UNIVERSIDADE FEDERAL DA FRONTEIRA SUL
CAMPUS CHAPECÓ
CURSO DE CIÊNCIA DA COMPUTAÇÃO

IAGO U. BERNDT

NAVEGAÇÃO DE MÚLTIPLOS AGENTES SOBRE SUPERFÍCIES ARBITRÁRIAS UTILIZANDO PLANARIZAÇÃO DINÂMICA

CHAPECÓ
2014

IAGO U. BERNDT

NAVEGAÇÃO DE MÚLTIPLOS AGENTES SOBRE SUPERFÍCIES ARBITRÁRIAS UTILIZANDO PLANARIZAÇÃO DINÂMICA

Trabalho de conclusão de curso de graduação apresentado como requisito para obtenção do grau de Bacharel em Ciência da Computação da Universidade Federal da Fronteira Sul.

Orientador: Prof. Dr. Rafael P. Torchelsen

CHAPECÓ
2014

Somente para TCC 2: Está página deve ser substituida pela ficha de catalogação antes de sua entrega na biblioteca.

Berndt, Iago U.

Navegação de múltiplos agentes sobre superfícies arbitrárias utilizando planarização dinâmica / por Iago U. Berndt. – 2014.
46 f.: il.

Orientador: Rafael P. Torchelsen
Trabalho de conclusão de curso (graduação) - Universidade Federal da Fronteira Sul, Ciência da Computação, Curso de Ciência da Computação, Chapecó, SC, 2014.

1. Planejamento de Caminho. 2. Planarização. 3. Superfícies Arbitrárias. 4. Distância geodésica. 5. CUDA. I. Torchelsen, Rafael P., orient. II. Universidade Federal da Fronteira Sul. III. Título.

E-mail: iagouilian@gmail.com

IAGO U. BERNDT

NAVEGAÇÃO DE MÚLTIPLOS AGENTES SOBRE SUPERFÍCIES ARBITRÁRIAS UTILIZANDO PLANARIZAÇÃO DINÂMICA

Trabalho de conclusão de curso de graduação apresentado como requisito para obtenção do grau de Bacharel em Ciência da Computação da Universidade Federal da Fronteira Sul.

Orientador: Prof. Dr. Rafael P. Torchelsen

Este trabalho de conclusão de curso foi defendido e aprovado pela banca em: 15 / 12 / 14

BANCA EXAMINADORA:

Dr. Rafael P. Torchelsen - UFFS

Dr. João L. D. Comba - UFRGS (VIDEOCONFERÊNCIA)

Me. Fernando Belvilacqua - UFFS

RESUMO

Navegação de múltiplos agentes em superfícies arbitrárias é um tópico de pesquisa recente que tem atraído crescente atenção, entretanto, estes trabalhos não são capazes de realizar a navegação em tempo real com a mesma qualidade dos trabalhos focados em superfícies planares. Um dos principais fatores limitantes é o custo computacional do cálculo de distância entre dois pontos sobre uma superfície arbitrária, o que em superfícies planares pode ser realizado em tempo constante através do cálculo de distância euclidiana. Neste trabalho é proposta uma abordagem para realizar a navegação de múltiplos agentes sobre superfícies arbitrárias, e ao contrário dos trabalhos anteriores, permitindo a utilização de técnicas de navegação sobre superfícies planares, por exemplo, Reciprocal Velocity Obstacles. A solução proposta utiliza o cálculo da distância geodésica na navegação global, responsável pela definição do caminho inicial. Para o desvio dos obstáculos dinâmicos são utilizadas técnicas consolidadas, criadas para superfícies planares, com comprovada qualidade. Para permitir a utilização desses métodos é proposta uma planarização da superfície próxima do agente de modo a substituir o uso de distância geodésica por distância euclidiana na navegação local. Além disso, é implementado um algoritmo híbrido CPU/GPU e assim explorando as melhores qualidades de cada processador.

Palavras-chave: Planejamento de Caminho. Planarização. Superfícies Arbitrárias. Distância geodésica. CUDA.

ABSTRACT

Path planning of multiple agents on arbitrary surfaces is a recent research topic. The quality of the paths generated to arbitrary surface is poor compared to method proposed to planar surfaces. A major limitation to improve the quality is related to the computational cost of computing distances over arbitrary surface. Traditional path planning algorithms designed to planar surface use Euclidian distance with has constant computational cost, in the other hand, arbitrary surfaces require Geodesic distances with in turn has a considerable higher computational cost. This project proposes novel dynamic planarization around each agent with will allow the use of Eucledian distance during the local navigation, that way, classic path planning algorithms designed for planar surface can be used over arbitrary surface. The main benefit is the increased quality of paths for agents. The proposed solution uses geodesic distance in the global navigation, responsible for defining the initial path. However, for the local navigation, or obstacle avoidance, our solution allows the use of classic techniques. To allow the use of these methods we propose a planarization of the surface adjacent to the agent in order to replace the use of geodesic distance with Euclidean distance in the local navigation. Additionally, the algorithm is massively parallel and implemented in the CPU and GPU to explore the best features of each.

Keywords: Path Planning, Planarization, Arbitrary Surfaces, Geodesic Distances, CUDA.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Um dos problemas mais desafiadores presente no desenvolvimento de jogos é a navegação de múltiplos agentes em tempo real. Onde cada agente deve encontrar um caminho até uma determinada posição e realizar uma navegação até este destino, evitando colisões com outros agentes e obstáculos do ambiente.

Questões como a interação entre os agentes para evitar colisões e o planejamento de rotas, bem como a escalabilidade em relação ao número de agentes e a complexidade das superfícies, foram abordados recentemente. No entanto a maioria dos trabalhos estão limitados a superfícies planares e não conseguem tratar de casos como na Figura 1.1.

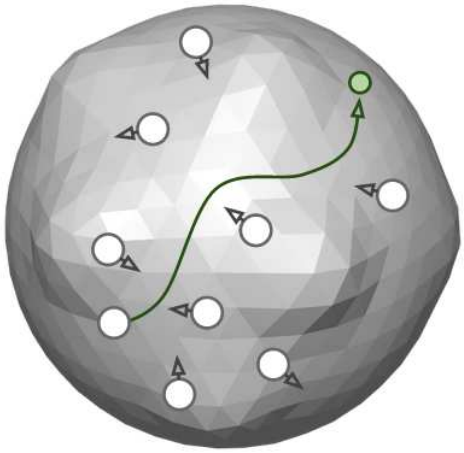


Figura 1.1: Navegação de múltiplos agentes sobre uma superfície arbitrária. O método de navegação deve ser capaz de orientar o agente até o destino desviando de obstáculos que possam surgir durante o percurso.

Técnicas capazes de suportar superfícies arbitrárias, apesar de terem um propósito semelhante, costumam ser diferentes e apresentar um custo computacional superior. Isso é causado principalmente pela necessidade de se utilizar distância geodésica ao invés de distância euclidiana para orientar os agentes.

Métodos para o cálculo de distâncias geodésicas exatas apresentam um alto custo computacional [31], muitas vezes é necessário realizar um pré-processamento para que a navegação possa ocorrer em tempo real [27]. Desta forma, a superfície não poderá mais sofrer alterações ou conter novos obstáculos após o processamento. Limitações como essa se tornam obstáculos ao realismo.

Outros métodos para o cálculo de distâncias geodésicas como algoritmos de busca em

grafos sobre a malha de triângulos, possuem um custo computacional que permite respostas em tempo-real. Porém esses métodos apresentam baixa precisão e resultam em caminhos *artificiais* e assim, também se torna um obstáculo para o realismo.

Navegação de múltiplos agentes em superfícies arbitrárias é um tema de pesquisa ainda em aberto e que tem atraído crescente atenção. Técnicas para múltiplos agentes tem sido tradicionalmente estudadas no domínio da robótica e nos últimos anos tem sido aplicadas aos jogos, filmes e simulações.

Neste trabalho é proposta uma abordagem para realizar a navegação de múltiplos agentes sobre superfícies arbitrárias. Para isso, será seguido a abordagem de Torchelsen et al [27], onde a navegação é dividida em duas etapas principais, a navegação global e a local.

Na navegação global é realizado o planejamento de uma trajetória até o destino do agente sem levar em conta os obstáculos dinâmicos que poderão interferir na navegação. Para esta etapa, é utilizado a distância geodésica discreta de cada ponto da superfície.

Na navegação local são identificados os vizinhos de cada agente e calculado o desvio necessário. Como a navegação local não utiliza toda a superfície, e sim apenas a região próxima de cada agente, este trabalho propõe a planarização desta região. Desse modo, permitindo que técnicas de navegação em superfícies planares possam ser utilizadas sobre superfícies arbitrárias, sendo essa a principal contribuição deste trabalho.

2 TRABALHOS RELACIONADOS

Visto que a navegação será dividida em duas etapas, e desta forma utilizando tanto técnicas para superfícies arbitrárias, como técnicas para superfícies planares, os trabalhos anteriores referentes a navegação serão divididos em duas sessões, navegação sobre superfícies planas e navegação sobre superfícies arbitrárias. Também visto que a região próxima de cada agente deverá ser planarizada, será apresentado uma sessão sobre as técnicas de parametrização de malhas.

2.1 Navegação sobre superfícies planares

Planejamento de trajetórias em planos geralmente são baseados em algoritmos de busca em grafos sobre malhas de navegação [5]. Para agilizar a busca sobre malhas de navegação, Bleiweiss [1] apresenta um trabalho sobre busca em grafos utilizando a GPU. Para isso foram implementados e comparados os algoritmos de Dijkstra, busca em largura e A*, todos utilizando a plataforma CUDA, para fila de prioridade é utilizado o Heapsort também implementada na GPU. Apesar do trabalho citado ser para superfícies planares, o método ainda pode ser aplicado em superfícies arbitrárias utilizando a malha de triângulos como grafo de navegação, porém, desta forma, resultando em caminhos limitados as bordas dos triângulos.

Com o objetivo de desviar de obstáculos dinâmicos, Fiorini e Shiller [8] propuseram o método *Velocity Obstacle* (VO), que através da posição e velocidade do obstáculo, consegue calcular o conjunto de todas as velocidades que resultarão em uma colisão dentro de um intervalo de tempo, como na Figura 2.1. Além dessa abordagem, foram propostos outros métodos para prevenção de colisões, planejamento e navegação entre obstáculos em movimento [9], [18], [30]. Entretanto estes estudos não consideram que os obstáculos possam ser outros agentes e que também tentarão realizar o desvio a medida que se aproximam. Logo, tais abordagens não são capazes de garantir uma navegação livre de colisões com múltiplos agentes.

Há também uma quantidade significativa de estudos sobre navegação de múltiplos agentes, onde cada agente pode navegar individualmente entre os outros, como [7], [14] e [25]. No entanto, estas técnicas também não consideram as reações que os agentes a serem desviados poderão tomar, assim, podendo ocorrer o desvios pelo mesmo lado e novamente necessitarem de uma troca de direção, causando uma navegação semelhante o que ocorre na Figura 2.2.

Para prevenir colisões durante a navegação de múltiplos agentes, Van den Berg, Lin e

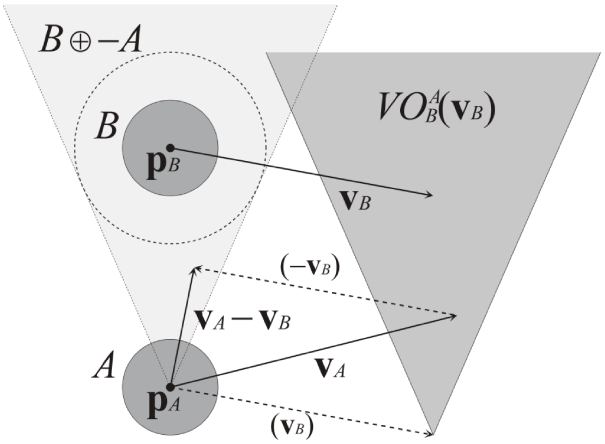


Figura 2.1: Velocity Obstacle $VO_A^B(v_B)$ do obstáculo B para o agente A. O agente A deverá escolher uma velocidade fora da região correspondente ao *Velocity Obstacle*, [8].

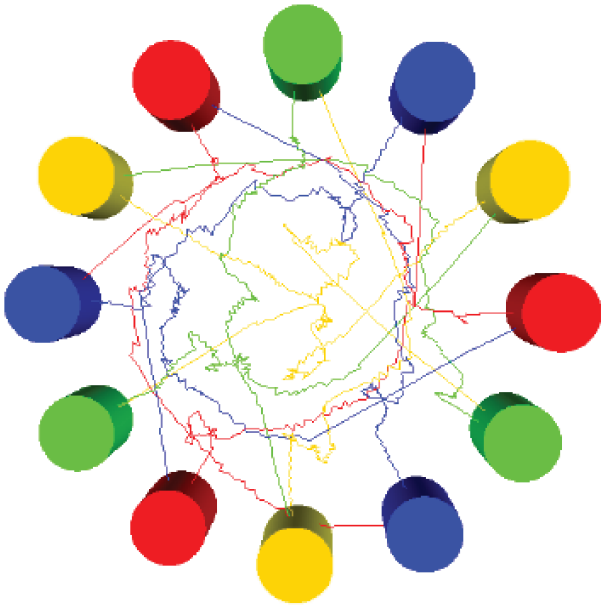


Figura 2.2: Caminhos percorridos por múltiplos agentes utilizando uma abordagem baseada em *Velocity Obstacle*. Percebe-se que durante a navegação houve mudanças bruscas nas rotas, causado principalmente por não ser considerado as possíveis atitudes dos outros agentes, [29].

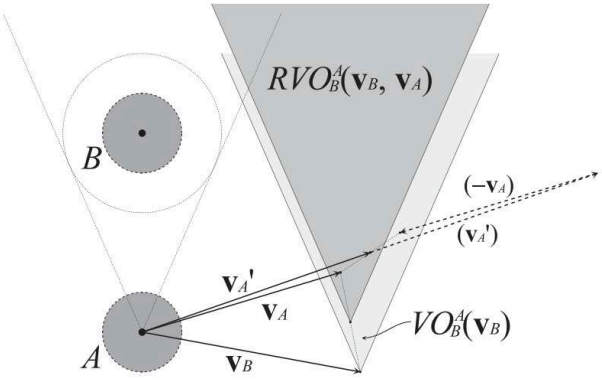


Figura 2.3: Reciprocal Velocity Obstacle $RVO^B_A(v_B, v_A)$ do agente B para o agente A. O agente A deverá escolher uma velocidade fora da região correspondente ao Reciprocal Velocity Obstacle, [29].

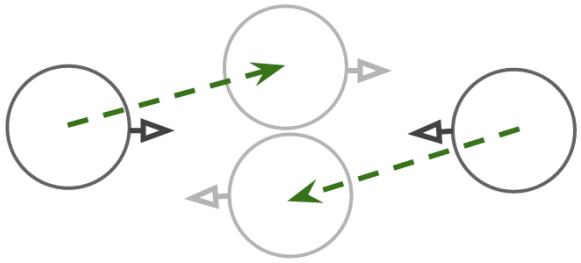


Figura 2.4: Tratamento de colisão realizado pelo método RVO, onde cada agente só realiza metade do movimento necessário para o desvio da colisão.

Manocha [29] basearam-se na abordagem *Velocity Obstacle* e formalizaram um novo método chamado Reciprocal Velocity Obstacle (RVO). A ideia básica deste método é simples: escolher uma nova velocidade que é a média de sua velocidade atual com uma velocidade que se situa fora do *Velocity Obstacle* de outro agente, como na Figura 2.3. Dessa forma, o agente só precisará fazer metade do desvio para evitar a colisão, pois assume que o outro agente cuidará da outra metade, como na Figura 2.4.

Apesar do RVO ter apresentado resultados como na Figura 2.5, ainda pode resultar em oscilações indesejáveis durante navegação quando três ou mais agentes se encontram. Outro problema presente no RVO é a utilização de amostragem aleatória para encontrar uma velocidade próxima da preferencial, causando impactos significativas no custo computacional.

Para corrigir os problemas encontrados no RVO, Berg et al. [28] propuseram um novo método chamado Optimal Reciprocal Collision Avoidance (RVO2). A principal diferença desta abordagem em relação ao RVO é a utilização de um semiplano para definir o conjunto de velocidades livres de colisão e que garantem uma navegação suave, como na Figura 2.6.

Para o RVO2 encontrar a velocidade mais próxima da preferencial e que respeite as res-

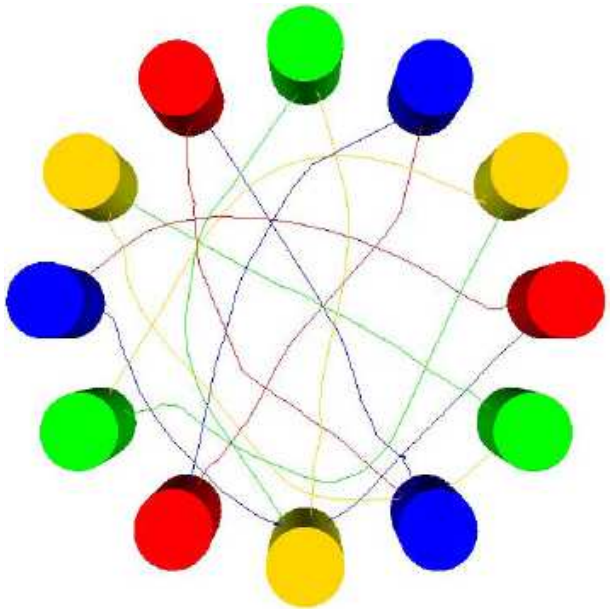
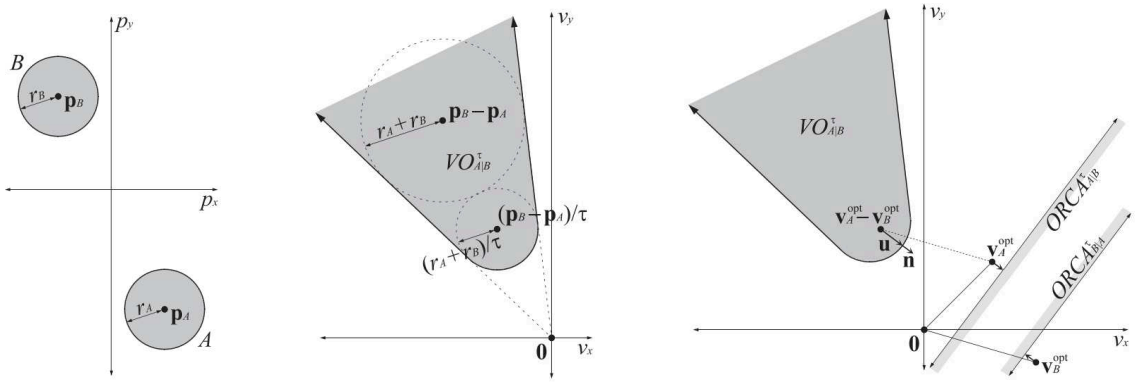
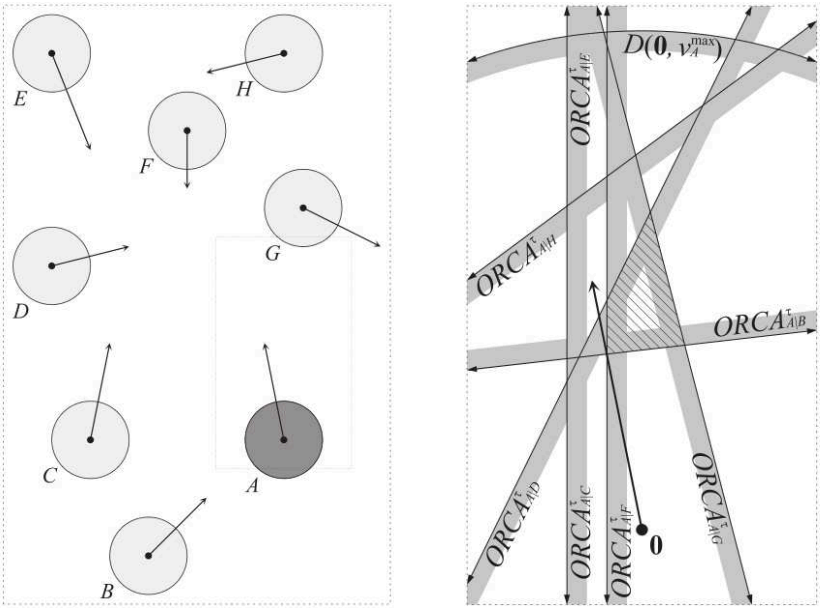


Figura 2.5: Caminhos percorridos por múltiplos agentes utilizando o método RVO, onde é considerado as possíveis atitudes dos os outros agentes, desta forma reduzindo as chances de novas colisões com os agentes a serem desviados, resultando em caminhos suaves, [29].



(a) (b) (c)

Figura 2.6: (a) Configuração com dois agentes A e B. (b) *Velocity Obstacle* $VO^t_{A|B}$ sendo interpretado geometricamente como um tronco de cone. (c) Cálculo do semiplano $ORCA^t_{A|B}$ que define um conjunto de velocidades capazes de evitar colisão com o agente B. A linha que delimita o semiplano $ORCA^t_{A|B}$ é perpendicular a u e passa pelo ponto $V_A^{opt} + \frac{1}{2}u$, onde u é vetor a partir de $V_A^{opt} - V_B^{opt}$ até o ponto mais proximo no limite de $VO^t_{A|B}$, [28].



(a) (b)

Figura 2.7: (a) Configuração com 8 agentes, onde as setas representam a velocidade atual de cada agente. (b) Os semiplanos indicando as velocidades permitidas para o agente A induzidas por cada um dos outros agentes. O agente A devera receber um vetor velocidade que aponte para a região tracejada, [31].

trições impostas pelos outros agentes, como na Figura 2.7, é utilizado um algoritmo de programação linear. Este algoritmo apresenta complexidade de tempo linear ao número de semiplanos, desta forma trazendo ganhos de desempenho computacional em relação ao RVO. Para cenários densos em que não há velocidade que respeite todas as restrições, é utilizado um algoritmo de programação linear tridimensional. Este algoritmo ira encontrar em complexidade de tempo linear uma velocidade segura e que penetre minimamente as limitações impostas.

Como o método RVO não necessita da negociação entre os agentes, Bleiweiss [2] propôs uma implementação do método baseada em CUDA, onde a navegação de vários agentes é realizada simultaneamente. Para melhorar o desempenho na GPU, foi necessário utilizar uma hash para o acesso aos agentes próximos. Nessa implementação também é realizado o cálculo do caminho global para evitar que os agentes corram o risco de ficar presos em obstáculos estáticos.

2.2 Navegação sobre superfícies arbitrárias

Abordagens para navegação sobre superfícies arbitrárias normalmente envolvem o cálculo de distâncias geodésicas para permitir o planejamento de caminhos, como na Figura 2.8.

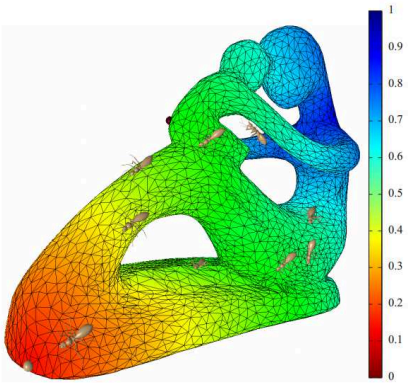
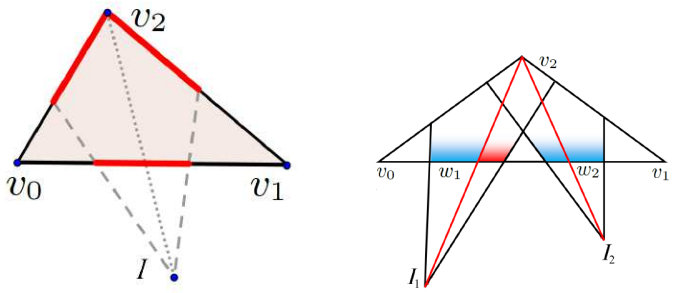


Figura 2.8: Campo de distância, onde a superfície foi colorida de acordo com a distância até o destino, desta forma o agente pode chegar ao destino apenas navegando em direção ao lado com menor distância, [27].



(a) (b)

Figura 2.9: Propagação do algoritmo de Chen e Han; (a) Uma janela gerando duas novas em bordas opostas. (b) Propagação de duas janelas que resultam em um conflito, neste caso, permanece a janela com menor distância até sua origem, [32].

O cálculo de distâncias geodésicas pode ser realizado utilizando algoritmos de busca em grafos sobre a malha de triângulos, como o algoritmo de Dijkstra [6], porém, por utilizarem apenas as arestas dos triângulos, estas soluções fornecem as distâncias geodésicas discretas, que são aproximações das exatas.

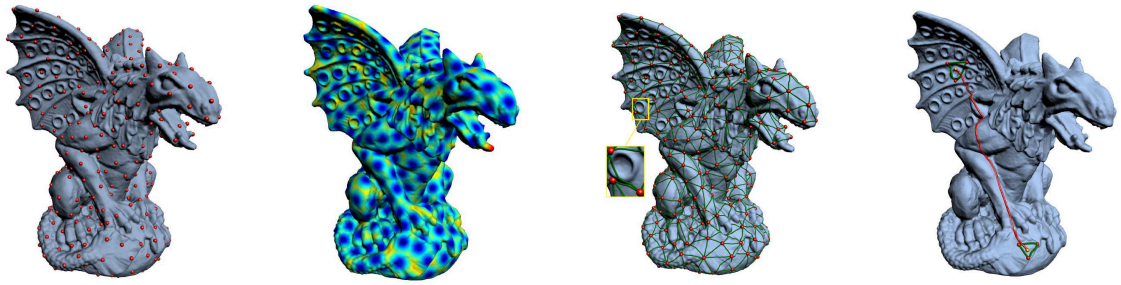
Com o objetivo principal de planejar caminhos sobre poliedros, Chen e Han [3] propuseram um algoritmo capaz de realizar uma busca em largura sobre a superfície passando pelo interior dos triângulos. Para isso, a malha de triângulos é utilizado como um grafo e particionado cada aresta em intervalos, estes intervalos passam a ser chamados de janelas, como na Figura 2.9. As janelas são armazenadas em forma de árvore que se propaga sobre a superfície, ao alcançar o destino se tem o caminho e sua distância geodésica.

Desde meados dos anos de 1980 também foram propostas novas abordagens e melhorias para o cálculo de distâncias geodésicas, tais como, [17], [13] e [26]. Estes trabalhos, porém, apresentam um alto custo computacional, dificultando sua aplicação em modelos complexos e

com restrições no tempo de processamento.

Para reduzir este problema, Xin, Ying e He [32] propuseram uma extensão do algoritmo de Chen e Han capaz de propagar as janelas de forma paralela e eficaz. Como durante a propagação podem ocorrer conflitos entre as janelas, tal como na Figura 2.9(b), foi proposta uma organização capaz de evitá-los, assim reduzindo o número de sincronizações e dependências.

Algumas das abordagens recentes foram criadas especialmente para computação paralela, como o trabalho de Xin, Ying e He [31], que apresenta um algoritmo para o cálculo de distâncias geodésicas próximas da exata, capaz de retornar a distância entre pontos em tempo constante. O método de Xin, Ying e He, porém, necessita de um pré-processamento que consiste em distribuir pontos uniformemente pela superfície (Figura 2.10(a)), realizar o cálculo da distância geodésica entre eles paralelamente (Figura 2.10(b)), assim formando uma triangulação de Delaunay (Figura 2.10(c)), que é utilizada junto com um algoritmo de busca em grafo para encontrar a distância entre todos os pontos (Figura 2.10(d)). Os valores encontrados pelo método são armazenados para consultas futuras em tempo constante, desta forma permitindo consultas em tempo real, porém, não permitindo que a superfície sofra alterações.



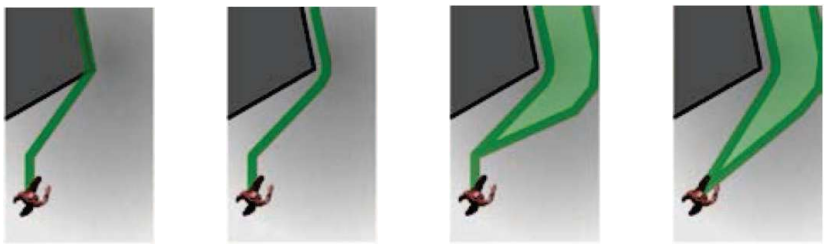
(a) (b) (c) (d)

Figura 2.10: Principais fases do algoritmo de Xin, Ying e He.(a) Distribuição de pontos uniformemente sobre a malha. (b) Cálculo das distâncias geodésicas entre os pontos próximos. (c) Geração de malha com a distância geodésica entre os pontos. (d) Planejamento de caminho utilizado busca em grafo, [31].

Com o objetivo de permitir que a navegação de múltiplos agentes ocorra sem impor limites ao gênero da superfície, Torchelsen et al. [27] apresentaram o primeiro trabalho para navegação de múltiplos agentes sobre superfícies arbitrárias. A solução divide a navegação em global e local. Na navegação global é calculado o caminho de cada agente até o seu destino sem considerar os possíveis obstáculos. Para que as distâncias geodésicas possam ser calculadas em tempo real, é utilizado técnicas de parametrização e realizado um pré-processamento. Para evitar colisões entre os agentes, é realizado de forma paralela a navegação local, onde é

calculado a direção de cada agente considerando os obstáculos próximos. Para identificar as áreas ocupadas pelos agentes é utilizado uma grade 3D.

Algumas das soluções recentes propuseram a utilização da distância geodésica discreta como forma de reduzir os custos de processamento na navegação global, como a abordagem de Ricks e Egbert [22], onde é realizado o cálculo de uma rota inicial utilizando busca em grafos (Figura 2.11(a)) e após é realizado melhorias no caminho para produzirem um movimento natural. Nesse tratamento é realizado o afastamento de obstáculos estáticos (Figura 2.11(b)), calculado um corredor para futuros desvios (Figura 2.11(c)) e suavizado os cantos presentes no caminho (Figura 2.11(d)). As colisões entre os agentes é identificada através da distância euclidiana com um tratamento para evitar a identificação de falsas colisões, que ocorrem quando estão próximos, mas em lados diferentes da superfície.



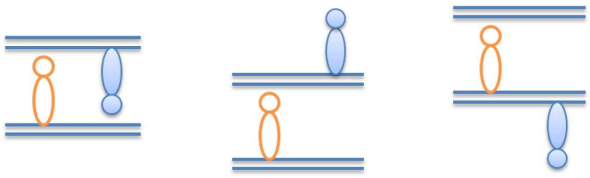
(a) (b) (c) (d)

Figura 2.11: (a) Cálculo do caminho utilizando busca em grafo. (b) Afastamento de caminhos proximos de obstáculos. (c), Cálculo de um corredor para ser utilizado durantes os desvios. (d) Suavização dos caminhos irregulares, [22].

Outro trabalho de Ricks e Egbert [20] apresenta uma forma de realizar a navegação de múltiplos agentes sobre superfícies arbitrárias considerando a capacidade de visão. Assim simula-se a incerteza dos agentes ao tomar decisões sobre a sua navegação. Esta abordagem foi implementada na GPU e utiliza um método semelhante ao que foi apresentado [22], porém, leva em conta uma superfície que é colorida conforme a navegação dos agentes, assim também é capaz de simular a comunicação entre eles.

O cálculo para identificar possíveis obstáculos em superfícies arbitrárias também apresenta diferenças em relação as técnicas para superfícies planares, isso porque podem ocorrer colisões mesmo quando os agentes estão distantes em relação a superfície, como na Figura 2.12(a), ou estarem muito próximos sem apresentarem riscos de colisão, como nas Figuras 2.12(b) e 2.12(c).

Para evitar falsos positivos e negativos Ricks e Egbert [21] propuseram um novo método para identificar obstáculos em superfícies arbitrárias, onde é levado em conta a superfície em



(a) (b) (c)

Figura 2.12: (a) Falso negativo utilizando distância geodésica, (b) e (c) Falso positivo utilizando distância euclidiana, [21].

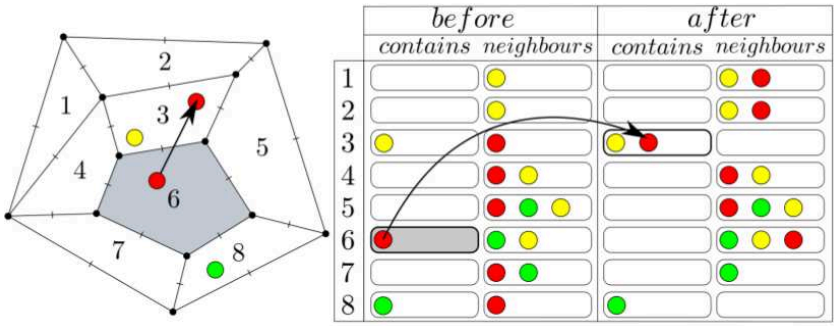


Figura 2.13: À esquerda um ambiente com três agentes, onde o agente em vermelho se move e troca de face. À direita uma tabela com as listas de cada face antes e depois da atualização. [11].

que os agentes estão.

Simulação de multidões impõe a utilização de estrutura de dados otimizadas para permitir a consulta de agentes próximos em tempo real. Técnicas como quadtrees, BSP e kd-trees, apesar de terem complexidade de tempo logarítmica, consultas e atualizações frequentes com grandes multidões de agentes dificultam a execução em tempo real.

Para resolver este problema, Jund et al. [11] propuseram um novo método de organização de agentes, onde a superfície é representada como uma malha multi-resolução e para cada face é armazenado uma lista com os agentes contidos, como na Figura 2.13. Esse método também é capaz de ajustar a resolução das faces conforme a densidade de agentes, assim melhorando a precisão na consulta de possíveis vizinhos.

2.3 Parametrização de Malhas

Técnicas de parametrização de malhas são geralmente utilizadas para o mapeamento de texturas, no entanto trabalhos recentes como o de Torchelsen et al. [27] e Sheffer en al.[24] mostram como estas técnicas podem ser exploradas para outras aplicações.

Nos últimos anos foram desenvolvidos várias abordagens para a parametrização de ma-

lhas, visando diversos domínios e com foco em diferentes propriedades de parametrização, como os de Liu et al. [15], Hormann et al. [10] e Goes et al. [4]. Também foram desenvolvidos métodos capazes de aproveitar os benefícios da computação paralela, como o de Levy et al. [16].

O trabalho de Levy et al. [16] apresenta um método baseado em otimização capaz de minimizar as deformações angulares dos triângulos. A parametrização começa projetando a região em um plano e após é realizado otimizações na malha para reduzir deformações e manter a orientação dos triângulos, o que é garantido.

Entretanto independente do método utilizado, na maioria dos casos haverá alguma distorção dos triângulos após a parametrização. Para medir a distorção causada nos triângulos, Sander et al. [23] propuseram a métrica L2. O método consiste em definir um circulo unitário dentro do triângulo parametrizado e o mapear sobre o triangulo original, gerando possivelmente uma elipse, como na Figura 2.14. Através da diferença dos lados da elipse obtida, é calculado a distorção do triângulo.

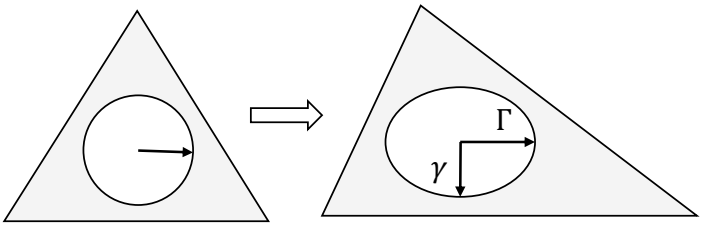


Figura 2.14: Métrica de distorção L2. A esquerda um circulo unitário dentro do triângulo parametrizado e a direita o mesmo circulo mapeado sobre o triangulo original. A distorção é obtida através da diferença entre Γ e γ .

3 OBJETIVO

3.1 Objetivo Geral

O objetivo deste projeto é permitir que técnicas de navegação planares possam ser utilizadas durante a navegação local em superfícies arbitrárias, assim aproveitando os métodos já existente para o tratamento de colisões e aperfeiçoamento de rotas.

3.2 Objetivos Específicos

O objetivo descrito acima se desdobra nos seguintes objetivos específicos:

- Propor um método para a planarização da área próxima do agente que permita a navegação local utilizando técnicas de navegação em superfícies planares.
- Apresentar uma estratégia que otimize o armazenamento das distâncias geodésicas.
- Desenvolver uma abordagem baseada em GPU capaz de ser executada para milhares de agentes em tempo real.

4 METODOLOGIA

Para elaborar este projeto e garantir que os seus objetivos fossem alcançados, primeiramente foi realizada a revisão bibliográfica, conforme o capítulo anterior.

Como dito anteriormente, este trabalho divide a navegação em duas etapas, a navegação global e a local, como na Figura 4.1, onde na navegação global é calculado o caminho até o destino, enquanto na navegação local é realizado o desvio de agentes utilizando uma técnica de navegação planar.

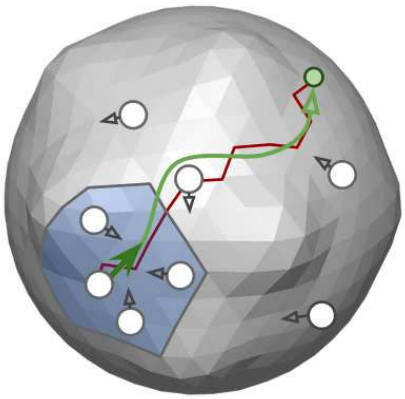


Figura 4.1: Navegação de múltiplos agentes. Em vermelho, o caminho encontrado através da navegação global, em verde, a rota que será indicada pela navegação local e em azul, a área que foi planarizada e utilizada pela navegação local.

Para que seja possível utilizar técnicas de navegação planar, este trabalho propõe a planarização da superfície próxima do agente, de modo que os agentes vizinhos possam ser representados sobre a superfície gerada. Assim, disponibilizando à navegação local um mapa que represente a superfície próxima do agente. Esta abordagem necessita gerar apenas um mapa para cada triângulo da superfície, pois os mapas são independentes da posição do agente no triângulo.

Para reduzir o custo computacional durante a simulação, é realizado um pré-processamento opcional para calcular o mapa de todos os triângulos. Caso ocorra alterações na superfície durante a simulação, os mapas afetados são recalculados.

Assim, a abordagem é dividida em quatro processos, simulação, navegação global, planarização e navegação local.

Como a navegação de cada agente pode ser realizada de forma independente, foi desenvolvido uma implementação híbrida entre a CPU e GPU, seguindo o diagrama da Figura 4.2,

aproveitando o melhor de cada unidade de processamento e assim melhorando a escalabilidade deste trabalho em relação ao número de agentes. Para as implementações na CPU, foi utilizado a linguagem de programação C++ junto com a API OpenMP. Enquanto que para a GPU, foi utilizado a biblioteca OpenNL [19] e a plataforma CUDA. Para o cálculo de desvios foi reimplementada em CUDA a consolidada RVO2 Library [12].

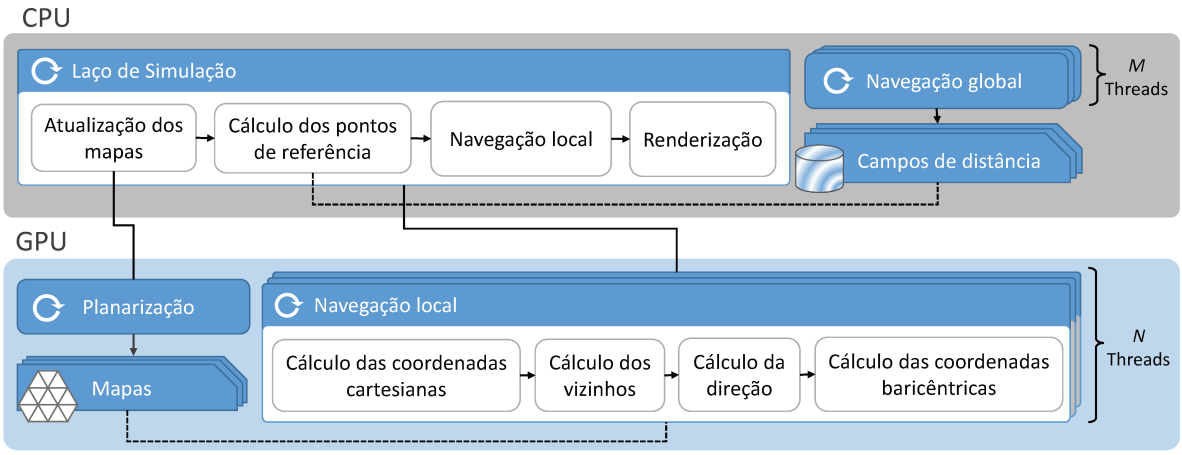
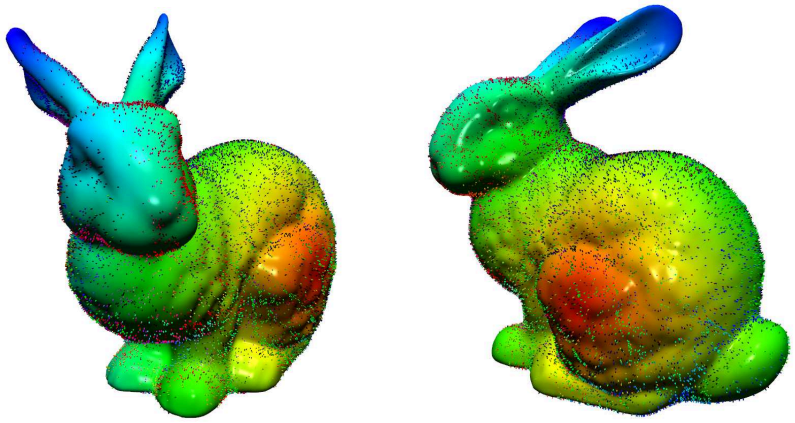


Figura 4.2: Hierarquia computacional. Na CPU é realizado o processamento principal da simulação, e em segundo plano por M threads a navegação global. Na GPU é realizado a planarização dos mapas e a navegação local por N threads.

4.1 Navegação Global

O processo de navegação global é responsável por planejar os caminhos dos agentes até os seus destinos. Para permitir que os agentes mudem de rota sem necessitar um replanejamento, é utilizado o campo de distância, como na Figura 4.3. Desta forma também permitindo que vários agentes com o mesmo destino utilizem o mesmo campo de distância. Para calcular o campo de distância, é utilizado o algoritmo de Dijkstra com heap binomial. Outros algoritmos, como A*, apesar de apresentarem um custo computacional menor para encontrar o melhor caminho, não apresentam vantagens de desempenho para o cálculo de todo o campo de distância.



(a) (b)

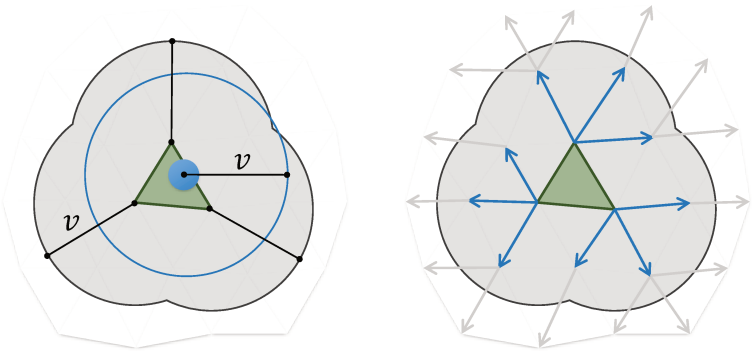
Figura 4.3: Campo de Distância, onde a superfície foi colorida de acordo com a distância até o destino. Para que o agente chegue até o destino é necessário que ele navegue em direção a um lado com distância menor.

Para reduzir o tempo de espera dos agentes por um caminho e aproveitar melhor todos os núcleos do processador, esta etapa é realizada por múltiplas threads. Cada thread da navegação global aguarda a solicitação de novos caminhos através de uma fila que contém os destinos necessários.

4.2 Planarização

O processo de planarização é dividido em duas etapas. Na primeira etapa é realizado a seleção dos triângulos que irão compor o mapa. Para garantir que a região selecionada cubra o campo de visão de qualquer agente no triângulo central, é realizado uma busca utilizado o algoritmo de Djikstra.

O algoritmo de Dijkstra é executado a partir dos pontos do triângulo central e percorre todos os pontos que tiverem distância menor que o raio de visão dos agentes. Após é selecionado os triângulos que tiveram algum ponto alcançado pelo algoritmo, como na Figura 4.4.



(a) (b)

Figura 4.4: Definição da região do mapa. (a) O mapa deve cobrir o raio de visão v para qualquer posição de um agente dentro do triângulo central. (b) Execução do algoritmo de Dijkstra percorrendo apenas os pontos com distância menor que o raio de visão dos agentes.

Na segunda etapa da planarização é realizado a parametrização da região selecionada. Para isso, foi utilizado a abordagem de Levy et al. [16], que além de minimizar as deformações angulares dos triângulos, é capaz de gerar mapas com bordas livres e utilizar computação paralela. O método utilizado não garante minimizar a deformação da escala dos mapas. Para corrigir este problema, é alterado a escala do mapa de forma que a área do triângulo central seja igual a original.

4.3 Simulação

O processo simulação é responsável por realizar o pré-processamento dos mapas, iniciar e integrar o restante dos processos e manter a estrutura dos agentes e da superfície, como no Algoritmo 1. O laço de simulação é dividido em quatro etapas: Atualização dos mapas, cálculo dos pontos de referência, navegação local e renderização.

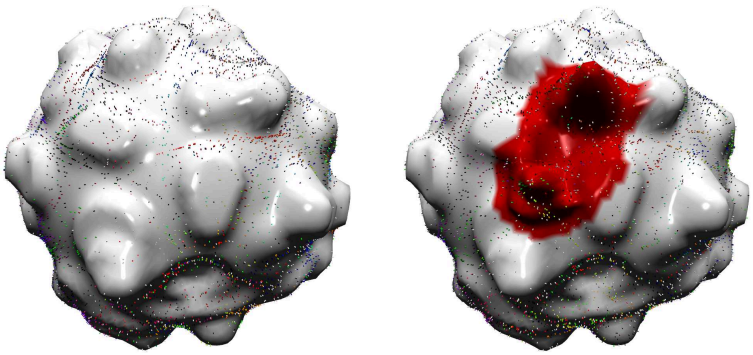
Algoritmo 1: Simulação

```
   // Pré-processamento (opcional)
 1 for t ∈ triangulos da malha do
 2 |   selecionar triângulos ao redor de t;
 3 |   mapas[t] ← parametrização da região de t;
 4 end
 5 for p ∈ pontos da malha do
 6 |   dist[p] ← Dijkstra(p);
 7 end
   // Laço de Simulação
 8 while True do
   |   // Navegação global
 9 |   for a ∈ agentes que aguardam caminho até o destino do
10 |   |   Dijkstra(a_destino);
11 |   end
   |   // Atualização dos Mapas
12 |   for t ∈ mapas que necessitam ser calculados do
13 |   |   selecionar triângulos ao redor de t;
14 |   |   mapas[t] ← parametrização de mapas[t];
15 |   end
   |   // Cálculo dos pontos de referência
16 |   for a ∈ agentes do
17 |   |   if a mudou de triângulo then
18 |   |   |   atualizar ponto de referência de a;
19 |   |   end
20 |   end
   |   // Navegação Local
21 |   for a ∈ agentes do
22 |   |   c ← mapas[a_triangulo] ;
23 |   |   RVO2 ← posição (2D) de a em c ;
24 |   |   RVO2 ← direção de a em c ;
25 |   |   RVO2 ← direção preferencial de a;
26 |   |   for i ∈ outros agentes em c e no raio de visão de a do
27 |   |   |   RVO2 ← posição de i em c;
28 |   |   |   RVO2 ← direção de i em c;
29 |   |   end
30 |   |   direção de a ← RVO2;
31 |   |   posição de a ← RVO2;
32 |   |   atualizar triângulo atual de a;
33 |   |   calcular coordenadas baricêntricas de a;
34 |   end
35 |   Renderizar ambiente;
36 end
```

A organização dos agentes no espaço é realizada de forma semelhante ao trabalho de Jund et al. [11]. A estrutura utilizada armazena uma lista de agentes para cada triângulo da superfície. Desta forma, permitindo que as consultas de vizinhos sejam realizas em tempo real, mesmo com o uso de mapas durante a navegação local.

4.3.1 Atualização dos Mapas

Em casos em que a superfície sofre alterações, como na Figura 4.5, é realizado a atualização dos mapas. A cada execução desta etapa é atualizado a parametrização de um número pré-definido de mapas. Para isso, é utilizado uma fila que armazena os pontos da superfície que foram alterados, também é utilizado uma estrutura que fornece todos os mapas que são afetados por um determinado ponto.



(a) (b)

Figura 4.5: A região em vermelho representa a superfície que foi gradualmente alterada durante a navegação dos agentes. Os mapas que tiveram pontos afetados pela deformação são recalculados.

4.3.2 Cálculo dos Pontos de Referência

Para que a navegação local possa ser realizada, além do mapa da superfície, também deve ser informado um ponto de referência que oriente o agente até o seu destino. Desta forma, não é necessário enviar os campos de distância para a GPU, assim reduzido consideravelmente as transferências e o consumo de memória na GPU.

O ponto de referência é calculado através de uma média ponderada pela perda de distância de cada direção do triângulo, como na Figura 4.6, desta forma suavizando os caminhos que são gerados sobre as bordas dos triângulos. Caso o destino já esteja presente no mapa, sua posição é utilizado como ponto de referência. Se o agente ainda não tiver um caminho planejado

pela navegação global, então é utilizado sua própria posição e assim permanecendo parado.

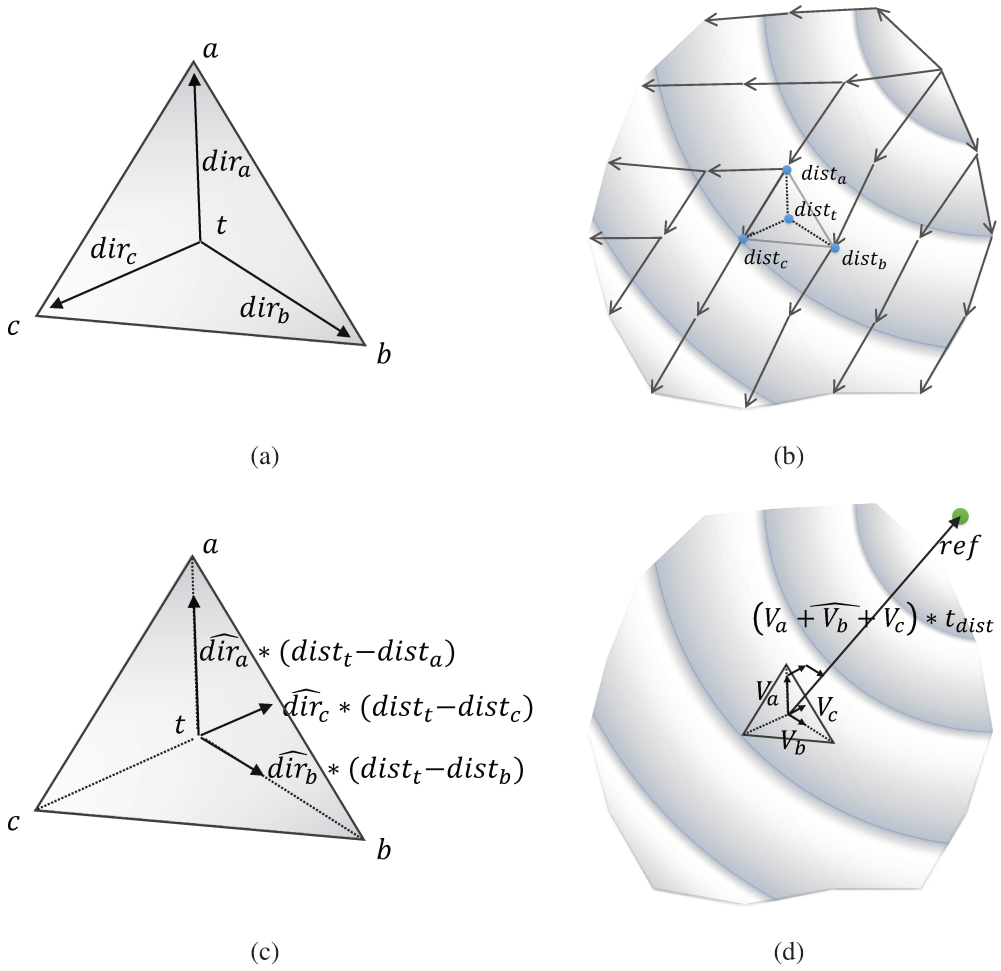


Figura 4.6: Cálculo do ponto de Referência. (a) Triângulo central de uma mapa, onde o ponto t representa o baricentro do triângulo e os vetores dir a direção até cada ponto. (b) Mapa gerado a partir do triângulo abc. O fundo em degrade representa a variação de distância até o destino e as setas indicam como o algoritmo de Dijkstra se propagou pela superfície durante a navegação global. Os valores $dist$ se referem a distância do ponto até o destino, $dist_t$ é calculado através da média dos $dist$ do triângulo. (c) Cada vetor de direção é normalizado e multiplicado pela diferença entre a distância do centro e a do seu ponto, resultando nos vetores V. Como cada direção foi multiplicada pelo seu ganho, todos vetores resultantes apontam para uma direção que reduz a distância até o destino e tem o comprimento de acordo com o seu ganho. (d) A direção do ponto de referência ref é calculada através da média dos vetores V ponderada pelos seus comprimentos.

Como o ponto de referência de cada agente pode ser calculado de modo independente, esta etapa é realizada de forma paralela com o uso da API OpenMP. Já que o ponto de referência não varia enquanto o agente estiver no mesmo mapa, este cálculo só é realizada para os agentes que tiverem trocado de mapa (triângulo).

4.3.3 Navegação Local

Nesta etapa é inicializado na GPU o processo de navegação local. Para isso, antes é realizado o envio para a GPU das informações necessárias de todos os agentes e da estrutura que os organiza sobre a superfície. Após é inicializado a Navegação local e aguardado sua finalização. Por último, é realizado a transferência dos dados da GPU, a atualização de cada agente é realizada de forma paralela com a API OpenMP. Para casos em que o agente troca de triângulo e é necessário atualizar a estrutura de organização dos agentes, é utilizado um semáforo para evitar acessos simultâneos.

4.3.4 Renderização

Nesta etapa é realizado a renderização do ambiente, para isso é calculado a posição global de cada agente a partir de suas coordenadas baricêntricas. Para facilitar a análise foram implementados diversos modos de visualização, como rastros na superfície, mapa do agente, distorção dos mapa, mapa de distância, triângulos da superfície e coloração dos agentes.

Para realizar a análise estatística, esta etapa também gera um arquivo de *log* contendo informações da navegação e performance.

4.4 Navegação Local

O processo de navegação local ocorre na GPU para todos agentes de forma paralela, cada thread é dividida em quatro etapas: Cálculo das coordenadas cartesianas, cálculo dos vizinhos, cálculo da direção e cálculo das coordenadas baricêntricas.

4.4.1 Cálculo das Coordenadas Cartesianas

Na primeira etapa da navegação local é realizado o posicionamento do agente sobre o mapa, para isso é calculado sua posição e direção utilizado as coordenadas baricêntricas. Para calcular a direção preferencial, é utilizado o ponto de referência, como na Figura 4.7.

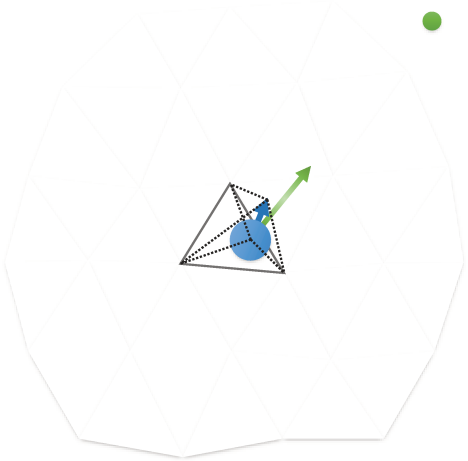


Figura 4.7: Posicionamento do agente no mapa. Em azul o agente e sua direção atual calculados a partir das coordenas baricêntricas. Em verde o seu ponto de referência e a seta representando sua direção preferencial.

4.4.2 Cálculo dos Vizinhos

Nesta etapa é realizado o posicionamento dos outros agentes que estejam dentro do raio de visão. Para isso, é percorrido a lista de agentes de todos os triângulos no mapa, para cada agente é calculado sua posição através das coordenadas baricêntricas, caso ele esteja dentro do raio de visão, também é calculado sua direção e enviado para o RVO2, como na Figura 4.8.

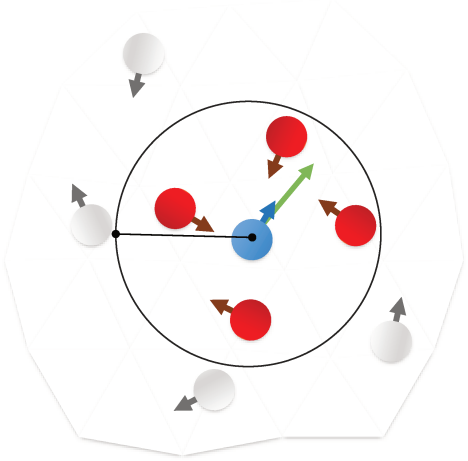


Figura 4.8: Posicionamento dos agentes vizinhos. Em vermelho os agentes que estão dentro do raio de visão do agente em azul, em cinza os agentes que foram ignorados.

4.4.3 Cálculo da Direção

Nesta etapa é calculado uma nova direção para o agente através de uma técnica de prevenção de colisões para superfícies planas. Para garantir navegação sem oscilações e realizar o cálculo de forma independente para cada agente, foi utilizado o método RVO2. O cálculo de uma nova direção para o agente é dividida em duas partes. Primeiro é percorrido todos os agentes vizinhos calculando os semiplanos do método RVO2 e os armazenando em um vetor. Na segunda parte é calculado a direção que esteja mais próxima da preferencial e que menos penetre nos semiplanos já calculados. Desta forma é obtido uma nova direção que evita colisões com os agentes vizinhos, como na Figura 4.9.

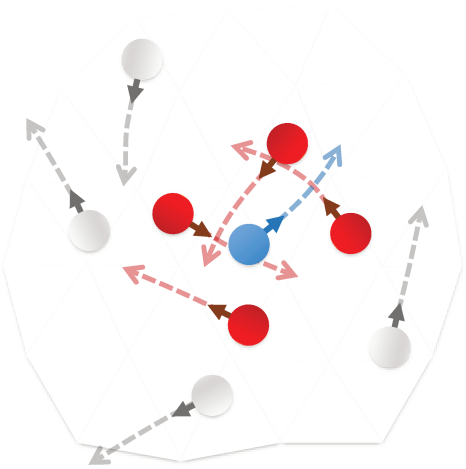


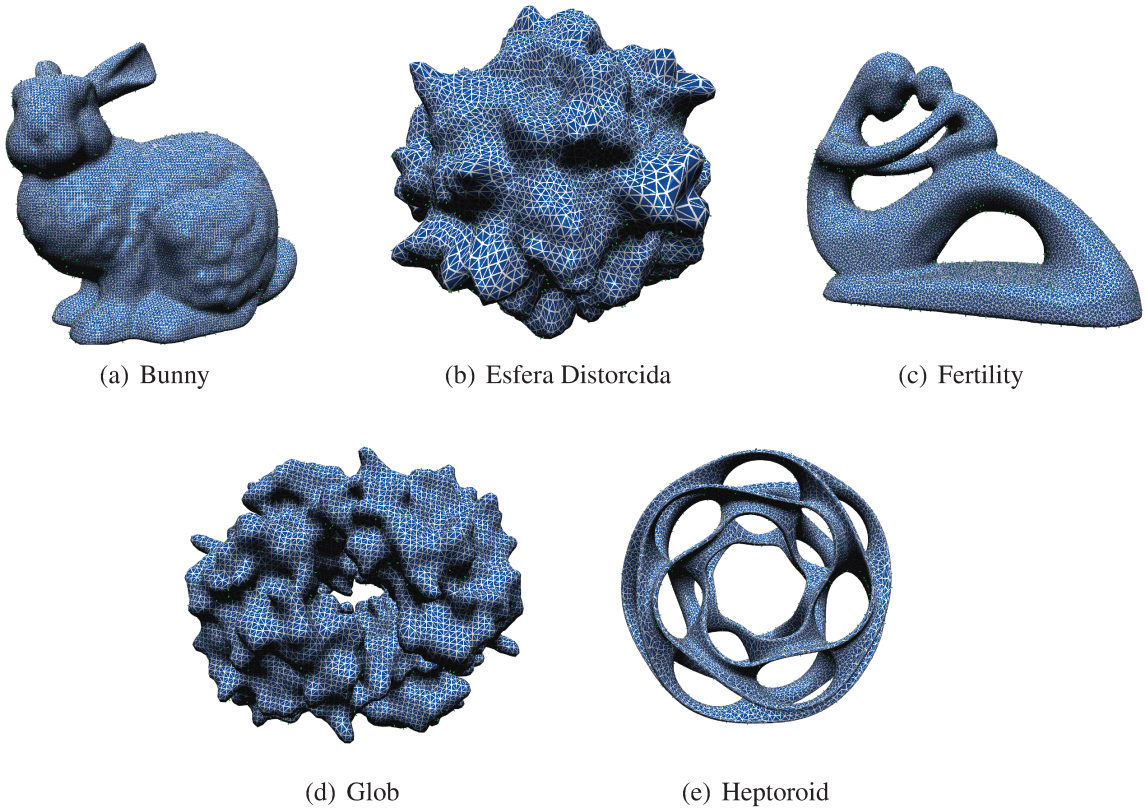
Figura 4.9: Cálculo da direção dos agentes, as setas próximas dos agentes indicam a sua nova direção e as setas tracejadas os possíveis caminhos que serão seguidos para evitar colisões.

4.4.4 Cálculo das Coordenadas Baricêntricas

Na última etapa é ajustado a posição do agente conforme a nova direção e calculado as coordenadas baricêntricas. Para as coordenadas baricêntricas, antes é identificado em qual triângulo o agente está e que deve ser utilizado como referência.

5 RESULTADOS

Os teste foram realizado em um computador Intel I7^{TM}4747k, 16GB ram e NVIDIA Titan^{TM}. Foram utilizados 5 modelos de superfícies, como na Figura 5.1, para cada superfície foram realizados seis testes com o número de agentes variando de 20160 a 120000.



(a) Bunny (b) Esfera Distorcida (c) Fertility

(d) Glob (e) Heptoroid

Figura 5.1: Modelos utilizados nos testes.

A Tabela 5.1 apresenta as malhas utilizadas e o número médio de triângulos por mapa.

Modelo	Triângulos	Pontos	Média de triângulos por mapa local
Stanford Bunny	68958	34481	13.37
Heptoriod	40084	20000	13.60
Fertility	40000	19994	13.48
Glob	29774	14875	13.42
Esfera Distorcida	20480	10242	12.98

Tabela 5.1: Informações dos modelos utilizados.

A qualidade da renderização e as malhas usadas para representar os agentes dependem da aplicação. Para que estes itens não interfiram nas estatísticas de desempenho da navegação, os seguintes números não consideram o custo de renderização.

Em todos os testes foram utilizados os pré-processamentos de mapas e campos de distância. Para superfícies com animação, o desempenho é reduzido de acordo com número de mapas atualizados a cada quadro. Note-se que os mapas podem ser recalculados gradualmente para evitar a instabilidade do FPS.

A Tabela 5.2 apresenta o custo computacional de cada malha para o pré-processamento dos gráficos e dos campos de distância.

Modelo	Tempo de processamento		Memória	
	Planarização	Navegação Global	Caminhos	Mapas
Stanford Bunny	0.0162ms	4.6971ms	4.43GB	24.88MB
Heptoriod	0.0166ms	3.0032ms	1.49GB	14.46MB
Fertility	0.0164ms	2.6485ms	1.49GB	14.43MB
Glob	0.0167ms	1.9496ms	0.82GB	10.74MB
Esfera Distorcida	0.0148ms	1.1191ms	0.40GB	7.39MB

Tabela 5.2: Modelos utilizados e seus custos computacionais. O tempo de planarização é o tempo médio necessário para gerar um único mapa. Além disso, a coluna Navegação Global é o tempo médio para calcular um único campo de distância. O consumo de memória para os campos de distância aumenta polinomialmente com o número de pontos.

Os mapas e desvios de obstáculos são calculados em GPU, enquanto os mapas de distância na CPU. Esta distribuição de trabalho permite que ambos os processadores trabalhem em paralelo, mas existe um custo computacional para transferir os dados. Além disso, o processo principal na CPU precisa aguardar o retorno dos processos na GPU, a Figura 5.2 ilustra esta distribuição de trabalho.

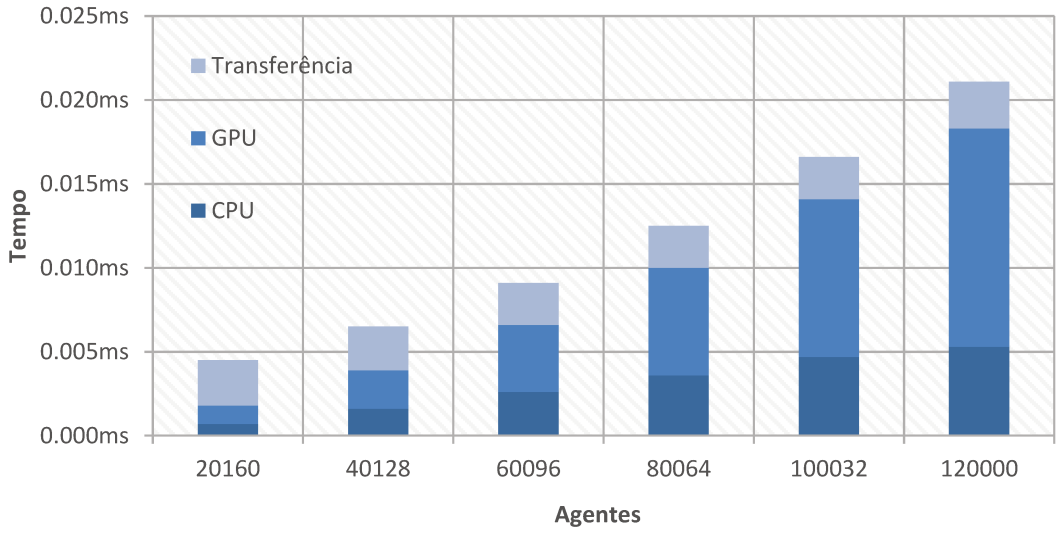


Figura 5.2: Este gráfico apresenta a distribuição do tempo computacional durante as simulações sobre o modelo Stanford Bunny. O tempo de transferência apresenta pouca variação mesmo tendo um aumento de tráfego linear ao número de agentes.

As seguintes Figuras 5.3, 5.4, 5.5, 5.8, 5.6 e 5.7 foram resultados das mesmas simulações.

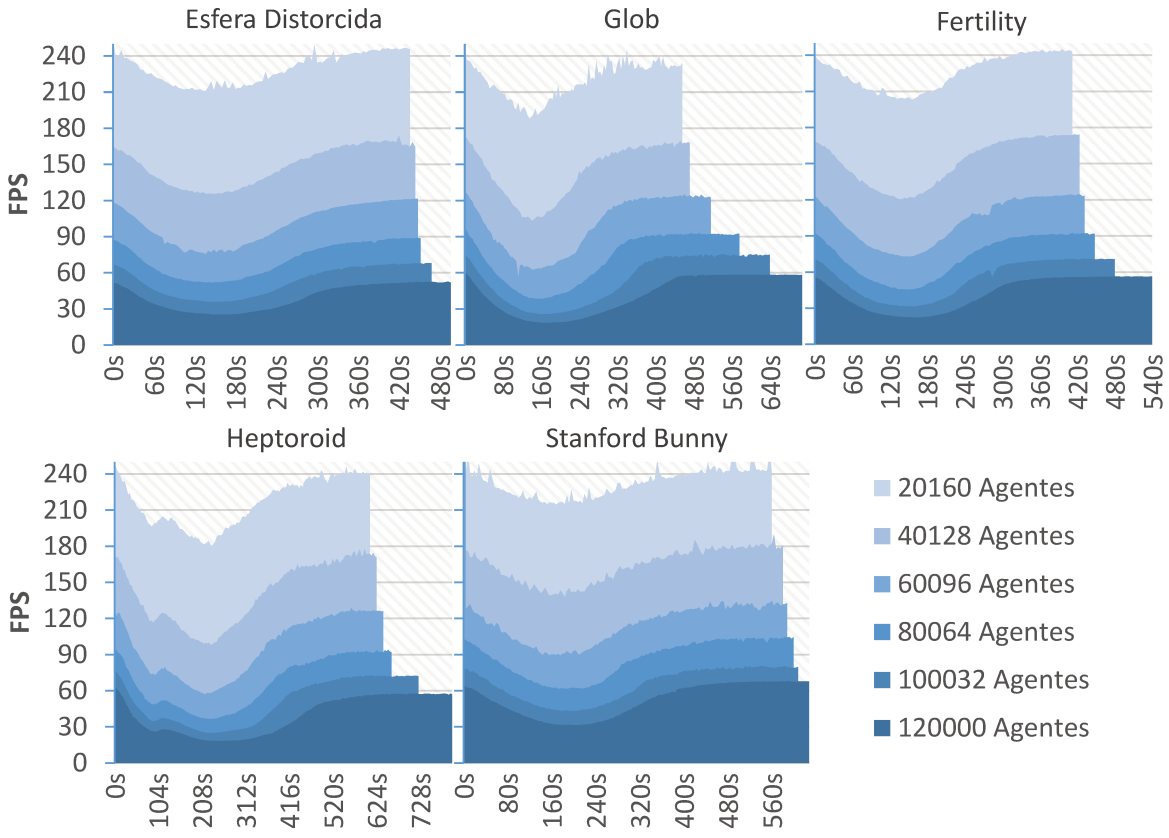


Figura 5.3: Estes gráficos apresentam a variação do FPS durante as simulações.

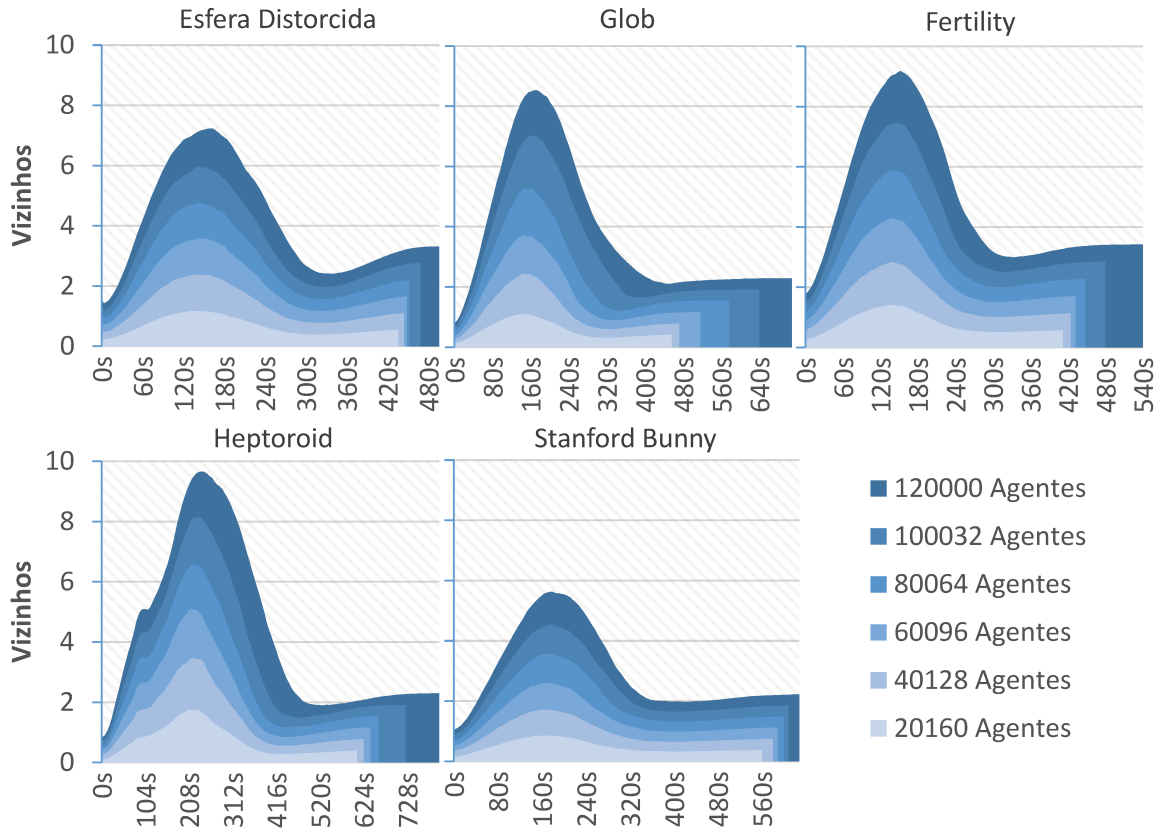


Figura 5.4: Estes gráficos apresentam a variação média do número de vizinhos durante as simulações.

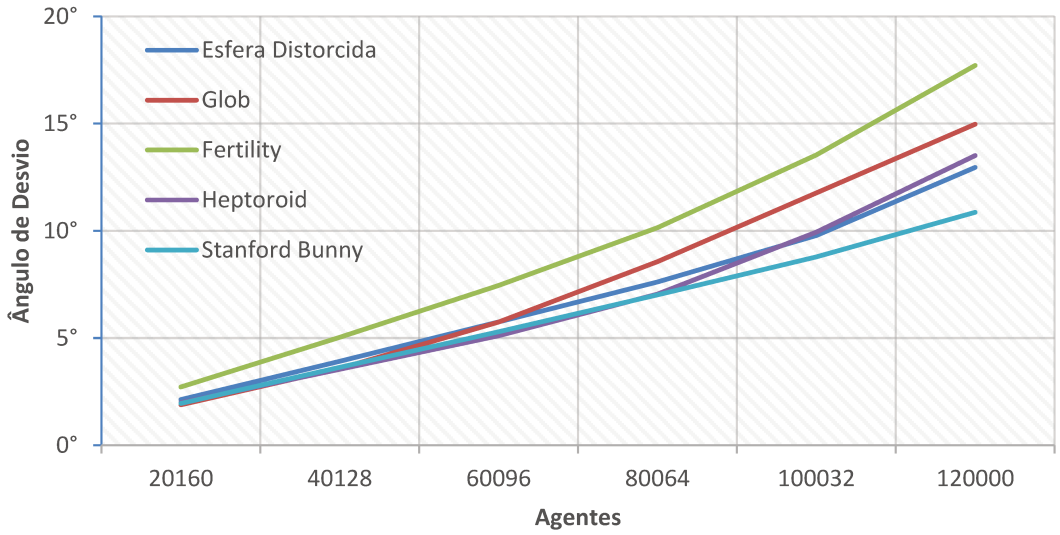


Figura 5.5: Este gráfico mostra a crescente necessidade de se desviar da direção preferencial conforme aumenta o número de agentes.

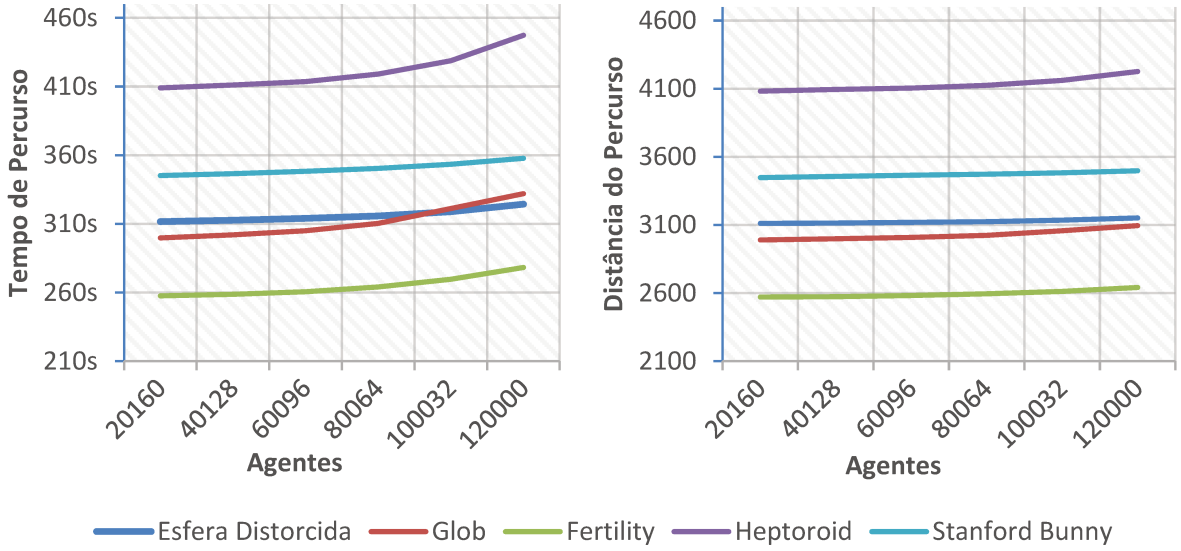


Figura 5.6: Este gráfico ilustra a eficiência do método. O tempo médio e a distância percorrida para os agentes chegarem ao destino são pouco afetados pelo aumento do número de agentes.

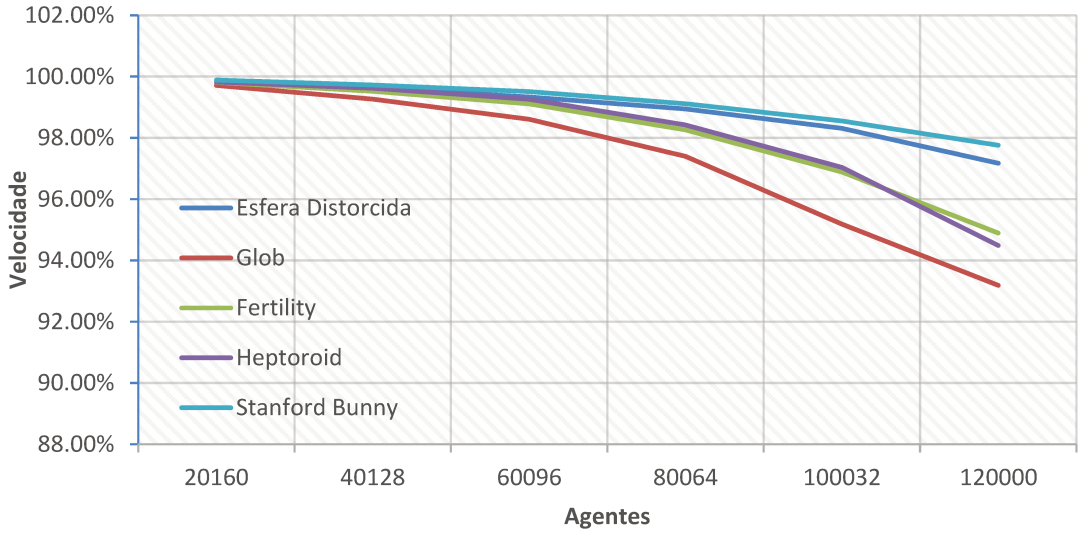


Figura 5.7: Este gráfico mostra a velocidade média do agente durante a navegação, em relação à velocidade máxima permitida.

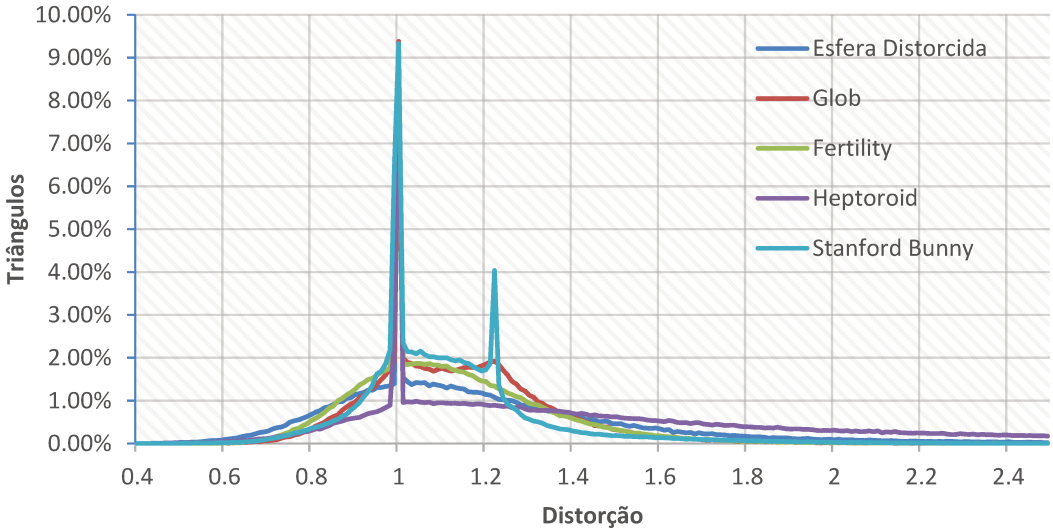


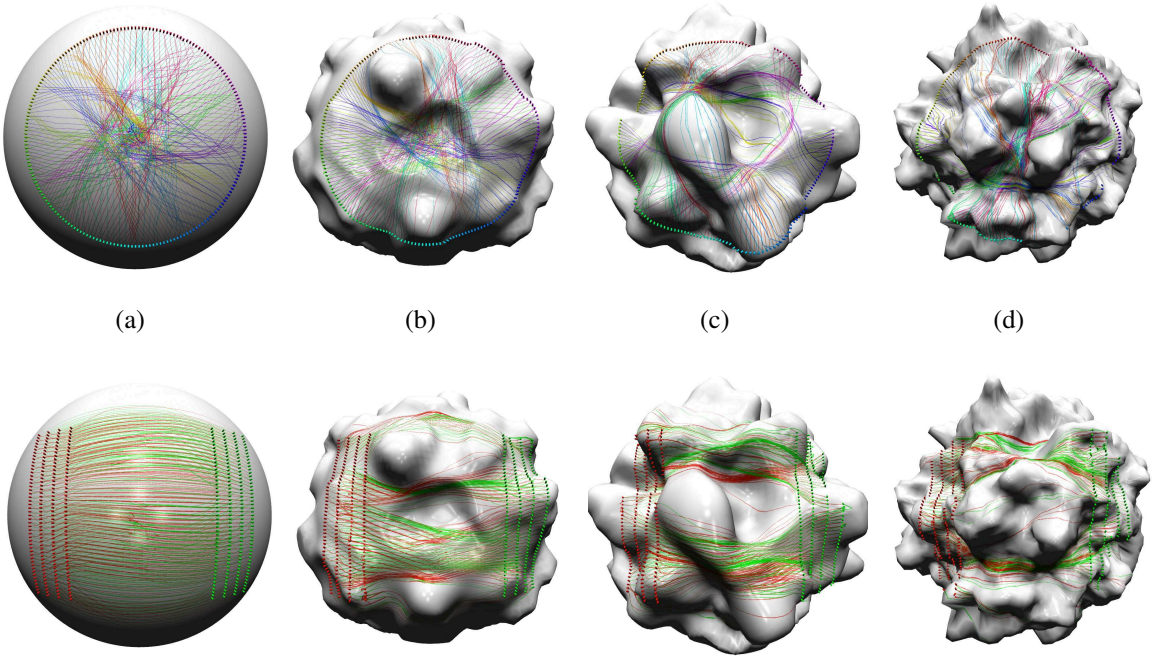
Figura 5.8: Porcentagem de triângulos parametrizados de acordo com a taxa L2 [23]. Distorção igual a 1 significa que o triângulo não foi distorcido com a planarização.

Observe na Figura 5.3 o efeito causado pelo número de vizinhos (Figura 5.4). Quanto maior o número de vizinhos, maior o custo computacional da navegação local.

As Figuras 5.7 e 5.5 também ilustram a eficiência do método. A Figura 5.7 apresenta a velocidade média, que é afetada pelo aumento do número de agentes, mas normalmente fica perto da velocidade desejada. A Figura 5.5 mostra o desvio médio da direção dada pelo ponto de referência. Note-se que a navegação local tem de fazer mais desvios conforme o número de agentes aumenta.

A Figura 5.8 ilustra a taxa de distorção L2 [23] medido para cada triângulo. Note-se que cada triângulo é contabilizadas várias vezes porque está presente em vários mapas. Além disso, a distorção em torno do agente é geralmente menor do que nas bordas do mapa, devido ao bloqueio do triângulo central.

A Figura 5.9 ilustra o comportamento do desvio de obstáculos com o método RVO2, combinado com a planarização. Em ambas as situações, o objectivo de cada agente foi o lado oposto da distribuição inicial, resultando em um gargalo no meio da navegação.



(a) (b) (c) (d)

(e) (f) (g) (h)

Figura 5.9: Dois modos de distribuição dos agentes em superfícies com diferentes níveis de deformação.

6 CONCLUSÃO E TRABALHOS FUTUROS

Este trabalho apresentou um método prático para a navegação de múltiplos agentes sobre superfícies 3D irregulares. Além disso, ele pode lidar com malhas com animação em tempo real. Para melhorar o desempenho, são apresentadas etapas de pré-processamento que permitem desempenho semelhante às simulações em superfícies planas. Também é apresentado resultados estatísticos para ilustrar a qualidade do planejamento de trajetória. Os resultados também mostram que o desempenho alcançado foi obtido devido a abordagem massivamente paralela.

Como trabalho futuro, sugere-se explorar uma abordagem como a de Xin, Ying e He [31] para melhorar a precisão dos mapas de distância. Também sugere-se implementar um método para evitar colisões acima da superfície . Estes método podem ser úteis para jogos, especialmente aqueles de estratégia em tempo real.

REFERÊNCIAS

[1] A. Bleiweiss. Gpu accelerated pathfinding. In *Proceedings of the 23rd ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware*, GH '08, pages 65–74, Aire-la-Ville, Switzerland, Switzerland, 2008. Eurographics Association.

[2] A. Bleiweiss. Multi agent navigation on gpu. In *http://developer.download.nvidia.com/presentations/2009/GDC/ MultiAgentGPU.pdf*, 2009.

[3] J. Chen and Y. Han. Shortest paths on a polyhedron. In *Proceedings of the Sixth Annual Symposium on Computational Geometry*, SCG '90, pages 360–369, New York, NY, USA, 1990. ACM.

[4] K. Crane, F. de Goes, M. Desbrun, and P. Schröder. Digital geometry processing with discrete exterior calculus. In *ACM SIGGRAPH 2013 Courses*, SIGGRAPH '13, pages 7:1–7:126, New York, NY, USA, 2013. ACM.

[5] Cui and Shi. An overview of pathfinding in navigation mesh. In *International Journal of Computer Science and Network Security*, 2012.

[6] E. W. Dijkstra. *Communication with an Automatic Computer*. PhD thesis, University of Amsterdam, 1959.

[7] F. Feurtey. Simulating the collision avoidance behavior of pedestrians. 2013.

[8] P. Fiorini and Z. Shillert. Motion planning in dynamic environments using velocity obstacles. *International Journal of Robotics Research*, 17:760–772, 1998.

[9] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics and Automation*, 4(1), 1997.

[10] K. Hormann, B. Lévy, and A. Sheffer. Mesh parameterization: Theory and practice video files associated with this course are available from the citation page. In *ACM SIGGRAPH 2007 Courses*, SIGGRAPH '07, New York, NY, USA, 2007. ACM.

[11] T. Jund, P. Kraemer, and D. Cazier. A unified structure for crowd simulation. *Comput. Animat. Virtual Worlds*, 23(3-4):311–320, May 2012.

[12] J. S. M. C. L. D. M. Jur van den Berg, Stephen J. Guy. Reciprocal collision avoidance for real-time multi-agent simulation. http://gamma.cs.unc.edu/RVO2/, 2014.

[13] R. Kimmel and J. A. Sethian. Computing geodesic paths on manifolds. In *Proc. Natl. Acad. Sci. USA*, pages 8431–8435, 1998.

[14] Y. Li and K. Gupta. Motion planning of multiple agents in virtual environments on parallel architectures. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 1009–1014, 2007.

[15] L. Liu, L. Zhang, Y. Xu, C. Gotsman, and S. J. Gortler. A local/global approach to mesh parameterization.

[16] B. Lévy, S. Petitjean, N. Ray, and J. Maillo t. Least squares conformal maps for automatic texture atlas generation. In ACM, editor, *ACM SIGGRAPH conference proceedings*, Jul 2002.

[17] J. S. B. Mitchell, D. M. Mount, and C. H. Papadimitriou. The discrete geodesic problem. *SIAM J. Comput.*, 16(4):647–668, Aug. 1987.

[18] S. Petti, T. Fraichard, I. Rocquencourt, and E. D. M. D. Paris. Safe motion planning in dynamic environments. In *in Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS*, pages 3726–3731, 2005.

[19] Project-Team-ALICE. Opennl (open numerical library). http://alice.loria.fr/index.php/software/4-library/23-opennl.html, 2014.

[20] B. C. Ricks and P. K. Egbert. Real-time synthetic vision visibility testing on arbitrary surfaces. 2012.

[21] B. C. Ricks and P. K. Egbert. Improved obstacle relevancy, distance, and angle for crowds constrained to arbitrary manifolds in 3d space. 2013.

[22] B. C. Ricks and P. K. Egbert. A whole surface approach to crowd simulation on arbitrary topologies. *IEEE Transactions on Visualization and Computer Graphics*, 20(2):159–171, 2014.

[23] P. V. Sander, J. Snyder, S. J. Gortler, and H. Hoppe. Texture mapping progressive meshes. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 409–416. ACM, 2001.

[24] A. Sheffer, E. Praun, and K. Rose. Mesh parameterization methods and their applications. *Found. Trends. Comput. Graph. Vis.*, 2(2):105–171, Jan. 2006.

[25] A. Sud, E. Andersen, S. Curtis, M. Lin, and D. Manocha. Real-time path planning for virtual agents in dynamic environments. In *ACM SIGGRAPH 2008 Classes*, SIGGRAPH '08, pages 55:1–55:9, New York, NY, USA, 2008. ACM.

[26] V. Surazhsky, T. Surazhsky, D. Kirsanov, S. J. Gortler, and H. Hoppe. Fast exact and approximate geodesics on meshes. *ACM Trans. Graph.*, 24(3):553–560, July 2005.

[27] R. P. Torchelsen, L. F. Scheidegger, G. N. Oliveira, R. Bastos, and J. a. L. D. Comba. Real-time multi-agent path planning on arbitrary surfaces. In *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '10, pages 47–54, New York, NY, USA, 2010. ACM.

[28] J. Van den Berg, S. Guy, M. Lin, and D. Manocha. Reciprocal n-body collision avoidance. In C. Pradalier, R. Siegwart, and G. Hirzinger, editors, *Robotics Research*, volume 70 of *Springer Tracts in Advanced Robotics*, pages 3–19. Springer Berlin Heidelberg, 2011.

[29] J. van den Berg, M. C. Lin, and D. Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION*, pages 1928–1935. IEEE, 2008.

[30] J. Vannoy. Real-time planning of mobile manipulation in dynamic environments of unknown changes. In *in Proc. Robotics: Science and Systems*, 2006.

[31] S.-Q. Xin, X. Ying, and Y. He. Constant-time all-pairs geodesic distance query on triangle meshes. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '12, pages 31–38, New York, NY, USA, 2012. ACM.

[32] X. Ying, S.-Q. Xin, and Y. He. Parallel chen-han (pch) algorithm for discrete geodesics. *ACM Trans. Graph.*, 33(1):9:1–9:11, Feb. 2014.