UNIVERSIDADE FEDERAL FLUMINENSE

LUCIANO ARNALDO ROMERO CALLA

An Iterative Parallel Algorithm for Computing Geodesic Distances on Triangular Meshes

NITERÓI

2017

UNIVERSIDADE FEDERAL FLUMINENSE

LUCIANO ARNALDO ROMERO CALLA

An Iterative Parallel Algorithm for Computing Geodesic Distances on Triangular Meshes

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Mestre em Computação. Área de concentração: Computação Visual

Orientador:
Prof. Anselmo Antunes Montenegro

NITERÓI

2017

Ficha Catalográfica elaborada pela Biblioteca da Escola de Engenharia e Instituto de Computação da UFF

R763 Romero Calla, Luciano Arnaldo
An iterative parallel algorithm for computing geodesic distances on triangular meshes / Luciano Arnaldo Romero Calla. – Niterói, RJ : [s.n.], 2017.
55 f.

Dissertação (Mestrado em Computação) - Universidade Federal Fluminense, 2017.
Orientador: Anselmo Antunes Montenegro.

1. Computação visual. 2. Algoritmo. 3. geodésica. I. Título.

CDD 006

Luciano Arnaldo Romero Calla

An Iterative Parallel Algorithm for Computing Geodesic Distances on Triangular Meshes

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Mestre em Computação. Área de concentração: Computação Visual

Aprovada em Fevereiro de 2017.

Niterói

2017

This works is dedicated to my parents.

Acknowledgements

First of all, I would like to thank my parents, without them I wouldn't be here. I would like to thank professor Cristian Lopez, who taught me the basics of Computer Science and taught me how to think properly. I would like to thank my advisor and friend Anselmo Montenegro for his invaluable help to guide this work, especially in the mathematical fundamentals. I would also like to thank my friend and co-worker Lizeth Fuentes for providing a creative way to solve the problems and sharing her ideas with me during these years of mutual collaboration.

Resumo

O cálculo de distâncias geodésicas é um importante tópico de pesquisa no processamento de geometria e análise de formas em malhas, pois é um componente básico de muitos métodos usados nessas áreas. Diferentes abordagens têm sido propostas para calcular distâncias geodésicas incluindo métodos exatos e aproximados. Métodos aproximados são em muitos casos a melhor escolha, pois são mais eficientes do que os métodos exatos e, ao mesmo tempo, produzem resultados próximos da distância exata. Muitos métodos têm sido propostos para o cálculo de distâncias geodésicas aproximadas, por exemplo, variações do método Fast Marching e mais recentemente por métodos espectrais e de fluxo de difusão. Estas abordagens são muito eficientes para a consulta da distância, mas geralmente dependem de uma etapa de pré-processamento que pode ser computacionalmente intensa. Neste trabalho, apresentamos um algoritmo paralelo iterativo para calcular distâncias geodésicas aproximadas em malhas que não requer nenhum passo de pré-processamento. A convergência do algoritmo iterativo proposto depende do número de anéis em torno dos pontos de origem, a partir dos quais a informação da distância se propaga. Assim, nosso método é particularmente eficiente para a computação de distâncias geodésicas de múltiplas fontes. Nos experimentos, mostramos como nosso método escala com o tamanho do problema e comparamos seu erro médio e os tempos de processamento com os de outros métodos encontrados na literatura. Também demonstramos seu uso para resolver dois problemas comuns de processamento de geometria: o problema de amostragem regular e a tesselação de Voronoi em malhas.

Palavras-chave: marcha rápida, distância geodésica, malhas triangulares, modelos tridimensionais.

Abstract

The computation of geodesic distances is an important research topic in geometry processing and shape analysis on meshes as it is a basic component of many methods used in these areas. Different approaches have been proposed for computing geodesic distances including exact methods and approximate ones. Approximate methods are in many cases the best choice as they are more efficient than the exact methods and yield results that are not far from the optimum distance. Many methods have been proposed for approximate geodesic distance computation, for instance, variations of the Fast Marching method and more recently by spectral and diffusion flow-based methods. These approaches are very efficient for distance query but usually depend on a pre-processing step which can be computationally intensive. In this work, we present an iterative parallel algorithm for computing approximate geodesic distances on meshes that do not require any pre-processing step. The convergence of our iterative algorithm depends on the number of rings around the source points from which distance information propagates. Hence, our method is particularly efficient for multisource geodesic distance computation. In the experiments, we show how our method scales with the size of the problem and compare its mean error and processing times with such measures computed with other methods found in the literature. We also demonstrate its use for solving two common geometry processing problems: the regular sampling problem and the Voronoi tessellation on meshes.

Keywords: fast marching, geodesic distance, triangular meshes, tridimensional models

List of Figures

5.1 Rings propagation with 3 samples; in **green** $\boldsymbol{V_0 = S}$; in **blue** $\boldsymbol{V_1}$, in **red** $\boldsymbol{V_2}$, in **purple** $\boldsymbol{V_3}$ and in **black** $\boldsymbol{V_4}$. 14

5.2 Lower bounds to m and T for $n \in [1 : 10000]$ 20

6.1 Geodesic distances map for each mesh in Table 6.1, from $m = 1$ sources. . . 23

6.2 FM (CPU) and PRP (GPU) execution times from $m = 1$ sources. 24

6.3 Parallel Ring Propagation speedup from $m = 1$ sources. 25

6.4 PRP speedup with $m \in [1 : 1000]$. 25

6.5 Execution times of the PRP algorithm with $m \in [1 : 1000]$ for each mesh. . 26

6.6 MAPE in function of number of iterations $k \in [\rho : 2\rho]$. 27

6.7 Rings map (rings propagation) in 6.7a and 6.7d, geodesics distances map in 6.7b and 6.7e. Bunny triangular mesh 6.7c. 28

6.8 Farthest Point Sampling speedup with $m \in [1 : 1000]$. 29

6.9 Computation of $m = 1000$ samples to bunny mesh 6.9a. Voronoi regions in 6.9b, 6.9c and 6.9d; from the calculation of $m = 100$ samples with the FPS algorithm. 30

A.1 Counting vertices at ring V_r. 36

B.1 Meshes topology. 40

B.2 Degree meshes. 41

B.3 Rings meshes distribution. 42

B.4 Sorted rings meshes distribution. 43

B.5 Analysis of the error with double precision. 44

List of Tables

5.1 Comparison of complexity algorithm analysis. 20

6.1 Comparison times, mean absolute percent error and speedup. 24

A.1 Vertices updated at iteration i'. 38

Contents

1 Introduction **1**

1.1 Contribution . . . 2

1.2 Outline . . . 2

2 Introdução **3**

2.1 Contribuição . . . 4

2.2 Organização . . . 4

3 Related Works **6**

4 Background **8**

4.1 Basic concepts . . . 8

4.2 Distance maps . . . 9

4.3 Discrete Distance Maps . . . 9

4.4 Fast Marching algorithm . . . 10

5 Proposed Method **13**

5.1 Rings propagation . . . 13

5.2 Ring Propagation-based algorithm . . . 14

5.3 Parallelization . . . 16

5.4 Algorithm Complexity and Comparison . . . 17

5.4.1 Application: Farthest Point Sampling . . . 18

5.4.2 Lower bound to m and T . . . 20

6 Experimental evaluation **23**

6.1 Performance 24

6.2 Accuracy 26

6.3 Farthest Point Sampling and Voronoi Diagrams 29

7 Conclusion **31**

References **33**

Appendix A – Proofs **35**

A.1 Theorem 5.1 35

A.2 Theorem 5.2 36

Appendix B – Results **40**

B.1 Mesh topology 40

B.2 Analysis of the degrees of the vertices 41

B.3 Analysis of the distribution of rings 42

B.4 Analysis of error double precision 44

Chapter 1

Introduction

The computation of geodesic distances on meshes is of paramount importance to many geometry processing and shape analysis algorithms, for example: Parameterization [11], shape retrieval [16], isometry-invariant shape classification [3], mesh watermarking [1], object recognition [8], skinning [19], just to mention a few. Because it is a basic step for many geometrical algorithms the efficiency of its computation is a very important issue.

Meshes are usually described as graphs. Consequently one natural approach to solve the problem of geodesic computation on meshes is to generalize the ideas of distances on graphs to compute distance maps on surface representations. This was the path pursued by [13] which proposed a continuous Dijkstra method that is able to yield exact results. Later, [20] refined such method and proposed an approximate version that is more efficient. Other researches approached the problem via physical phenomena analogy, for instance, based on models for propagation of waves and diffusion of heat. The Fast Marching approach belongs to the class of those first methods and aims to solve the so called Eikonal Equation. The Geodesics on Heat belongs to the second class and explores the relation between heat kernel computation and distances on surfaces. Recently a new class of spectral methods was proposed in [4].

These methods, although powerful, have its idiosyncrasies and different drawbacks. We describe in details these drawbacks in Chapter 3 and how we tackled them by proposing a new method in Chapter 5.

Here, we propose a parallel algorithm for the computation of distance maps on meshes that produces very competitive results and is simple to implement. As the Fast Marching method, it is also inspired by the grassfire propagation but does not require any priority queue. Instead, we propagate distances simultaneously in a band around the frontier of the propagation in a sequence of phases. Our experiments show that no more than $c\sqrt{n}$ iterations, where c is a small constant and n is the number of vertices, are necessary for the convergence of the method. We provide a theoretical computational complexity congruent with the experimental results. Our method is specially appropriate to the solution of the multisource distance map computation.

1.1 Contribution

We introduce an iterative parallel algorithm called Parallel Rings Propagation to compute distances maps from a set of multiple sources on triangular meshes. Our method is based on the propagation of distance information from the inner to the outer rings of the sources where each vertex is updated in parallel independently, exploring the powerful parallel architectures in the GPUs. It is specially appropriate for problems where multiple sources are required and queries will not be performed later. We present our results for problems with such characteristics: the *Regular Point Sampling* and the *Voronoi Diagram on Meshes*.

1.2 Outline

This work is organized as follows. In Chapter 3, we describe some of the most important works related to distance computation on graphs and minimal geodesic computation on meshes. Next, in Chapter 4, we describe and define some concepts and results that were used to inspire and develop our algorithm. The proposed method is presented in Chapter 5. In Chapter 6, we present the results produced by our algorithm. We first show two applications of our method: a parallel solution to the Farthest Point Sampling Problem and a parallel solution to the problem of computing the Voronoi Diagram on meshes. Next, we present the speedup of our parallel method for different meshes and measure the distance error for each experiment. We also compare our results with the exact method of [13] and the Fast Marching algorithm of [10]. Finally in Chapter 7, we present the conclusions.

Chapter 2

Introdução

O cálculo de distâncias geodésicas em malhas é de suma importância para muitos problemas nas áreas de processamento geométrico e análise de formas, por exemplo: parametrização [11], recuperação de formas [16], classificação de formas isométricas invariantes [3], Watermarking em malhas [1], reconhecimento de objetos [8], skinning [19], apenas para mencionar alguns. A eficiência do cálculo de distâncias geodésicas é uma questão muito importante, porque ela é um passo básico em muitos algoritmos geométricos e portanto, ainda é alvo de investigação.

Um grafo é a estrutura de dados normalmente utilizada para descrever uma malha. Consequentemente, uma abordagem natural para resolver o problema de computação geodésica em malhas é generalizar as ideias de distâncias em grafos para calcular mapas de distância em representações de superfícies. Este foi o caminho proposto por Mitchell [13], quem propôs um método de Dijkstra contínuo capaz de calcular distâncias exatas. Mais tarde, Surazhsky [20] refinou esse método e propôs uma versão mais eficiente, calculando uma aproximação do mapa de distâncias.

Outras pesquisas abordaram o problema por meio de analogias com fenômenos físicos, por exemplo, baseados em modelos de propagação de ondas e difusão de calor. A abordagem dos algoritmos de Marcha Rápida (*Fast Marching*) pertence aos primeiros métodos que resolvem a equação Eikonal. A Geodésica sobre o Calor (*Heat on Flow*) pertence à segunda classe e explora a relação entre o núcleo de calor (*Heat Kernel*) e o cálculo de distâncias em superfícies. Recentemente, uma nova classe de métodos espectrais foi proposto em [4].

Estes métodos, embora poderosos, têm suas idiossincrasias e diferentes desvantagens. Nós descrevemos em detalhes estas desvantagens no Capítulo 3 e como as tratamos através do desenvolvimento de um novo método apresentado no Capítulo 5.

Aqui, propomos um algoritmo paralelo para o cálculo de mapas de distâncias em malhas que produz resultados muito competitivos e é simples de implementar. Semelhante ao método de Marcha Rápida, nosso método também é inspirado pela propagação de fogo, mas não requer uma fila de prioridade. Em vez disso, propagamos distâncias simultaneamente em uma faixa ao redor da fronteira da propagação em uma sequência de fases. Nossos experimentos mostram que não mais que $c\sqrt{n}$ iterações, onde c é uma pequena constante e n é o número de vértices, são necessárias para a convergência do método. Nós fornecemos uma complexidade computacional teórica congruente com os resultados experimentais. Mostramos também que nosso método é especialmente apropriado para a solução do cálculo do mapa de distâncias a partir de muitas fontes.

2.1 Contribuição

Introduzimos um algoritmo paralelo iterativo chamado *Parallel Rings Propagation* para calcular mapas de distâncias a partir de um conjunto de múltiplas fontes em malhas triangulares. Nosso método baseia-se na propagação de informações de distância do interior para os anéis externos das fontes onde cada vértice é atualizado em paralelo de forma independente, explorando as poderosas arquiteturas paralelas nas GPUs. O Parallel Rings Propagation é especialmente apropriado para problemas onde várias fontes são necessárias e que não envolvem muitas consultas posteriormente ao cálculo das distâncias. Apresentamos nossos resultados para problemas com tais características: *Amostragem Regular em Malhas* e *Diagrama de Voronoi sobre Malhas.*

2.2 Organização

Este trabalho é organizado da seguinte forma. No Capítulo 3, descrevemos alguns dos trabalhos mais importantes relacionados à computação de distâncias geodésicas em malhas. Em seguida, no Capítulo 4, descrevemos e definimos alguns conceitos e resultados que foram utilizados para inspirar e desenvolver nosso algoritmo. O método proposto é apresentado no Capítulo 5. No Capítulo 6, apresentamos os resultados produzidos pelo nosso algoritmo. Mostramos duas aplicações do nosso método: uma solução paralela para o problema de amostragem do ponto mais distante e uma solução paralela para o problema da computação do diagrama de Voronoi em malhas. A seguir, apresentamos a aceleração do nosso método paralelo para diferentes malhas e medimos o erro de distância para cada experimento. Também comparamos nossos resultados com o método exato de [13] e o algoritmo Marcha Rápida (*Fast Marching*) de [10]. Finalmente no Capítulo 7, apresentamos as conclusões.

Chapter 3

Related Works

The exact computation of geodesic distances on surfaces was proposed by Mitchell et al. [13]. The MMP algorithm, proposed by them, is based on a continuous Dijkstra algorithm and its computational complexity is $O(n^2 \log n)$.

Due to its high computational cost, approximate methods were developed. These methods present a better performance and maintain a comparable level of accuracy. A fast implementation of the MMP algorithm was presented in [20]; they also proposed an approximate and faster algorithm that requires less memory with a computational complexity of $O(n \log n)$.

Over the last decade, several approximate methods were proposed to compute distances by solving the Eikonal equation:

$$|| \bigtriangledown \phi|| = 1 \tag{3.1}$$

where ϕ is a distance function. We can divide these approaches into two families, the Fast Marching, and the Fast Sweeping.

The Fast Marching methods were introduced by Sethian to solve distance computation on regular grids [18] and later were extended to triangular meshes by Kimmel and Sethian [10].

The Fast Marching (FM) is a popular algorithm which maintains the spirit of the Dijkstra algorithm because it uses a priority queue. It is a single-source to all-vertices algorithm, whose main advantage is the fast calculation of distances of vertices that are close to the source vertices. However, due to the sequential requirement of the priority queue, it is not possible to parallelize this algorithm.

The Fast sweeping approaches [15] have linear computational complexity $O(n)$. However they require a lot of sweeps to converge, specially when the grids are unstructured [9].

A parallel version of the Fast Marching algorithm on parametric surfaces was proposed by Weber [21]. In this method, the surface must be divided into several regular grids. Then, the distance map is computed for each grid, using a parallel version of Fast Marching algorithm based on the Raster Scan algorithm [5]. Finally, the reconstruction is done by joining the distance maps of each grid via the Dijkstra algorithm. Up to now, this method is the fastest and highly parallelizable for computing geodesic distances on triangular meshes but the error of the computed distance map depends on the distortion of the parameterization.

Recently a different technique called the Heat method was introduced by Crane [4]. The Heat Method requires the solution of the Poisson equation and has its spirit based on the spectral graph theory because it requires computing the Laplacian operator. This method is adaptable to many kinds of representations because it is possible to compute the Laplacian operator for many different models including triangular meshes, point clouds and polygonal meshes. However, the accuracy of the distance map computation is sensitive to the choice of a parameter, which is done experimentally.

Chapter 4

Background

In the next Chapters we present some of the basic concepts and results in which our method is based on.

4.1 Basic concepts

The distance between two points x and y in a set of points X can be defined by a function $d : X \times X \rightarrow \Re$ satisfying:

(a) $d(x, y) \geq 0$

(b) $d(x, y) = 0$, if and only if $x = y$

(c) $d(x, y) = d(y, x)$

(d) $d(x, z) \leq d(x, y) + d(y, z)$

The function d is a *metric* and the tuple (X, d) is a *metric space*. For subsets of Euclidean spaces, d can be defined by the Euclidean norm $||x - y||_2^2$. Meanwhile, for more general spaces, the notion of distance cannot be well described by a metric induced by the Euclidean norm. This is the case for $n-d$ *manifolds*, i.e., sets of points which locally are homeomorphic to $n-d$ Euclidean open subsets.

The notion of distance on a manifold $\mathcal{M}$ can be defined, though, in terms of *length of paths* on $\mathcal{M}$. A *path* on a manifold $\mathcal{M}$ is a parameterized curve given by a map $\gamma(t) : t \rightarrow \mathcal{M}$, where $t \in I = [a, b]$. According to [6], γ is a *geodesic* at $t \in I$ if the field of its tangent vectors $\gamma'(t)$ are parallel along γ at t. In other words, if the covariant derivative $\frac{D\gamma'(t)}{dt} = 0$. We can say that a parameterized curve γ is a geodesic if it is a geodesic at all $t \in I$ and a *minimizing geodesic* or *shortest path* when it is given by $arg\ inf_{\gamma} L(\gamma) = \int_a^b \sqrt{\gamma(t)'^T G \gamma(t)'} dt$. Based on these concepts we define a *length metric* as $d_L(x, y) = inf L(\gamma), \gamma : [a, b] \rightarrow M, \gamma(a) = x, \gamma(b) = y$.

4.2 Distance maps

The problem of computing distance maps on manifolds from multiple sources is defined as follows. The single source distance map problem is a particular case of the multiple source problem and will not be explicitly defined.

Definition 4.1 (Multiple Source Distance Map Problem - MSDMP)**.** Given a metric space $(\mathcal{M}, d_L)$ and a set of source points $\mathcal{S} \subset \mathcal{M}$ define a map $d_{\mathcal{S}}(x) = d_L(s, x)$ that associates to each $x \in \mathcal{M}$ the distance to a point $s \in \mathcal{S}$.

One way to solve the problem of computing distance maps is to simulate the propagation of a signal (the distance information) from the source points towards all other points in the space. For the sake of argument, consider that the signal propagates with constant speed $|v| = 1$ where v is a velocity vector. In nature, signals like light and sound travel according to the *Fermat Principle*, i.e., by choosing the quickest path. Hence, the signal that propagates from a point source s will choose the direction of propagation that causes the greatest increase in distance. This direction is the *gradient of the distance function* $||\nabla d_{\mathcal{S}}||$. One can show that the *gradient of the distance function* is parallel to the tangent of the minimal geodesic $\gamma(t)'$. Thus, as $||\gamma(t)'|| = 1$, it is also possible to show (see [2]) that for a manifold $\mathcal{M}$,

$$||\nabla_{\mathcal{M}} d_V|| = 1 \tag{4.1}$$

where $||\nabla_{\mathcal{M}} d_V|| = 1$ is the *intrinsic gradient* of distance function on $\mathcal{M}$. Equation 4.1 is the *Eikonal Equation* seen before defined on manifolds.

4.3 Discrete Distance Maps

Many graphical objects are described by 2-d manifolds, embedded in a three dimensional space, that is, *embedded surfaces.* Surfaces are usually represented by piecewise linear representations known as *triangle meshes* $T = (V, E, F)$, where V is the set of vertices, E

the set of edges and F the set of faces. We now state the problem of computing distance maps on a triangulated mesh T.

Definition 4.2 (Multiple Source Discrete Distance Map Problem - MSDDMP)**.** Let T be a triangulated mesh. Given a subset $S \subset V$ called source set, compute a map $d_S(v) : V \to \Re$ that associates to each $v \in V$ the minimum distance to S.

In the literature, one can find many ways to solve the MSDDMP problem as it was summarized in Chapter 3. Our method is closer in essence to the Fast Marching method and it also uses a modification of its update function. Hence, for the sake of comprehensiveness, we present it in the next subsections.

4.4 Fast Marching algorithm

Algorithm 4.1 Fast Marching algorithm [21, 2]

```
Require: Triangular mesh $(V, F)$, source vertices $S \subset V$
Ensure: Distances map $d : V \to \mathbb{R}$
 1: $\forall v \in V : d(v) \Leftarrow \infty$, $\forall s \in S : d(s) \Leftarrow 0$
 2: $R \Leftarrow S$
 3: $G \Leftarrow V \setminus R$
 4: $B \Leftarrow \{\}$
 5: while $B \neq V$ do
 6:     $v \Leftarrow \arg\min_{v \in R} d(v)$
 7:     $R \Leftarrow R \setminus \{v\}$
 8:     $B \Leftarrow B \cup \{v\}$
 9:     for all $v_0 \in \mathcal{N}(v)$ do
10:         $G \Leftarrow G \setminus \{v_0\}$
11:         $R \Leftarrow R \cup \{v_0\}$
12:         for all $(v_0, v_1, v_2) \in F(v_0)$ do
13:             update$(d, v_0, v_1, v_2)$
14:         end for
15:     end for
16: end while
17: return $d$
```

The Fast Marching algorithm (Algorithm 4.1), simulates the propagation of the distance information in a discrete set. It is possible to make an analogy with the propagation of fire in a grass land, what is called here in the text as *fire propagation.* In the beginning, each source vertex $s \in S$ has its distance fixed to zero ($d(s) = 0, s \in S$) and is inserted on a priority queue R (red vertices), whose priority is defined in terms of the smallest distance. All other vertices $v \notin S$ are labeled with distance equal to infinity ($d(v) = \infty$).

At each step, one vertex v is selected from the priority queue and inserted in the list of processed vertices B (black vertices); for all neighbor vertices $v_0 \in \mathcal{N}(v)$ of v, all triangles incident to it $F(v_0)$ are updated using Algorithm 4.2, which was proposed by Kimmel and Sethian in [10]. The new vertices on the updated triangles are inserted in the priority queue R and the algorithm proceeds until all vertices are processed, that is, $B = V$.

Algorithm 4.2 Calculate the planar update to (v_0, v_1, v_2) [21, 2]

```
Require: Distance map $d : V \to \mathbb{R}$, triangular face $(v_0, v_1, v_2) \in F$
Ensure: A new distance $d(v_0)$
 1: $x_1 \Leftarrow v_1 - v_0$
 2: $x_2 \Leftarrow v_2 - v_0$
 3: $X \Leftarrow (x_1, x_2)$
 4: $t \Leftarrow (d(v_1), d(v_2))^T$
 5: $Q \Leftarrow (X^T X)^{-1}$
 6: $p \Leftarrow \dfrac{1^T Q t + \sqrt{(1^T Q t)^2 - 1^T Q 1 \cdot (t^T Q t - 1)}}{1^T Q 1}$
 7: $n \Leftarrow XQ(t - 1)$
 8: if $QX^T n < 0$ then
 9:     $d(v_0) \Leftarrow \min\{d(v_0), p\}$
10: else
11:     $d(v_0) \Leftarrow \min\{d(v_0), d(v_1) + \|x_1\|, d(v_2) + \|x_2\|\}$
12: end if
```

The update step (see Algorithm 4.2) is one of the distinctiveness of the Fast Marching method. It yields a linear local approximation to the continuous distance and guarantees that the solution obeys both the *consistence condition* and the *monotonicity condition* ($QX^T n < 0$) of the signal propagation. When considered together, they guarantee that, for a given triangle, defined by three vertices x_0, x_1, x_2, if x_0 and x_1 are closer to the source set, then x_3 cannot be reached by the signal before x_0 and x_1, producing a correct solution for the Eikonal Equation. In geometrical terms this means that the triangles in the mesh cannot be obtuse [2]. One solution for dealing with meshes that have such triangles is to subdivided or to locally unfold the mesh [10]. Further in the text, we show that our method yields correct results even when dealing with meshes with obtuse triangles.

Both exact and Fast Marching based methods rely, to a lesser or greater extent, on the use of priority queues which makes it tough to devise their corresponding parallel versions. Thus, we decided to completely abandon the use of priority queues in the proposed method. Instead of fixing the final distance for the closest vertex, at each iteration, we update the distances on subsets of vertices that are good candidates for the propagation of distance information. More precisely we take advantage of the discrete topological structure of the mesh which can be decomposed in rings around the source vertices to propagate the information simultaneously and in an independent way in multiple phases until the distances converge.

Chapter 5

Proposed Method

Inspired by the intuitive propagation idea of the Fast Marching algorithm, we developed an algorithm that simulates the fire propagation as a set of sequential iterations. The key idea to simulate this behavior is to sort the vertices by levels (rings), which mark off the limit scope of the fire propagation.

In this Chapter, we define the rings propagation and also provide an upper bound to the number of rings from a source vertex, for a given triangle mesh. Then, we describe the proposed algorithm and its parallel version. Finally, we present a complexity analysis and explain how to leverage the properties of our method to solve the multi-source version of the problem.

5.1 Rings propagation

The rings propagation relies on the concept of rings; essentially, a ring in a graph G is composed of all vertices v of G, such that the length of the shortest path p, from v to the source s, in the associated unweighted graph G', is constant and equal to k. For example, the ring $V_1 \in V$ is considered a level one ring because the shortest path from each $v \in V_1$ to s is comprised by only one edge. We can extend this definition to rings $V_2, V_3, ..., V_\rho$, where ρ is the number of rings on a mesh.

Definition 5.1 (Vertices at ring ***r***)**.** Let $T = (V, E, F)$ be a triangular mesh and $S \subset V$ the set of source vertices; $V_r \subset V$ is the set of vertices at ring r and is defined by the recurrence relation

$$V_r = \left\{ v \in \mathcal{N}(V_{r-1}) : v \notin \bigcup_{r'=0}^{r-1} V_{r'} \right\}$$

where $V_0 = S$. We can define the following properties of ring:

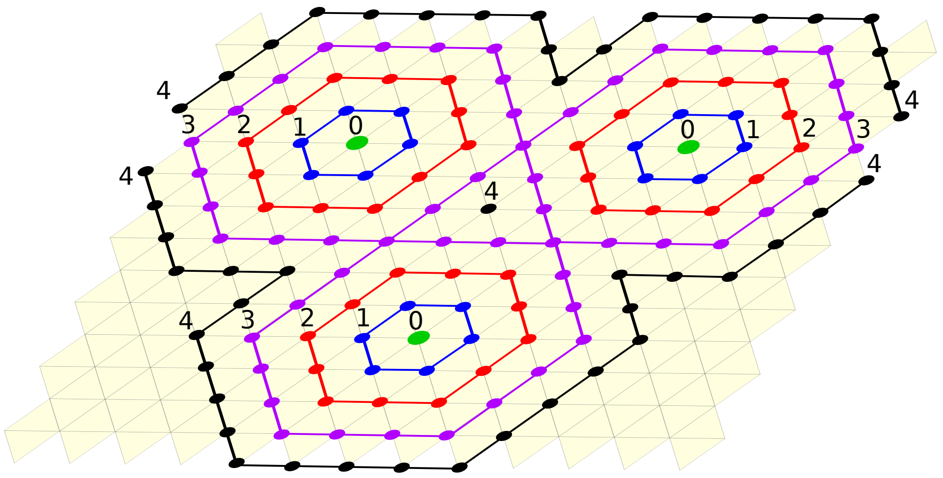


Figure 5.1: Rings propagation with 3 samples; in **green $V_0 = S$**; in **blue V_1**, in **red V_2**, in **purple V_3** and in **black V_4**.

1. $\bigcup_{r=0}^{\rho} V_r = V$
2. $\forall r, r' \in [0, \rho], r \neq r' : V_r \cap V_{r'} = \emptyset$
3. $\sum_{r=0}^{\rho} |V_r| = |V|$

where ρ is the number of rings from $S \in V$ in a triangular mesh.

Figure 5.1, shows the rings at different levels, defining a ring propagation from three sources. The color blue marks the vertices that belong to ring V_1, in red are the vertices that belong to V_2, in purple and black the vertices that belong to V_3 and V_4 respectively.

Theorem 5.1. [Number of rings ρ] Let $T = (V, E, F)$ be a triangular mesh, $s \in V$ be a source vertex such that it is possible order all rings V_r by their number of vertices in a crescent way. Then the number of rings ρ from s satisfies $\rho = O(\sqrt{n})$, where $n = |V|$.

A proof to Theorem 5.1 is give in Appendix A, in the same way, we can prove that for a triangular mesh $T = (V, E, F)$ and a set of sources vertices $S \subset V$, the number of rings is $\rho = O\left(\sqrt{\frac{n}{m}}\right)$, where $m = |S|$.

5.2 Ring Propagation-based algorithm

We present our approach based on the frontier propagation which is determined by the propagation of distance information through the meshes' rings.

In the proposed method, we define a band formed by a set of consecutive rings of the mesh, around each ring V_r. The band will have its distances updated for a number of iterations K. For a given predefined K we show that the distances converge.

Algorithm 5.1 Proposed method: Parallel Ring Propagation (PRP).

```
Require: Triangular mesh $(V, F)$, source vertices $S \subset V$, $V_r$ and $r \in [0, \rho]$
Ensure: Distances map $d : V \to \mathbb{R}$
 1: $d_k$ current distance map at iteration $k$
 2: $\forall v \in V : d_0(v) \Leftarrow \infty$
 3: $\forall s \in S : d_0(s) \Leftarrow 0$
 4: $d_0 \Leftarrow d_1$
 5: for $k \Leftarrow 1$ to $K$ do
 6:   for all $v_0 \in \bigcup_{r=\lfloor k/2 \rfloor}^{k} V_r$ (in parallel) do
 7:     for all $(v_0, v_1, v_2) \in F(v_0)$ do
 8:       update$(d_k, v_0, v_1, v_2)$
 9:     end for
10:   end for
11: end for
12: return $d \Leftarrow d_K$
```

Algorithm 5.1 implements the fire propagation, updating in each k iteration, the current distance map d_k, as a function of the previous distance map d_{k-1} at iteration $k-1$, for all vertices; (see Algorithm 5.2). This propagation requires a number K of iterations to make the distance map converge to the minimum error; K is proportional to the number of rings ρ from the sources vertices.

Algorithm 5.2 Planar update FM modified.

```
Require: Distance map $d_k : V \to \mathbb{R}$, triangular face $(v_0, v_1, v_2) \in F$
Ensure: A new distance $d_k(v_0)$
 1: $x_1 \Leftarrow v_1 - v_0$
 2: $x_2 \Leftarrow v_2 - v_0$
 3: $X \Leftarrow (x_1, x_2)$
 4: $t \Leftarrow (d_{k-1}(v_1), d_{k-1}(v_2))^T$
 5: $Q \Leftarrow (X^T X)^{-1}$
 6: $p \Leftarrow \dfrac{1^T Q t + \sqrt{(1^T Q t)^2 - 1^T Q 1 \cdot (t^T Q t - 1)}}{1^T Q 1}$
 7: $n \Leftarrow XQ(t - 1)$
 8: if $QX^T n < 0$ then
 9:   $d_k(v_0) \Leftarrow \min\{d_{k-1}(v_0), p\}$
10: else
11:   $d_k(v_0) \Leftarrow \min\{d_{k-1}(v_0), d_{k-1}(v_1) + \|x_1\|, d_{k-1}(v_2) + \|x_2\|\}$
12: end if
```

One of the most important operations in the rings propagation algorithm is the distance update operation. Our distance update is a modification of the distance update of the Fast Marching method, considering a previous distance map d_{k-1}. Consequently, it is possible to update all vertices independently (see Line 6 in Algorithm 5.1), since each vertex v_0 updates its new distance $d_k(v)$ according to the distance map d_{k-1} computed in the preceding iteration $k-1$.

We can take advantage of the mesh's ring structure to sort the vertices and avoid unnecessary updates in a given iteration k. It is only necessary to update the vertices $v \in \bigcup_{r=1}^{k} V_r$ because the distances of the sources vertices $V_0 = S$ are equal to zero and the distances of the vertices $v \in \bigcup_{r=k+1}^{K} V_r$ are $d(v) = \infty$; they have not been reached yet by the fire propagation and all vertices in their neighborhood have infinite distance values.

Theorem 5.2. [Maximum number of iterations K] Let $T = (V, E, F)$ be a triangular mesh, $s \in V$ be a source vertex such that it is possible order all rings V_r by their number of vertices in a crescent way. Then, the maximum of iterations is $K = O(\sqrt{n})$, where $n = |V|$.

An sketch of the proof of Theorem 5.2 is giving in the Appendix A.2 and also it is supported by the experimental evaluation in Chapter 6. Based on the Theorem 5.2, the proposed PRP algorithm has an upper bound for the estimate of the number of iterations that are necessary to update the distances of all vertices of the mesh. Furthermore, the PRP algorithm sets an updating layer, because it is not necessary to update all the vertices $v \in \bigcup_{r=1}^{k} V_r$. As the frontier travels forward, there will be vertices whose distances have already converged; these vertices are located in the first rings of the propagation. Based on the proof A.2, by Lemma A.2, the number of iterations k to update all vertices $v \in V_r$ is $k = 2r - 1$; we can observe that in the iteration k the vertices in the ring $r = \frac{k-1}{2}$ have computed their final distance; then we can defined an updating layer between all vertices included in the rings $r = \left[\left\lfloor \frac{k}{2} \right\rfloor, k\right]$; Algorithm 5.1 updates all vertices in this layer (see line number 6).

5.3 Parallelization

The Parallel Ring Propagation algorithm completely eliminates the dependency of the priority queue which is necessary for the classical Fast Marching algorithm. Also, we introduced an upper bound to estimate the number of iterations that are necessary for the convergence of our algorithm.

These advantages of our method permits us to reuse the calculations of the previous distance maps. Furthermore, as the calculation of distances is independent for each vertex

v_0, the loop (see Line 6 in Algorithm 5.1) is highly parallelizable in SIMD and GPU processors.

5.4 Algorithm Complexity and Comparison

In a first analysis of the Parallel Ring Propagation algorithm, the number of threads T will not be considered, because T is a constant that potentially will improve the performance of the PRP algorithm. We further discuss the impact of T, which will have a strong effect when $T \approx \sqrt{n}$.

The Parallel Ring Propagation (Algorithm 5.1) sorts the vertices by levels (the rings) and updates the vertices in each iteration. We claim that the number of levels is $O(\sqrt{n})$ by Theorem 5.1, and that the vertices sorting by levels has complexity of $O(n)$.

We start the analysis of the complexity of the PRP algorithm with the number of operations $f(K)$ in the main loop of the Algorithm 5.1:

$$f(K) = c\sum_{k=1}^{K} \sum_{r=\lfloor \frac{k}{2} \rfloor}^{k} |V_r| \tag{5.1}$$

where K is the number of iterations and c is a constant.

Let $r' = \arg\max\limits_{r\in[0:\rho]} |V_r|$ be the ring with the highest number of vertices, and $|V_{r'}|$ the maximum number of vertices in r'. Thus, we affirm that:

$$c\sum_{k=1}^{K} \sum_{r=\lfloor \frac{k}{2} \rfloor}^{k} |V_r| \leq c\sum_{k=1}^{K} \sum_{r=\lfloor \frac{k}{2} \rfloor}^{k} |V_{r'}|$$

$$f(K) \leq c|V_{r'}|\sum_{k=1}^{K} \sum_{r=\lfloor \frac{k}{2} \rfloor}^{k} 1$$

$$f(K) \leq c|V_{r'}|\sum_{k=1}^{K} \frac{k}{2} = \frac{c}{2}|V_{r'}|\sum_{k=1}^{K} k = \frac{c}{4}|V_{r'}|K(K+1)$$

$$f(K) = O\left(|V_{r'}|(K^2 + K)\right)$$
$$f(K) = O\left(|V_{r'}|K^2\right)$$

and by Theorem 5.2, $K = O(\sqrt{n})$, consequently the number of operations f in function of n is:

$$f(n) = O\left(|V_{r'}|n\right)$$

For the sake of argument, we consider $|V_{r'}| \leq cr'$ where c is a constant (see Appendix A.1), and in the worst case we have that $r' = \rho$ and $\rho = O(\sqrt{n})$. Hence, we can conclude that the complexity of the main loop is:

$$f(n) = O(n\sqrt{n}). \tag{5.2}$$

This complexity is impacted by the number of sources m; as the number of rings, for $m > 1$ sources, is $\rho = O\left(\frac{\sqrt{n}}{\sqrt{m}}\right)$ and the number of iterations $K = O\left(\frac{\sqrt{n}}{\sqrt{m}}\right)$, we can affirm that the complexity of the PRP algorithm with $m > 1$ sources is $O\left(\frac{n\sqrt{n}}{\sqrt{m}}\right)$.

5.4.1 Application: Farthest Point Sampling

The Farthest Point Sampling (FPS) is a generic algorithm introduced by Eldar [7]. It generates a regular sampling and calculates the farthest vertex and inserts in S a new sample in each iteration, as described in the Algorithm 5.3.

Algorithm 5.3 Farthest Point Sampling (FPS)

```
Require: Triangular mesh $(V, F)$, source vertex $s \in V$, number of samples $m$
Ensure: Sampling vertices set $S \subset V$
1: $S \Leftarrow \{s\}$
2: while $|S| < m$ do
3:     $d \Leftarrow$ compute_geodesics($V, F, S$)
4:     $s \Leftarrow \arg\max_{v \in V} d(v)$
5:     $S \Leftarrow S \cup \{s\}$
6: end while
7: return $S$
```

The FM algorithm (Algorithm 4.1) has a complexity of $O(n \log n)$ similar to the Dijkstra algorithm. The complexity of the FPS algorithm without taking into consideration the cost of calculating distances is $O(mn)$, where m is the number of samples S. However, to compute a sub-sampling in a triangular mesh, we must compute the distance map with the FM algorithm in each iteration. Hence the FPS algorithm complexity is $O(mn \log n)$.

The number of operations $f(n)$ of the FPS algorithm using the PRP algorithm is

$$f(n) = \sum_{i=1}^{m} \left(c_1 \frac{n\sqrt{n}}{\sqrt{i}} + c_2 n \right) \tag{5.3}$$

where c_1 and c_2 are constants, the terms inside the sum in Equation 5.3 represent the operations in each iteration: the PRP algorithm (Algorithm 5.1) which computes the geodesic distance map (first term) and the selection of the vertex with the maximum distance from the i sources (second term). Equation 5.4 reduces Equation 5.3:

$$\sum_{i=1}^{m} \left(c_1 \frac{n\sqrt{n}}{\sqrt{i}} + c_2 n \right) \leq c \sum_{i=1}^{m} \frac{n\sqrt{n}}{\sqrt{i}} = cn\sqrt{n} \sum_{i=1}^{m} \frac{1}{\sqrt{i}} \tag{5.4}$$

Proof. We can prove that:

$$\sum_{i=1}^{m} \frac{1}{\sqrt{i}} = O(\sqrt{m}) \tag{5.5}$$

as a consequence of the following facts:

$$\sum_{i=1}^{m} \frac{1}{\sqrt{i}} = \frac{1}{1} + \frac{1}{\sqrt{2}} + \frac{1}{\sqrt{3}} + \frac{1}{2} + \frac{1}{\sqrt{5}} + \frac{1}{\sqrt{6}} + \frac{1}{\sqrt{7}} + \frac{1}{\sqrt{8}} + \frac{1}{3} + \frac{1}{\sqrt{10}} + \cdots + \frac{1}{\lfloor\sqrt{m}\rfloor} + \cdots + \frac{1}{\sqrt{m}}$$

we use the fact that different $(k+1)^2 - k^2 = 2k+1$ where $k \in \mathbb{N}$, to limit:

$$\sum_{i=1}^{m} \frac{1}{\sqrt{i}} \leq \sum_{i=1}^{\lfloor\sqrt{m}\rfloor} \frac{2i+1}{i} \tag{5.6}$$

$$\sum_{i=1}^{m} \frac{1}{\sqrt{i}} \leq 2\sqrt{m} - \sum_{i=1}^{\lfloor\sqrt{m}\rfloor} \frac{1}{i}$$

$$\sum_{i=1}^{m} \frac{1}{\sqrt{i}} \leq 2\sqrt{m} - H_{\lfloor\sqrt{m}\rfloor} \tag{5.7}$$

where $H_{\lfloor\sqrt{m}\rfloor} \sim \ln\lfloor\sqrt{m}\rfloor$ is a Harmonic number. Hence, we can affirm that:

$$\sum_{i=1}^{m} \frac{1}{\sqrt{i}} = O(\sqrt{m}). \tag{5.8}$$

□

Combining Equation 5.4 with Equation 5.5 we obtain

$$cn\sqrt{n}\sum_{i=1}^{m}\frac{1}{\sqrt{i}} = O(n\sqrt{n}\sqrt{m}) = O(\sqrt{m}n^{3/2}) \tag{5.9}$$

In this partial complexity analysis, we did not consider the effects of the number of threads in the parallelization.

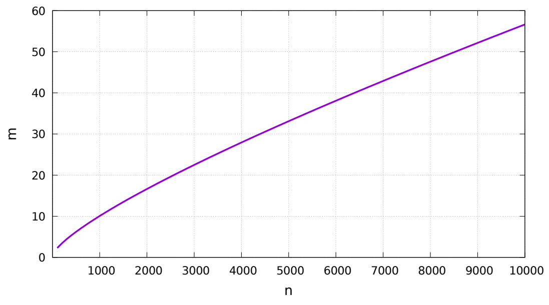
Algorithm	**Complexity**
Update Fast Marching	$O(1)$
Fast Marching	$O(n\log n)$
Farthest Point Sampling	$O(mn\log n)$
Update Fast Marching modified	$O(1)$
Parallel Ring Propagation	$O\left(\sqrt{\frac{n}{m}}\frac{n}{T}\right)$
Farthest Point Sampling with PRP	$O\left(\sqrt{m}\frac{n^{3/2}}{T}\right)$

Table 5.1: Comparison of complexity algorithm analysis.

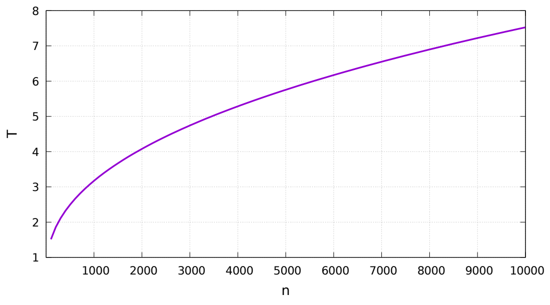
The computational complexity analysis of FPS is $O(mn\log n)$ using FM, whereas the parallel version of FPS using PRP has complexity of $O(\sqrt{m}n^{3/2})$. As the number of samples grows, there is an increase in the performance of the algorithm.

Table 5.1 summarizes the analyzed algorithms. We include the term T to the proposed algorithm because the parallelization is a feature in our algorithm, and also has impact in the FPS algorithm.

5.4.2 Lower bound to m and T



(a) *Function* $m_0 = \frac{n}{\log^2 n}$ *with* $c = 1$



(b) *Function* $T_0 = \frac{\sqrt{n}}{\log n}$ *with* $c = 1$

Figure 5.2: Lower bounds to m and T for $n \in [1 : 10000]$

The number of samples $m = |S|$ is less than the total vertices of the mesh $n = |V|$, i.e., $m < n$. Given the complexities calculated above, we can estimate the minimum number of samples m as a function of n, necessary to make the parallel version of the FPS algorithm, using the Parallel Ring Propagation, outperform the original FPS algorithm.

Notice that the calculation of the distances using a priority-queue-based Fast Marching algorithm is more efficient than our approach without parallelization, as the time complexity of PRP algorithm is $O\left(\frac{n^{3/2}}{\sqrt{m}}\right)$ with $m = 1$. Nevertheless, the PRP algorithm is highly parallelizable, because it discards the use of the priority queue. In addition, the performance increases according to the number of samples m, as shown in the analysis that follows.

The PRP algorithm is more efficient that FM algorithm for some $m_0 \leq m$ where $m < n$; we compare the complexities between the FM algorithm and the PRP algorithm with the next equation:

$$c_1 n \log n > c_2 \frac{n\sqrt{n}}{\sqrt{m_0}} \tag{5.10}$$

where c_1 and c_2 are constants. Reducing this comparison, we obtain

$$m_0 > \frac{cn}{\log^2 n} \tag{5.11}$$

where c depends on c_1 and c_2.

As the result does not contradict the condition $m < n$, we can conclude that when $m > \frac{cn}{\log^2 n}$, the PRP algorithm has a better performance, without considering the performance gained with the parallel implementation. Figure 5.2a, illustrates the behavior of Equation 5.11, for values of $n \in [1 : 10000]$; we can observe that for meshes with 10000 vertices, the FPS using the Rings Propagation algorithm outperforms the original FPS when the number of samples is greater than 60.

The previous analysis did not contemplate the potential of the Parallel Ring Propagation algorithm. The time complexity of our approach and the FPS algorithm using the PRP algorithm are shown in Table 5.1. The number of cores is T. In our approach, the update step is performed simultaneously until it reaches a band, which increases its size with the number of iterations. In this step, it is important that all the vertices update their distances before it proceeds to the next iterations.

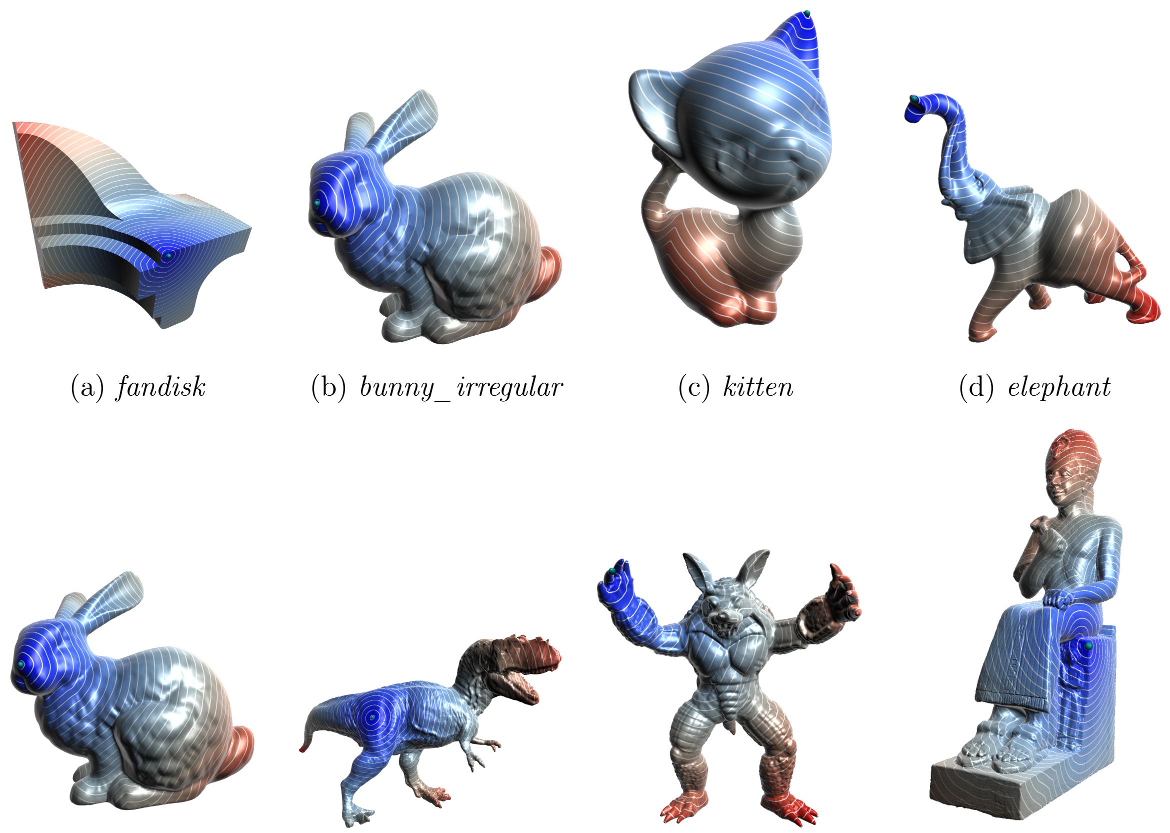
We can analyze a lower bound to the number of threads $T_0 \leq T$, in the same way we estimated m_0:

$$T_0 = \frac{c\sqrt{n}}{\sqrt{m}\log n} \tag{5.12}$$

The function associated to Equation 5.12 is ploted in Figure 5.2b, for $n \in [1 : 10000]$, considering $m = 1$; this function is a lower bound to the number of threads that are required. The parallelization in multicores and GPU improves considerably the time complexity of our approach against the original FM. In the Chapter 6, we prove the results of these analysis by presenting results including execution times, speedup and errors for both the PRP and the FPS based on PRP algorithms.

Chapter 6

Experimental evaluation



(a) *fandisk* (b) *bunny_irregular* (c) *kitten* (d) *elephant*

(e) *bunny* (f) *tyra* (g) *armadillo* (h) *ramesses*

Figure 6.1: Geodesic distances map for each mesh in Table 6.1, from $m = 1$ sources.

We implemented the proposed PRP algorithm using CUDA, a parallel computing platform and programming model developed by NVIDIA; we used the version 8.0 of the CUDA toolkit and the GCC 6.2 compiler. The experiments were performed using an Intel Core i7-6700HQ and a NVIDIA GeForce GTX 960M with 640 cuda cores. We implemented the classical version of the Fast Marching algorithm that uses a priority

Filename	Vertices	Serial Time (s)	GPU Time (s)	Speedup	Serial Error %	GPU Error %
fandisk	6475	0.154	0.006	27.327	1.066e+00	1.065e+00
bunny_irregular	7500	0.188	0.007	28.322	2.138e+00	2.101e+00
kitten	14472	0.335	0.016	20.718	1.134e+00	1.132e+00
elephant	24955	0.580	0.035	16.712	8.954e-01	8.912e-01
bunny	34835	0.803	0.034	23.425	9.578e-01	9.549e-01
tyra	100002	2.462	0.260	9.484	9.283e-01	9.004e-01
armadillo	172974	4.220	0.562	7.504	7.182e-01	6.991e-01
ramesses	826266	20.560	4.243	4.846	6.379e-01	4.584e-01

Table 6.1: Comparison times, mean absolute percent error and speedup.

queue, in order to compare the performance of our algorithm. To compare the accuracy of the algorithms we used the algorithm to compute exact geodesics [13], included in the MeshLP package [1].

The performance and accuracy are evaluated over the meshes shown in Figure 6.1. All experiments used single precision floating point arithmetic. Table 6.1 resumes the experiments of distance computation from $m = 1$ sources.

6.1 Performance

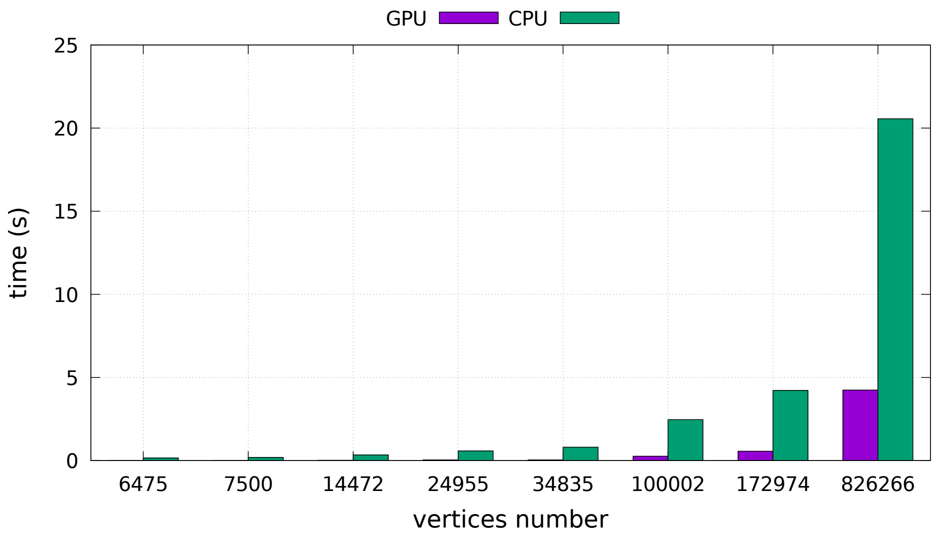


Figure 6.2: FM (CPU) and PRP (GPU) execution times from $m = 1$ sources.

Table 6.1 and Figure 6.2 show the execution times for the FM algorithm implemented in CPU, and the PRP algorithm implemented in GPU to compute the geodesics distances for each mesh, from $m = 1$ sources. It also Figure 6.3 presents the speedup of the PRP

[1]MeshLP package: https://github.com/areslp/matlab/tree/master/MeshLP/MeshLP, Geodesics code: http://code.google.com/p/geodesic/

algorithm over the FM algorithm. We can observe that the speedup decreases in meshes with a large number of vertices, but there are GPUs with more cuda cores and better theoretical performance with which these values will be outperformed.

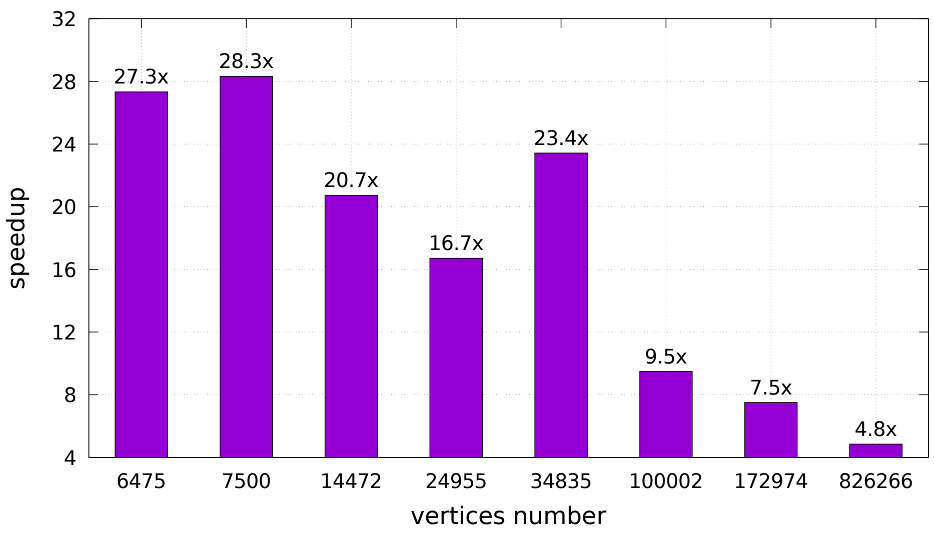


Figure 6.3: Parallel Ring Propagation speedup from $m = 1$ sources.

In Chapter 5, we showed the importance of the inverse relation between the complexity of the PRP algorithm and the number of sources m, in addition to the independence of distance computation in each iteration, which allows the parallel implementation.

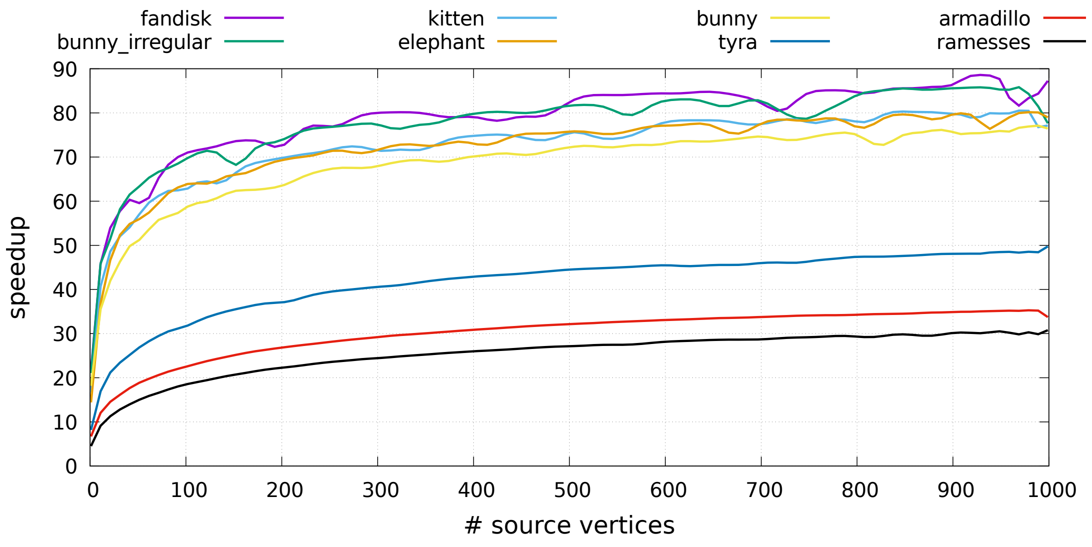
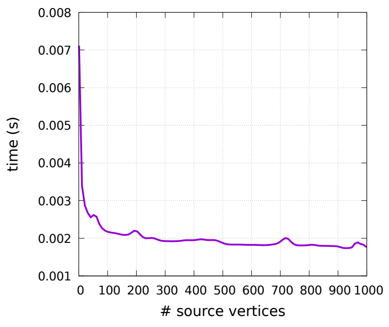
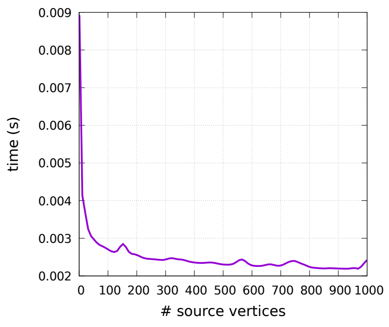


Figure 6.4: PRP speedup with $m \in [1 : 1000]$.

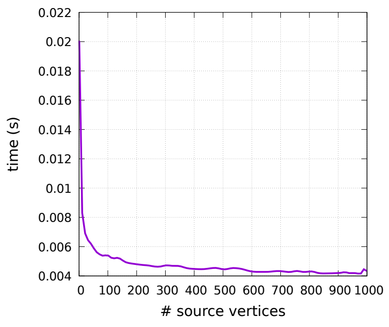
Figure 6.4 presents the speedup of the PRP algorithm for each mesh from m sources. Note that the speedup of meshes with large number of vertices increase with the number of sources m; this is an advantage to compute distances map from many sources. The execution times for the computation of the geodesics distances from $m = [1 : 1000]$ sources



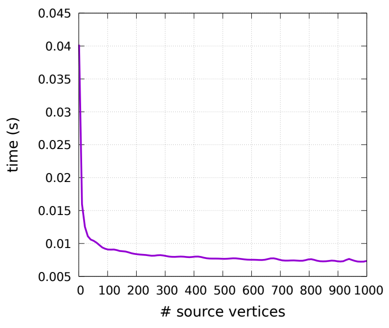
(a) *fandisk*



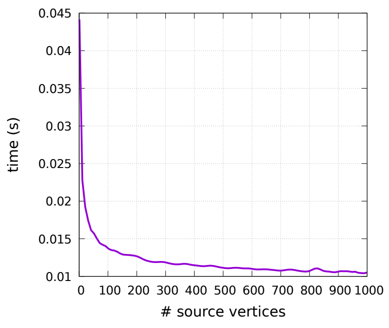
(b) *bunny_irregular*



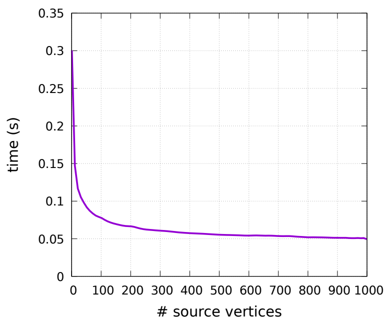
(c) *kitten*



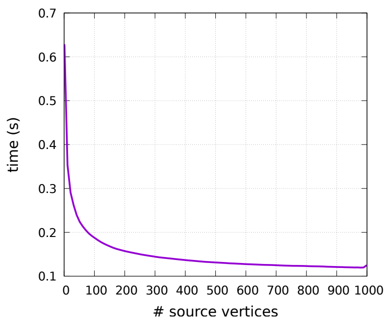
(d) *elephant*



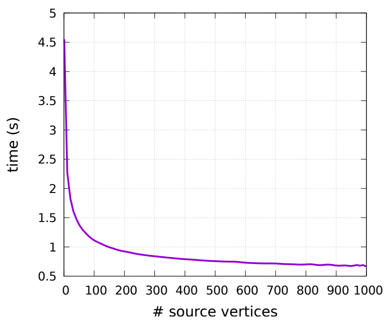
(e) *bunny*



(f) *tyra*



(g) *armadillo*



(h) *ramesses*

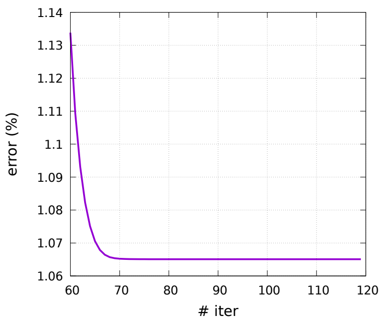
Figure 6.5: Execution times of the PRP algorithm with $m \in [1 : 1000]$ for each mesh.

for each mesh is shown in Figure 6.5. We observe that this execution times decrease when m increase; this behavior is as expected and we can verify the inverse relation between performance and number of sources in the proposed algorithm PRP.

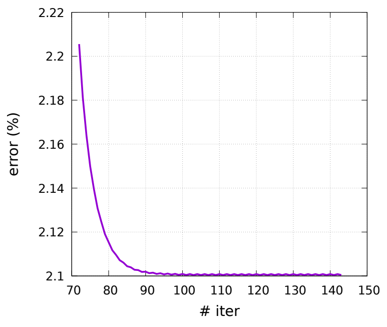
6.2 Accuracy

We tested the accuracy of the PRP algorithm, performing a comparison of *mean absolute percent error* (MAPE) between the FM algorithm and the PRP algorithm, summarized in Table 6.1. We can observe that PRP algorithm have a MAPE less than of the FM algorithm; this difference is even greater in the meshes with a large number of vertices.

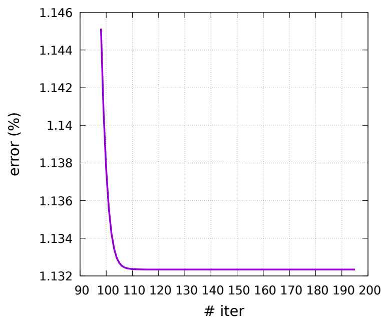
We evaluate the convergence of the PRP algorithm in relation to the number of iter-



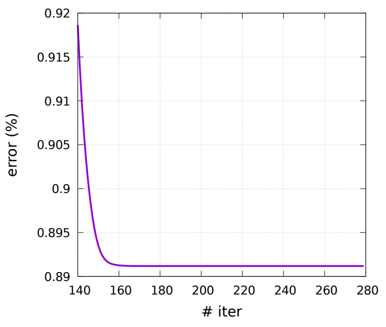
(a) *fandisk*



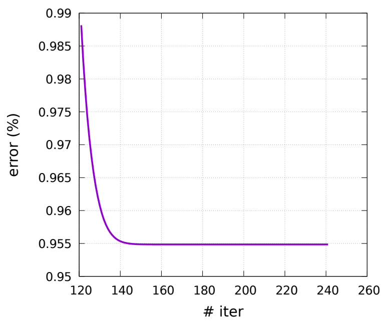
(b) *bunny_irregular*



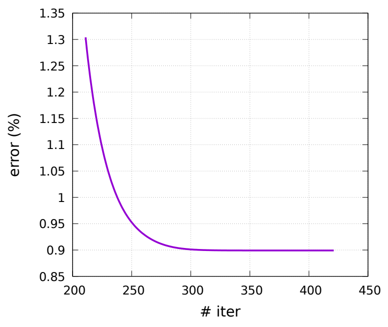
(c) *kitten*



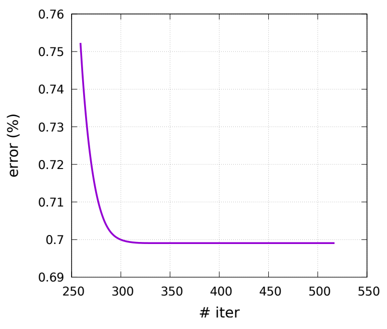
(d) *elephant*



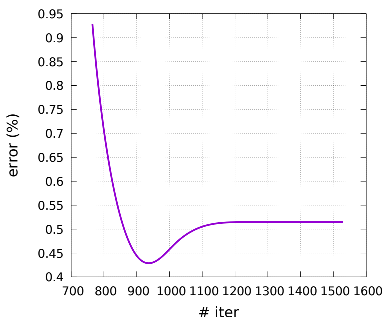
(e) *bunny*



(f) *tyra*



(g) *armadillo*



(h) *ramesses*

Figure 6.6: MAPE in function of number of iterations $k \in [\rho : 2\rho]$.

ations. In the previous Chapter, we have proved that the number of rings in a triangular mesh is $\rho \leq \sqrt{n}$, where n is the number of vertices; and we have established a relation between ρ and the maximum number of iterations K to compute the distances map.

Figure 6.6 presents the absolute mean errors in function to the number of iterations; each chart starts with an iteration $k = \rho$ (number of rings of the mesh) and finished in $K = 2\rho$, (Theorem 5.2). Note that the error decreases as the number of iterations is increased and it converges in some $k \in [\rho : 2\rho]$. In Appendix B.4 presents also the absolute mean errors with double precision.

We set the maximum number of iterations $K = 2\rho$, although this value depends on of the topology of the triangular mesh and the relation of the propagation of rings and the geodesic distance map. A more rigorous analysis of this relation has not been done

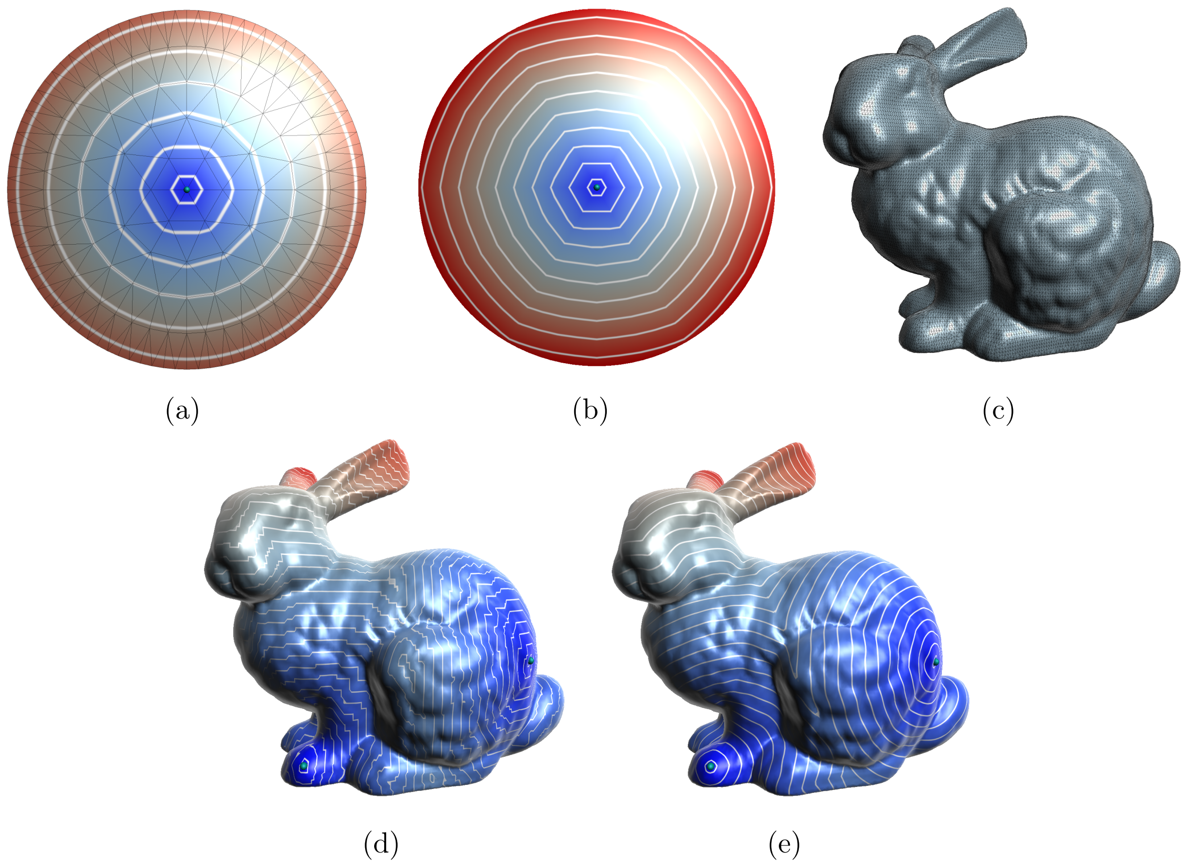


Figure 6.7: Rings map (rings propagation) in 6.7a and 6.7d, geodesics distances map in 6.7b and 6.7e. Bunny triangular mesh 6.7c.

yet in this work, but we have a preliminary analysis and we proved in Theorem 5.2 that maximum number of iterations $K \leq 2\rho$.

We believe that, when the distances defined by the rings (which are defined by the combinatorial topology of the mesh) are already close to the expected distances produced by the fire propagation of PRP, the maximum number of iterations will be ρ, because the rings propagation will be proportional to the real distance map.

The relationship between the rings propagation and the geodesic distance map is depicted in Figure 6.7. The rings propagation and the geodesic distance map over the circle mesh is proportional and similar, in this case, our algorithm only needs ρ iterations to compute the correct distance map and update the vertices in the last ring for each iteration. For the bunny mesh, we observe a similar behavior between the rings map and the distance map, but this mesh needs more iterations that the number of rings and recompute the distances not only in the last ring established for each iteration.

6.3 Farthest Point Sampling and Voronoi Diagrams

Figure 6.8 presents the speedup of the computation of m samples, using the FPS algorithm (Algorithm 5.3) with the GPU implementation of the PRP algorithm to compute the geodesics distances in each iteration. The FPS algorithm exploits the fact that in the PRP algorithm, the performance increases when the number of samples grows, because the FPS algorithm in each iteration computes a new sample, which is added to the sources set S. Figure 6.9 presents some visual results to the computation of m samples using the FPS algorithm with the geodesics distances computed with the PRP algorithm; also presents the voronoi regions with m sources, which were computed with the FPS algorithm.

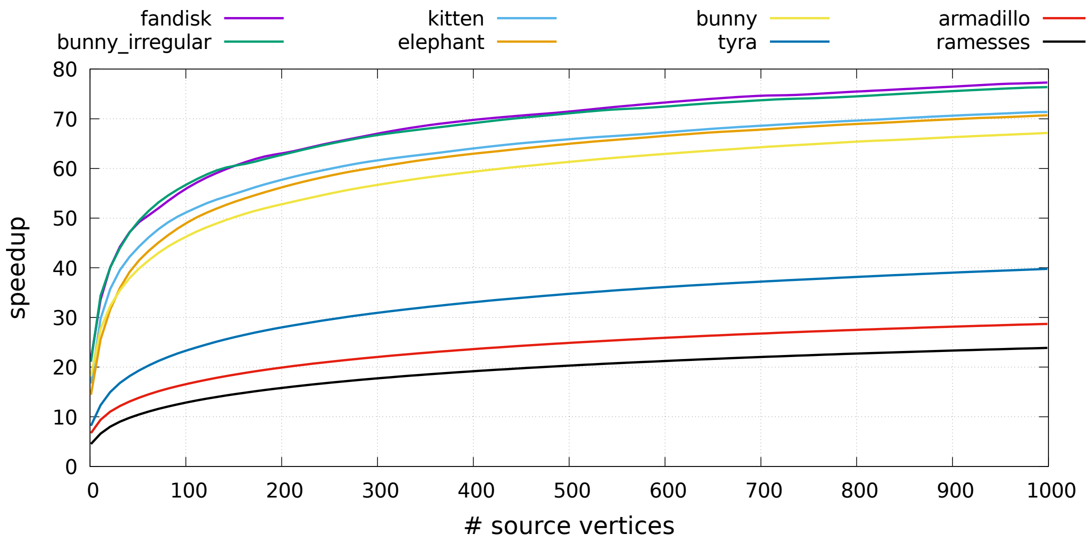
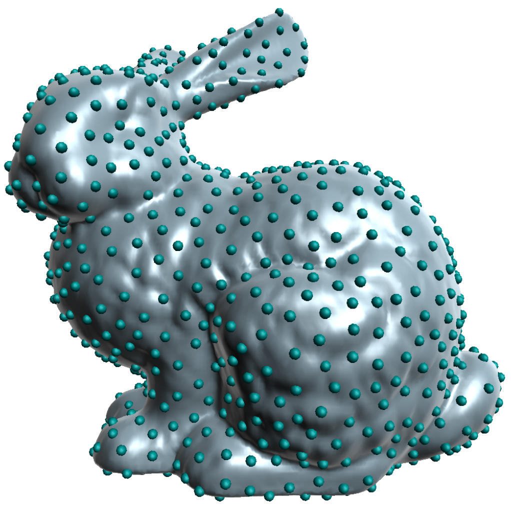
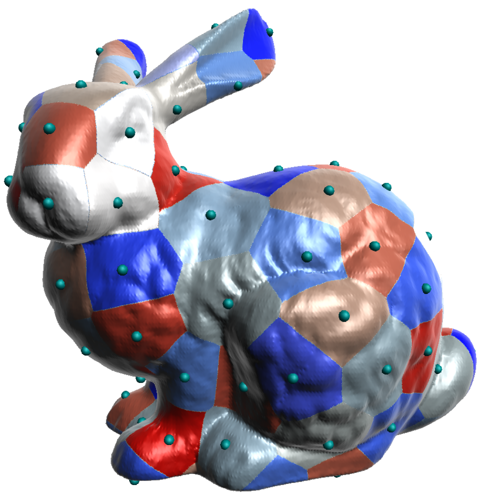
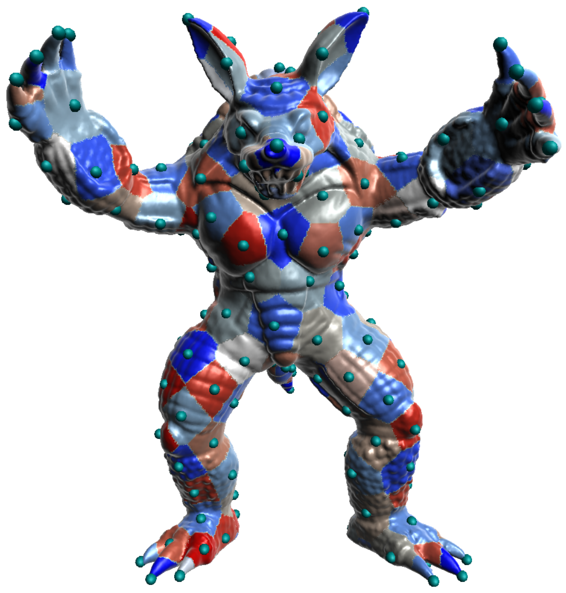


Figure 6.8: Farthest Point Sampling speedup with $m \in [1 : 1000]$.


(a)


(b)


(c)

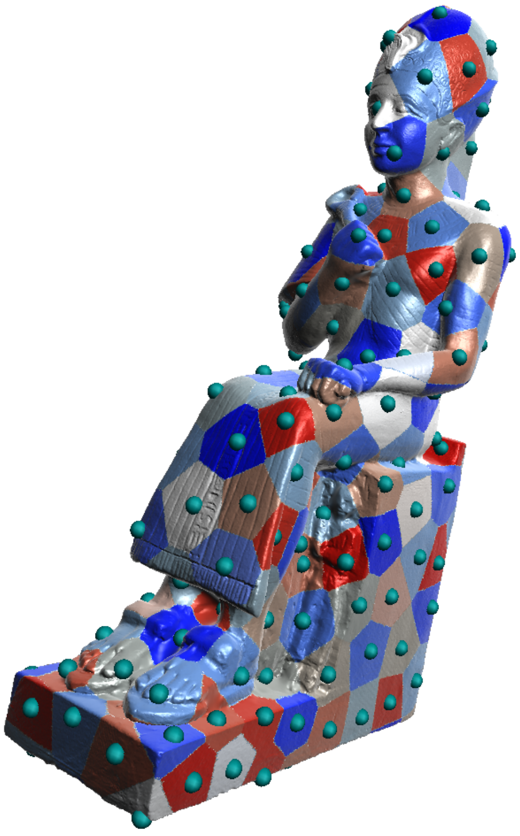

(d)

Figure 6.9: Computation of $m = 1000$ samples to bunny mesh 6.9a. Voronoi regions in 6.9b, 6.9c and 6.9d; from the calculation of $m = 100$ samples with the FPS algorithm.

Chapter 7

Conclusion

We presented an efficient and highly parallelizable algorithm to compute approximate distance maps on triangular meshes with time complexity of $O\left(\sqrt{\frac{n}{m}}\frac{n}{T}\right)$. The experiments show that the distances computed have error around the 1% from the exact method.

The main advantages of our approach are:

- **Simplicity:** Our approach does not require complicated pre-processing steps that could lead to introducing some error in the accuracy, for example, parameterization.
- **Highly parallelizable:** Because the distances in each iteration can be computed simultaneously and it does not use a priority queue. Furthermore, the proposed method improves the speedup considerably when used to solve the multisource problem.
- **Accuracy:** Our method achieves an accuracy similar to the classical Fast Marching method.

From the experiments, we can conclude that our method, Parallel Ring Propagation, achieves good speedup values without any pre-processing time. Especially for problems where multiple sources are required and queries will not be performed later, such as the Farthest Point Sampling algorithm, the speedup increases from 20 to 70, as the number of sources increases. This algorithm has a time complexity of $O\left(\sqrt{m}\frac{n^{3/2}}{T}\right)$ using the Parallel Ring Propagation algorithm.

The presented method is not the best to calculate intensive queries of pair-wise distances, but if the pairs are close to each other, the computation of the distances is faster, because the propagation is bounded by a certain distance radius.

Finally, as future works, a more rigorous analysis of the number of iterations required by our algorithm shall be developed and also, we plan perform a robustness analysis of the proposed method, considering mesh quality criteria.

References

[1] Bennour, J.; l. Dugelay, J. Protection of 3d object visual representations. In *2006 IEEE International Conference on Multimedia and Expo* (July 2006), pp. 1113–1116. 1, 3

[2] Bronstein, A.; Bronstein, M.; Kimmel, R. *Numerical Geometry of Non-Rigid Shapes*, 1 ed. Springer Publishing Company, Incorporated, 2008. 9, 10, 11

[3] Bronstein, A. M.; Bronstein, M. M.; Kimmel, R. Generalized multidimensional scaling: a framework for isometry-invariant partial surface matching. *Proc. National Academy of Sciences (PNAS) 103*, 5 (January 2006), 1168–1172. 1, 3

[4] Crane, K.; Weischedel, C.; Wardetzky, M. Geodesics in heat: A new approach to computing distance based on heat flow. *ACM Trans. Graph. 32*, 5 (Oct. 2013), 152:1–152:11. 1, 3, 7

[5] Danielsson, P.-E. Euclidean distance mapping. *Computer Graphics and Image Processing 14*, 3 (1980), 227 – 248. 7

[6] do Carmo, M. P. *Differential geometry of curves and surfaces.* Prentice Hall, 1976. 8

[7] Eldar, Y.; Lindenbaum, M.; Porat, M.; Zeevi, Y. Y. The farthest point strategy for progressive image sampling. In *Pattern Recognition, 1994. Vol. 3 - Conference C: Signal Processing, Proceedings of the 12th IAPR International Conference on* (Oct 1994), pp. 93–97 vol.3. 18

[8] Hamza, A. B.; Krim, H. *Geodesic Object Representation and Recognition.* Springer Berlin Heidelberg, Berlin, Heidelberg, 2003, pp. 378–387. 1, 3

[9] Hysing, S.; Turek, S. *The Eikonal Equation: Numerical Efficiency Vs. Algorithmic Complexity on Quadrilateral Grids.* Ergebnisberichte angewandte Mathematik. Techn. Univ., 2004. 7

[10] Kimmel, R.; Sethian, J. A. Computing geodesic paths on manifolds. In *Proc. Natl. Acad. Sci. USA* (1998), pp. 8431–8435. 2, 5, 6, 11

[11] Kurtek, S.; Klassen, E.; Ding, Z.; Avison, M. J.; Srivastava, A. *Parameterization-Invariant Shape Statistics and Probabilistic Classification of Anatomical Surfaces.* Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 147–158. 1, 3

[12] Martínez, D.; Velho, L.; Carvalho, P. C. Computing geodesics on triangular meshes. *Comput. Graph. 29*, 5 (Oct. 2005), 667–675.

[13] Mitchell, J. S. B.; Mount, D. M.; Papadimitriou, C. H. The discrete geodesic problem. *SIAM J. Comput. 16*, 4 (Aug. 1987), 647–668. 1, 2, 3, 5, 6, 24

[14] Morera, D.; Carvalho, P. *Geodesic-based Modeling on Manifold Triangulations*. Tese de Mestrado em Computacao Grafica - IMPA. IMPA, 2006.

[15] Qian, J.; Zhang, Y.; Zhao, H. Fast sweeping methods for eikonal equations on triangular meshes. *SIAM Journal on Numerical Analysis 45*, 1 (2007), 83–107. 7

[16] Rabin, J.; Peyré, G.; Cohen, L. D. *Geodesic Shape Retrieval via Optimal Mass Transport*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 771–784. 1, 3

[17] Sedgewick, R.; Vitter, J. S. Shortest paths in euclidean graphs. *Algorithmica 1*, 1 (1986), 31–48.

[18] Sethian, J. A. *Level set methods and fast marching methods : evolving interfaces in computational geometry, fluid mechanics, computer vision, and materials science*. Cambridge monographs on applied and computational mathematics. Cambridge University Press, Cambridge, GB, 1999. Première édition parue sous le titre : Level set methods, 1996. 6

[19] Sloan, P.-P. J.; Rose, III, C. F.; Cohen, M. F. Shape by example. In *Proceedings of the 2001 Symposium on Interactive 3D Graphics* (New York, NY, USA, 2001), I3D '01, ACM, pp. 135–143. 1, 3

[20] Surazhsky, V.; Surazhsky, T.; Kirsanov, D.; Gortler, S. J.; Hoppe, H. Fast exact and approximate geodesics on meshes. In *ACM SIGGRAPH 2005 Papers* (New York, NY, USA, 2005), SIGGRAPH '05, ACM, pp. 553–560. 1, 3, 6

[21] Weber, O.; Devir, Y. S.; Bronstein, A. M.; Bronstein, M. M.; Kimmel, R. Parallel algorithms for approximation of distance maps on parametric surfaces. *ACM Trans. Graph. 27*, 4 (Nov. 2008), 104:1–104:16. 7, 10, 11

APPENDIX A – Proofs

A.1 Theorem 5.1

Theorem 5.1. [Number of rings ρ] Let $T = (V, E, F)$ be a triangular mesh, $s \in V$ be a source vertex such that it is possible order all rings V_r by their number of vertices in a crescent way. Then the number of rings ρ from s satisfies $\rho = O(\sqrt{n})$, where $n = |V|$.

Proof. Let a triangular mesh $T = (V, E, F)$ and the sets of vertices V_r at ring $r \in [1 : \rho]$. We can stablish that:

$$|V_r| - |V_{r-1}| \geq c \tag{A.1}$$

where $c \geq 1$ and $V_0 = \{s\}$ is the set containing the source vertex s. Without loss of generality we can choose a constant $c = 1$. Then we have

$$|V'_r| = |V'_{r-1}| + 1 \tag{A.2}$$

where V'_r is a new crescent distribution of rings with the minimum difference such that $V'_r \leq V_r$, $r \in [1 : \rho']$, note that $\rho \leq \rho'$; then, we can solve this recurrence: $|V_r| = r$ for all $r \in [1 : \rho']$. Now, by $\sum_{r=0}^{\rho'} |V'_r| = n$ we have:

$$1 + \sum_{r=1}^{\rho'} r = n$$

$$1 + \frac{\rho'(\rho' + 1)}{2} = n$$

$$\rho^2 \leq 2 + \rho'^2 + \rho' = 2n$$

$$\rho \leq O(\sqrt{n})$$

□

A.2 Theorem 5.2

To prove Theorem 5.2, we need the next lemmas:

Lemma A.1. Let $T = (V, E, F)$ be a triangular mesh, and $s \in V$ a source vertex; the number of rings ρ is $\Omega(\log n)$.

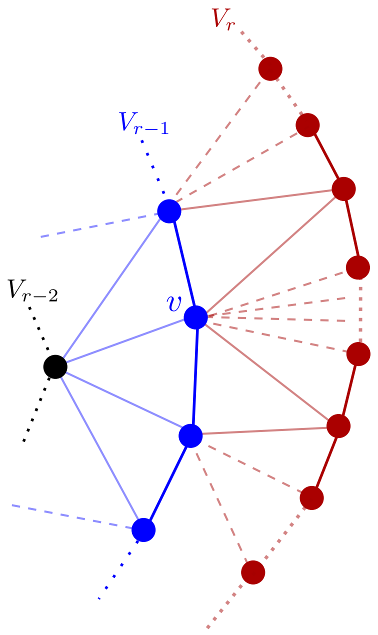


Figure A.1: Counting vertices at ring V_r.

Proof. To count the number of vertices $|V_r|$ at ring $r \in [1 : \rho]$, we first consider all possible vertices in ring V_r that must be connected to ring V_{r-1}. We can affirm that their sum is at least equal to degree $\deg(v) - 3$, for each $v \in V_{r-1}$. The number 3 in $\deg(v) - 3$ accounts for the two mandatory vertices connecting neighbors in the same ring V_{r-1} together with the neighbor vertex at ring V_{r-2}. This is the maximum number of vertices that V_r must have satisfying the constraints given by the degrees of the vertices $v \in V_{r-1}$ (see Figure A.1). This is represented by Equation A.3:

$$|V_r| \leq \sum_{v \in V_{r-1}} (\deg(v) - 3) - |V_{r-1}|, \qquad |V_0| = 1 \tag{A.3}$$

We use the maximum degree $\Delta_V = \max_{v\in V} \deg(v)$, to limit the number of vertices at ring r:

$$|V_r| \leq \sum_{v \in V_{r-1}} \deg(v) - 3|V_{r-1}| - |V_{r-1}|$$

$$|V_r| \leq \sum_{v \in V_{r-1}} \Delta_V - 4|V_{r-1}|$$

$$|V_r| \leq \Delta_V |V_{r-1}| - 4|V_{r-1}|$$

$$|V_r| \leq (\Delta_V - 4)|V_{r-1}|. \tag{A.4}$$

Let $b = \Delta_V - 4$. Solving recurrence (A.4) we obtain:

$$|V_r| \leq b^r, \qquad |V_0| = 1$$

Now, knowing that $\sum_{r=0}^{\rho} |V_r| = n$, where $n = |V|$ we have

$$\sum_{r=0}^{\rho} |V_r| \leq \sum_{r=0}^{\rho} b^r$$

$$n \leq \frac{b^{\rho+1} - 1}{b - 1} \leq b^{\rho+1}$$

$$\log_b n \leq \rho + 1$$

therefore, as long as b is limited, we can conclude that the number of rings ρ is $\Omega(\log n)$. □

Lemma A.2. All vertices $v \in V_r$ need $k_r \leq 2r + \log\left(\frac{\Delta_v - 3}{3}\right) r - 1$ iterations to compute their final distance [1].

Proof. We will prove by induction that:

$$k_r \leq 2r + \log\left(\frac{\Delta_v - 3}{3}\right)(r-1) - 1 \tag{A.5}$$

Base case: When $r = 1$, all vertices $v \in V_1$ compute their final distance at iteration 1, then $k_1 = 1 = 2(1) - \log\left(\frac{\Delta_V - 3}{3}\right)(1-1) - 1$ and (A.5) is true.

Induction step: All vertices at ring V_{r-1} compute their final distances at iteration k_{r-1}, and the vertices at ring V_r need at least i iterations from k_{r-1} to compute their final distances. The next recurrence equation represents the number of iterations k_r that the vertices $v \in V_r$ need to compute their final distances:

$$k_r \leq k_{r-1} + i, \qquad k_1 = 1 \tag{A.6}$$

solving this recurrence, we obtain:

$$k_r \leq i(r-1) + 1 \tag{A.7}$$

[1]Considering $\log = \log_2$

Note this behavior of the recurrence at the beginning of iteration $k_{r-1}+i'$, $i' \in [1:i]$, in Table A.1:

Table A.1: Vertices updated at iteration i'.

i'	Vertices updated in $V_r \cup V_{r-1}$	Remaining vertices in $\lvert V_r \rvert$
1	$\lvert V_{r-1}\rvert$	$\lvert V_r\rvert$
2	$2\lvert V_{r-1}\rvert$	$\lvert V_r\rvert - \lvert V_{r-1}\rvert$
3	$4\lvert V_{r-1}\rvert$	$\lvert V_r\rvert - (2-1)\lvert V_{r-1}\rvert$
4	$8\lvert V_{r-1}\rvert$	$\lvert V_r\rvert - (4-1)\lvert V_{r-1}\rvert$
$\vdots$		
i	$2^{i-1}\lvert V_{r-1}\rvert$	$\lvert V_r\rvert - (2^{i-2}-1)\lvert V_{r-1}\rvert$

This means that at each iteration i', the computed distances in V_{r-1} define the final distances of a total of $|V_{r-1}|$ vertices in V_r at iteration $i' = 1$. When $2^{i-1}|V_{r-1}| \geq |V_r| - (2^{i-2}-1)|V_{r-1}|$ all vertices in V_r have computed their final distances at end of iteration $k_r \leq k_{r-1} + i$. Now, we need to find the value of i:

$$2^{i-1}|V_{r-1}| \geq |V_r| - (2^{i-2}-1)|V_{r-1}|$$

$$(2^{i-1} + 2^{i-2} - 1)|V_{r-1}| \geq |V_r|$$

$$\left(2^i \frac{3}{4} - 1\right)|V_{r-1}| \geq |V_r|$$

using Equation A.4 we obtain:

$$2^i \frac{3}{4} - 1 \geq \Delta_V - 4$$

$$\log 2^i \geq \log\left(4 \frac{\Delta_V - 3}{3}\right)$$

$$i \geq \log 4 + \log\left(\frac{\Delta_V - 3}{3}\right)$$

$$i \geq 2 + \log\left(\frac{\Delta_V - 3}{3}\right) \tag{A.8}$$

replacing (A.8) in (A.7):

$$k_r \leq \left(2 + \log\left(\frac{\Delta_V - 3}{3}\right)\right)(r-1) + 1$$

$$k_r \leq 2r - 2 + \log\left(\frac{\Delta_V - 3}{3}\right)(r-1) + 1$$

$$k_r \leq 2r + \log\left(\frac{\Delta_V - 3}{3}\right)(r-1) - 1$$

Conclusion: By the principle of induction, (A.5) is true.

□

Note that in the majority of triangular meshes, the expected value of the maximum degree Δ_v for each ring is 6 , then the number of iterations that the vertices in V_r need to compute their final distance is:

$$k_r = 2r - 1$$

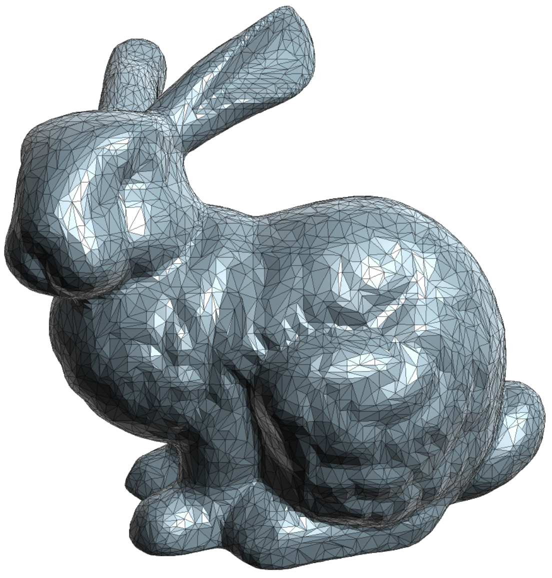
this reasoning is supported by the experimental evaluation.

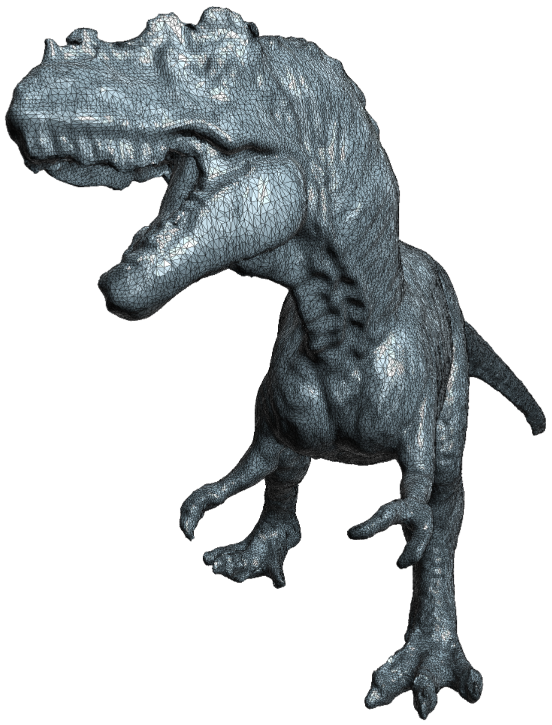
Theorem 5.2. [Maximum number of iterations K] Let $T = (V, E, F)$ be a triangular mesh, $s \in V$ be a source vertex such that it is possible order all rings V_r by their number of vertices in a crescent way. Then, the maximum of iterations is $K = O(\sqrt{n})$, where $n = |V|$.

Proof. To proof the Theorem 5.2 we use the Lemma A.2 to count the number of iterations K to compute the final distance to last vertices ring set V_ρ is $K = 2\rho - 1$ and $\rho \leq \sqrt{n}$, then we can affirm that $K \leq 2\sqrt{n}$ and $K = O(\sqrt{n})$. □

APPENDIX B - Results

B.1 Mesh topology


(a)


(b)

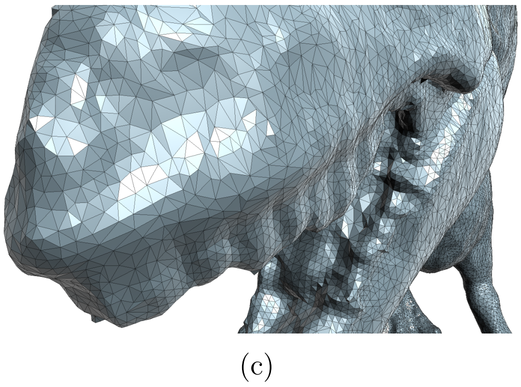
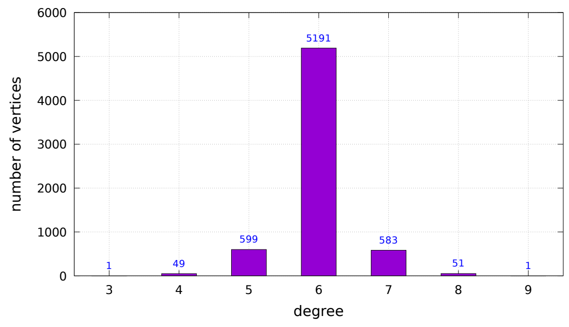

(c)

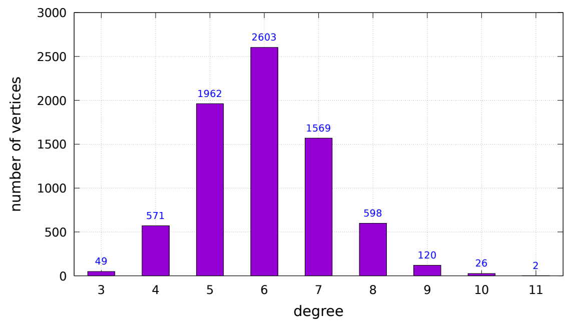
Figure B.1: Meshes topology.

B.2 Analysis of the degrees of the vertices

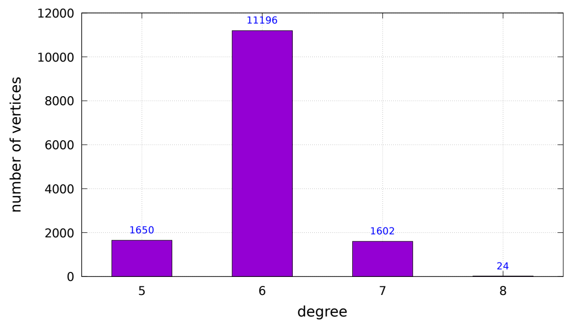
Figure B.2 shows the distribution of mesh degrees considering all the meshes used in the experiments. We can observe that the predominant degrees are 6, 4 and 5.



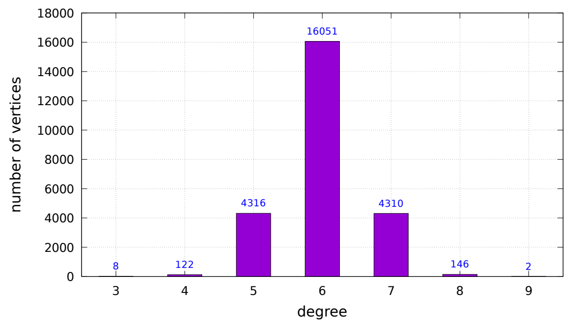
(a) *fandisk*



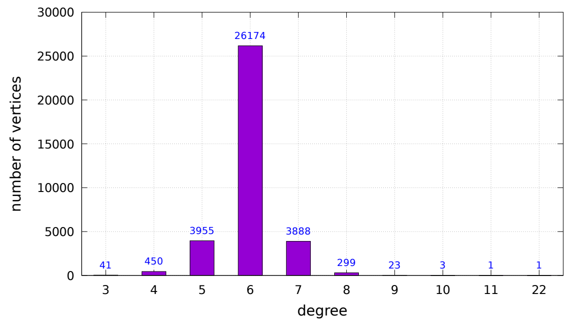
(b) *bunny_irregular*



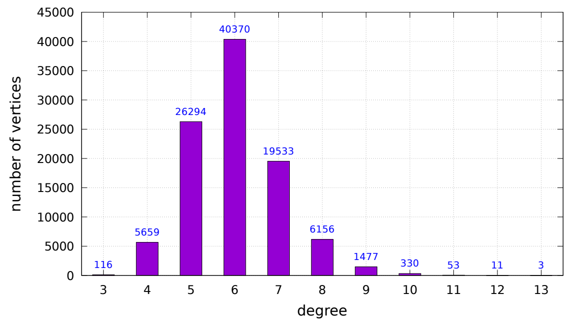
(c) *kitten*



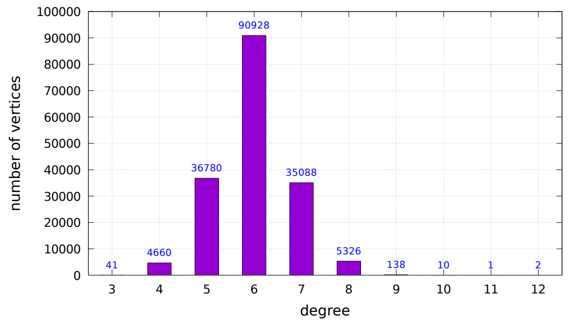
(d) *elephant*



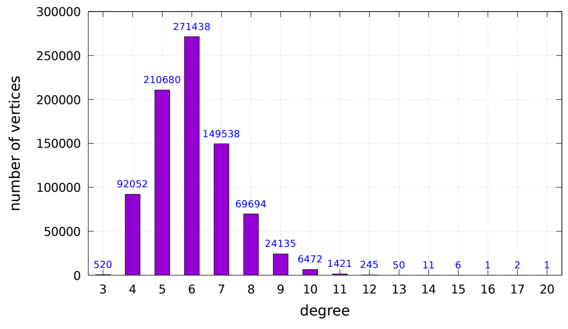
(e) *bunny*



(f) *tyra*



(g) *armadillo*

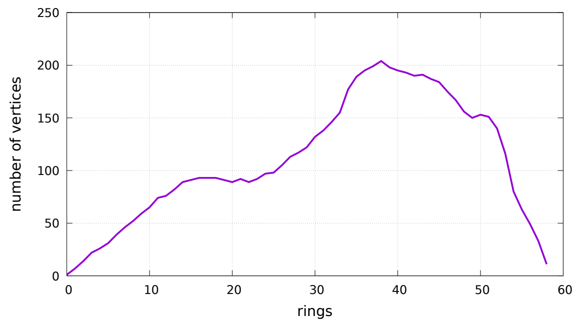


(h) *ramesses*

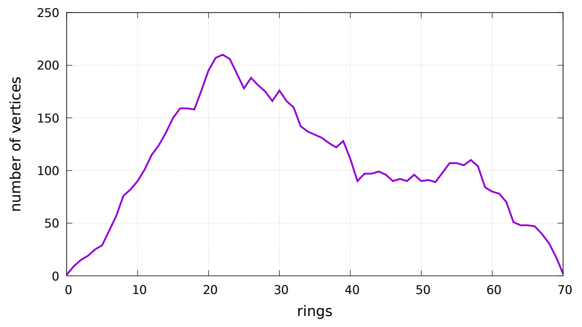
Figure B.2: Degree meshes.

B.3 Analysis of the distribution of rings

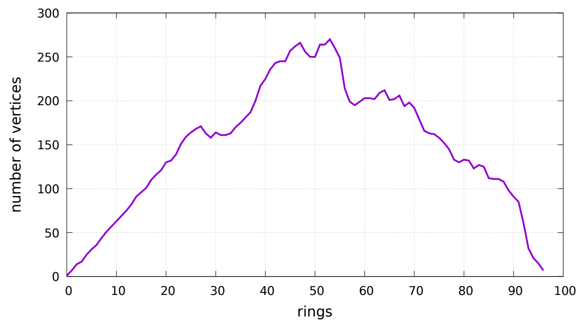
Figure B.3 shows the distribution of rings for each mesh in the experiments. We observe that the curves are non-constant and the vertices are distributed in each ring.



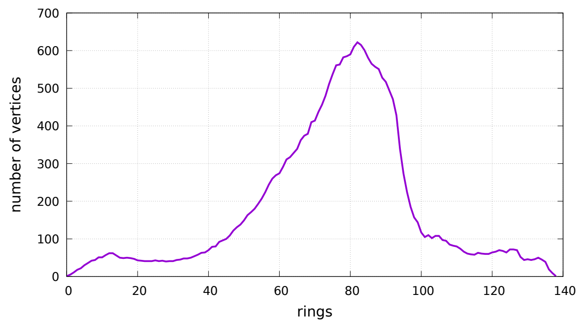
(a) *fandisk*



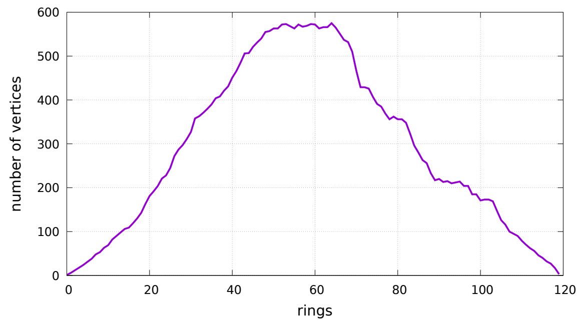
(b) *bunny_irregular*



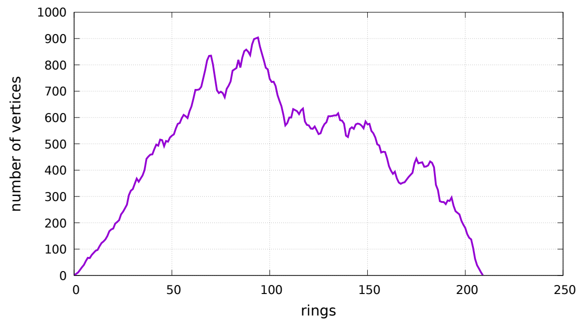
(c) *kitten*



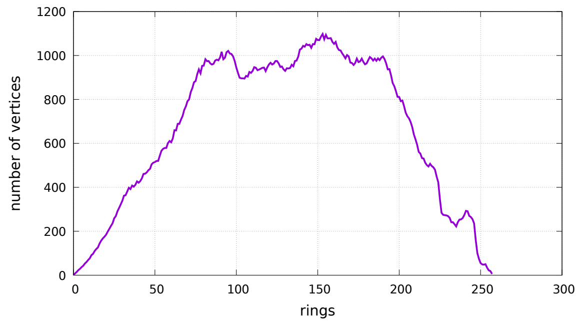
(d) *elephant*



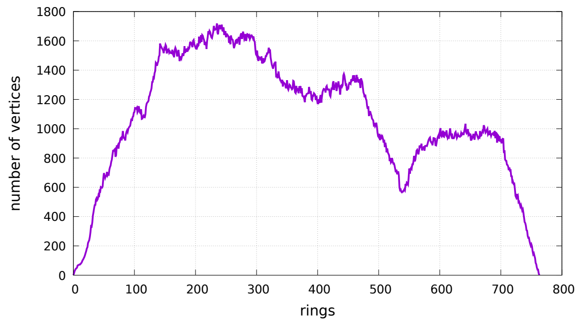
(e) *bunny*



(f) *tyra*



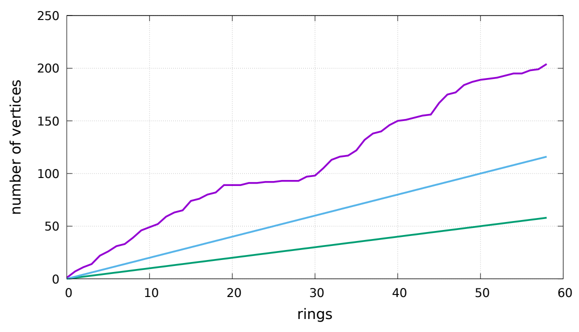
(g) *armadillo*



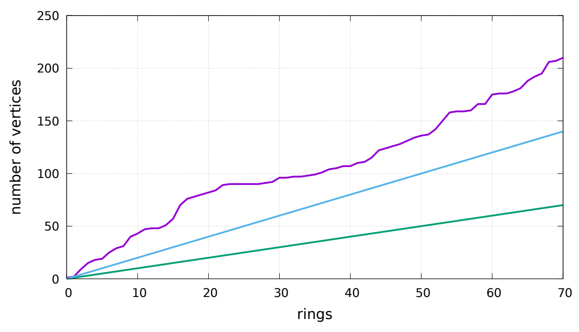
(h) *ramesses*

Figure B.3: Rings meshes distribution.

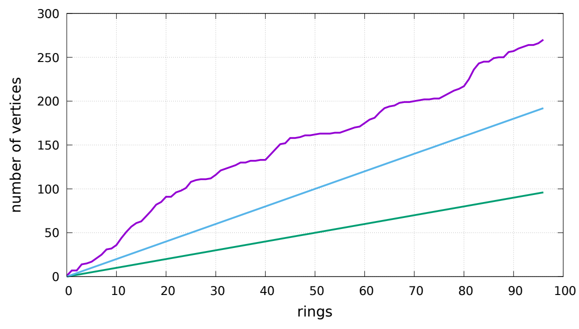
Figure B.4 shows the distribution of rings sorted by the number of vertices, and the functions $y = x$ and $y = 2x$. We can observe that the curve increases over the minimum increase constant $c = 1$, which is considerate in the proofs. This plots confirm that the number of rings $\rho \leq \sqrt{n}$.



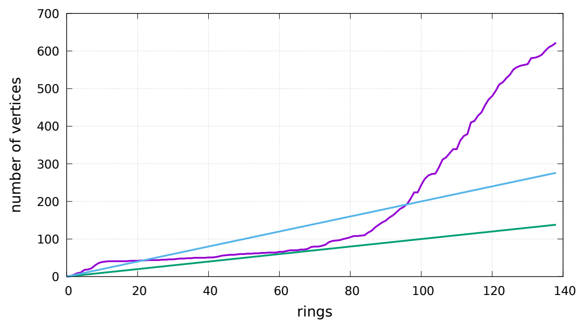
(a) *fandisk*



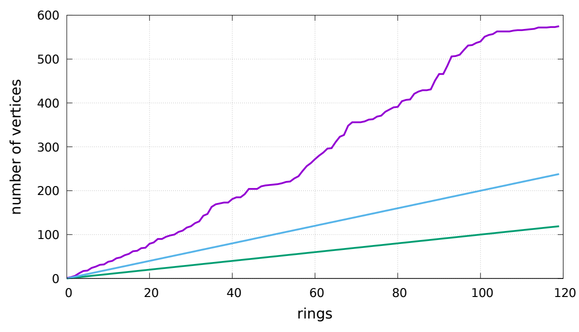
(b) *bunny_ irregular*



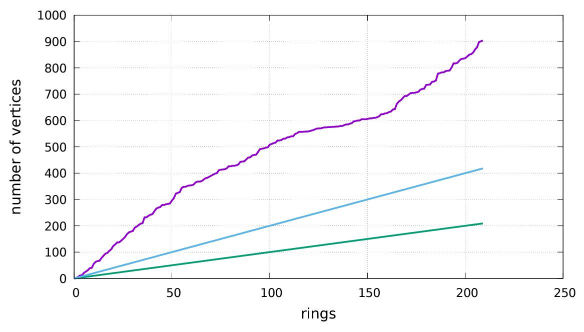
(c) *kitten*



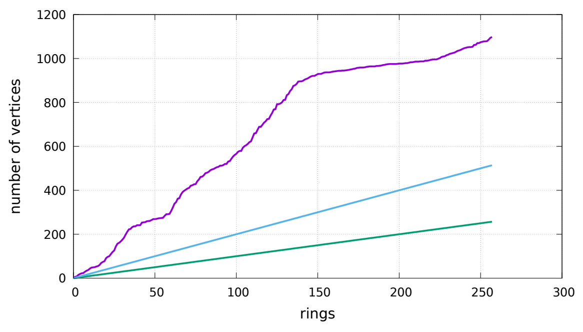
(d) *elephant*



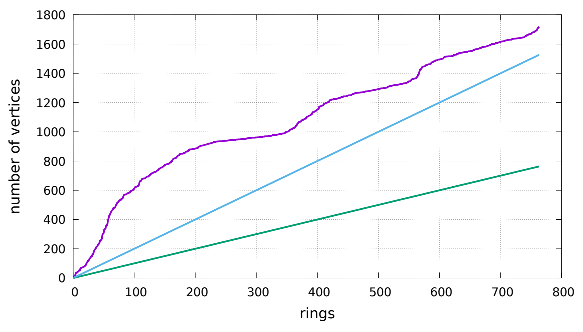
(e) *bunny*



(f) *tyra*



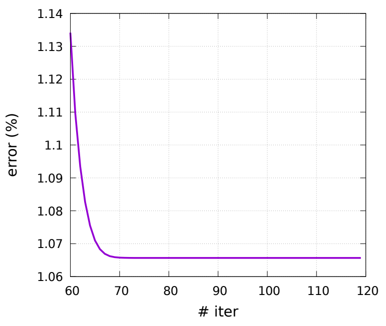
(g) *armadillo*



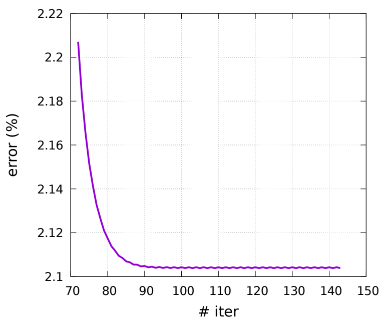
(h) *ramesses*

Figure B.4: Sorted rings meshes distribution.

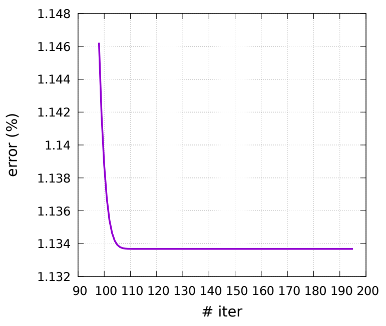
B.4 Analysis of error double precision



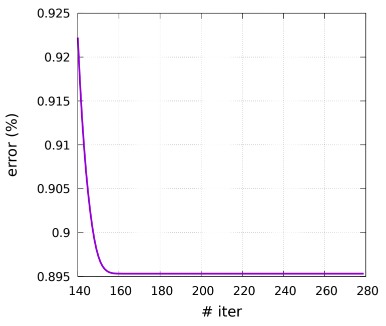
(a) *fandisk*



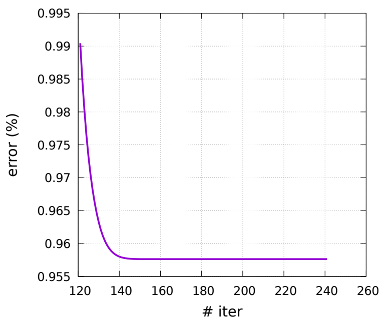
(b) *bunny_irregular*



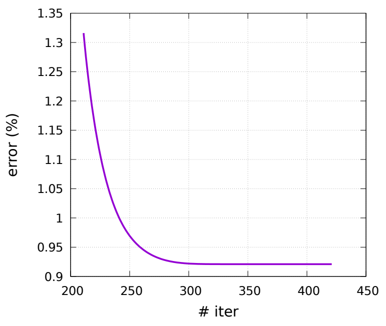
(c) *kitten*



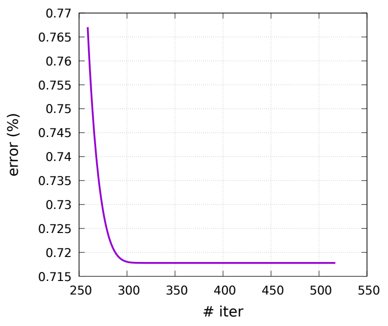
(d) *elephant*



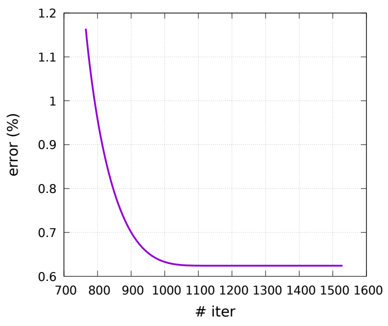
(e) *bunny*



(f) *tyra*



(g) *armadillo*



(h) *ramesses*

Figure B.5: Analysis of the error with double precision.