Recibido: 25/5/2022 / Aceptado: 8/6/2022
doi: https://doi.org/10.26439/interfases2022.n015.5886

A DISTRIBUTED MODEL FOR COMPUTING 3D MESH LOCAL DESCRIPTORS BASED ON *K*-RINGS

FRANCI SUNI-LOPEZ
fsunilo@unsa.edu.pe / ORCID: 0000-0002-4212-7910
Universidad Nacional de San Agustín de Arequipa, Peru

JAN HURTADO
jhurtadoj@unsa.edu.pe / ORCID: 0000-0003-3422-3117
Universidad Nacional de San Agustín de Arequipa, Peru

ALEJANDRA MÁRQUEZ
amarquezhe@unsa.edu.pe
Universidad Nacional de San Agustín de Arequipa, Peru

LEONARDO GUZMÁN
lguzmanz@unsa.edu.pe
Universidad Nacional de San Agustín de Arequipa, Peru

Abstract

In order to facilitate 3D object processing, it is common to use high-level representations such as local descriptors that are usually computed using defined neighborhoods. K-rings, a technique to define them, is widely used by several methods. In this work, we propose a model for the distributed computation of local descriptors over 3D triangular meshes, using the concept of *k*-rings. In our experiments, we measure the performance of our model on huge meshes, evaluating the speedup, the scalability, and the descriptor computation time. We show the optimal configuration of our model for the cluster we implemented and the linear growth of computation time regarding the mesh size and the number of rings. We used the Harris response, which describes the saliency of the object, for our tests.

KEYWORDS: 3D local descriptor / geometry processing / distributed computing / large meshes

Un modelo distribuido para calcular descriptores locales de malla 3D basados en *k-rings*

Resumen

Para facilitar el procesamiento de objetos 3D, es común utilizar representaciones de alto nivel, como los descriptores locales que generalmente se calculan utilizando vecindarios definidos. *K-rings* es una técnica para definirlos y es ampliamente utilizada por varios métodos. En este trabajo, proponemos un modelo para el cálculo distribuido de descriptores locales sobre mallas triangulares 3D, utilizando el concepto de anillos *k*. En nuestros experimentos, medimos el rendimiento de nuestro modelo en mallas enormes, evaluando la aceleración, la escalabilidad y el tiempo de cálculo del descriptor. Mostramos la configuración óptima de nuestro modelo para el clúster que implementamos y el crecimiento lineal del tiempo de cálculo con respecto al tamaño de la malla y el número de anillos. Usamos la respuesta de Harris, que describe la prominencia del objeto, para nuestras pruebas.

PALABRAS CLAVE: descriptor local 3D / procesamiento geométrico / computación distribuida / mallas grandes

1. INTRODUCTION

3D repositories are growing rapidly (Gao et al., 2015) and, consequently, there are many techniques for 3D data acquisition (e.g. 3D modeling, 3D scanners, 3D reconstruction, depth-sensing cameras, etc.). In this context, processing this type of data is an important task for computer scientists. For instance, areas such as videogames, medicine, archeology, biology, engineering, physics, and others use this kind of data for real object representation. There are some tasks that are very important for 3D data analysis: registering (Pavlakos et al., 2018; Maquart et al., 2021; O' Sullivan et al., 2022; Van Kaick et al., 2011; Li et al., 2015), symmetry detection (Mitra et al., 2013; Areias & Rabczuk, 2017), segmentation (Figueiredo et al., 2021; Zhou et al., 2017), sampling and compression. These tasks depend on the representation of the object; a 3D object can be represented by point clouds, polygon meshes, polygon soups, connectivity graphs, volumetric pixels (voxels), etcetera (low-level representations). To facilitate processing, it is common to use high-level representations such as descriptors, graphs, skeletons, key points, components, etc.

In this paper, we focus on local descriptor representation. A local descriptor is usually defined on each atomic structure of low-level representations; we use polygon mesh representations and their vertices as atomic structures (i.e., we work on triangular meshes). Usually, when computing local descriptors, it is necessary to establish the neighborhood of each vertex of the mesh. This neighborhood allows us to define local features. For example, if we want to define the curvature of a single vertex, it is not necessary to know the distribution of all vertices. Gelfand et al. (2005) proposed the Integral Volume Descriptor, which evaluates the curvature of a region of the surface immersed in a sphere (determines the neighborhood) of a given radius. Lee et al. (2005) proposed a descriptor based on the mean curvature. Gaussian filters are used in different scales to obtain the saliency value of each vertex. They determine neighborhoods using geodesic or Euclidean distance on the mesh. Then, Castellani et al. (2008) applied the Gaussian filter directly to the points of the object instead of the values.

Another way to determine a neighborhood is using the concept of rings. Zaharescu et al. (2009) proposed a descriptor based on the discrete curvature; they defined neighborhoods to obtain the approximate curvature, these neighborhoods were composed by the set of the k-level adjacent vertices of each vertex. Sipiran and Bustos (2011) implemented a technique called Harris 3D, which is based on the corner detection technique proposed by Harris and Stephens (Harris & Stephens, 1988).

They compute the Harris response for each point regarding a locality conformed by different rings; also, they approximate a continuous surface for this locality and transform it using Principal Component Analysis (PCA) to address rotation invariance. The saliency is computed evaluating the derivatives on the surface.

It is possible to generate high definition meshes with millions of vertices. For example, the digital Michelangelo project (Levoy et al., 2000), has meshes with more than 900 million faces. Using a sequential model to process large meshes involves considerable computation time. Multi-thread and GPU based algorithms are very useful when we want to speed up this task. The other problem we want to solve when working with large meshes is the insufficient memory space. Sometimes this problem is addressed using compact data structures for space optimization (Chen et al., 2009; Zamolo et al., 2022; Herath et al., 2020; Cignoni et al., 2003; Aleardi et al., 2005; Gurung et al., 2011; Luffel et al., 2014; Gurung et al., 2013; Rocca et al., 2011), but it decreases processing speed.

Distributed data processing is a computer-networking method in which multiple computers across different locations share computer-processing capability with all connected systems. That is the reason why distributed models have been widely used in different areas of computer science, taking advantage of the reduction of processing time when working on a distributed manner. The distributed paradigm has been addressed in many works concerning image processing (Le Tien et al., 2021; Verma et al., 2018; Squyres et al., 2000; Prajapati & Vij, 2011; Warn et al., 2009), video processing (Pereira et al., 2010), molecular data analysis (Gurung et al., 2011), etc. In the case of 3D data processing, there are multiple frameworks for specific tasks. Balman (2006) proposed a distributed method for non-uniform triangular mesh refinement. Vo et al. (2011) used a MapReduce model for 3D data visualization. Cabiddu and Attene (2015) proposed a web-based system for the generally distributed processing of triangular meshes.

Determining the locality or neighborhood is commonly associated with the k nearest neighbors (kNN) problem. Sankaranarayanan et al. (2007) proposed an extension of their previous work for distributed processing; they used large point clouds for their experiments and calculated the nearest neighbors using Euclidean distance. We can use this to estimate a neighborhood, but the problem of a Euclidean range search is that we can get wrong results if two different parts of the mesh are very close regarding this metric and very far following the topological structure. That is why it is important to define a geodesic metric. The easiest way to discretize it is using adjacency level distance. This will be explained later (k-rings).

In this work, we propose a distributed model for computing 3D triangular mesh local descriptors based on k-rings. We consider a multi-process architecture to address parallel processing and a distributed disk storage for insufficient memory capacity problems. We implemented this model using MPI and applied the Harris response descriptor on it (Sipiran & Bustos, 2011). We conducted several experiments to obtain an optimal configuration for our cluster. Also, we evaluated the speedup, the scalability, and the descriptor computation time.

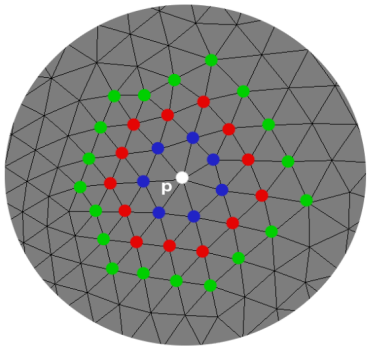
The paper is organized as follows. In section 2, we explain how to represent a 3D object, how to define a descriptor, how to define a neighborhood, and how the Harris response works. The proposed model is explained in section 3; each step is presented in detail here. We explain our experiments and show our results in section 4. Finally, in section 5, we present our conclusions and future work.

2. BACKGROUND

A 3D object can be formally represented as a 2-manifold. Discretizing, we can represent it as a triangular mesh. A triangular mesh is composed by a set of vertices and a set of faces (triangles) . The faces define the connectivity of the vertices, this representation preserves geometric and topological features. A point-wise descriptor is a function , that describes all the object; these kinds of descriptors work independently on each vertex and its locality. First, we must compute the neighborhood and then the descriptor values. Each neighborhood can be defined using the concept of *k*-rings (an example of this concept is shown in figure 1, used by Sipiran and Bustos (2011).

Figure 1

K-rings



Note. White: evaluated vertex. Blue: 1-ring. Red: 2-ring. Green: 3-ring. Adapted from *Harris 3D: a robust extension of the Harris operator for interest point detection on 3D meshes* by I. Sipiran & B. Bustos, 2011, *The Visual Computer*, 27 (https://link.springer.com/content/pdf/10.1007/s00371-011-0610-y.pdf)

A *k*-ring is defined as follows:

$$ring_k(p) = \{p' \epsilon\ M / |\text{minimun\_path}(p, p')| = k\} \qquad (1)$$

We assume that the length of the edges is one. A neighborhood with radius equal to *k* is the set formed by the union of all rings with radius . Formally we have:

$$neighborhood_k(p) = \bigcup_{i=1}^{k} ring_i(p) \quad (2)$$

2.1 Harris 3D

In our experiments, we use the Harris response to get our results. Harris response is a local point-wise descriptor that uses neighborhoods to get the saliency value of each vertex. Given a neighborhood N of a vertex , defined using the concept of *k*-rings, it is possible to approximate a surface. To address rotation invariance, a fitting plane is computed using Principal Component Analysis (PCA). The neighborhood is rotated with respect to the normal of this plane. Sipiran & Bustos (2011) fit a quadratic surface to the set of transformed vertices. This surface has the following form:

$$f(x,y) = \frac{p_1}{2}x^2 + p_2xy + \frac{p_3}{2}y^2 + p_4x + p_5y + p_6 \quad (3)$$

Then, with this continuous surface, the derivatives can be calculated on the vertex by:

$$fx = \frac{\delta f(x,y)}{\delta x} \quad (4)$$

$$fy = \frac{\delta f(x,y)}{\delta y} \quad (5)$$

To avoid noise problems, these authors propose continuous Gaussian functions that can be expressed as follows:

$$A = p_4^2 + p_1^2 + p_2^2 \quad (6)$$

$$B = p_5^2 + p_2^2 + p_3^2 \quad (7)$$

$$C = p_4p_5 + 2p_1p_2 + 2p_2p_3 \quad (8)$$

With these terms, the matrix E is defined:

$$E = \begin{pmatrix} A & C \\ C & B \end{pmatrix} \quad (9)$$

The Harris response is computed by:

$$h = \det(E) - q(tr(E))^2 \quad (10)$$

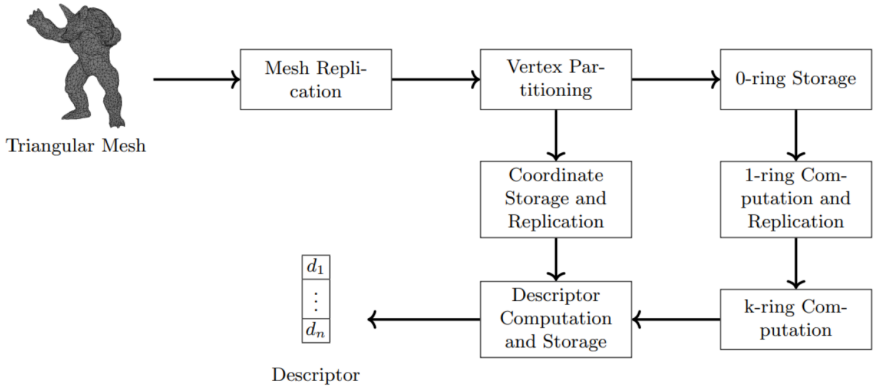
Where *q* is a constant.

3. DISTRIBUTED MODEL

In this paper, we introduce a distributed model for 3D mesh local descriptor computation. The cluster executes in parallel the same tasks with different data and different parameters. First, we define a neighborhood based on *k*-rings; using this technique, it is necessary to find different levels of adjacency (0-k levels). The adjacent vertices, which make up the neighborhood, are necessary to compute the descriptor.

Figure 2

Workflow of the proposed algorithm



Note. Adapted from *Harris 3D: a robust extension of the Harris operator for interest point detection on 3D meshes,* by I. Sipiran & B. Bustos, 2011, *The Visual Computer,* 27, (https://link.springer.com/content/pdf/10.1007/s00371-011-0610-y.pdf)

We address two problems: insufficient memory capacity and parallel processing. To solve the first one, our method uses disk I/O operations for final and temporal data, which is helpful for partial results storage. These operations are executed in parallel, on each node of the cluster. The second one is solved using a multi-process architecture. Our model consists of seven steps. The input is a 3D triangular mesh and the output, its respective descriptor. 1) Mesh replication: distributes the mesh file. 2) Vertex partitioning: regular partitioning using appearance order. 3) Coordinate storage and replication: extracts coordinates from mesh file and replicates them on each cluster node. 4) 0-ring storage: the 0-ring is integrated by the evaluated vertex. 5) 1-ring computation and replication: the 1-ring is integrated by the adjacent vertices, and it is replicated on all the cluster. 6) *K*-ring computation: the *k*-ring is computed using the (*k*-1)-ring, the (*k*-2)-ring and the 1-ring. 7) Descriptor computation and storage: using the computed rings we can load the coordinates for descriptor computation. The workflow of our model is shown in figure 2.

The mesh replication step replicates the mesh input file on each node of the cluster (a node can have multiple processes). Each process reads the same file, which has a set of vertices (with their respective coordinates) and a set of faces (triangles). The vertices are not sorted or clustered, so we can't split them by proximity. In the next step, the set of vertices is partitioned according to their order of appearance. These partitions are assigned to each process. Then, we must store two types of data: global data and local data. The global data is replicated on each node of the cluster, and the local data is stored in the corresponding node.

The 0-ring and the coordinates can be loaded at the same time. In a typical mesh file, the coordinates of each vertex are located at the top. These coordinates are replicated (global data) and will be useful for descriptor computation. The 0-ring is integrated by the vertices that have 0 distance from the source, therefore the 0-ring is conformed only by the evaluated vertex. In the mesh file each triangle is represented by the indices of the three vertices that conform it. We use that information to compute the 1-ring of each vertex and then replicate this information all over the cluster (global data). This ring is replicated because we have to access it several times for the computation of the following rings.

In our implementation, we compute and store the coordinates, the 0-ring and the 1-ring before doing the replication process. Each node of the cluster generates files for their corresponding partitions. We execute one process for each partition and each process is composed by sequential steps depending on the memory usage we want. Support data structures are used to optimize the computation time. Each node executes its processes in parallel. When this task is finished, the coordinates and the 1-ring are replicated all over the cluster.

The next step is the computation of the *k*-rings. We make use of the 1-ring to obtain the adjacent vertices of a specific vertex. If we are computing the *k*-ring, we need to access to the (*k*-1)-ring and (*k*-2)-ring. We use the first one in order to get the adjacent vertices of the previous ring (external ring). And the second one because the adjacent vertices we select should not be in the rings that have already been computed. We use data structures to speed up this step. As in the previous steps, we assign a process for each partition and use sequential steps depending on the memory usage. We compute the *k*-rings sequentially regarding the partitions. Figure 3 shows in detail the algorithm used in this step .

Figure 3

Algorithm of computation of k-ring

```
1: procedure K-RING(k,partitions,steps)
2:     for all p ∈ partitions do
3:         split p in a set of chunks C, regarding the number of steps
4:         for all chunk c ∈ C do
5:             for all vertex v ∈ c do
6:                 load (k-1)-ring and (k-2)-ring, regarding v
7:             for all vertex v ∈ c do
8:                 for all vertex a such that (v, a) is an edge do
9:                     if a ∉ (k-1)-ring and a ∉ (k-2)-ring then
10:                        include a in k-ring set, for the evaluated vertex v
11:        write k-ring partition
```

Note. Adapted from *Harris 3D: a robust extension of the Harris operator for interest point detection on 3D meshes* by I. Sipiran & B. Bustos, 2011, *The Visual Computer,* 27, (https://link.springer.com/content/pdf/10.1007/s00371-011-0610-y.pdf)

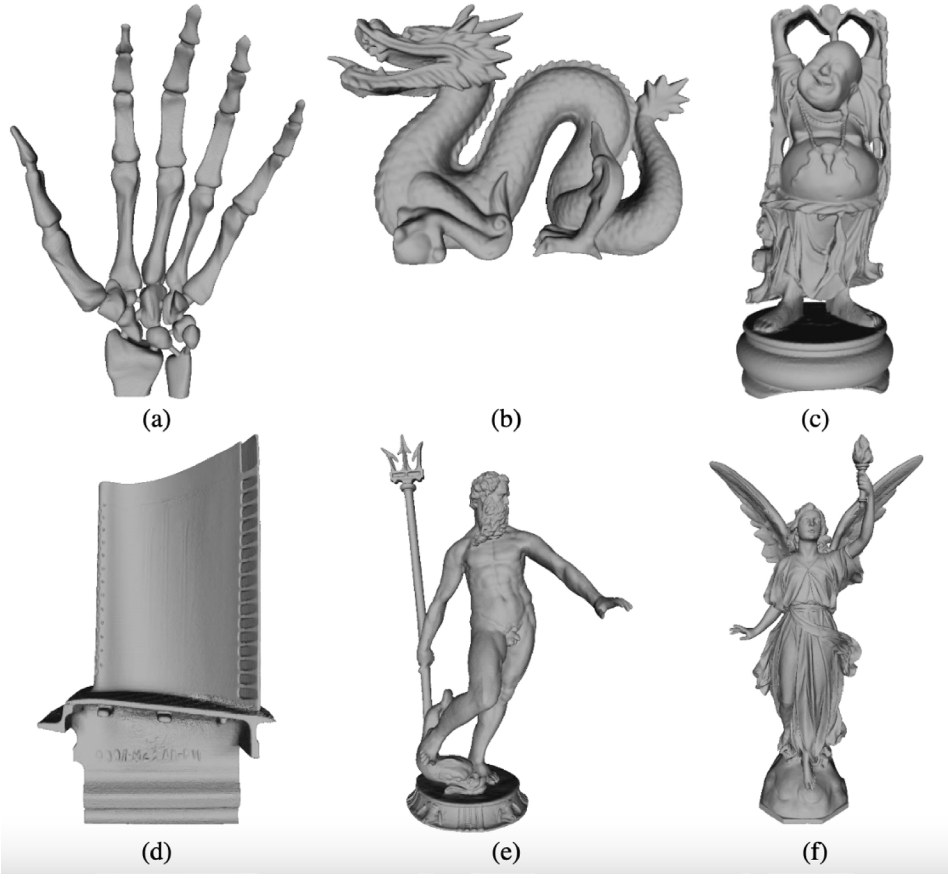
Given the set of rings and the coordinates, we can define a neighborhood and the descriptor values. In the same way as in a single ring computation of the *k*-rings, we load the coordinates instead of the adjacent vertices. For each partition, the corresponding descriptor is computed regarding the vertices involved. The result consists of files storing the descriptor values of each vertex of the mesh.

4. EXPERIMENTS AND RESULTS

We implemented a cluster of ten computers, each with the following specifications: Ubuntu 14.04 LTS OS, Intel Core i7-4770 CPU @ 3.40GHz x8 processor architecture and 8 Gb of RAM. We did several tests to obtain an optimal configuration of the number of processes and steps. For testing reasons, we generated uniform plane meshes of different sizes. Also, we used 3D models which were directly obtained from the Stanford 3D Scanning Repository1, the AIM@SHAPE Shape Repository2 and the GIT Large Geometry Models Archive3. These models are shown in figure 4.

Figure 4

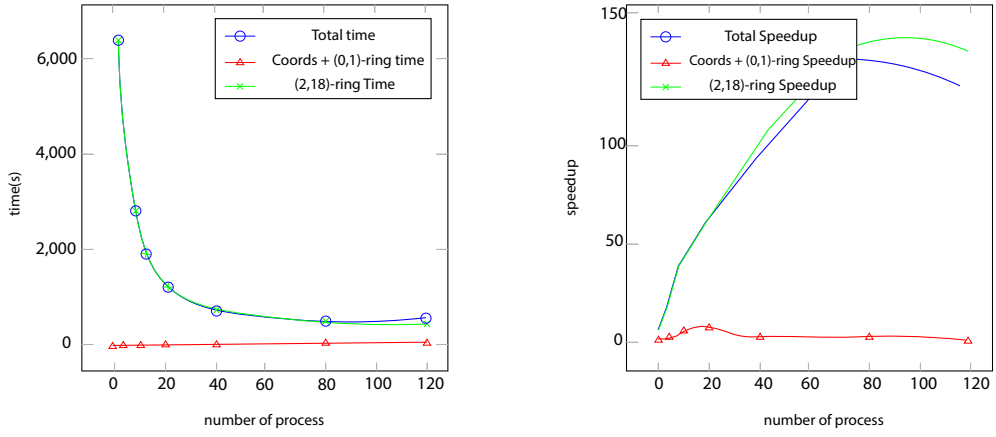
Models used for the experiments



Note. The models were obtained from the Stanford 3D Scanning Repository, the AIM@SHAPE Shape Repository, and the GIT Large Geometry Models Archive. (a) Skeleton Hand. (b) Dragon. (c) Happy Buddha. (d) Turbine Blade. (e) Neptune. (f) Lucy.

Figure 5

Execution time and speed up obtained over a mesh with 16 million vertices using different number of processes



(a) Execution time

(b) Speedup

Note. Blue: total time/speedup of k-rings computation. Red: time/speedup of coordinates and (0,1)-ring computation. Green: time/speedup of (2-k)-ring computation.

4.1 Speedup

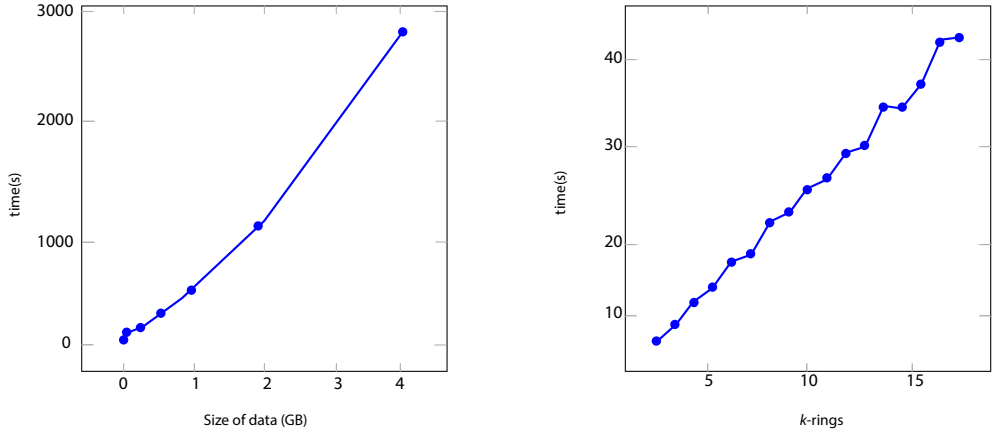
First, using a mesh conformed by 16 million vertices we calculated the execution time of the coordinates and (0-k)-ring computation for different number of processes. This task is the critical part in our implementation. In the case of the coordinates and (0,1)-ring computation, we obtained better results using one step for all number of processes except when we used only one. This occurred since we didn't have enough memory capacity. For the (2-*k*)-ring computation we calculated an optimal number of steps depending on the number of processes used. Here we did not consider the computation time of the descriptor because the process of extracting the neighborhoods is very similar to a single ring computation and calculating the values for each neighborhood will depend on the descriptor we choose. In table 1 we show the total and partial execution times for different number of processes. Also, we show the speedup obtained regarding the sequential implementation. In figure 5 (a) and 5 (b) we show the graphs obtained from the execution time values and the speedup values respectively. The optimal number of processes for the total computation time is 80. But in the case of coordinates and (0,1)-ring computation, the optimal number of processes is 20. This occurs because these steps don't have too much processing cost.

4.2 Scalability

Using the optimal number of processes (80), we tested the task mentioned on the previous section on plane regular meshes of different sizes. The number of rings that we calculated was 18. The number of steps we used were selected according to the size of the mesh. In table 2 we show the number of vertices and the size of the meshes, the number of steps of each process, the size of the computed data (rings and coordinates), and the total computation time. In figure 6 (a) we show the respective behavior. The graph shows that the growth is linear with respect to the number of vertices.

Figure 6

K-rings computation time using plane meshes of different sizes



(a) Size of data

(b) *k*-rings

Note. (b) shows the computation time of k-rings with 16 million of vertices and 80 processes.

We calculated the computation time for each ring over a regular plane mesh of 16 million vertices, using 80 processes. In table 3 we show the results from 2-Ring to 18-ring, and in figure 6(b) its respective graph. The growth regarding the number of rings is also linear.

Table 1

Results describing the execution time and speedup obtained using different number of processes over a mesh with 16 million vertices

Proc.	Steps	Coords + (0,1)- ring (s)	(2-18)-ring (s)	Total (s)	Speedup
1	25	130,083	50 849,972	50 980,055	1
2	18	43,023	7203,385	7246,408	7,035
5	16	44,918	3093,867	3138,785	16,242
10	15	29,118	1448,57	1477,688	34,5
20	14	17,057	931,066	948,123	53,769
40	14	31,782	588,037	619,819	82,25
80	13	34,834	371,161	405,995	125,568
120	11	63,030	385,741	448,771	113,559

Table 2

K-ring computation time using plane meshes of different sizes

Vertices	Size (GB)	Steps	Data (GB)	Time (s)
1 million	0,052	1	0,666	28,404
2 million	0,109	2	1,5	52,492
4 million	0,224	4	3,1	90,751
8 million	0,454	7	4,8	151,881
16 million	0,948	13	14	405,995
32 million	2	20	28	1090,801
64 million	4	30	56	2744,154

Table 3

Computation time in seconds of (1-18) – ring using 80 processes in a 16 million vertex plane mesh

Ring	Time (s)	Ring	Time (s)	Ring	Time (s)
2-ring	4,856	8-ring	19,367	14-ring	33,587
3-ring	6,927	9-ring	20,677	15-ring	33,567
4-ring	9,649	10-ring	23,472	16-ring	36,394
5-ring	11,523	11-ring	24,840	17-ring	41,592
6-ring	14,564	12-ring	27,583	18-ring	42,052
7-ring	15,557	13-ring	29,920		

4.3 Descriptor

In this test we calculated the execution times for all the tasks involved in the descriptor computation. We used the Harris response as a descriptor, which is in Sipiran & Bustos, (2011).

Table 4

K-ring based descriptor computation

Mesh	# Vertices	Size (GB)	Replication (s)	(2-10) – ring (s)	Descriptor (s)	Data size (GB)	Total (s)
Skeleton Hand	327 323	0,020	1,084	3,800	1,101	0,091	9,7
Dragon	437 645	0,029	1,504	4,109	1,939	0,153	14,449
Happy Buddha	543 652	0,037	2,082	4,499	2,469	0,199	17,689
Turbine Blade	882 954	0,037	2,267	5,771	2,806	0,254	19,74
Neptune	2 003 932	0,150	6,736	10,445	8,774	0,626	55,552
Lucy	1 402 7872	0,969	36,641	65,878	45,061	4,9	285,682
Plane (16)	16 000 000	0,948	34,834	72,603	34.5	4,8	238,607
Plane (32)	32 001 649	1,957	68,781	156,604	140,193	10,5	582,54
Plane (64)	64 000 000	3,977	135,969	324,011	419,891	21,5	1533,345

The neighborhoods are conformed by rings with $k \leq 10$. In the test we used six meshes obtained from the repositories we previously mentioned, and three plane meshes with different sizes. The results we obtained are shown in table 4. We show the name of the mesh, the number of vertices, the size of the mesh, the coordinates and (0,1)-ring execution time, the replication time, the (2-10)-ring execution time, the descriptor computation time, the size of the generated data, and the total time of the whole computation. Our implementation works better when the partition vertices are near, since less disk operations are needed to obtain the adjacent vertices and the coordinates.

5. CONCLUSION AND FUTURE WORK

The main contribution of this paper is the distributed computation of 3D mesh local descriptors based on the concept of *k*-rings. In our implementation, we work with three main resources: processors, RAM and hard disk. We have parameters that allow us to fix the usage of these resources. Getting an optimal configuration will also depend on the cluster features. Automating the selection of these parameters is a future work.

The partitioning method used on this model is very simple (appearance order), Using a smart partitioning, such as a Euclidean distance-based partitioning, would optimize the execution time. In that case, a preprocessing step is needed in order to obtain balanced partitions.

We showed that our method is scalable, so increasing the cluster capacity will decrease the execution time. Also, there are some operations that can be executed using GPU. This will reduce considerably the execution time of the descriptor computation. We can extend our model for the computation of local descriptors over point clouds. In that case, we would have to use spatial data structures and a Euclidean distance-based partitioning with overlapping.

REFERENCES

Aleardi, L., Devillers, O., & Schaeffer, G. (2005). Succinct representation of triangulations with a boundary. In F. Dehne, A. López-Ortiz, & J.-R. Sack (Eds.), *Algorithms and data structures.* WADS 2005. Lecture Notes in Computer Science, vol 3608 (pp. 134-145). Springer. https://doi.org/10.1007/11534273_13

Areias, P., & Rabczuk, T. (2017). Steiner-point free edge cutting of tetrahedral meshes with applications in fracture. *Finite Elements in Analysis and Design*, *132*, 27-41. https://doi.org/10.1016/j.finel.2017.05.001

Balman, M. (2006). Tetrahedral mesh refinement in distributed environments. In T. M. Pinkston & F. Ozguner (Eds.), Proceedings of the *2006 International Conference on Parallel Processing Workshops (ICPPW'06)* (pp. 498-504). IEEE Computer Society. https://doi.org/10.1109/ICPPW.2006.72

Cabiddu, D., & Attene, M. (2015). Distributed processing of large polygon meshes. In A. Giachetti, S. Biasotti, & M. Tarini (Eds.), *Smart tools and apps for graphics—Eurographics Italian chapter conference* (pp. 139-148). The Eurographics Association. https://doi.org/10.2312/stag.20151301

Castellani, U., Cristani, M., Fantoni, S., & Murino, V. (2008). Sparse points matching by combining 3D mesh saliency with statistical descriptors. *Computer Graphics Forum*, *27*(2), 643-652. https://doi.org/10.1111/j.1467-8659.2008.01162.x

Chen, X., Golovinskiy, A., & Funkhouser, T. (2009). A benchmark for 3D mesh segmentation. *ACM TransActions on Graphics, 28*(3), 1-12. https://doi.org/10.1145/1531326.1531379

Cignoni, P., Montani, C., Rocchini, C., & Scopigno, R. (2003). External memory management and simplification of huge meshes. *IEEE Transactions on Visualization and Computer Graphics, 9*(4), 525-537. https://doi.org/10.1109/TVCG.2003.1260746

Figueiredo, L., Ivson, P., & Celes, W. (2021). Deep learning-based framework for Shape Instance Registration on 3D CAD models. *Computers & Graphics, 101*, 72-81. https://doi.org/10.1016/j.cag.2021.08.012

Gao, L., Cao, Y.-P., Lai, Y.-K., Huang, H.-Z., Kobbelt, L., & Hu, S.-M. (2015). Active exploration of large 3D model repositories. *IEEE Transactions on Visualization and Computer Graphics, 21*(12), 1390-1402. https://doi.org/10.1109/TVCG.2014.2369039

Gelfand, N., Mitra, N. J., Guibas, L. J., & Pottmann, H. (2005). Robust global registration. In M. Desbrun & H. Pottmann (Eds), *Europgraphics Symposium on Geometry Processing* (pp. 197-206). Alvety Vision Club. http://vecg.cs.ucl.ac.uk/Projects/SmartGeometry/global_registration/paper_docs/global_registration_sgp_05.pdf

Gupta, O., & Rani, S. (2013). Accelerating molecular sequence analysis using distributed computing environment. *International Journal of Scientific & Engineering Research–IJSER*, *4*(10), 262-265. https://www.ijser.org/onlineResearchPaperViewer.aspx?Accelerating-Molecular-Sequence-Analysis-using-Distributed-Computing-Environment.pdf

Gurung, T., Laney, D., Lindstrom, P., & Rossignac, J. (2011). SQuad: Compact representation for triangle meshes. *Computer Graphics Forum, 30*(2), 355-364. https://doi.org/10.1111/j.1467-8659.2011.01866.x

Gurung, T., Luffel, M., Lindstrom, P., & Rossignac, J. (2013). Zipper: A compact connectivity data structure for triangle meshes. *Computer-Aided Design, 45*(2), 262-269. https://doi.org/10.1016/j.cad.2012.10.009

Harris, C., & Stephens, M. (1988). A combined corner and edge detector. In C. J. Taylor (Ed.), *Proceedings of the Alvey Vision Conference* (pp. 23.1-23.6). http://dx.doi.org/10.5244/C.2.23

Herath, U., Tavadze, P., He, X., Bousquet, E., Singh, S., Muñoz, F., & Romero, A. H. (2020). PyProcar: A Python library for electronic structure pre/post-processing. *Computer Physics Communications, 251*, 107080. https://doi.org/10.1016/j.cpc.2019.107080

Lee, C. H., Varshney, A., & Jacobs, D. W. (2005). Mesh saliency. *ACM Transactions on Graphics*, *24*(3), 659-666. https://doi.org/10.1145/1186822.1073244

Le Tien, M., Tan, K. N., & Raffin, R. (2021, 15-16 December). Analysis of geometrical features of 3D model based on the surface curvature of a set of point cloud. In *The 5th International Conference on Future Networks & Distributed Systems* (pp. 17-23). The Association of Computing Machinery. https://doi.org/10.1145/3508072.3508076

Levoy, M., Pulli, K., Curless, B., Rusinkiewicz, S., Koller, D., Pereira, L., Ginzton, M., Anderson, S., Davis, J., Ginsberg, J., Shade, J., & Fulk, D. (2000). The digital Michelangelo project: 3D scanning of large statues. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques* (pp. 131-144). SIFFGRAPH. https://doi.org/10.1145/344779.344849

Li, B., Lu, Y., Li, C., Godil, A., Schreck, T., Aono, M., Burtscher, M., Chen, Q., Chowdhury, N. K., Fang, B., Fu, H., Furuya, T., Li, H., Liu, J., Johan, H., Kosaka, R., Koyanagi, H., Ohbuchi, R., Tatsuma, A., Wan, Y, Zhang, C., & Zou, C. (2015). A comparison of 3D shape retrieval methods based on a large-scale benchmark supporting multimodal queries. *Computer Vision and Image Understanding*, *131*, 1-27. https://doi.org/10.1016/j.cviu.2014.10.006

Luffel, M., Gurung, T., Lindstrom, P., & Rossignac, J. (2014). Grouper: A Compact, Streamable Triangle Mesh Data Structure. *IEEE Transactions on Visualization and Computer Graphics*, *20*(1), 84-98. https://doi.org/10.1109/TVCG.2013.81

Maquart, T., Elguedj, T., Gravouil, A., & Rochette, M. (2021). 3D B-Rep meshing for real-time data-based geometric parametric analysis. *Advanced Modeling and Simulation in Engineering Sciences, 8*, 8. https://doi.org/10.1186/s40323-021-00194-5

Mitra, N. J., Pauly, M., Wand, M., & Ceylan, D. (2013). Symmetry in 3D geometry: Extraction and applications. *Computer Graphics Forum, 32*(6), 1-23. https://doi.org/10.1111/cgf.12010

O' Sullivan, E., Van de Lande, L. S., Papaioannou, A., Breakey, R. W. F., Jeelani, N. O., Ponniah, A., Duncan, C., Schievano, S., Khonsari, R. H., Zafeiriou, S., & Dunaway, D. J. (2022). Convolutional mesh autoencoders for the 3-dimensional identification of FGFR-related craniosynostosis. *Scientific Reports, 12*, 2230. https://doi.org/10.1038/s41598-021-02411-y

Pavlakos, G., Zhu, L., Zhou, X., & Daniilidis, K. (2018). Learning to estimate 3D human pose and shape from a single color image. In L. O'Conner (Ed.), *Proceedings of the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 459-468). IEEE Computer Society; Conference Publishing Services. https://doi.org/10.1109/CVPR.2018.00055

Pereira, R., Azambuja, M., Breitman, K., & Endler, M. (2010). An architecture for distributed high performance video processing in the cloud. In R. Bilof (Ed,), Proceedings of

the *2010 IEEE 3rd International Conference on Cloud Computing* (pp. 482–489). IEEE Computer Society; Conference Publishing Services. https://doi.org/10.1109/CLOUD.2010.73

Prajapati, H. B., & Vij, S. K. (2011). Analytical study of parallel and distributed image processing. In R Siddavatam & S. P. Ghrera (Eds.), *2011 International Conference on Image Information Processing* (pp. 1-6). The Institute of Electrical and Electronics Engineers. https://doi.org/10.1109/ICIIP.2011.6108870

Rocca, L., De Giorgis, N., Panozzo, D., & Puppo, E. (2011). Fast neighborhood search on polygonal meshes. In A. F. Abate, M. Nappi, & G. Tortora (Eds.). *Eurographics Italian Chapter Conference 2011* (pp.15-21). The Eurographics Association. https://doi.org/10.2312/LocalChapterEvents/ItalChap/ItalianChapConf2011/015-021

Sankaranarayanan, J., Samet, H., & Varshney, A. (2007). A fast all nearest neighbor algorithm for applications involving large point-clouds. *Computers & Graphics, 31*(2), 157-174. https://doi.org/10.1016/j.cag.2006.11.011

Sipiran, I., & Bustos, B. (2011). Harris 3D: a robust extension of the Harris operator for interest point detection on 3D meshes. *The Visual Computer, 27*, 963. https://doi.org/10.1007/s00371-011-0610-y

Squyres, J. M., Lumsdaine, A., McCandless, B. C., & Stevenson, R. L. (2000). *Parallel and distributed algorithms for high speed image processing*. University of Notre Dame. https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.939.2869&rep=rep1&type=pdf

Van Kaick, O., Zhang, H., Hamarneh, G., & Cohen-Or, D. (2011). A survey on shape correspondence. *Computer Graphics Forum, 30*(6), 1681-1707. https://doi.org/10.1111/j.1467-8659.2011.01884.x

Verma, N., Boyer, E., & Verbeek, J. (2018). FeaStNet: Feature-steered graph convolutions for 3d shape analysis. In L. O'Conner, (Ed.), *Proceedings of the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 2598-2606). IEEE Computer Society; Conference Publishing Services. https://doi.org/10.1109/CVPR.2018.00275

Vo, H. T., Bronson, J., Summa, B., Comba, J. L. D., Freire, J., Howe, B., Pascucci, V., & Silva, C. T. (2011). Parallel visualization on large clusters using MapReduce. In D. Rogers & C. T. Silva. *Proceedings of the 2011 IEEE Symposium on Large Data Analysis and Visualization* (pp. 81-88). IEEE Computer Society Press. https://doi.org/10.1109/LDAV.2011.6092321

Warn, S., Emeneker, W., Cothren, J., & Apon, A. (2009, 31 de Agosto-4 de setiembre). *Accelerating SIFT on parallel architectures* [written presentation]. 2009 IEEE

International Conference on Cluster Computing and Workshops, Nueva Orleans, Louisiana, USA.. https://doi.org/10.1109/CLUSTR.2009.5289155

Zaharescu, A., Boyer, E., Varanasi, K., & Horaud, R. (2009, 20-25 de junio). *Surface feature detection and description with applications to mesh matching*. [written presentation] 2009 IEEE Conference on Computer Vision and Pattern Recognition, Miami,Florida, Estados Unidos. https://doi.org/10.1109/CVPR.2009.5206748

Zamolo, R., Miotti, D., & Nobile, E. (2022). Numerical analysis of thermo-fluid problems in 3D domains by means of the RBF-FD meshless method. *Journal of Physics: Conference Series*, *2177*, 012007. https://doi.org/10.1088/1742-6596/2177/1/012007

Zhou, R., Song, Z., & Lu, Y. (2017). 3D mesoscale finite element modelling of concrete. *Computers & Structures, 192*, 96–113. https://doi.org/10.1016/j.compstruc.2017.07.009