3D Delaunay Triangulation of 1 Billion Points on a PC

S.H. Lo

Department of Civil Engineering, the University of Hong Kong, Pokfulam, Hong Kong

(Email: hreclsh@hku.hk)

ABSTRACT

Of course, there is not enough memory on a PC with 16 GB RAM, and tetrahedra constructed have to be output to leave rooms for the creation of new tetrahedra in the next round of point insertion. A segmental zonal insertion scheme is developed, in which large data sets of more than 100 million points are partitioned into zones, each of which is triangulated in turn by the parallel zonal insertion module. An overlapping zone between two steps of insertion has to be allowed to ensure Delaunay tetrahedra formed at the boundary between two insertion zones.

Tetrahedra between zones can be easily eliminated by the minimum vertex allocation method. The collection of all the tetrahedra from each insertion zone/step will produce the required triangulation for the point set. As the work of each typical step for the insertion of an equal number of points is very much similar, the processing time bears roughly a linear relationship with the number of points in the set, at a construction rate of more than 5 million Delaunay tetrahedra per second for the triangulation of 1 billion randomly generated points.

KEY WORDS: 3D *Parallel Delaunay Triangulation, segmental zonal point insertion, 1 billion points*

1. INTRODUCTION

Based on a sound geometrical concept, Delaunay triangulation has important applications in many fields, including data visualization and imaging, terrain modeling, finite element mesh generation, surface reconstruction and structural networking for arbitrary point sets, etc. [1,2] The popularity of Delaunay triangulation is attributed to its nice geometric properties as a dual of Voronoi tessellation and the speed with which it can be constructed in two and higher dimensions. In view of its diverse applications, many strategies for its construction have been proposed [3-7]. With a rapid increase in the problem size from thousands of points to millions of points, it is necessary to devise ever more efficient schemes for the construction of Delaunay triangulations. Today's microcomputers, PC, are all equipped with more than one processors, and a standard machine with four cores and 16GB memory is quite common. Using one single processor for Delaunay triangulation represents a pitiful 25% usage of the total capacity of the machine, and an efficient parallel Delaunay algorithm making the full use of all the processors will simply boost the speed by more than four times, cutting down the triangulation time to one quarter of that by a serial process.

In 1999, Blelloch *et al* [8] presented a dividing path of Delaunay edges by projecting points on paraboloid surface for parallel triangulation of 2D points using *divide-and-conquer* algorithm. There was about 50% or four times speed up running in parallel with eight processors for uniformly or non-uniformly distributed points. In 2002, Kolingerova and Kohout [9] introduced the "optimistic method" based on the idea that the probability of collision of threads on the same triangle is relatively low for large data set. 2D points were divided into subsets, and concurrent insertions by several processors were handled by synchronization.

Based on the *divide-and-conquer* algorithm, Chen *et al* in 2004 [10] presented a parallel procedure for the near Delaunay triangulation of 2D points. The main challenge of the method was the subsequent merge of isolated triangulated patches into one coherent piece. Nave *et al* in 2004 [11] proposed a parallel Delaunay refinement algorithm by a synchronized point insertion with guaranteed quality provided certain boundary constraints are fulfilled. A parallel *divide-and-conquer* scheme for 2D Delaunay triangulation was proposed by Chen *et al* in 2006 [12], in which the "affected zone" was introduced to combine "sub-Delaunay" triangulations. Wu *et al* in 2011 [13] introduced *ParaStream*: a parallel streaming Delaunay triangulation algorithm for LiDAR points on multi-core architectures, in which kd-tree was applied to distribute workload among processors.

Parallel mesh refinement to triangular and tetrahedral meshes is a direct application of the parallel Delaunay triangulation to mesh generation. Chrisochoides and Nave [14] in 2003 presented a parallel Bowyer-Watson insertion for mesh generation by synchronization of processors in case the cavities associated with two or more concurrent inserted points intersect. It was reported that code complexity might cause issues of stability near domain boundary. A template for developing parallel Delaunay refinement was also presented by Chernikov and Chrisochides in 2010 [15], in which rigorous analysis on how to avoid conflicting Delaunay insertion in mesh refinement was discussed.

Research works have only just been started in the parallelization of Delaunay triangulation over three and higher dimensions in recent years. In 2003, Kohout and Kolingerova [16] put forward a parallel Delaunay triangulation based on a randomized incremental insertion with edge and face swaps in 3D space, in which synchronization for multiple point insertion was applied at various stages of point insertion. A survey of parallel Delaunay triangulation algorithms was presented by Kohout *et al* in 2005 [17], in which a parallel insertion algorithm was also proposed based on a synchronization scheme using thread priorities. Beyer *et al* in 2005 [18] presented a procedure for parallel dynamic and kinetic regular triangulation in three dimensions, based on an incremental construction with parallel flipping of tetrahedra arising from the idea of accessible and non-accessible simplices. Batista *et al* in 2010 [19] studied and implemented several parallel geometric algorithms for multi-core computers. YingLiang Ma [20] presented a surface extraction algorithm for large binary image data set based on parallel 3D Delaunay subdivision strategy. Points and surface data were partitioned into zones, in which Delaunay triangulation was carries out over each zone in parallel, which could then be connected to form the skeleton of the required surface. However, no details were given in the algorithm or in the implementation of the parallel 3D Delaunay triangulation.

A parallel insertion scheme by zonal subdivision has recently been proposed by Lo [21], in which points are partitioned into cells which are then grouped into zones for parallel insertion. A high efficiency of this method is attributed to its absolute independence for zonal insertion by each processor and the ease and robustness in the elimination of redundant tetrahedra at the boundary between zones by the elegant minimum vertex allocation rule. By means of the parallel zonal insertion, on a PC i7 CPU 870@2.93GHz with 16GB RAM, it is able to construct 350 million Delaunay tetrahedra in less than 100 seconds.

However, 16 GB RAM is what we can have nowadays on a PC. In case, we would like to tackle a large point set of more than 50 million 3D points, a more powerful machine of more than 16 GB RAM is needed. From this, we can see that the bottleneck for the construction of Delaunay triangulation for a large point set is not the speed but the memory restriction. In the light of the parallel zonal insertion scheme, each processor works independently within its assigned zone, and the tetrahedra so constructed will be subsequently put together to form the final triangulation. This gives us the idea that a large point set can be first divided into zones, which are to be handled by one or more processors as a typical parallel zonal insertion step. A segmental zonal insertion scheme can thus be formulated by taking one zone at a time until all the zones have been processed.

Nevertheless, in each step, tetrahedra have to be output to allow rooms for the creation of new tetrahedra in the next step of zonal insertion. The coordinates and the zonal label of the points have to be synchronized with the step by step insertion process as well, so that the number of coordinate points could be controlled to within a reasonable limit, just good enough for each step of zonal insertion. An overlapping zone between two steps of insertion has to be provided too to ensure Delaunay tetrahedra are formed at the boundary between two insertion zones.

In this paper, a segmental zonal insertion scheme will be introduced to triangulate large sets of randomly generated points and typical non-uniformly distributed points. In theory, there is no limit to the number of points in a triangulation, and it is only a matter of computer running time and the amount of tetrahedra that can be stored in the available permanent memory. By the nature of the segmental zonal insertion, the time complexity is basically linear, and about 3.1 million Delaunay tetrahedra per second can be created in the triangulation of 1 billion randomly generated points.

2. DELAUNAY TRIANGULATION

The Delaunay triangulation of a set of points on a plane is defined to be a triangulation such that the circumcircle of every triangle in the triangulation contains no point from the set in its interior. Such a triangulation exists for a given set of points, and it is the dual of the Voronoi tessellation. The triangulation is unique if the points are in general position, i.e. no four points are cyclic. A triangle T is said to be Delaunay with respect to a point **p** if **p** does not lie inside the circumcircle of T. A triangle T in a triangulation of a set of points is call Delaunay triangle if T is Delaunay with respect to every point in the set. A triangulation of a set of points is called the Delaunay triangulation of the point set if every triangle in the triangulation is a Delaunay triangle as shown in Figure 1. The notion of Delaunay triangulation is very general, which can be easily extended to higher dimensions. For instance, the Delaunay triangulation in three dimensions is given by replacing triangle by tetrahedron, circle by sphere and 2D plane by 3D space. The following lemma provides the basis for many algorithms in the construction and verification of Delaunay triangulation.

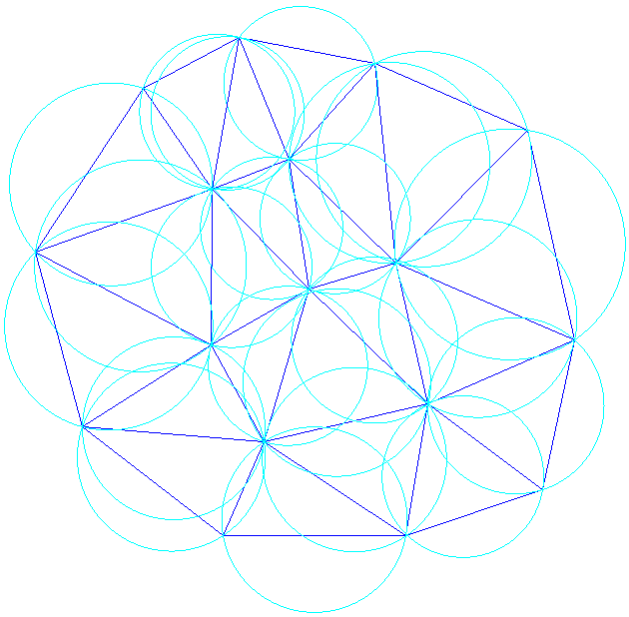


Figure 1. Delaunay triangulation of 17 points

Lemma of Delaunay [22]

*Let **T(S)** be a triangulation of the point set **S**. The necessary and sufficient condition that no point of **S** is contained in the circumsphere of any tetrahedron in the triangulation is that any two adjacent tetrahedra in the triangulation are Delaunay with respect to each other's vertices.*

2.1 The Insertion Algorithm

For the construction of Delaunay triangulation in three and higher dimensions, point insertion algorithm is the most popular, and many interesting methods have been proposed [3-7]. For a set of 3D points, the initial triangulation is a cuboid consisting of five or six Delaunay tetrahedra large enough to contain all the given points as shown in Figure 2. The Delaunay triangulation is achieved by inserting points one by one into the initial triangulation. Each cycle of point insertion can be divided into three steps.

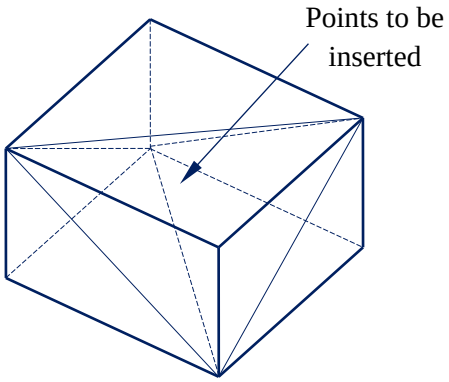


Figure 2. Initial triangulation of five tetrahedra

(i) For a newly inserted point, identify all the tetrahedra whose circumsphere contains the point in its interior. The cavity left behind upon removal of these tetrahedra forms a star-shaped insertion polyhedron.

(ii) Owing to the finite precision arithmetic, the triangulation facets on the boundary of the cavity have to be verified with the visibility check and corrected before they are connected with the inserted point to form tetrahedra.

(iii) The triangulation of the insertion polyhedron should be trivial. However, the adjacency relationship of the tetrahedra has to be established, which will be frequently referred to throughout the triangulation process.

When a new point **p** is inserted in a Delaunay triangulation, it is required to find all the tetrahedra whose circumsphere contains the point **p**. A simple method to determine those non-Delaunay tetrahedra is to scan through all the existing tetrahedra for those circumspheres containing the point **p**. However, a more efficient approach is to start with the tetrahedron which contains the inserted point **p** and find the others by means of the adjacency relationship. In this way, the boundary of the insertion polyhedron is given by the common faces of two tetrahedra for which one is positive in the sphere inclusion test while the other fails. The tetrahedron which contains the insertion point **p** is called the *base*, which is an integral part of the insertion polyhedron.

In the point insertion algorithm, there are two basic steps, namely, (i) the location of the base tetrahedron containing the inserted point and (ii) to ensure the circumsphere criterion is verified for all the tetrahedra connected to the inserted point. If points are inserted cell by cell as shown in Figure 3, the searching path in determining the *base* is a constant depending on the number of points in a cell, and the verification of the circumsphere criterion is a local process if we follow the adjacency relationship of the tetrahedra, thanks to lemma of Delaunay. From this observation, it can be seen that Delaunay triangulation by point insertion is one of the most efficient algorithms for its simplicity and linearity provided that points are inserted in clusters in a contiguous manner.

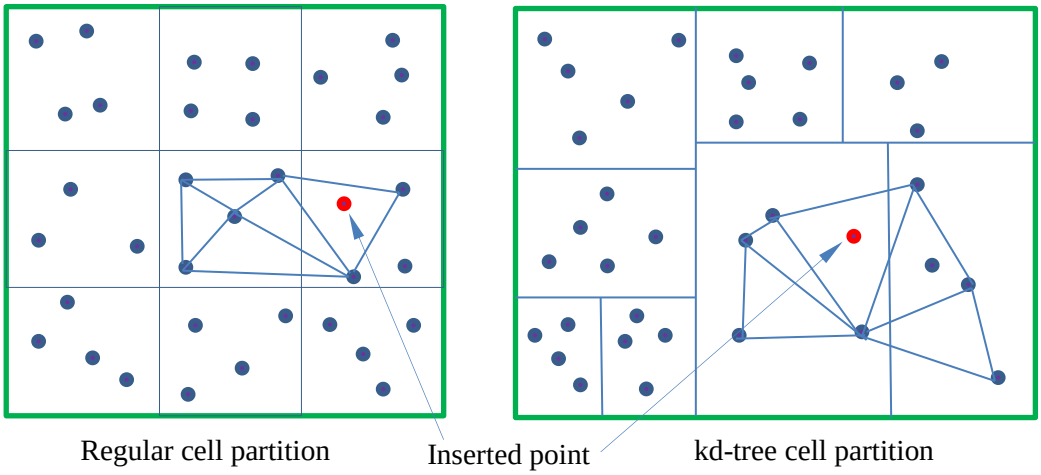


Figure 3. Points sorted into cells

Point insertion following an ordered cell sequence for Delaunay triangulation of points in clusters is a very simple and efficient scheme as the number of operations is almost optimal. Consider 3D point insertion, it takes *n* scans of tetrahedra on average in locating the base tetrahedron, where *n* is the number of points in a cell, and on average a point is connected to 27 tetrahedra for which circumsphere criterion has to be verified. A simple count illustrates that on average the number of operations for each point insertion is n+27=35 if each cell contains roughly 8 points. Numerical tests of the insertion of 1 million randomly generated points by single-processor insertion shows that over 1 million Delaunay tetrahedra can be constructed per second on a PC.

3. PARALLEL DELAUNAY TRIANGULATION IN 3D

3.1 General Considerations and Strategies

A robust and efficient parallel 3D Delaunay triangulation algorithm has to be based on a sound, reliable and fast sequential scheme, and hence parallelization of point insertion is considered for the possibility of multiple point simultaneous insertions by several processors.

There are three main strategies proposed so far for Delaunay triangulation by a multiple insertion of points concurrently with several processors. (i) *Creation of Delaunay boundary edges*. Delaunay cutting lines are introduced by local Delaunay triangulation or by some more sophisticated method such as the projection onto a paraboloid [8] to partition the two-dimensional domain into two roughly equal portions. Further division can be done by introducing more cut lines to produce as many regions as necessary for triangulation by point insertion or any other techniques within each isolated region in parallel. (ii) *Triangulation of points within zones in parallel* and the patches of triangulation are connected by filling up gaps between zones. The given points for triangulation are first allocated to various zones, the ensemble of which is a partition of space covering all the points. Delaunay triangulation can now be carried out using points within the zones in parallel. The result is a collection of isolated patches of triangles, which have to be connected properly to form the final Delaunay

triangulation [23]. (iii) *Synchronization*. Parallel point insertion controlled by some verified sequence/location of insertion, or by means of synchronized insertion for several processors by blocking the access of a particular point or triangles from other processors to avoid conflicts [9,14]. This idea can be readily extended to three dimensions [16,17] as the procedure is not based on any two-dimensional features of the triangulation. However, single-processor operations and the checking for the occurrence of such events will substantially slow down the overall performance of parallel insertion.

3.2 The Zonal Insertion Scheme

In theory, a processor can be assigned to each point for the maximum speed up. Nevertheless, machines with so many processors are not yet available, and even they exist it may not be economical (most efficient) to have so many processors working together as conflicts are bound to occur for the insertion of neighbouring points. It is impractical and unnecessary to assign one processor to each point as there is a lot of redundancy in doing so, and an optimized scheme is to group several cells together into a zone for a single processor insertion. As a result, cells of spatial partitions of points are grouped into a number of zones depending on the number of processors available. There are two major issues to be addressed in such a scenario: (i) make sure that Delaunay tetrahedra are constructed at the boundary between zones and (ii) how to get rid of the redundant tetrahedra efficiently in a rigorous manner. Such a zonal parallel insertion scheme has recently been proposed by Lo in a paper entitled "Parallel Delaunay Triangulation in three dimensions" [21]. The major steps are outlined as follows.

3.2.1 Points partitioned into cells

Let N be the number of points in a 3D Delaunay triangulation, and n be the average number of point desirable in a cell, then

$$nN_xN_yN_z = N \tag{1}$$

where N_x, N_y, N_z are respectively the number of cell division along the x, y, and z-axis and the number of cells, $N_c=N_x N_y N_z$

Let x_{min}, x_{max} y_{min}, y_{max}, z_{min}, z_{max} be the bounds of the (x,y,z) coordinates of the point set, compute $R_x = x_{max} - x_{min}$, $R_y = y_{max} - y_{min}$ and $R_z = z_{max} - z_{min}$, then

N_x, N_y and N_z can be determined by substituting $\mu R_x = N_x$, $\mu R_y = N_y$ and $\mu R_z = N_z$ into (1).

The most important requirement in a spatial partition of points into cells is to ensure that each point belongs to one and only one cell, and the sum of points in all the cells equal to the total number of points, i.e.

$$N = \sum_{i=1}^{N_c} n_i \qquad \text{where } n_i = \text{number of points in cell i}$$

and $N_c = N_x N_y N_z$ is the number of cells in the partition

3.2.2 Grouping cells into zones

Let $N_{Zone}=D_xD_yD_z$ be the number of zones, which is usually a multiple of the number of processors available for parallel zonal insertion, where D_x, D_y and D_z are the zonal divisions along x-axis, y-axis and z-axis respectively as shown in Figure 4, then cells are grouped into zones such that each zone will consist of m cells given by

$$m = \text{NINT}\left(\frac{N_x}{D_x}\right)\text{NINT}\left(\frac{N_y}{D_y}\right)\text{NINT}\left(\frac{N_z}{D_z}\right)$$

For the best performance of the parallel insertion, the number of zones has to be an integral multiple of the number of processors available; for example, division into 2x2x3=12 zones for 4 or 6 processors is a sound division. However, division into 12 zones for 8 processors may not be that desirable, because 12=8+4 and not all processors will be working to their full capacity all the time. However, since we are working with millions of points, the partition into cells and the division into zones which are multiples of the number of processors will never be a problem. The number of cells in some zones near the boundary may have fewer or more cells if N_x/D_x, N_y/D_y or N_z/D_z is not a whole number. Such a variation would not cause any problem in the subsequent operations, as a zone is identified by the bounding cells in x-, y- and z-directions, i.e. a zone I, I=1~N_{Zone}, is specified by $\text{Zone I} = \text{Zone}(N_{x1}, N_{x2}, N_{y1}, N_{y2}, N_{z1}, N_{z2})$, where ($N_{x1}$,$N_{x2}$), ($N_{y1}$,$N_{y2}$) and ($N_{z1}$,$N_{z2}$) are respectively the starting and ending cell division lines along x-, y- and z-directions.

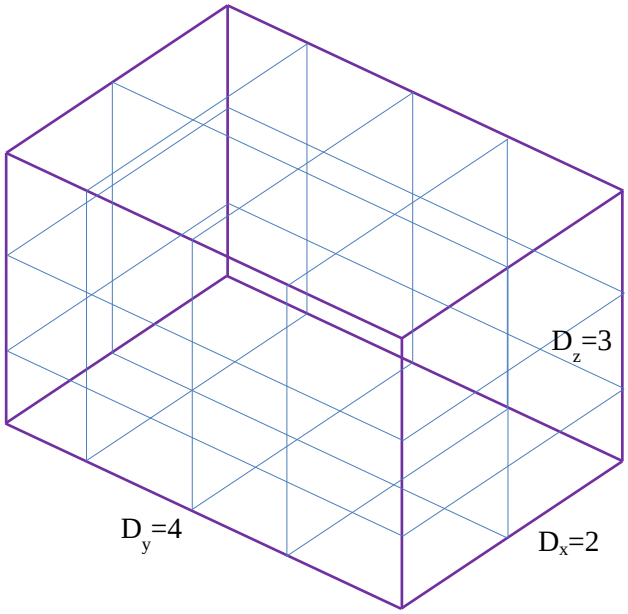


Figure 4. Partitioned into 2x4x3=24 zones

3.2.3 Simultaneous insertion in 3D

A division into 2x2x2=8 zones for the insertion of 2000 three-dimensional points is shown in Figure 5, in which the average number of points in a cell, n=8, R_x, R_y and R_z are in the ratios 1:1:1, hence $N_x=N_y=N_z=6$, and $nN_xN_yN_z=1728\approx2000$. The initial triangulation for each zone is a cuboid of five tetrahedra large enough to contain all the points. Point insertion in each zone will be handled by one single processor in a completely independent manner. All the points within each zone I, I=1~8, will be processed concurrently by inserting points in those cells belonging to the zone under consideration. This will generate all the Delaunay tetrahedra within the zone; however, Delaunay tetrahedra crossing the zonal boundary are missing as shown in Figure 6. The next step is to construct all the Delaunay tetrahedra at the boundary between zones. To do so, boundary cells are added around the zone to all the boundary surfaces of the zone as shown in Figure 7. For easy visualization, only those boundary tetrahedra with the neighbouring zones and their associated circumspheres are shown, whereas tetrahedra formed with the auxiliary corner points and their associated circumspheres are not shown. Boundary tetrahedra are defined as those tetrahedra supported on vertex or vertices from the current zone and vertex or vertices from neighbouring zone(s) as shown in Figure 7. This simple definition is also applicable to Octree and kd-tree spatial partitions.

Layers of cells can be added to the boundary surfaces of the augmented zone until all the circumspheres of the boundary tetrahedra are bounded to ensure all boundary tetrahedra are Delaunay as shown in Figure 7. As each cell contains roughly equal number of points, the process converges fairly rapidly and evenly on all the zonal boundary faces in one or two layers of cells. In this particular example shown in Figure 7, no additional layer of cells is needed, as all circumspheres of the boundary tetrahedra are already bounded by the augmented zone. Similar to the two-dimensional case, points considered before need not be reconsidered after further point insertions as the union of circumspheres will always shrink for the introduction of a new point.

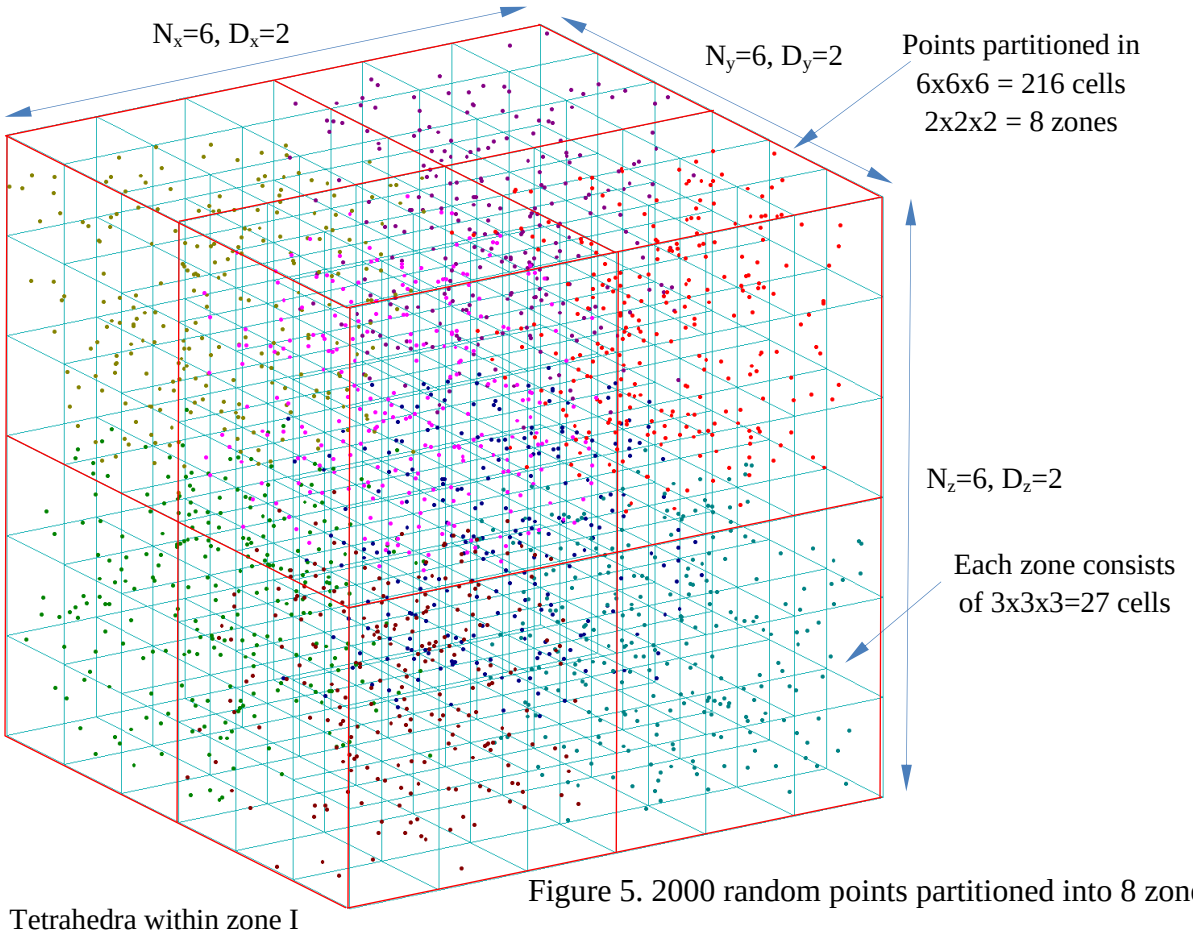


Figure 5. 2000 random points partitioned into 8 zones

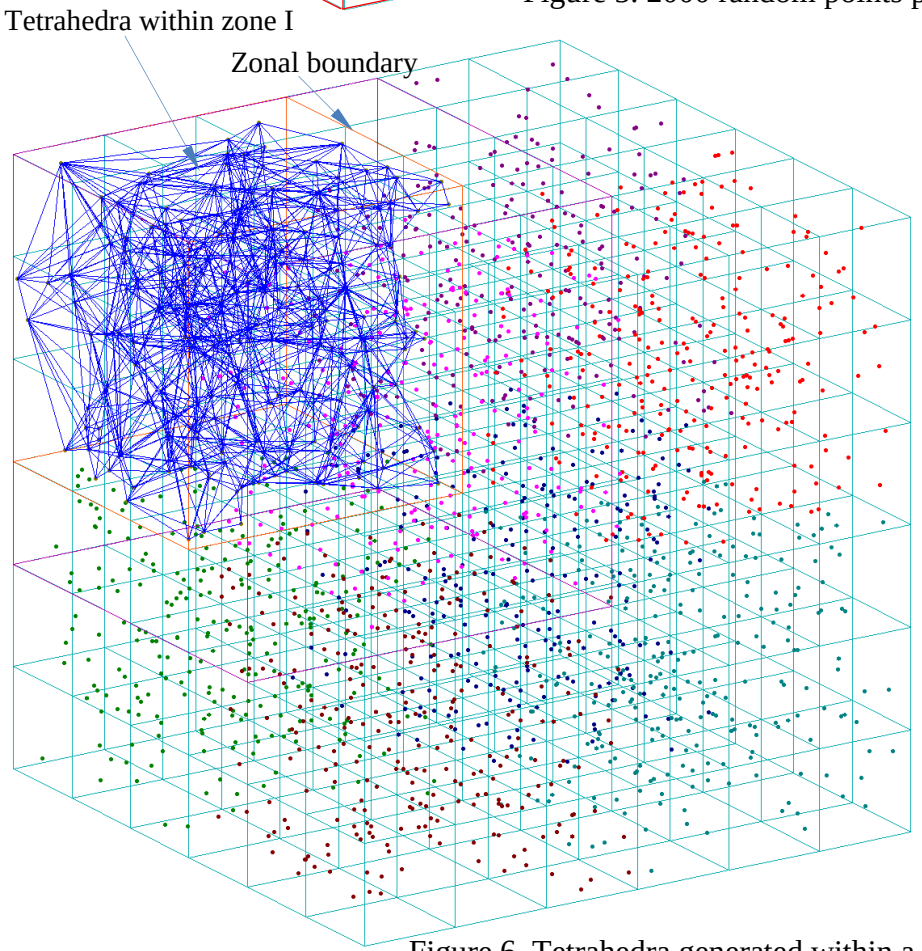


Figure 6. Tetrahedra generated within a zone

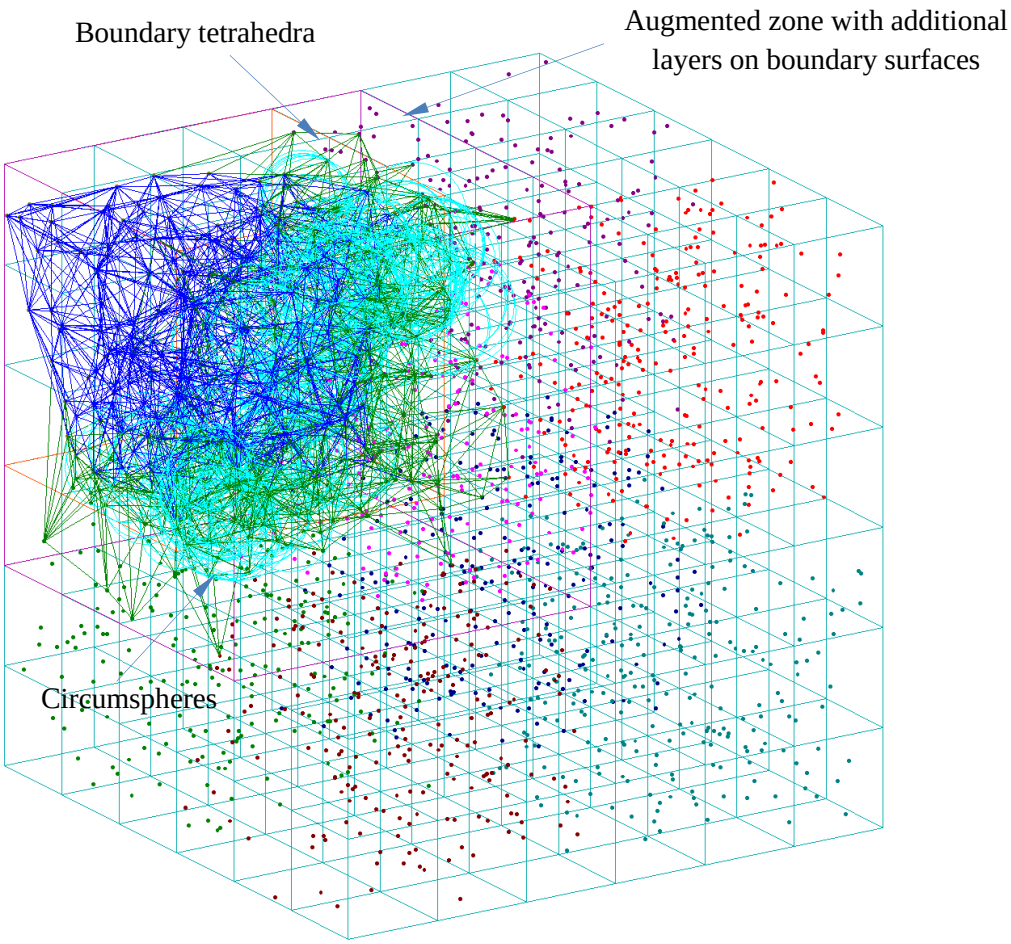


Figure 7. Boundary tetrahedra and circumspheres

3.2.4 Elimination of redundant tetrahedra

Delaunay triangulation by zonal insertion is complete in the sense that it contains all the Delaunay tetrahedra of the point set as Delaunay tetrahedra in a patch around each point are constructed in the insertion process for all the points partitioned into zones. As each zone is processed by a processor independently, all the tetrahedra within a zone are generated by one processor. However, tetrahedra on the boundary between zones may be generated by one or more processors. A general rule for assigning tetrahedra to zones can be formulated such that tetrahedra with vertex zone labels z_1, z_2, z_3 and z_4 will be assigned to zone z given by

$$z = \min(z_1, z_2, z_3, z_4)$$

This is a surprisingly simple and elegant scheme, by which redundant tetrahedra can be eliminated independently almost without effort. By the minimum vertex allocation, redundant tetrahedra for the parallel insertion of 2000 3D points are eliminated and distributed into 8 zones as shown in Figure 8. This is not the convex hull of the given points as the auxiliary corner points are not placed at infinity, but a Delaunay triangulation of the points with some missing boundary edges and faces.

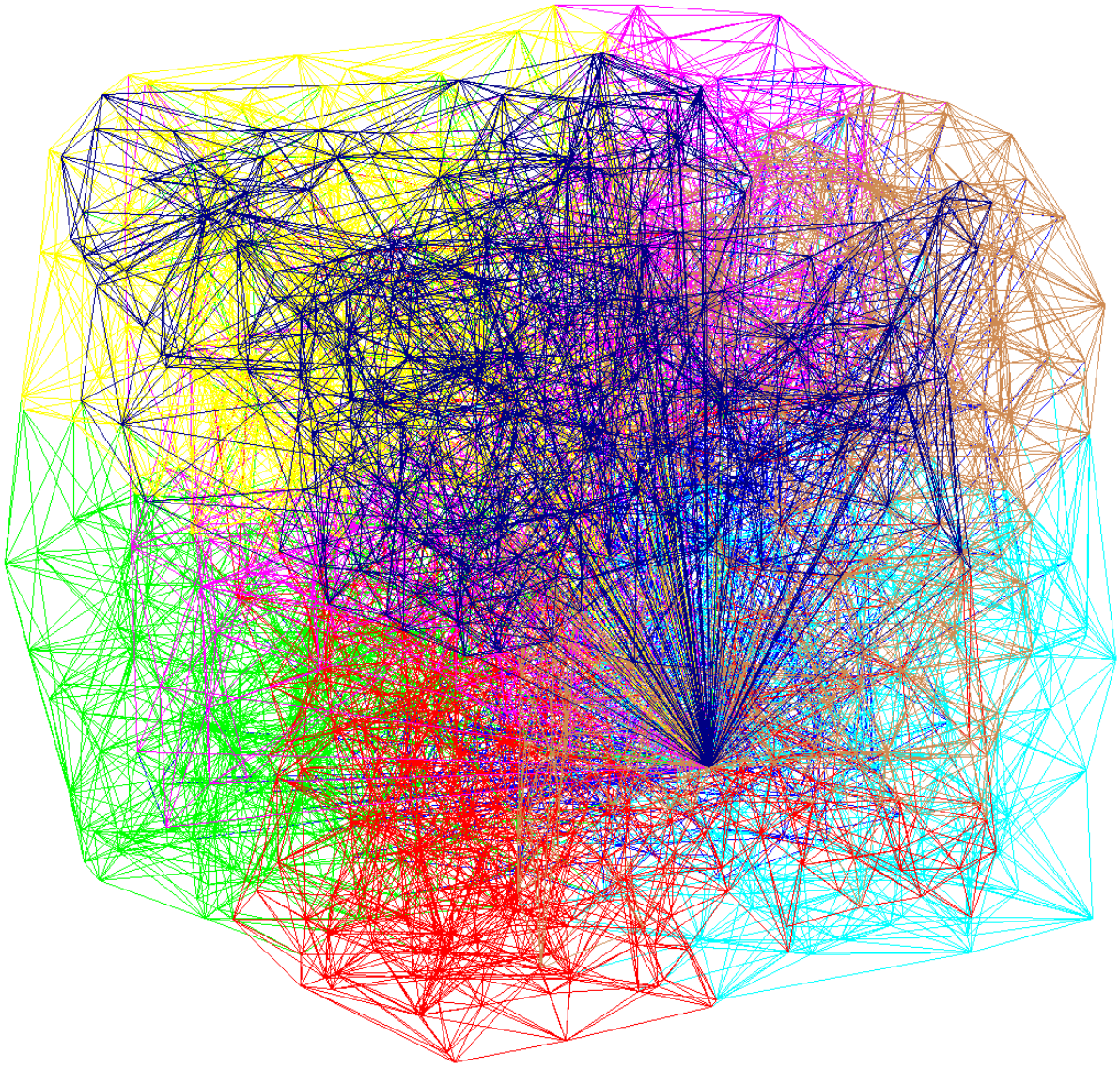


Figure 8. Partitioning of tetrahedra into 8 zones by the minimum vertex scheme

The simple insertion scheme by means of a regular grid for uniformly or mildly non-uniformly distributed points is slightly more efficient (about 10% on average) compared to the well-established triangulation algorithm CGAL4.2 [24]. For highly non-uniform point distributions, the multi-grid insertion scheme can be used, which is one of the most efficient triangulation methods ever proposed [24]. The scalability of the parallel zonal point insertion has been fully described in reference [21]. The purpose of this paper is to make use of these advanced techniques just developed in conjunction with the novel segmental triangulation algorithm to be introduced in the next section into a single triangulation scheme for ultra large point sets on a standard PC. In principle, even for the limited resources of a PC, there is no restriction to the size of the point set to be triangulated so long as sufficient time is allowed for its execution.

4. SEGMENTAL TRIANGULATION

4.1 Partition into zones

In case the set of points is too large to be handled all at the same time, points can be partitioned into zones which are read in progressively step by step for triangulation in a segmental insertion process zone by zone. Following the idea of parallel triangulation described in Section 3, Delaunay triangulation of all the points in a zone can be assured if Delaunay point insertion is applied to all the points within the zone and the entire zone is covered by a layer of Delaunay tetrahedra. Depending on the value of R_x, R_y and R_z, the set of points is divided into zones along the longer dimension. Without loss of generality, suppose R_z is the largest and the point set is divided along the z-direction. As shown in Figure 9, the number of point per layer along the z-direction is given by

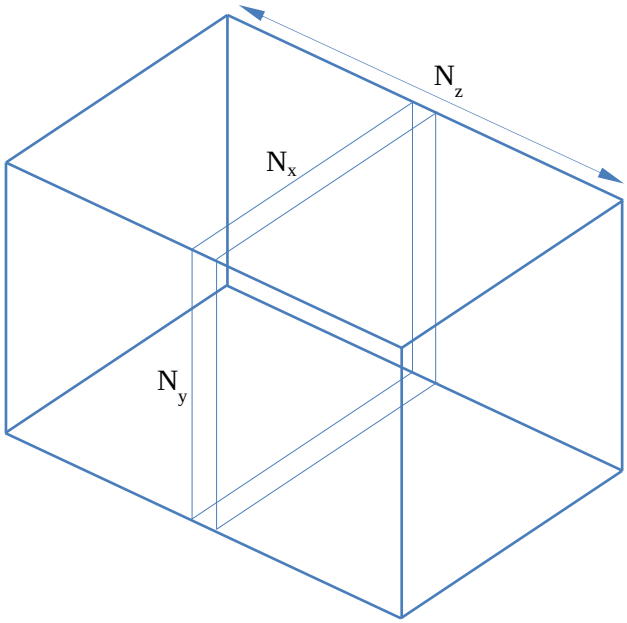


Figure 9. Partitioned into cells

$N_L = nN_xN_y$ where n is the average number of points in a cell

From the experience of parallel zonal insertion for randomly generated points, 2 to 3 layers are usually sufficient to ensure a Delaunay cover of the zone. Hence, if an overlapping zone of m layers of cells is allowed between two insertion zones, the number of points in this buffer zone is given by

$$N_B = mnN_xN_y$$

where m can be conveniently set to 5, which is about twice the number of layers needed for Delaunay triangulation. In the actual implementation, Delaunay triangulation at the zonal boundary is achieved by adding layers of cells from the buffer zone in a progressive manner, and it would be easy to find out if the buffer zone has been exceeded or not. Let N_s be the number of points that can be handled in one insertion step. If N_I is the number of points to be inserted in a step, we have

$$N_I + 2N_B = N_S$$

N_B has to be multiplied by 2 as there are overlapping zones both in the front and at the back of the insertion zone as shown in Figure 10.

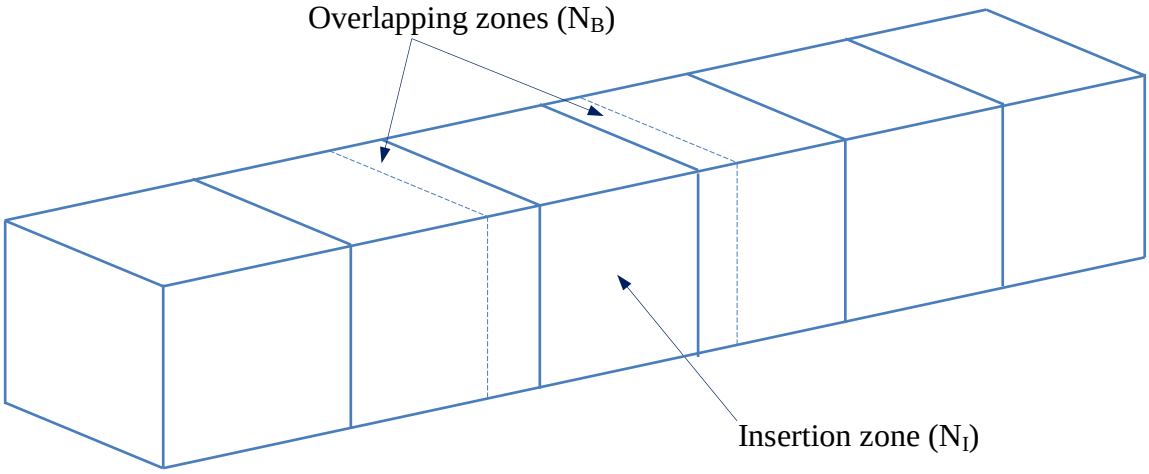


Figure 10. Insertion zone bounded by buffer zones on both sides

Knowing N_I, it is not difficult to estimate the number of steps needed for the entire segmental insertion process.

$$N_{STEP} = N/N_I + 1$$

and the number of points to be inserted in each step is then adjusted to

$$N_I^* = N/N_{STEP}$$

In a typical insertion step, points in the insertion zone are fed into a parallel insertion module to create a Delaunay triangulation for the insertion zone. Tetrahedra not belonging to the zone can be readily eliminated by the minimum vertex allocation scheme.

4.2 Memory Management

Perhaps, the most difficult part of the segmental zonal insertion for a large set of point is to make the best use of the available memory in a simple robust manner. No extra memory is needed for the parallel zonal insertion scheme compared to the classical sequential insertion process, except a zonal label for each point stored as a 2-byte integer in a linear array. To minimize the use of memory, the circumcentre and the circumradius of each tetrahedron are not stored, which are re-calculated whenever necessary. For each point insertion, about $7 \times (4 + 4) \times 4 + 3 \times 8 = 248 \approx 250$ bytes memory are required to store the vertices and the neighbours of the 7 tetrahedra (rounded up from an average of 6.75) generated and the x, y and z-coordinates of the point. Thus, a PC with 16G Bytes, apart from the memory taken up by the operating system, can generate quite comfortably 350 million tetrahedra for an insertion of more than 50 million points without being appreciably slowed down due to a lack of memory. In the segmental zonal insertion process, about 60 million points can be handled in one single insertion step, i.e. $N_S = 6 \times 10^7$, subtracting points in the overlapping zones, points in the insertion zone is about 50 million, or $N_I = 5 \times 10^7$.

The quantities that are needed in a parallel insertion process are the (x, y, z) coordinates of the points, the tetrahedra constructed, the adjacency information of the tetrahedra, namely, their neighbours and the zonal label of the points. Once triangulation is done for the insertion zone in an insertion step, the tetrahedra constructed can be output to allow room for new tetrahedra to be created in the next insertion step. However, points in the overlapping zones have to be retained for the next round of point insertion, and the label to identify individual point will continue to increase as the segmental insertion process advances with more insertion steps.

A simple solution to access correctly the coordinates of the points is as follows. Coordinates of point i, $i \in \{1,2,\dots,N\}$ with $N >> N_S$, is given by

$$(x_i, y_i, z_i) = (x_j, y_j, z_j)$$

where $j = \mathrm{mod}(i-1, N_s) + 1, \qquad j \in \{1,2,\dots,N_S\}$.

As the number of points in an insertion step is given by

$$N_I^* + 2N_B$$

which is always smaller than N_S, and each point in the insertion zone will be unique and within the range $\{1,2,\dots,N_S\}$. Of course, when points are input for the next zonal insertion, coordinates of point i will occupy coordinates of the point with label j, such that most coordinates (x_j, y_j, z_j), $j \in \{1,2,\dots,N_S\}$, will be over written for each insertion step. The zonal label, which is also an attribute of the points, has to be treated in a similar manner as the spatial point coordinates.

4.3 Dynamic Grid Update

Based on this modulus point labeling scheme, there is no limit to the number of points in a triangulation, and it is only a matter of computer running time to handle very large point sets. However, the number of cells will keep increasing when more and more insertion zones are processed as shown in Figure 11. Hence, it is necessary to reset the cell label to smaller numbers for each insertion step; otherwise the maximum number of cells will be exceeded in the segmental insertion process.

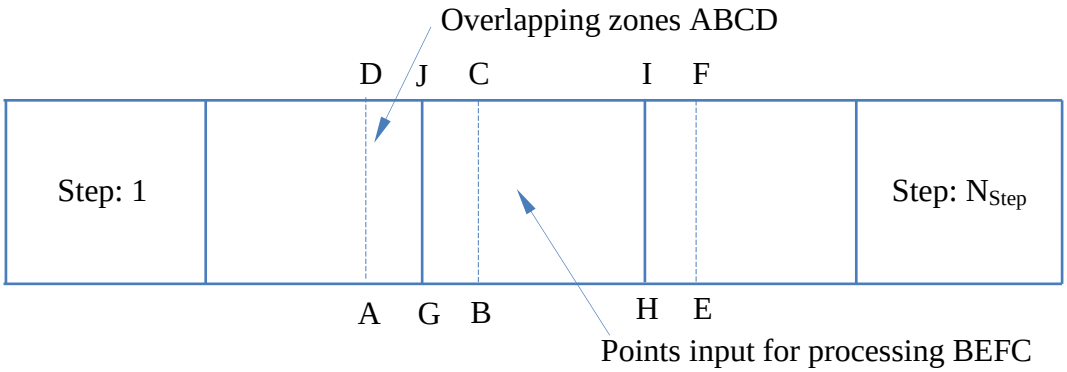


Figure 11. Typical step of segmental zonal insertion

As shown in Figure 11, the bounding region of segmental insertion AEFD consists of two portions, the overlapping zone ABCD from the previous insertion and the current input zone BEFC, which can be re-partitioned into three zones for parallel insertion, namely the buffer zone in the front AGJD, the insertion zone GHIJ and the buffer zone at the back HEFI. The first cell of the overlapping zone ABCD from the last round of point insertion has to be reset to 1, and cells in the input zone BEFC will be labeled sequentially following those in the overlapping zone ABCD. The number of cells in the input zone BEFC and the overlapping zone ABCD should not exceed the number of cells allowed in the segmental insertion process.

To reset the cell label of overlapping zone ABCD, simply copy the content of the cells in the zone ABCD to cells with label starting from 1. In the present implementation, points in cell k is given by a pointer array P such that points in cell k = $\{P_i, i=P_k+1, P_{k+1}\}$, k=1,2, . . . , N_c, where N_c is the number of cells. Let k_1 and k_2 be respectively the first cell and the last cell of the overlapping zone ABCD. Then the number of cells in the zone ABCD, No = k_2-k_1+1. Hence, the contents in the cells k_1 to k_2 have to be transferred to cells 1 to No.

(i) The number of points (offsets) in each cell:

$$\{P_{k1}, P_{k1+1}, \ldots, P_{k2+1}\} \longmapsto \{P_1, P_2, \ldots, P_{No+1}\}$$

(ii) The points in each cell from k_1 to k_2:

Points in cells k_1 to k_2 are given by $\{P_{j1}, P_{j1+1}, \ldots, P_{j2}\}$ where $j_1 = P_{k1}+1$, $j_2 = P_{k2+1}$.

Number of points in cells k_1 to k_2, $N_P = j_2 - j_1 + 1$. These points have to be relocated within pointer array P so that P would not be exhausted in the insertion process.

Let L be the allowable size of array P, usually $L=2N_I$ is good enough for the segmental insertion as for a regular grid the memory requirement for the pointer array P is given by

$$\text{Memory required} = N_c + N_P$$

where N_c is the number of cells and N_P is the number of points.

As, on the average, there are more than one point in each cell, $N_c < N_P$, and memory required is less than $2N_P$. If each cell contains roughly $n \approx 20$ points, it is pretty safe to set $L = 2N_I$. As shown in Figure 12, points in cells k_1 to k_2, $\{P_{j1}, P_{j1+1}, \ldots, P_{j2}\}$, are relocated in array P to positions starting from s given by

$$s = L - N_I - N_P$$

where N_I is the number of points read in for the current round of insertion, such that

$$\{P_{j1}, P_{j1+1}, \ldots, P_{j2+1}\} \mapsto \{P_{s+1}, P_{s+2}, \ldots, P_{s+NP}\}$$

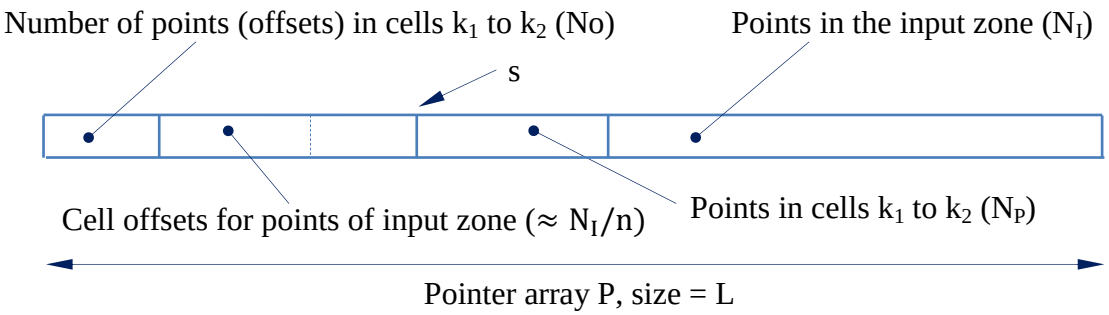


Figure 12. Memory arrangement of pointer array P for zonal partition into cells

Remark: The performance of the zonal insertion scheme is not very sensitive to the number of points in a cell (n), as long as it is of the right order for evenly distributed points on a larger scale. The theoretical optimal value for n is not known, as it is only a statistical average depending also on the point distribution. Indeed, for randomly generated points, the number of points in a cell is not a constant but varies from 1 or 2 up to 2n or more.

5 EXAMPLES

The segmental parallel Delaunay triangulation scheme was tested with 200 million to 1 billion randomly and non-uniformly distributed points on a PC, Intel®, Core™, i7 CPU, 870@2.93GHz with 16 GB RAM running on Window 7 using Intel Fortran VS2010 on 64 bit, which supports OpenMP directives for parallel processing on shared memory architecture. The validity of the segmental insertion scheme has been thoroughly checked against a single processor insertion process with up to 50 million randomly generated points triangulated all in one zone. As shown in Table 1, 6 sets of randomly generated points ranging from 200 million to 1 billion points were triangulated by the segmental insertion scheme. In these tests, only memory for handling 10% of the points was allowed for the triangulation of the point sets, i.e. for the triangulation of 200 million points only 20 million points were read in at a time, and as a result, the point set was triangulated in 14 steps. As for the triangulation of 400 million points, 40 million points were allowed for each insertion, and 13 steps were required for the triangulation of the point set.

Example	Distribution	NP	Step	NI	NB	NT	NT*	T1	T2	$NT^{\#}$
U1	Random	200	14	14.3	1.55	1562.6	1350.7	431.4	459.9	3.13
U2	Random	400	13	30.8	2.46	3012.3	2702.6	845.7	910	3.20
U3	Random	600	15	40	3.22	4495.1	4054.9	1300.2	1403.9	3.12
U4	Random	800	17	47.1	3.9	5990.6	5407.4	1717.8	1854.8	3.15
U5	Random	1000	22	45.5	4.52	7576.6	6760.3	2191.9	2360.6	3.08
U6	Random	1000	38	26.3	4.52	8029	6761.1	2328.3	2475.8	2.90
N1	Mixed Patterns	200	14	14.3	1.77	1627.6	1343.1	522.3	552.1	2.57
N2	Mixed Patterns	400	13	30.8	2.46	3084.8	2691.2	937.1	1001.2	2.87
N3	Ellipsoid	600	20	30	3.22	4719.2	4018	1419.5	1514.8	2.83
N4	Ellipsoid	800	21	38.1	3.9	6220.5	5360.7	1915.7	2047.2	2.80
N5	Ellipsoid	1000	22	45.5	4.52	7724.3	6703.6	2363.5	2530.2	2.84

Table 1. CPU times for segmental triangulation

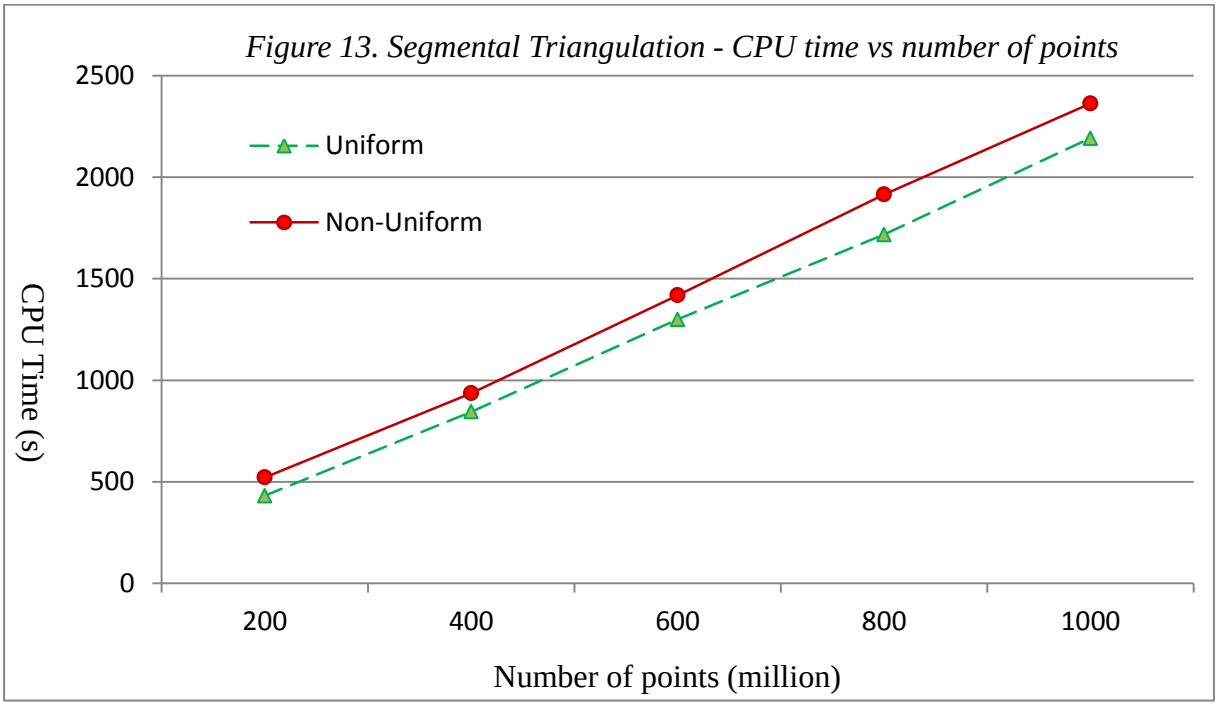
NP = Number of points in million
Step = Number of steps in the segmental triangulation process
NI = Number of points in million input for each insertion step
NB = Number of point in million in the buffer zone
NT = Total number of tetrahedra in million created in the segmental insertion process
NT* = Number of Delaunay tetrahedra in million retained in the convex hull
T1 = CPU time for segmental triangulation
T2 = Total CPU time including data input/output, grid construction, etc.
$NT^{\#}$ = Number of Delaunay tetrahedra in million constructed per second

With 16GB RAM, the maximum number of points that could be handled is about 60 million, which was employed to triangulate 800 million and 1 billion points in 17 and 22 steps respectively. Fewer points of 40 million (4% of 1 billion) for each insertion step were also tested, and the number of steps increased from 22 to 38. However, the number of tetrahedral constructed per second only dropped slightly from 3.08 to 2.9 million, showing the overhead for handling the overlapping zones between two insertion zones was not significant. Indeed,

the speed of segmental Delaunay triangulation is pretty fast using just 10% memory for each insertion step, and 3.2 million tetrahedral per second can be constructed for the insertion of 400 million points. As shown in Figure 13, this rate is quite linear with respect to the number of points in the set, which is however the expected result of the segmental insertion process as the work for each insertion step is fairly similar and probably would take just equal amount of CPU time.

As for the non-uniform point distributions, as long as the loading could be evenly distributed and points are not highly concentrated in a tiny volume, the performance of the segmental insertion scheme is only slightly lower compared to random point distributions, at a speed of creating about 2.8 million tetrahedra per second. For instance, for the triangulation of 200 and 400 million points of mixed patterns of non-uniform distributions, i.e. line, ellipsoid and spiral distribution patterns (with 1% spread relative to the large dimension) were repeatedly triangulated by a segmental insertion process as shown in Figure 14.

Figure 13. Segmental Triangulation - CPU time vs number of points



The slow down for the segmental triangulation of non-uniformly distributed points was probably due to two reasons; (i) slightly more CPU time was needed for the triangulation of non-uniformly distributed points, and (ii) a slightly larger overlapping zone had to be allowed between two insertion zones. For the segmental triangulation of 600, 800 and 1000 million non-uniformly distributed points, the distribution over an ellipsoidal surface was tested. With an almost balanced load for each processor and locally uniformly distributed points, the triangulation times taken were very similar to those of the randomly distributed points.

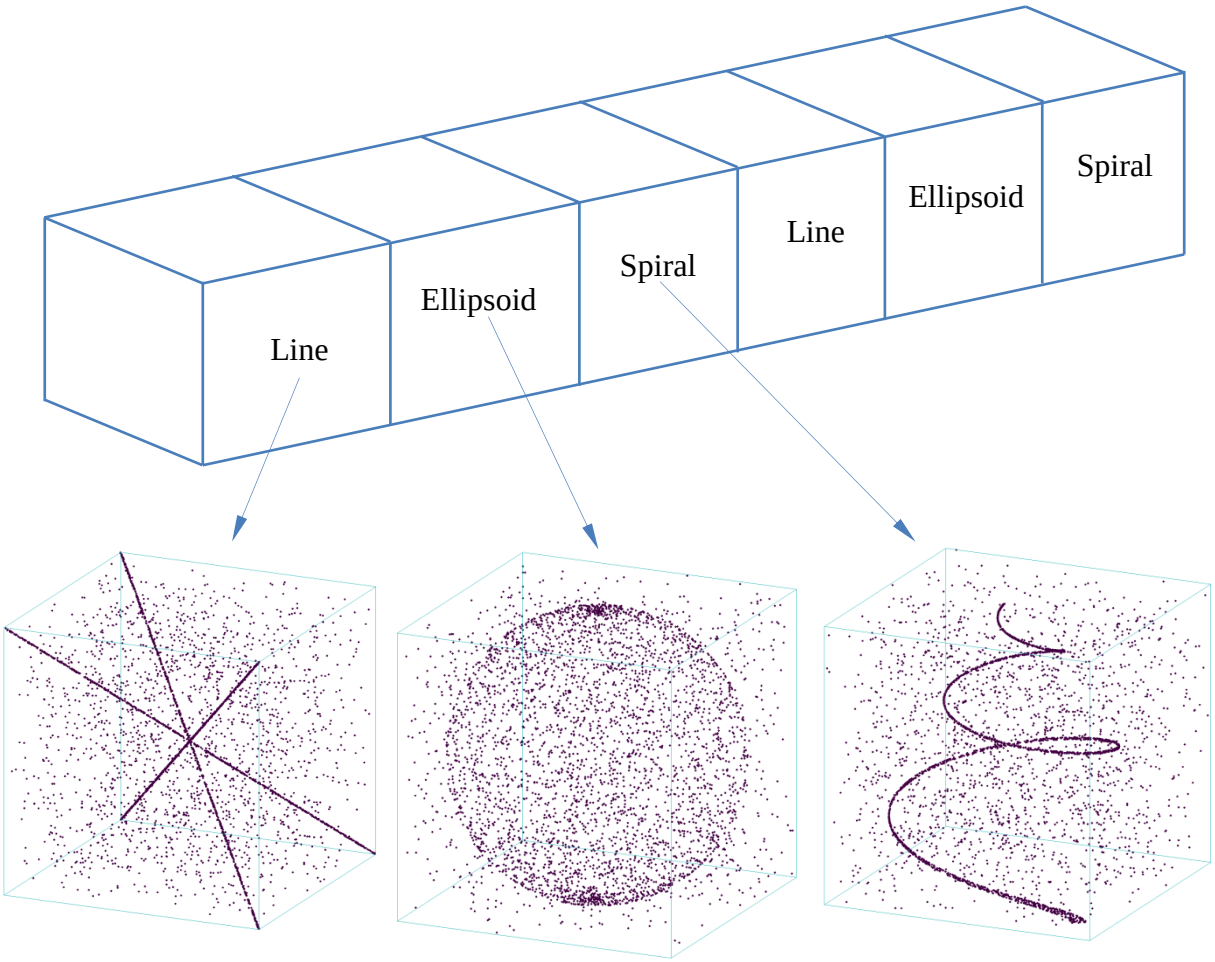


Figure 14. Non-uniform point distribution of mixed patterns

6 CONCLUSIONS AND DISCUSSIONS

A segmental point insertion method has been proposed for the Delaunay triangulation of exceptionally large point sets of more than 1 billion points on a PC in three dimensions. The points to be triangulated are partitioned into zones (segments), and points in each zone are triangulated by a parallel insertion module, step by step until all the points in the set are processed. Based on a modulus labeling scheme to access point coordinates and a dynamic update of points stored in cells, in theory, there is no limit to the number of points that can be handled by a machine with limited memory.

As the work of each typical step for the insertion of an equal number of points is very similar, the processing time bears, more or less, a linear CPU time relationship with the number of points in the set, with a construction rate of about 3.1 million Delaunay tetrahedra per second for randomly generated points. 3.1 million Delaunay tetrahedra is based on earlier test runs done on a relatively old PC, and later tests indicated that more than 5 million tetrahedra per second could be achieved in a faster PC, Intel Core i7-4770 @ 3.4GHz with 32GB RAM. The performance of the segmental insertion scheme has also been tested on non-uniformly

distributed points. Provided that a balanced load could be set up for the parallel insertion kernel, the performance is only slightly lower compared to that of randomly distributed points at about 2.8 million tetrahedra per second. More complicated segmental partition into zones can also be considered, instead of a line subdivision, the points set can as well be partitioned by a block subdivision scheme into $n \times n \times n$ zones to reduce common inter-zonal boundary. However, this may not be more effective, as synchronization in the use of memory is bound to be more complicated for insertion zones having common boundary with many other adjacent zones.

In theory, there is no limit to the number of points in the triangulation. However, there are two bottlenecks in the practical implementation. The first concern is whether the current Windows operating system and the available C++/FORTRAN programming language can effectively handle memory addresses more than 32GB RAM. The second concern is that for the present computer code 4-byte integers have been used, whose dynamic range is ± 2147483648, and dividing this by 4, the number of tetrahedral elements in a zone is limited to 536870912. In other words, using 4-byte integers, the array indices have to be carefully controlled to avoid overflow of indices for handling large array of tetrahedra, their neighbours and vertices.

ACKNOWLEDGEMENTS

The Area of Excellence Scheme "Theory, Modeling and Simulation on Emerging Electronics" granted by UGC, Hong Kong SAR, is gratefully acknowledged. The author would also like to thank Ms. Lillian Chan of the Computer Centre, the University of Hong Kong, for her help in the parallel computing of the examples, which was conducted using the HKU Computer Centre research computing facilities that are supported in part by the Hong Kong UGC Special Equipment Grant (SEG HKU09).

REFERENCES

[1] N Amenta and M. Bern, *"Surface reconstruction by Voronoi filtering"*, Discrete Computational Geometry, **22 4** (1999) 481-504

[2] Hang Si, *"Constrained Delaunay tetrahedral mesh generation and refinement"*, Finite Elements in Analysis and Design, **46** (2010) 33-46

[3] H. Borouchaki and S.H. Lo, *"Fast Delaunay triangulation in three dimensions"*, Comput. Methods Appl. Mech. Engrg. **128** (1995) 153-167

[4] S.H. Lo and W.X. Wang, *"Generation of tetrahedral mesh of variable element size by sphere packing over an unbounded 3D domain"*, Computer Methods in Applied Mechanics and Engineering, Volume **194**, Issues 48-49 , 15 November 2005, 5002-5018

[5] H. Borouchaki, P. L. George, and S. H. Lo, "*Optimal Delaunay point insertion"*, Int. J. for Num. Methods in Engg., **39**, 3407-3437 (1996)

[6] Olivier Devillers and Monique Teillaud, *"Perturbations for Delaunay and weighted Delaunay 3D triangulations"*, Computational Geometry, **44** (2011) 160-168

[7] J.D. Boissonnat, M. Sharir, B. Tagansky, M. Yvinec, *"Voronoi diagram in higher dimensions under certain polyhedral distance functions"*, Discrete Comput. Geom. **19**:485-519 (1998)

[8] G.E. Blelloch, J.C. Hardwick, G.L. Miller and D. Talmor, *"Design and implementation of a practical parallel Delaunay algorithm"*, Algorithmica (1999) **24**: 243-269

[9] Ivana Kolingerova, Josef Kohout, *"Optimistic parallel Delaunay triangulation"*, Visual Computer (2002) **18**:511-529

[10] Min-Bin Chen, Tyng-Ruey Chuang and Jan-Jan Wu, *"Efficient parallel implementations of near Delaunay triangulation with high performance Fortran"*, Concurrency Computation: Practice and Experience, 2004; **16**: 1143-1159

[11] Demian Nave, Nikos Chrisochoides, L. Paul Chew, *"Guaranteed-quality parallel Delaunay refinement for restricted polygonal domains"*, Computational geometry **28** (2004) 191-215

[12] Min-Bin Chen, Tyng-Ruey Chuang and Jan-Jan Wu, *"Parallel divide-and-conquer scheme for 2D Delaunay triangulation"*, Concurrency and Computation: Practice and Experience, 2006, **18**: 1595-1612

[13] Huayi Wu, Xuefeng Guan, Jianya Gong, *"ParaStream: A parallel Delaunay triangulation algorithm for LiDAR points on multicore architectures"*, Computers & geosciences **37** (2011) 1355-1363

[14] Nikos Chrisochoides and Demian Nave, *"Parallel Delaunay mesh generation kernel"*, Inter. J. Numer. Methods Engrg., 2003; **58**: 161-176

[15] Andrey N. Chernikov, Nikos P. Chrisochoides, *"A template for developing next generation parallel Delaunay refinement methods"*, Finite Elements in Analysis and Design **46** (2010) 96-113

[16] Josef Kohout, Ivana Kolingerova, *"Parallel Delaunay triangulation in E^3: make it simple"*, Visual Computer (2003) **19**:532-548

[17] Josef Kohout, Ivana Kolingerova, Jiri Zara, *"Parallel Delaunay triangulation in E^2 and E^3 for computers with shared memory"*, Parallel Computing **31** (2005) 491-522

[18] Tilo Beyer, Gernot Schaller, Andreas Deutsch, Michael Meyer-Hermann, *"Parallel dynamic and kinetic regular triangulation in three dimensions"*, Computer Physics Communications **172** (2005) 86-108

[19] Vicente H.F. Bastista, David L. Millman, Sylvain Pion, Johannes Singler, *"Parallel geometric algorithms for multi-core computers"*, Computational Geometry **43** (2010) 663-677

[20] YingLiang Ma, *"A Parallel surface extraction algorithm for large binary image data sets based on an adaptive 3D Delaunay subdivision strategy"*, IEEE Transaction and visualization and Computer Graphics, Vol. **14**, No. 1, Jan/Feb 2008, 160-172

[21] S.H. Lo, "*Parallel Delaunay Triangulation in three dimensions*", Comput. Methods Appl. Mech. Engrg., **237** (2012) 88-106

[22] B. Delaunay, *"Sur la sphere vide"*, Bull. Acad. Sci. URSS, Class. Sc. Nat., 793-800, 1934

[23] Lemaire C and Moreau JM (2000), "*A probabilistic result on multi-dimensional Delaunay triangulations, and its application to the 2D case*", Computational Geometry, **17**, 69-96

[24] S.H. Lo, "*3D Delaunay triangulation of non-uniform point distributions*", Finite Elements Analysis and Design, **90** (2014) 113-130