the remarkable advantages of unstructured grids is possibility of controlling the grid resolution through refinement. Taking into account, the geometrical simplicity and the accessible procedures for the numerical simulation of physical problems (Liseikin, 2010), unstructured grids are generally composed of triangles in two dimensions and tetrahedrons in three dimensions. Basically, three methods exist to construct unstructured grids including Octree methods, Delaunay procedures and Advancing Front techniques (Liseikin, 2010).

Delaunay tessellation is a more common geometric decomposition tool to generate triangular and tetrahederal grids that has found wide use in engineering applications. Two and three-dimensional Delaunay tessellations are used for the analysis of porous media, and the modeling of flow in porous media (Thompson, 2002). Liseikin's book is a good reference to get information about the two other methods for unstructured mesh generation.

The present work is chiefly included obtaining a reasonable Delaunay mesh for a flexible discrete fracture network in 2D and 3D. There are different open access programs to mesh complex geological media. Some of them are listed and explained in brief in the following.

DistMesh is a simple MATLAB code for generation of unstructured triangular and tetrahedral meshes uses Delaunay triangulation and tetrahedralization. It was developed by the Department of Mathematics at MIT. A detailed description of the program is provided in Per-Olof Persson and Gilbert Strang, 2004 (DistMesh weblog). This code could be modified based on needs.

G23FM is another grid generator for 2D and 3D fractured media which generates unstructured grids automatically. It was introduced as a flexible tool including different options like adding points, controlling triangles sizes, refinement and scaling (Mustapha, 2010) and developed by a computer science specialist at McGill University.

FracMesh is a 2-D and 3-D structured mesh generator developed by Earth Sciences Division at Berkley. The most complicated part of fracture network gridding is creating the connectivity list. FracMesh produces the connectivity lists necessary for the input file of the multiphase flow numerical simulator TOUGH2 (Ito, 2003).

TetGen and Triangle are two well-tested mesh generators for 3D tetrahedral and 2D triangular meshing. TetGen was developed by Numerical Mathematics and Scientific Computing group at WIAS which could generate conforming Delaunay tetrahedra and apply volume control. Triangle was developed by Computer Science Division at Berkley (Triangle weblog: http://www.cs.cmu.edu/~quake/triangle.html).

MeshPy combines TetGen with Triangle into one package, which is very practical and was developed at the Institute of Mathematical Sciences at New York University (http://mathema.tician.de/software/meshpy).

Because of no availability of G23FM software when requested, considering this point that FractMesh is for structured grid generation while we are interested in unstructured mesh generation and mainly TetGen good quality manual, we selected TetGen for mesh generation in 3-D.

This paper is organized as follows; in the second part, Delaunay criterion will be explained. In the third part, unstructured mesh generation for 2D DFM will be described in detail. In fourth part, mesh generation for a 3D DFM is discussed and in the fifth part, 3-D grids are populated with permeability.

2. Delaunay Triangulation and Tetrahedralization

Based on Delaunay triangulation, neighboring points are connected to form triangular or tetrahederal cells in such a way that the circumcircle through the three vertices of a triangular cell in two-dimension and the circumsphere through the four vertices of a tetrahedral cell in three-dimension do not contain any other point. This condition is called Delaunay criterion (see Figure 1). In fact by applying Delaunay criterion in 2D, the minimum angle of all the angles of the triangles in the triangulation will be maximized and the maximum circumradius will be minimized thus they tend to avoid skinny triangles which will result in optimal triangulation in 2D and more refinement will be unnecessary (Liseikin, 2010). Note that, the Delaunay Triangulation is not unique. Optimal properties of Delaunay triangulation are not true in 3D, since a measurement for optimality in 3D is not agreed on. Slivers are special type of badly-shaped tetrahedrons which are flat and nearly degenerate and they will cause instability and inconsistency. In 3D, slivers could survive even after Delaunay refinement. To measure quality of tetrahedrons in 3D, two criterion is considered: 1) aspect ratio of an element which should be as small as possible and defined as ratio of the maximum side length to the minimum altitude, 2) radius edge ratio, which is the ratio of the radius of the circumsphere of the tetrahedron to the length of the shortest edge. This value should be small for a well-shaped tetrahederon.

3. Unstructured Mesh Generation in 2-D

In the first part, two cases are considered, including fracture with thickness and fracture without thickness. Subsequently, the focus of the second part is on the generation of a flexible fractured model in 2D by finding suitable points distribution on fractures to populate them as straight lines and moreover finding suitable background point distribution to show the matrix. Then, the resulted points distribution follows by automatic Delaunay triangulation done by Matlab DelaunayTri uses CGAL, the Computational Geometry Algorithms Library. To display the triangles defined by DelaunayTri, Matlab triplot function is used

3.1. Part I: Considering a Single Fracture

Fractures are lines including two endpoints in 2D and planes including four endpoints in 3D. Investigations started in two-dimension. Points were distributed uniformly in two steps, in background and on fractures for two different cases: 1) Fractures with thickness and 2) Fractures without thickness.

The following considerations play significant roles in accurate point distribution:

1) Removing background points close to the distributed points on fractures is essential.
2) In case of fracture with thickness, one should consider three parallel lines to represent the fracture and its thickness.
3) Boundary treatment is inevitable to calculate the accurate starting points for point distributions started from boundaries.

3.1.1. Boundary Treatment

Goal is obtaining equilateral triangles around fracture. For starting a correct distribution of points from boundaries, a geometric problem have to be solved. Consider left side of three parallel lines representative of one fracture with thickness (see Figure 2). Firstly, points on the middle line is to be distributed (points a, d …) and based on the resulted distribution, the exact place of starting point on upper and lower lines (around middle line) is calculated. X1 and X2 are target unknowns in Figure 2.

$$\text{in } \Delta \text{ abc: } ab = X1 \text{ , } ab = ac * \cos(<cao + <dak)$$

$$< dak = \arctan(\text{fracture slope})$$

$$\text{in } \Delta \text{ aco : } ac = \text{sqrt}(ht \wedge 2 + ao \wedge 2) = \text{sqrt}(ht \wedge 2 + (\text{spacing} / 2) \wedge 2)$$

$$\langle cao = \arctan(ht / ao)$$

$$\text{in } \Delta \text{ anm: } nm = X3,\ nm = am * \cos(\langle dak - \langle oam) = ac * \cos(\langle dak - \langle cao)$$

$$X3 - X2 = em * \cos(\langle dak) = ao * \cos(\langle dak)$$

In Figure 2, ht is equal to half of the fracture thickness and “cm” is the perpendicular bisector of “ad”. By calculating the angles <dak, <cao and the length ac, X1 and X2 is obtained which are starting points on upper line and lower line respectively.

After applying uniform point distribution, boundary treatment and removing close background points around fracture. Mesh is generated and visualized by using Matlab “DelaunayTri” and “triplot” commands respectively (see Figure 3). It is important to note that figure 3 can be easily representative of fracture without thickness if background mesh and fracture thickness mesh have relatively close size. While for fractures with thickness, there is a transition (buffer) zone around fracture with medium grid size showing the transition of fine grid size around fractures to background coarse mesh size. To represent the transition zone, around three main lines (in the middle) two more lines were populated by points considering suitable point distances q times bigger than the half of the main fracture thickness (ht in Figure 2). “q” is named scaling factor (see Figure 4).

Generally, considering fracture thickness at point distribution stage is not necessary because fracture grids can be added based on the technique proposed by Karimi Fard (SPE 79699) to consider flow along the fractures so we are ignoring the thickness in the next section.

3.2.Part II: 2D Discrete Fractured Network Modeling

Main goal in this section is to generate a flexible discrete fracture network without considering the fractures thickness and more importantly populating them with suitable point distribution to have acceptable mesh. Most of the required programs for this part had been written in part I and just some generalizations were necessary.

The same as previous part, distribution of background points and fractures population should be done separately (each of them in a separate function). Before fractures population, considering several fractures in fractured model, the intersections of n fractures are calculated by using the endpoints of fractures (see Figure 5 as a sample) slope and interception for each fracture. Subsequently, points are distributed on each fracture between endpoints and intersections and based on a feasible distance entered by user as “d” (see Figure 6). “d” which is the distance between distributed points have to be corrected on each segment of a fracture, for instance, fracture in Figure 5, includes 5 segments). This correction is done by defining a correction factor. In Figure 7, “L” is the length of each segment on one fracture. The endpoints of the solid line in Figure 7 are the endpoints of on fracture. The middle white point is an intersection and red point is the first distributed point on the RHS segment. Consider that this fracture has two segments.

$$\text{Correction Factor (CF)} = L / (d * \text{round}(L / d))$$

Note that “d” is the same for all fractures and the projection of d multiplied by CF on X axis should be used for point distribution. Figure 8 is shown both point distribution and mesh results before and after applying this correction. When one intersection coincided with an endpoint (Figure 9), fracture population stops. This problem is solved by using a separate function applied to the matrix which lists all points on the related fracture. In fact, in the point lists matrix including the endpoints, intersections and distributed points, repeated points are removed.

By applying all the mentioned modifications, the following fractured model was created as a sample (Figure 10). Fractures are generated stochastically and randomly. Point distribution on a fracture, which is orthogonal to x axis, needs different attention and it is done in a separate function.

In the next step, the close background points should be removed. This is done by a flexible criterion which has dependency on “d”. Background points after removing the close ones is shown in Figure 11.

Figure 13 is showing mesh result for combination of fractures and background points (shown in Figure 12) generated by Delaunay tessellation. Mesh is generated by Matlab DelaunayTri and visualized by Matlab “triplot”.

As a quick note, the input data for running the program are the number of fractures, endpoints of fractures (planes could be generated randomly as well), feasible distance between distributed points on fractures

(d), Length of the square area and the number of background points in x and y directions (nx and ny), which are regularly the same. Finding a general relation between nx (ny) and d and additionally a relation between the criterion for removing the BG points and d is recommended as future work to improve the results.

4. Unstructured Mesh Generation in 3-D

4.1. Developing Codes

In 3D, the same procedure could be applied to generate random 3D fracture planes, populating them with suitable uniform point distribution, distribute points on 3D background and removing the BG points close to the fracture planes based on a suitable criterion. A single randomly generated fracture plane populated uniformly with points, a sample generated random DFM with 30 fractures and distributed points on 3D background can be seen in Figures 14, 15 and 16. For now, TetGen will be used to generate mesh for a 3D DFM.

4.2. Unstructured 3D Mesh Generation by TetGen

TetGen produces tetrahedral meshes suitable for numerical simulation which are using finite volume or finite element methods. Terahedralization is done based on Delaunay criterion. In TetGen, constrained Delaunay Triangulation could be applied which decomposes a 3D domain into a tetrahedral mesh, such that the boundary is presented in the faces of the mesh.

The quality of tetrahedrons is measured by two criteria, aspect ratio of the element and radius edge ratio. It is important to note that while some of the degenerated tetrahedra, slivers, can be removed by TetGen using local flip technique; still slivers could survive even after Delaunay refinement. In the following, you will see some quick notes for using TetGen: As an input, TetGen uses a general input called piecewise linear complex (PLC). A PLC is including a set of vertices, a set of segments and facets. See Figure 18. The pink area shows a facet which is non-convex, has a hole, segments and vertices in it, (Si, 2006). Then by using the suitable command executable in Cygwin, mesh will be generated. A simple TetGen command is as follows:

Program	Switch/Switches	Input
./tetgen.exe	–p	*.poly

Both input file and output files could be visualized. To visualize the input and output of TetGen, TetView program is normally used. Input file is a "*.poly" file including the information of the facets, segments and points of the PLC. A typical output will include a list of mesh nodes (*.node), a list of tetrahedral (*.ele) and a list of boundary faces/convex hull faces (*.face). It is important to note that convex hull is represented by a list of triangles as well. To generate mesh for our case which is a DFM, fracture planes should be created randomly in the first step and then intersections of the fracture planes have to be found.

4.2.1. Generating Fracture Planes

To define a DFM in TetGen, each fracture plane is defined as a facet. The vertices used to define a facet of a PLC have to be precisely coplanar, but it is hard to achieve (Si, 2006). So, TetGen accepts the endpoints of a plane which are approximately coplanar and this approximation is defined by a tolerance. The suitable endpoints of a fracture plane can be achieved based on the following criterion for coplanar test:

$$V / L^3 < \text{Tolerance}$$

Where V is the Volume of the tetrahedron abcd created by four endpoints a, b, c and d and L is the average edge length of tetrahedron abcd. Default tolerance value in TetGen is 1e-8. One can set a tolerance for coplanar test by "-T" switch. It is appealing that by applying "-M" switch, TetGen does not merge coplanar fracture planes (facets).

4.2.2. Detecting the Intersections Between Fracture Planes

There is a very useful possibility in TetGen which helps to detect intersections of the PLC facets. For a DFM including numerous fracture planes, we need to know the intersection of each two fracture planes to add that intersection segment as a polygon when defining each fracture plane for TetGen. Finding the intersections is done

by creating a "*.node" file including the endpoints of the generated DFM and running TetGen program with "-d" switch on the "*.node" file as an input.

Program	Switch	Input
./tetgen.exe	–d	*.node

4.2.3. Creating TetGen Input File

After generating the fracture planes and finding the intersections, "*.poly" file should be created. It is including four parts and the information of all the nodes, segments and facets. The intersection nodes will be added to the first part of the file which is node list and the intersecting segments will be added in the second part which is a list of facets. Facet list should be created very carefully. See two invalid facets in the Figure 17. The facets are invalid because one segment (which is intersecting line) including two intersection points is missing in both planes. Each of the planes should be defined as two polygons not a single polygon. The first polygon is including the four endpoints and the 2nd one is the segment including two intersecting points.

A typical input file for a sample DFM with 7 fractures can be seen in Figure 19. Hole list and region list which are parts 3 and 4 of the input file, are empty for this example. TetGen visualized input file created by Tetview is shown in Figure 20. The right hand side image is a typical input DFM of FracMesh. The curved-shaped fracture plane (the green one in the RHS image) was created by combination of small pieces of rectangular fracture planes with different normal vectors and corner points (Si, 2006).

4.2.4. Getting Output and Visualization

In the following, is a sample command executed in Cygwin to generate tetrahedral mesh.

Program	Switch/Switches	Input
./tetgen.exe	–pq1.414a0.5f	*.poly

It is really important to note that to have a list of all facies "-f" switch should be used. By "-p" command mesh is generated and "-q" imposes the quality constraint and adds more points rather than using "-p" only. Radius edge ratio should be written after "-q" switch and default value for this ratio is considered 2 by TetGen. Maximum volume constraint is applied by entering a number after "-a" switch. In the above command, by applying a0.5, we constrain the maximum volume of the tetrahedrons to be less than 0.5 cubic meters. It is possible to reconstruct and refine previously generated mesh by "-r" and adding a list of additional points by applying "-i" switch. To visualize the input and output files, the following sample commands could be used:

	Program	Input
Input visualization:	./tetview-win	*.poly
Output visualization:	./tetview-win	*.1

.1 is including all output information (nodes, edges, faces) for ".poly" input file. TetView-win is a version of TetView compatible with windows operation system. TetGen typical outputs including a list of nodes, faces and tetrahedrons are shown in Figure 21. A sample visualized output for a DFM including 7 fractures can be seen in Figure 22. Fracture planes are recognizable by marking them using boundary marker option (Si, 2006). It is possible to visualize the output of TetGen in Medit or Gid as well. To see TetGen results under Medit "-g" and to see the results under Gid, "-G" switches should be used. Quality report of generated mesh is obtainable by "-V" switch or using the following command. A sample mesh quality statistics can be seen in Figure 23.

Program	Switches
. / tetgen.exe	–rNEFV

5. Permeability Distribution of Elements

Geostatistical modeling of petrophysical properties is done by using a GSLib program named "psgsim" (Manchuk, 2010). PSGSIM is to perform sequential Gaussian simulation of continuous variables and sequential indicator simulation of categorical properties on irregular sets of points. These sets of points are the location of the unknowns which are petrophysical properties. These locations in the triangular/tetrahedral control volumes in 2D/3D could be:

A) At the center of the unique circumcircle/circumsphere of each triangle/tetrahedron or,
B) At the centroid, named also geometric center, center of mass, center of gravity or barycenter of each triangle/tetrahedron.

For our case, grids' barycenter have been calculated and then populated with permeability values (See Figure 24).

6. Conclusions

Unstructured meshing was done for randomly generated 2D DFM. Codes are flexible and details have been presented in the first part of the paper. By comparison with similar works done previously using G23FM (Mustapha, 2010) and DistMesh (Bajaj, 2009), our results are remarkably practical while still some modifications could be done to improve the results. Unstructured mesh generation for a 3D DFM is done by applying TetGen program. A sample DFM model with seven fractures meshed by TetGen and resulted tetrahedral grids were populated with permeability values using geostatistical modeling.

References

Bajaj, R., 2009, "*An Unstructured Finite Volume Simulator For Multiphase Flow Through Fractured-Porous Media*", MSc Thesis, MIT University,

Ito K. and Seol Y., 2003, "*A 3-Dimensional Discrete Fracture Network Generator to Examine Fracture-Matrix Interaction Using TOUGH2*", Berkeley, California,

Liseikin, V.D., 2010, "*Grid Generation Methods*", Book, Springer, Second Edition,

Manchuk, J., 2010, "*Geostatistical Modeling of Unstructured Grids for Flow Simulation*", PhD thesis, University of Alberta,

Mustapha, H., 2010, "*G23FM: a tool for meshing complex geological media*", Springer,

Persson P. and Strang G., 2004, "*A Simple Mesh Generator in Matlab*", http://persson.berkeley.edu/distmesh/,

Si, H., 2006, "*A Quality Tetrahedral Mesh Generator and Three-Dimensional Delaunay Triangulator*", http://tetgen.berlios.de/,

Thompson, K.E., 2002, "*Fast and robust Delaunay tessellation in periodic domains*", Wiley Publication.

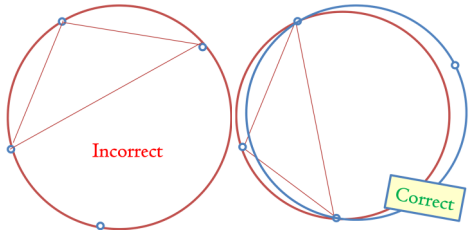


Figure 1: Delaunay criterion in 2D

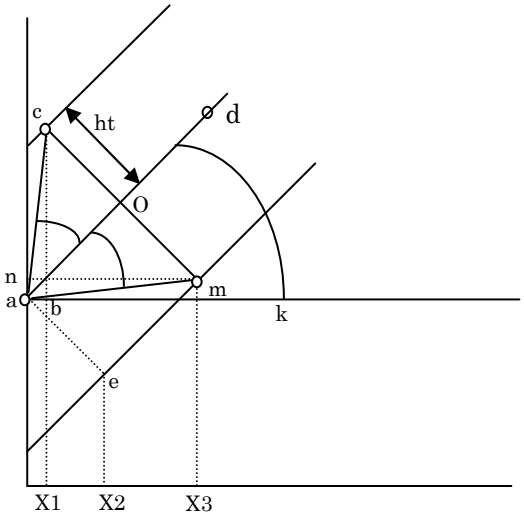


Figure 2: Finding the start points X1 and X2

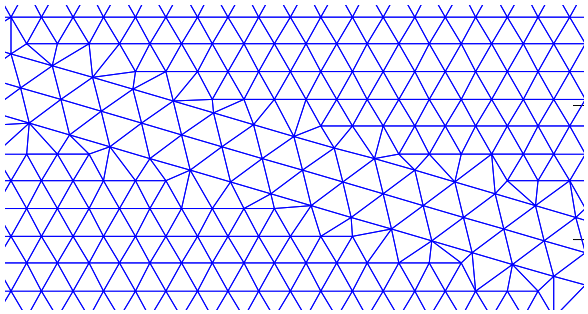


Figure 3: Mesh for fracture thickness exaggerated in size for better visualization

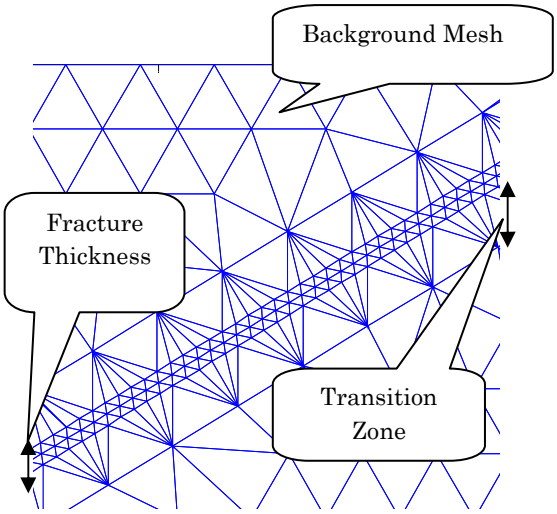


Figure 4: Transition zone for one fracture with thickness (q = 5)

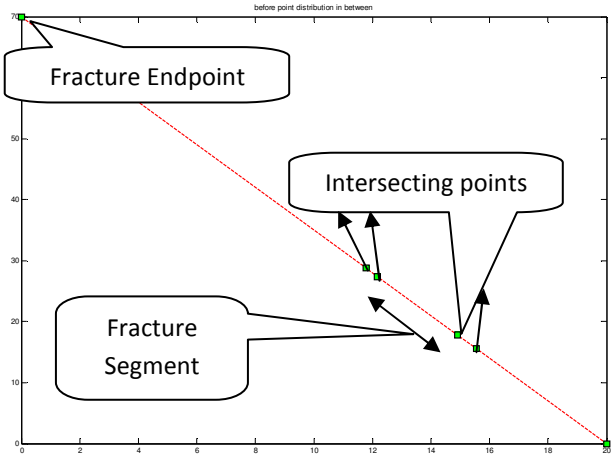


Figure 5: Intersections and endpoints on one sample fracture

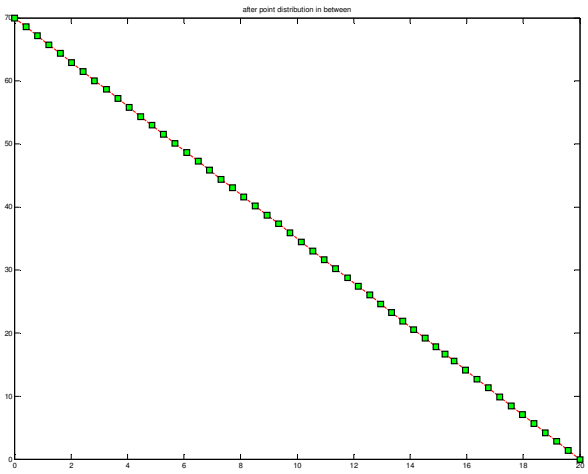


Figure 6: Population of fracture in figure 4

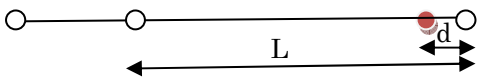
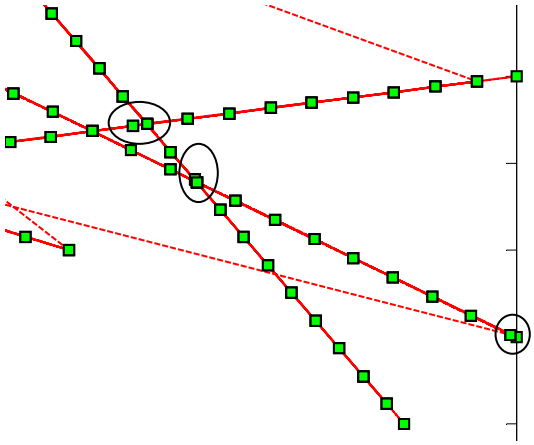
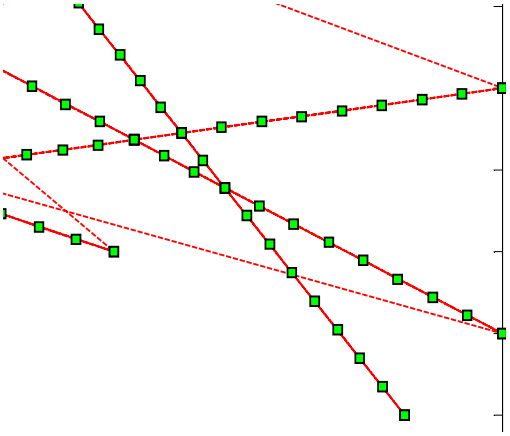


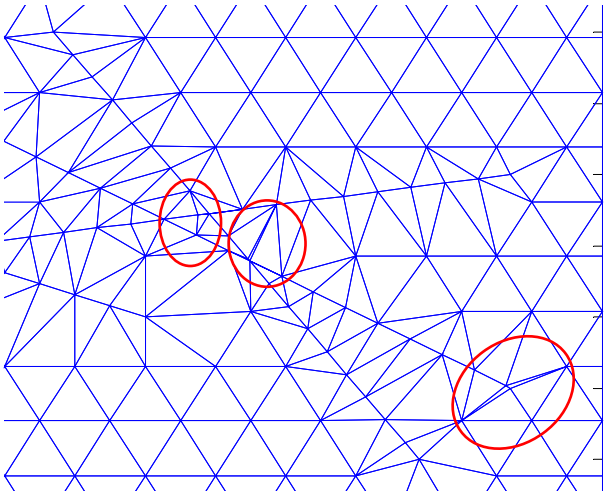
Figure 7



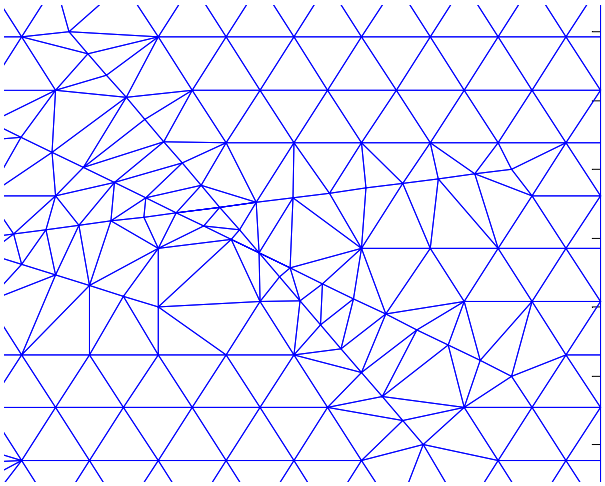
a) Before applying CF



b) After applying CF



a) Before applying CF



b) After applying CF

Figure 8: Removing degenerated triangles by applying correction factor

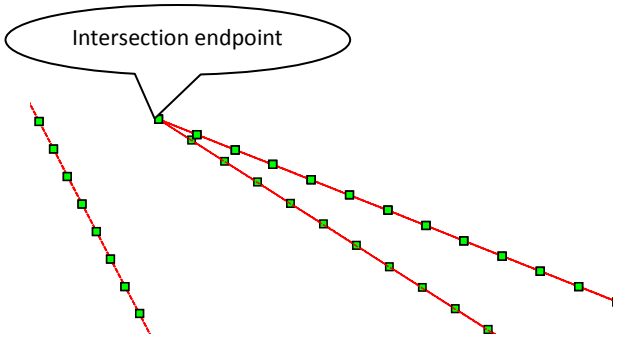


Figure 9

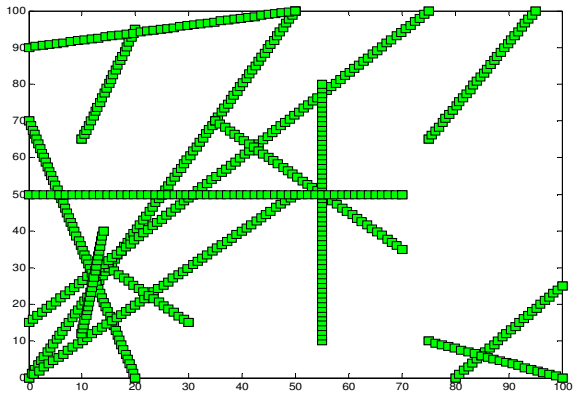


Figure 10: a sample DFM in two-dimension including horizontal and vertical fractures

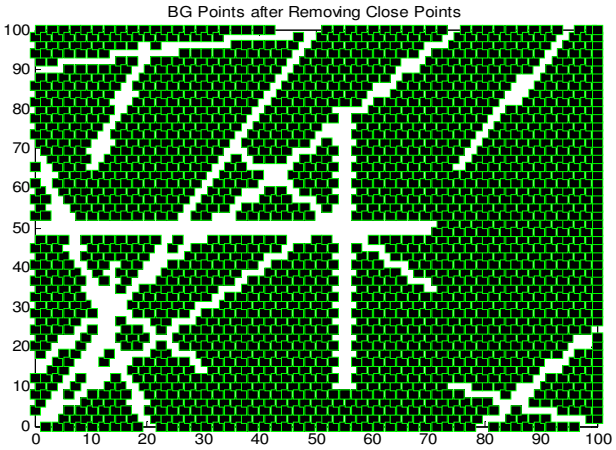


Figure 11: BG points after removing points close to the fracture points

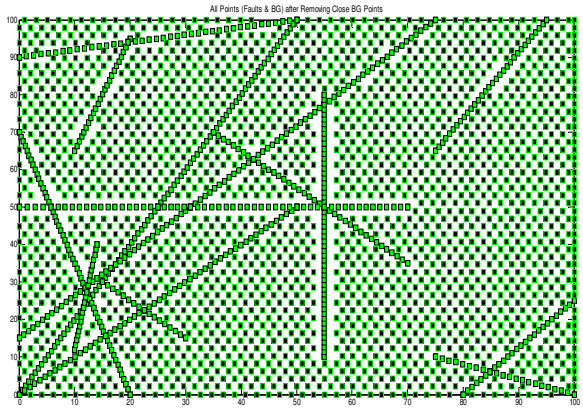


Figure 12: Both fractures and BG points

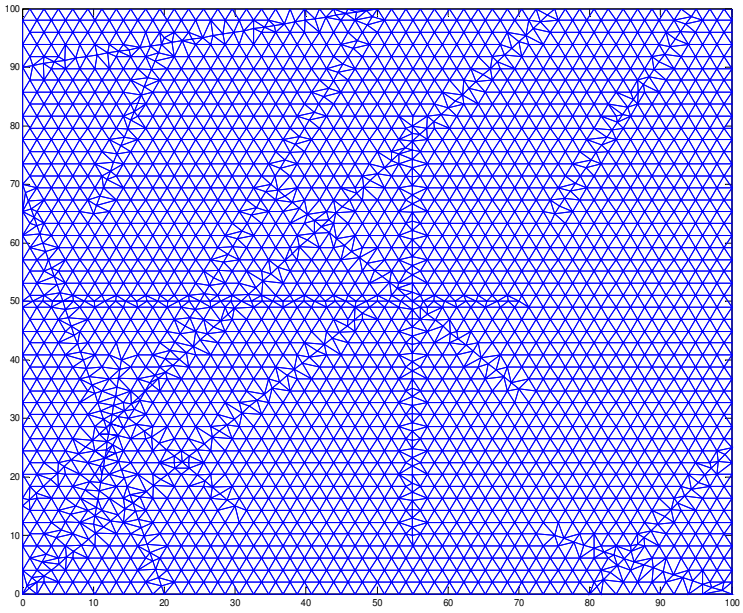


Figure 13: Delaunay mesh results
(Area Length = 100, nx = ny = 50, d = 1.5, criteria < 0.5d)

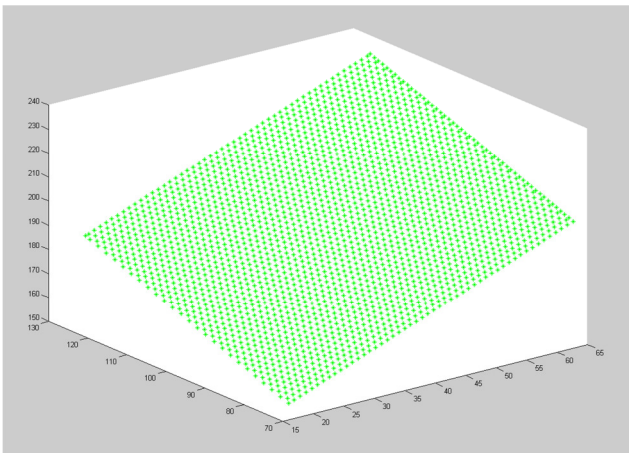


Figure 14: A sample randomly generated fracture plane where points distributed on it uniformly

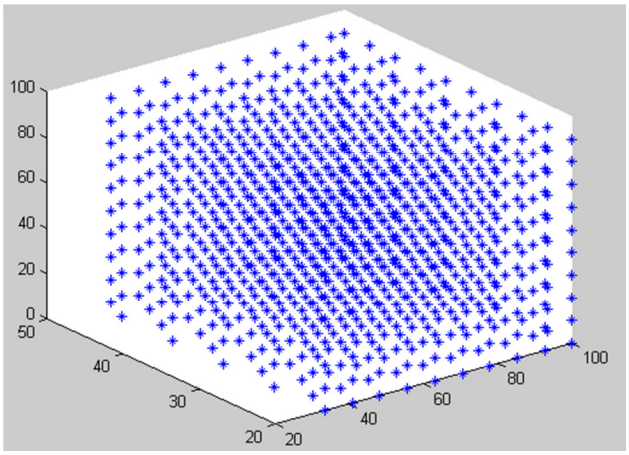


Figure 15: 3D background point distribution

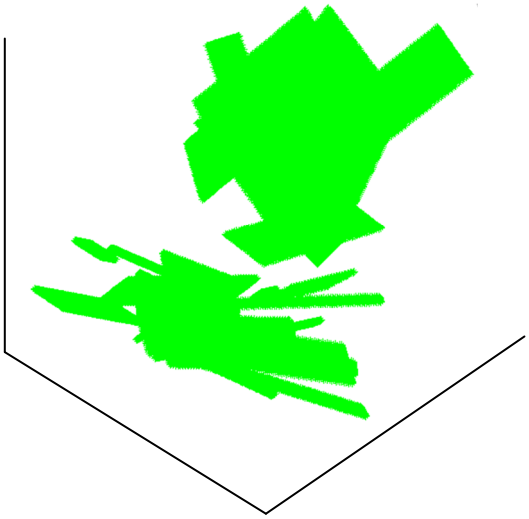


Figure 16: A randomly generated DFM with 30 fractures
(The green color is because of point distribution on the planes)

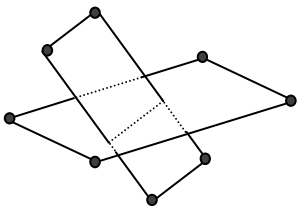


Figure 17: Two invalid facets (fracture planes) as TetGen input

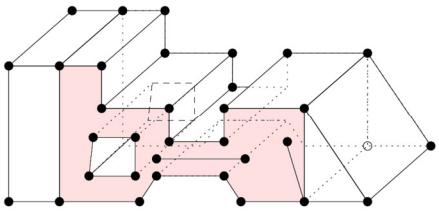


Figure 18: A sample PLC: TetGen input (Si, 2006)

```
# Part 1 - node list
24   3   0   1
 1   2  2  49
 2   2  9  49
 3   25  2  30
 4   25  9  30
 5   5  2  15
 6   5  9  15
 7   40  2  2
 8   40  9  2
 9   60  9  20
 10  60  2  20
 11  75  9  40
 12  75  2  40
 13  99  2  45
 14  99  9  45
 15  90  2  49
 16  90  9  49
 17  99  2  2
 18  99  9  2
 19  2  2  49
 20  2  9  49
```

```
# Part 2 - facet list
# facies , considering boundary markers
13 1
# Definition of 7 Fracture Planes
2 0 1
4 1 2 4 3
2 1 4
2 0 2
4 5 6 4 3
2 5 4
2 0 3
4 3 4 11 12
2 3 11
2 0 4
4 3 4 8 7
2 3 8
2 0 5
4 10 9 11 12
2 9 12
2 0 6
4 11 12 15 16
2 11 15
2 0 7
4 12 11 14 13
2 12 14
# Definition of 6 Boundary Facies
```

```
# Definition of 6 Boundary Facies
1 0 8
4 19 20 22 21
2 0 8
4 23 24 18 17
2 13 14
2 0 8
4 19 20 24 23
2 16 15
2 0 8
6 21 22 8 18 17 7
2 7 8
8 0 8
4 19 23 17 21
2 1 3
2 5 3
2 3 7
2 3 12
2 12 13
2 12 15
2 12 10
8 0 8
4 20 24 18 22
2 2 4
2 4 6
2 4 8
2 4 11
2 11 16
2 11 14
2 9 11
```

Figure 19: TetGen typical input file (*.poly)

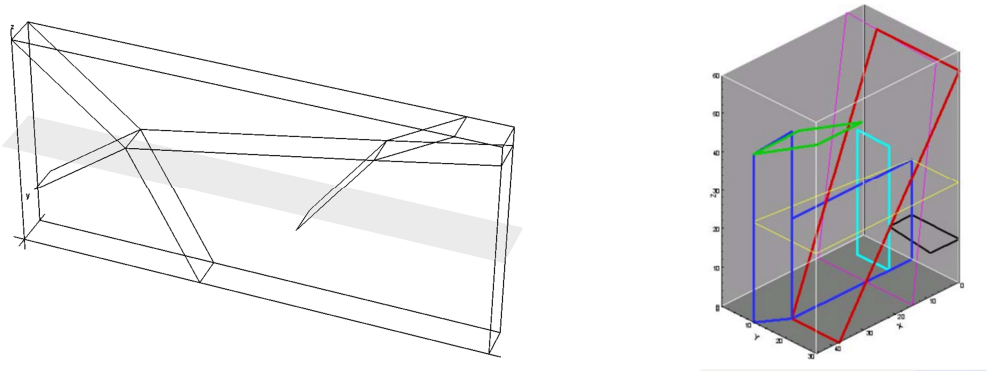


Figure 20: An example of input 3D DFM in a) TetGen and b) FracMesh (Ito, 2003)

fractestTri2.1.node

7203	3	0	1	
0	2	2	49	8
1	2	9	49	8
2	25	2	30	8
3	25	9	30	8
4	5	2	15	8
5	5	9	15	8
6	40	2	2	8
7	40	9	2	8
8	60	9	20	8
9	60	2	20	8
10	75	9	40	8
11	75	2	40	5
12	99	2	45	8
13	99	9	45	8

fractestTri2.1.face

73528	1			
0	189	130	141	6
1	189	845	130	0
2	130	845	141	0
3	141	845	189	0
4	257	897	314	8
5	257	1463	897	0
6	897	1463	314	0
7	314	1463	257	0
8	1662	2545	1041	0
9	1662	3721	2545	0
10	2545	3721	1041	0
11	1041	3721	1662	0
12	668	2118	1344	0
13	668	2122	2118	0

fractestTri2.1.ele

35083	4	1			
0	189	130	141	845	0
1	257	897	314	1463	0
2	1662	2545	1041	3721	0
3	668	2118	1344	2122	0
4	1146	1811	675	3665	0
5	1298	2862	1299	3103	0
6	449	4333	2020	4361	0
7	819	1343	313	2118	0
8	355	699	96	3200	0
9	7	145	352	903	0
10	2741	5334	3999	6413	0
11	22	54	104	199	0
12	902	1027	348	1629	0
13	948	1516	471	2366	0

Figure 21: TetGen typical output including nodes, faces and tetraheda lists

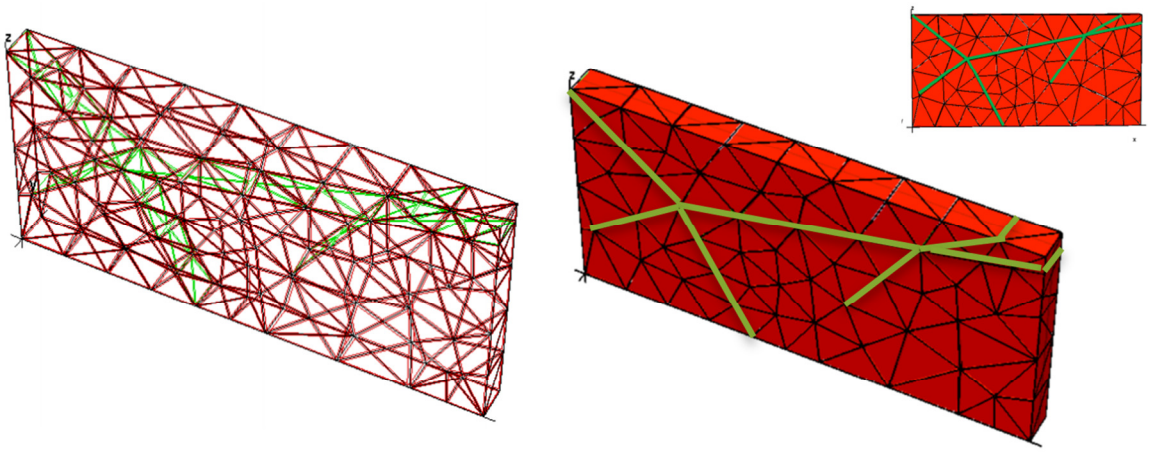


Figure 22: TetGen output visualized by TetView: a DFM with 7 fractures

```
Smallest volume:         0.0043848   |  Largest volume:          0.41467
Shortest edge:             0.32632   |  Longest edge:             1.9335
Smallest aspect ratio:      1.2436   |  Largest aspect ratio:      15.04
Smallest facangle:          16.388   |  Largest facangle:         140.3073
Smallest dihedral:          6.7059   |  Largest dihedral:         164.9972

Aspect ratio histogram:
       < 1.5        :      23729     |      6 - 10         :       6726
   1.5 - 2          :     254869     |     10 - 15         :        185
     2 - 2.5        :     194034     |     15 - 25         :          1
   2.5 - 3          :      63818     |     25 - 50         :          0
     3 - 4          :      35648     |     50 - 100        :          0
     4 - 6          :      17851     |    100 -            :          0
(A tetrahedron's aspect ratio is its longest edge length divided by its
  smallest side height)

Face angle histogram:
    0 -  10 degrees:          0      |     90 - 100 degrees:     136279
   10 -  20 degrees:         42      |    100 - 110 degrees:      38373
   20 -  30 degrees:      12522      |    110 - 120 degrees:       5201
   30 -  40 degrees:     270357      |    120 - 130 degrees:        320
   40 -  50 degrees:     616001      |    130 - 140 degrees:         23
   50 -  60 degrees:     317747      |    140 - 150 degrees:          1
   60 -  70 degrees:     270634      |    150 - 160 degrees:          0
   70 -  80 degrees:     461653      |    160 - 170 degrees:          0
   80 -  90 degrees:     304185      |    170 - 180 degrees:          0
Minimum input face angle is 19.1955 (degree).

Dihedral angle histogram:
    0 -   5 degrees:          0      |     80 - 110 degrees:     392946
    5 -  10 degrees:        895      |    110 - 120 degrees:      76322
   10 -  20 degrees:      19110      |    120 - 130 degrees:      48269
   20 -  30 degrees:      43626      |    130 - 140 degrees:      30541
   30 -  40 degrees:     107016      |    140 - 150 degrees:      19456
   40 -  50 degrees:     208582      |    150 - 160 degrees:      12388
   50 -  60 degrees:     185621      |    160 - 170 degrees:       3885
   60 -  70 degrees:      32010      |    170 - 175 degrees:          0
   70 -  80 degrees:      13055      |    175 - 180 degrees:          0
Minimum input facet dihedral angle is 19.1955 (degree).
```

Figure 23: A sample mesh quality statistics generated by TetGen

```
Back Transform:Simulated Output
5
x of barycenter
y of barycenter
z of barycenter
Simulated values by "psgsim" in Gaussian unit
Permability values: Back Transform
         97.500000          3.750000         46.687500      0.59523891 0.23534900
         35.750000          8.562500          5.500000      0.80376893 0.24768689
         29.303505          8.125000         17.665940     -0.15178446 0.19716169
          8.662253          6.375000         34.688150     -2.10633595 0.89178190E-01
         97.500000          5.500000         45.687500      1.56967262 0.28467008
         41.854730          6.812500         35.101351      0.23364580 0.21742645
          5.598150          5.971852         18.636112     -1.13771528 0.13564494
         53.500000          7.687500         37.625000     -0.75337723 0.15989928
```

Figure 24: 5th column: permeability values at the barycenters of the tetrahedrons