International Journal of Informatics and Applied Mathematics
e-ISSN:2667-6990 Vol. 7, No. 2, 71-85

Parallel Computing for 3D Delaunay Triangulation of Non-Uniform Cloud Points

Zahida Tchantchane, Mohamed Bey, and Khadidja Bouhadja

Centre de Développement des Technologies Avancées (CDTA), Baba Hassen, Algiers, Algeria.
ztchantchane@cdta.dz mbey@cdta.dz kbouhadja@cdta.dz

Abstract. 3D acquisition technologies have favored the development of geometric modelling of 3D objects based on data from their digitization. The aim is to use the Delaunay triangulation (DT) approach to generate a digital model of the external surfaces of a physical object from point clouds. The generation of a DT from a non-uniform point clouds is an arduous and time-consuming task. Moreover, point clouds are very large and computationally intensive, which increases processing time and costs, especially if only one processor is used. The fastest DT algorithm is based on the divide-and-conquer, which is generally designed to be used for parallelism. This algorithm is carried out in two steps. The first step recursively partitions the points set into sub-regions; each is assigned to a processor. Independently, these regions are further triangulated simultaneously. The second step merges the sub-regions into the final mesh, which is applied in the reverse order of points set partitioning. This work deals with the generation of a 3D triangulation from any point cloud, which is partitioned to several sub-points using cells. Independently, the sub points are further triangulated simultaneously by parallelizing the calculations on several processors. After that, an allocated area of each cell is determined, as well as the strategy for the fusion. Finally, this solution is tested and validated through many unstructured point clouds.

Keywords: Cloud of Points · Cells · 3D Delaunay Triangulation · Merging · Parallel Computing.

Received: 16 December 2024 | Revised: 20 December 2024 | Accepted: 24 January 2025
DOI: 10.53508/ijiam.1602342

1 Introduction

With the technological evolution in different industries such as automotive, robotics or medical, the need to produce more complex objects with more details and a higher degree of precision is always present. Depending on their geometric complexity, these objects are designed according to two approaches: the so-called classical one, based on the modelling of the object according to a specification using design and manipulation functions offered by computer-aided design (CAD) software. The disadvantage of this approach lies in the difficulty of obtaining the desired shapes without resorting to successive modification and validation phases. This leads to very long processing times. The other approach, known as Reverse Engineering, is used when the shapes of the objects are very complex and cannot be designed in CAD software, or if the object description model is unavailable. It is based on the modelling of the object from point clouds. This approach has become a very common practice in the industrial world now days, as it allows to reduce the development time of new products from prototypes.

Object reconstruction is a relatively young discipline. It allows the conversion of the point clouds representing the object into simple geometric elements such as triangles, tetrahedra, etc. Several methods have been developed for object reconstruction from point clouds. Rego [1] and Kazhdan [2] mention that the point cloud plays a very important role in the choice of the reconstruction method. It can be a uniform or non-uniform point cloud. In [3], a comparative study between several reconstruction algorithms was presented by taking into account several criteria: algorithm complexity, point cloud, surface topology, etc., the authors chose the Delaunay triangulation (DT) as the most used approach. As a result, many techniques for the generation of DT have been developed, the incremental insertion method presented by [4] and [5], the Sweep-line method proposed by [6] and [7] and the Divide and Conquer method first presented by [8]. Subsequently improved in [9] and [10]. The only drawback of these techniques is the long processing time. With the advent of multi-core machines, the construction time can be reduced by using more than one processor. Several parallel algorithms for DT construction have been proposed to improve the performance and overcome the drawbacks of existing techniques.

In [11], the point cloud is decomposed into a single dimension, and then each processor performs sub-cloud triangulation and boundary fusion. The authors in [12] propose an affected area solution. It consists in determining the area that can be modified when merging two sub-triangulations to reduce communication between processors. Since point clouds are often very large, the authors in [13] present an algorithm called Para Stream that uses k-tree to distribute the task of triangulation and merging over the processor cores to improve the load balance. Also, the authors in [14,15] present the generation of the parallel DT based on the partition of the points into zones. The points are first divided into cells of approximately equal number of points. The cells are grouped into zones, in which the DT is constructed by simultaneously inserting points cell by cell in each zone. Some techniques have been merged and presented as a hybrid method [16]. Authors in [16] used a double divide and conquer to increase the efficiency when

processing a dense point cloud. The obtained results show that the proposed parallel algorithm is efficient in constructing the DT with good speedup.

Point clouds are often very large to be processed with a single processor, this situation increases the processing time and consequently the cost. Based on this analysis, this paper proposes a methodology for the generation of the DT that allows the reduction of the processing time by parallelizing the computations on several processors or cores. So, given an unstructured point cloud resulting from the digitization of an object, an approach based on the principle of DT is pro posed. In this approach, the point cloud representing the skin of the object is subdivided into cells along three directions (X-axis, Y-axis, and Z-axis) by introducing the number of cell on each direction. Then, the cells are assigned to the different processors. Subsequently, the DT of the cells is generated in parallel. Finally, the assigned areas of each cell are generated to reduce the communication between the processors, and the fusion strategy is determined as well.

2 Proposed Approach

The general architecture of the proposed approach is composed of three steps (Figure 1). The first step takes as input the point cloud to generate cells, then the points are assigned to the corresponding cells and then the cells are assigned to the processors. The second step is to generate the DT of the cells in parallel. At the end, the last step involves the definition of the assigned area of each cell, followed by the merging between cells according to a well-defined order.

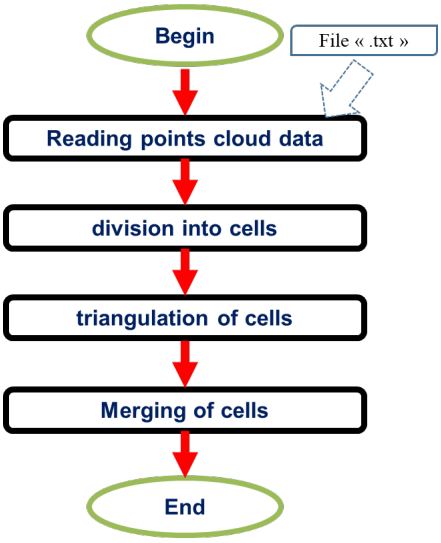
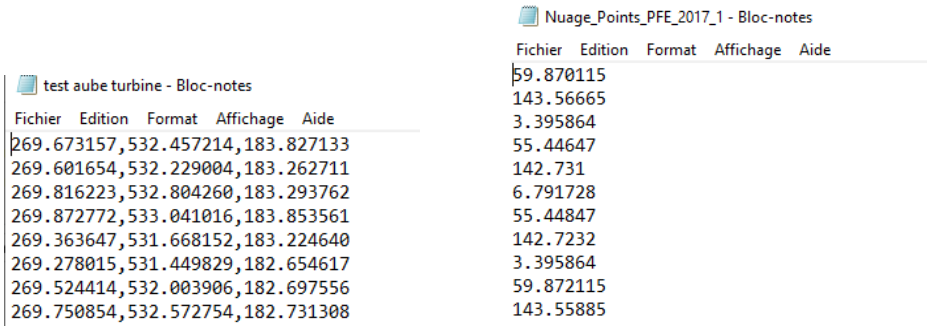


Fig. 1: Proposed approach

2.1 Reading the point cloud

After the acquisition of the file containing the points cloud representing the object, the file is verified syntactically and semantically. Figure 2 shows two types of file that can be read in this work.



(a) Three coordinates per line. (b) One coordinate per line.

Fig. 2: Types of file.

The algorithm in figure show the steps to verify the conformity of the file (see Figure 2). First, the file name is retrieved; it must contain the '.txt' extension. Then, the number of lines or the number of coordinates are read. It must be found either the number of lines or the number of coordinates multiple of number three. Otherwise, the file lacks information and therefore, I inform the user by message that the data file lacks information. If this file is correct, all the points will be stored in an array of point structure.

2.2 Partition of point cloud using cells

After reading the file, the subdivision of the point cloud into cells is carried out in two steps:

1. First step: consists in creating cells of the same size.
2. Second step: assign each point to the appropriate cell.

Creating cells of the same size To achieve this goal, the raw area surrounding the point cloud is subdivided into cells of equal size using Algorithm 1 illustrated below.

Once the user has saved the points in a three-dimensional array in the reading step, the user might enter the number of cells on each axis (nx, ny, and nz). Next, the limits of the raw part are calculated, which are: x_{max} and x_{min} on the x axis, y_{min} and y_{max} on the y axis, and z_{max} and z_{min} on the z axis. From these values of limits, the length, width, and height dimensions are calculated.

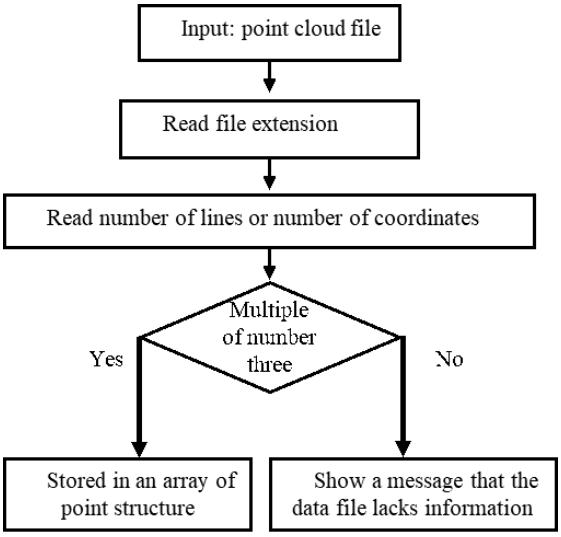


Fig. 3: Check file conformity algorithm.

Afterward, the dimensions of the cell ($step_x$, $step_y$, $step_z$) are calculated using these formulas:

$$step_x = \frac{length}{nx} \tag{1}$$

$$step_y = \frac{width}{ny} \tag{2}$$

$$step_z = \frac{height}{nz} \tag{3}$$

With nx, ny, and nz being the number of cells in the x, y, and z directions respectively. Finally, a 3D array of cells called `tab_c` is created, representing all the cells created.

Algorithm 1 Subdivision of the raw part into cells

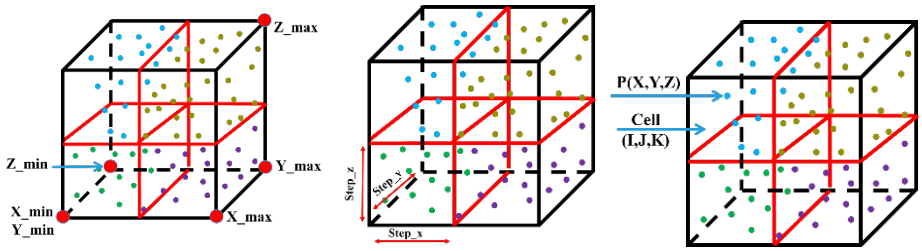
Require: Point cloud, number of cells n_x, n_y, n_z
Ensure: Set of cells Tab_C $(C_1, C_2, C_3, \ldots, C_k)$
1: Calculate the limits of the raw part: $x_{\min}$, $x_{\max}$, $y_{\min}$, $y_{\max}$, $z_{\min}$, $z_{\max}$
2: Calculate the dimensions of the raw part: *length*, *width*, and *height*
3: Calculate the dimensions of a cell: $step_x = \frac{\text{length}}{n_x}$, $step_y = \frac{\text{width}}{n_y}$, $step_z = \frac{\text{height}}{n_z}$
4: Create a three-dimensional array of cells `Tab_C`

Assigning points to cell To identify the cell for each point, three indices I, J and K that define each cell must be calculated. The formulae below allow each point to be assigned directly to the appropriate cell without having to test these three coordinates (Fig. 4). Where x represent the point coordinate along the x axis, x_{min} represent minimum raw limit and $step_x$ represent cell size along the x axis. From these values, the index I is calculated using formula (4). The same thing is repeated for two indices J and K representing the Y and Z axes respectively. Therefore, the point $P(x, y, z)$ is assigned to the cell (I, J, K)

$$I = \frac{x - x_{min}}{step_x} \tag{4}$$

$$J = \frac{y - y_{min}}{step_y} \tag{5}$$

$$K = \frac{z - z_{min}}{step_z} \tag{6}$$



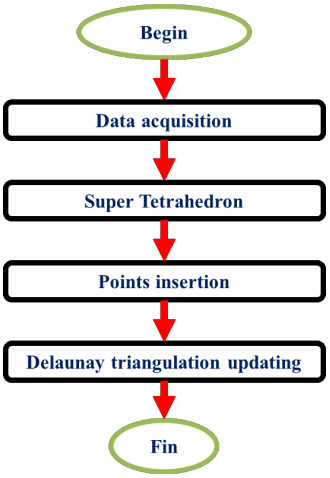
(a) Limits (b) Cell dimensions (c) Affected points.

Fig. 4: Assigning points to cells.

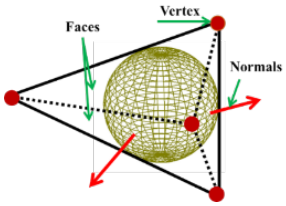
2.3 Parallel Delaunay triangulation of cells

3D Delaunay triangulation generation steps It consists of creating tetrahedrons from the unstructured point cloud. The process of triangulation is illustrated in Figure 5.

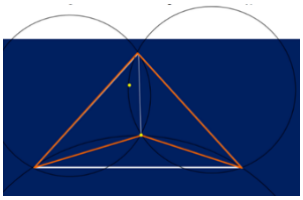
Parallel Delaunay triangulation In this step, the DT is called as many times as necessary, depending on the number of cells. This call is made automatically to generate DTs in parallel. This step is illustrated by an example of two cells along each axis, resulting in eight (08) cells (Figure 6).



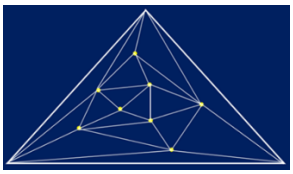
(a) Limits



(b) Cell dimensions



(c) Affected points.



(d) Affected points.

Fig. 5: Assigning points to cells.

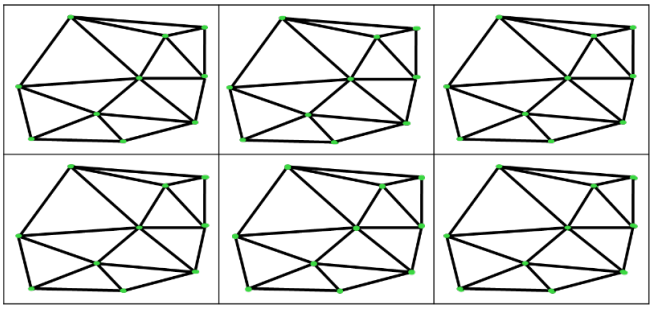


Fig. 6: Parallel Delaunay triangulation of eight cells.

2.4 Merging sub-triangulation

Affected zone The affected area is the region of the DT of each cell that is likely to be modified when the triangulations are merged. These regions represent the portions of the adjoining areas between the different cells along the

three directions X, Y and Z. To avoid processing all the tetrahedra of the cells, only the tetrahedra whose circumscribed sphere contacts the boundaries of the neighboring cells, along the three directions X, Y and Z, will be considered during processing (Figure 7). For each cell, six (06) affected zones must be determined; two (02) along each axis. The determined areas are: *right affected area*, *left affected area*, *top affected area*, *bot tom affected area*, *front affected area*, and *back affected area*.

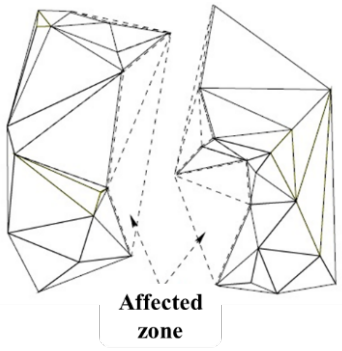
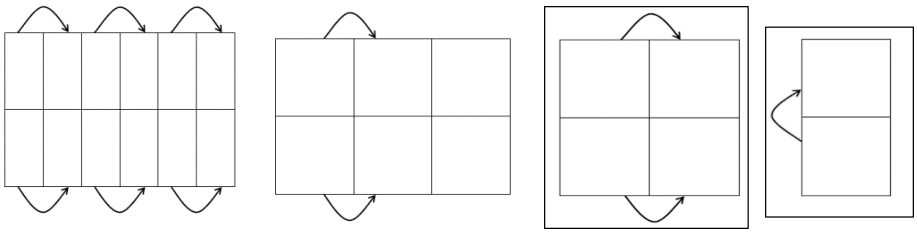


Fig. 7: Affected zone.

Cells triangulations merging Merging is the last phase after subdividing the point cloud into cells and triangulating the cells into parallels. The cell merging strategy adopted is to concatenate the cell triangulations two by two from the affected areas of adjacent cells. The merging strategy is done in all three directions. Firstly, in the X-axis direction, secondly in the Y-axis direction and thirdly in the Z-axis direction (Figure 8).

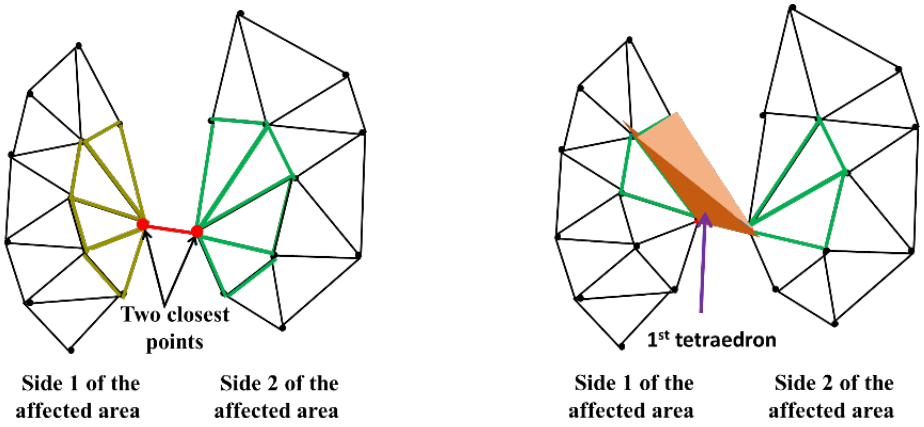


(a) X direction. (b) Y direction. (c) Z direction.

Fig. 8: Three directions of merging.

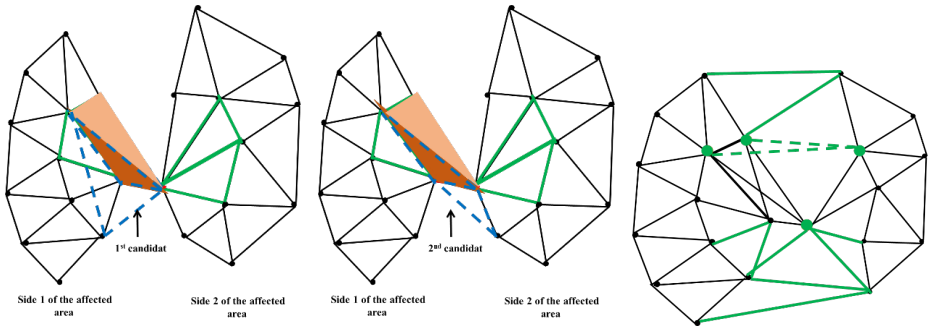
The merging process involves the following steps [17]:

– Generating of the first tetrahedron, which will serve as a support for the generation of the other tetrahedrons (see Figure 9).
– Generating a generic tetrahedron (see Figure 10).



(a) Two closest points. (b) First tetrahedron.

Fig. 9: Generating of the first tetrahedron.



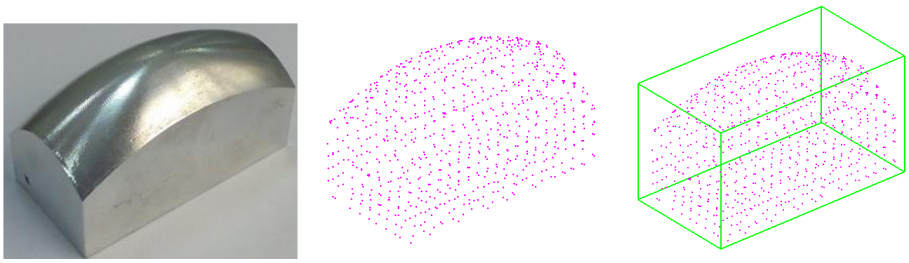
(a) First candidate. (b) Second candidate. (c) Full merging.

Fig. 10: Generating of the generic tetrahedron.

3 Results and Discussion

The proposed approach is implemented in object-oriented mode under Windows using the C++ Builder language and the OpenGL graphics library. The approach is validated on the test part (convex shape). Figure 11a and Figure 11b

shows the acquired points. The first step is to read the point cloud, calculate the boundaries of the raw part and its dimensions (length, width, and height). During the reading step, the number of point represented the convex part is equal to eight hundred and nine (809) and the overall dimensions of the blank are 90.49 mm×49.68 mm×47.74 mm.

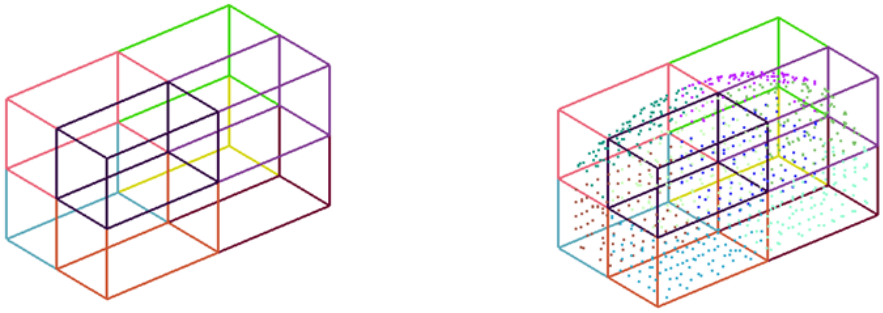


(a) Convex part. (b) Cloud of points. (c) Raw part dimensions.

Fig. 11: Test part (Convex shape).

3.1 Obtained results

The subdivision of the raw part is carried out as follows: the number of cells in all directions (X, Y and Z) is chosen equal to two (2). Then, a three-dimensional array of eight cells is generated. Figure 12 shows the generated cells and their points. Table 3 gives the number of points in each cell.



(a) Cells creation. (b) Affected points to theirs cells.

Fig. 12: Cells and their points.

Table 1: Number of points for each cell

Cell index	**Number of point**
Cell **0_0_0**	89
Cell **0_1_0**	108
Cell **0_0_1**	88
Cell **1_0_0**	93
Cell **1_1_0**	108
Cell **1_0_1**	92
Cell **0_1_1**	118
Cell **1_1_1**	113

From Table 3, the minimum and the maximum number of points in cells is equal to eighty-eight and one hundred and eighteen respectively. Therefore, the parameter Δpoint, which represents the difference between the maximum and minimum number of points in cells (Δpoint=118-88) is equal to thirty (30). This means that a little difference exists between cells, and therefore the number of points per cell is more or less homogeneous.

To demonstrate this performance, the number of points assigned for cell is checked against the average number of points. In this case, the average number of points inside a partition is 101 points (809 points on 8 clusters). Using the following equation (7), the obtained values of V are shown in Table 2.

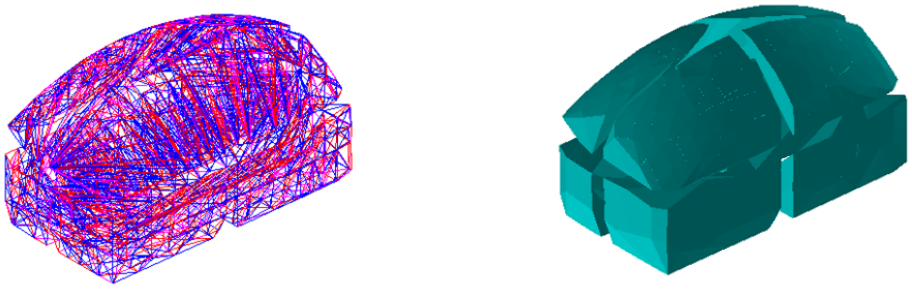
$$V_i = |\text{Number of assigned ponts to } cell_i - \text{Avrage number of points}| \quad (7)$$

Table 2: Obtained values of V (*****: denotes the minimum, ******: denotes the maximum)

Cell/number	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	**Sum**
number of points	89	108	88	93	108	92	118	113	809
number of points	-12*****	69	-13	-8	7	-9	17******	12	1

Table 2 shows that the maximum values of V is 17. Regarding the minimum values of V, the minimum values of V is -12. The last column of Table 2 shows that the sum of the values V is equal to one ($Sum = 1$). This affirms better homogeneous performances. However, this number can vary depending on the size and geometry of the object under consideration.

Once the point cells have been generated, the DT is run in parallel on a PC with two cores. The triangulation of the cells is shown in Figure 13. At the end, the process of generating the affected area generates six affected areas, and they are merged two by two. Table 3 shows the results obtained from this last step.



(a) Mesh sub-triangulation. (b) Rendered sub-triangulation.

Fig. 13: Sub-triangulation of eight cells.

Table 3: Number of points for each cell

Cell index	**Affected zones**	**Number of affected tetrahedra**
Cell **0_0_0**	Right	44
Cell **0_1_0**	Left	69
Cell **0_0_1**	Top	103
Cell **1_0_0**	Bottom	196
Cell **1_1_0**	Front	114
Cell **1_0_1**	Back	149
Cell **0_1_1**	Right	44
Cell **1_1_1**	Left	69

3.2 Comparative study

Table 4 illustrates the times of the DT for the three cases: global DT, Subdivision into eight cells in sequential mode, and subdivision into eight cells in parallel mode. For Global without subdivision, significant times are consumed compared with subdivision with sequential mode and parallel mode. The greatest improvement is also seen in parallel mode, where DT times are the lowest compared with the other modes. Hence, the interest of the parallel calculation.

Table 4: Number of points for each cell

Triangulation mode	**Running time**
Global without subdivision	1h 04min
Subdivision into eight cells in sequential mode	2min 13s
Subdivision into eight cells in parallel mode	1min 12s

For further validation of the proposed approach, three (03) PCs are used:

- PC 1: ACER Laptop, Windows 7 Intel® Core™ I3–3217U 1.8, 4Go de RAM.
- PC 2: HP Laptop, Windows 10, Intel® Core™ I5-4200 CPU @1.60 GHz 2.30 GHz, 4Go de RAM.
- PC3: PC Intel, Windows 7 Intel® Core i7-13700K, 6Go de RAM.

To show the influence of the number of processors in parallel computing, a comparison on the same example with 809 points is done (Table 5).

Table 5 illustrates the times of the DT using three computers PC1, PC2 and PC3 with two, four and six cores numbers respectively. The last line shows the percentage gain, which represents the difference between sequential and parallel execution times for each PC. For the PC1, the percentage gain is equal to 35%, for PC2 and PC3, the percentage gain is equal to 38% and 19% respectively. From this, the influence of the number of processors is significant.

Table 5: Execution time vs. number of cores.

Cores Number	**2**	**4**	**6**
Points Number	809	809	809
Cell Number	2	2	2
Sequential Running time	1h 48min	38min 29s	22min
Parallel Running time	1h 10min	24min 07s	17min 55s
Gain	35%	38%	19%

A second evaluation was performed. This evaluation focused on how the number of cells affects the performance of a parallel computing system. The findings of this evaluation are presented in Table 6.

Table 6: Number of points for each cell

Cores Number	**4**	**4**	**4**
Points Number	809	809	809
Cell Number	1	2	3
Sequential Running time	38min 29s	11min 12s	7min 51s
Parallel Running time	24min 07s	8min 02s	5min 06s
Gain	38%	29%	28%

Table 6 illustrates the times of the DT using PC2 computer with four cores numbers. Three cases are considered with three value of number of cell 1, 2 and 3. The last line shows the percentage gain, which represents the difference between sequential and parallel execution times for the case. From this, the influence of the number of cell is very significant

From the result obtained from Table 5 and Table 6, a very important difference be tween the execution times is observed. The higher the number of cells or the number of processors, the lower the execution time. The user make a good choice of the number of cells in order to obtain a considerable reduction of the processing time. As a result, it can ensure a balanced workload at different processors.

4 Conclusion

This paper proposes and implements an approach to parallelize DT computations. The approach uses as input a non-uniform point cloud obtained by digitizing an object in three main steps. The first step is to read the point cloud, then generate cells and assign the points to the corresponding cells. The second step is to allocate the cells to the processors and then to generate the DT of the cells in parallel. Finally, the last step consists in defining the assigned area of each cell, followed by the fusion of the cells two by two in three dimensions. The approach has been validated on a real part and the obtained results are encouraging, as the processing time is reduced as the number of cells and the number of cores or processors are increased. In the future, it is planned to apply this approach on a GPU.

References

1. Renata LME do Rego and Aluizio FR Araujo. A surface reconstruction method based on self-organizing maps and intrinsic delaunay triangulation. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2010.
2. Michael Kazhdan and Hugues Hoppe. Screened poisson surface reconstruction. *ACM Transactions on Graphics (ToG)*, 32(3):1–13, 2013.
3. Kavita Khanna and Navin Rajpal. Survey of curve and surface reconstruction algorithms from a set of unorganized points. In *Proceedings of the Third International Conference on Soft Computing for Problem Solving: SocProS 2013, Volume 1*, pages 451–458. Springer, 2014.
4. Charles L Lawson. Transforming triangulations. *Discrete mathematics*, 3(4):365–372, 1972.
5. NA Golias and RW Dutton. Delaunay triangulation and 3d adaptive mesh generation. *Finite elements in analysis and design*, 25(3-4):331–341, 1997.
6. L Yonghe, F Jinming, and S Yuehong. A simple sweep-line delaunay triangulation algorithm. *J. Algorithms Optim*, 1:30–38, 2013.
7. Peter Su and Robert L Scot Drysdale. A comparison of sequential delaunay triangulation algorithms. In *Proceedings of the eleventh annual symposium on Computational geometry*, pages 61–70, 1995.
8. Michael Ian Shamos and Dan Hoey. Closest-point problems. In *16th Annual Symposium on Foundations of Computer Science (sfcs 1975)*, pages 151–162. IEEE, 1975.
9. Jyrki Katajainen and Markku Koppinen. Constructing delaunay triangulations by merging buckets in quad tree order. *Fundamenta Informaticae*, 11(3):275–288, 1988.

10. Paolo Cignoni, Claudio Montani, and Roberto Scopigno. Dewall: A fast divide and conquer delaunay triangulation algorithm in ed. *Computer-Aided Design*, 30(5):333–341, 1998.
11. Min-Bin Chen, Tyng-Ruey Chuang, and Jan-Jan Wu. Efficient parallel implementations of near delaunay triangulation with high performance fortran. *Concurrency and Computation: Practice and Experience*, 16(12):1143–1159, 2004.
12. Min-Bin Chen. A parallel 3d delaunay triangulation method. In *2011 IEEE Ninth International Symposium on Parallel and Distributed Processing with Applications*, pages 52–56. IEEE, 2011.
13. Huayi Wu, Xuefeng Guan, and Jianya Gong. Parastream: A parallel streaming delaunay triangulation algorithm for lidar points on multicore architectures. *Computers & geosciences*, 37(9):1355–1363, 2011.
14. SH Lo. Parallel delaunay triangulation in three dimensions. *Computer Methods in Applied Mechanics and Engineering*, 237:88–106, 2012.
15. Tchantchane Zahida, Khadidja Bouhadja, Ouahiba Azouaoui, and Nassira Ghoualmi-Zine. Enhanced unstructured points cloud subdivision applied for parallel delaunay triangulation. *Cluster Computing*, 26(3):1877–1889, 2023.
16. Wenzhou Wu, Yikang Rui, Fenzhen Su, Liang Cheng, and Jiechen Wang. Novel parallel algorithm for constructing delaunay triangulation based on a twofold-divide-and-conquer scheme. *GIScience & Remote Sensing*, 51(5):537–554, 2014.
17. Min-Bin Chen. The merge phase of parallel divide-and-conquer scheme for 3d delaunay triangulation. In *International Symposium on Parallel and Distributed Processing with Applications*, pages 224–230. IEEE, 2010.