RENDERIZAÇÃO EFICIENTE DE NUVENS DE PONTOS BASEADA NO OPERADOR HPR

Renan Galvão Machado e Silva

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientador: Claudio Esperança

Rio de Janeiro

Março de 2012

RENDERIZAÇÃO EFICIENTE DE NUVENS DE PONTOS BASEADA NO OPERADOR HPR

Renan Galvão Machado e Silva

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

Prof. Claudio Esperança, Ph.D.

Prof. Ricardo Guerra Marroquim, D.Sc.

Prof. Guilherme Dias da Fonseca, Ph.D.

RIO DE JANEIRO, RJ – BRASIL

MARÇO DE 2012

Machado e Silva, Renan Galvão

Renderização Eficiente de Nuvens de Pontos Baseada no Operador HPR/Renan Galvão Machado e Silva. – Rio de Janeiro: UFRJ/COPPE, 2012.

XIV, 48 p.: il.; 29, 7cm.

Orientador: Claudio Esperança

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2012.

Referências Bibliográficas: p. 45 – 48.

1. Representação por pontos. 2. Visibilidade. 3. Visualização de nuvem de pontos. 4. Visualização interativa. 5. Reconstrução de Superfície. I. Esperança, Claudio. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

À minha família e à Juliana.

Agradecimentos

Agradeço à minha família, em particular aos meus pais, Antonio e Virginia, por todo amor, incentivo e orientação passados em todos os anos de minha vida. Aos meus irmãos, Diogo e Victor, pela união e exemplo de vida.

À minha namorada Juliana, pelo amor e enorme compreensão nas horas de estudo.

Ao professor e orientador Claudio Esperança, responsável por grande parte da minha formação acadêmica, pelos ensinamentos e orientação durante a execução deste trabalho e por sua amizade.

Aos professores do LCG, Antonio Oliveira, Ricardo Marroquim e Paulo Roma, pela amizade, ensinamentos e dicas.

Aos amigos e colegas do LCG, Alberto, Bruno, Carlos Eduardo, Fábio, Felipe, Luciano, Rafael, Rubens, Tiago e Yalmar pelos momentos de descontração e pela ajuda nas horas de trabalho.

Ao professor Diego Nehab pelos ensinamentos de programação em GPU.

Agradeço aos professores Ricardo Marroquim e Guilherme Fonseca pela participação na banca examinadora e contribuição neste trabalho.

Ao CNPq pelo financiamento da pesquisa.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

RENDERIZAÇÃO EFICIENTE DE NUVENS DE PONTOS BASEADA NO OPERADOR HPR

Renan Galvão Machado e Silva

Março/2012

Orientador: Claudio Esperança

Programa: Engenharia de Sistemas e Computação

Recentemente, Katz et al. [1] mostraram como a informação de visibilidade de uma nuvem de pontos pode ser extraída, independentemente da renderização e sem realizar a reconstrução de superfície, pelo chamado operador HPR (*Hidden Point Removal*). Em suma, o operador consiste de uma simples transformação da nuvem de pontos, seguida por uma computação de fecho convexo. Uma vez que que para computar o fecho convexo leva tempo $O(n \log n)$ no pior caso, este método não pode ser usado em aplicações de tempo real para nuvens de pontos médias e grandes. Neste trabalho, é descrita uma implementação em GPU de um algoritmo de fecho convexo aproximado baseado no algoritmo de Kavan et al. [2]. Além disso, é descrito uma maneira de computar uma reconstrução parcial de superfície a partir de uma simples triangulação dos pontos visíveis. Experimentos mostraram que o método proposto pode ser usado em aplicações como a renderização de nuvem de pontos e reconstrução parcial de superfície a taxas interativas.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

EFFICIENT HPR-BASED RENDERING OF POINT CLOUDS

Renan Galvão Machado e Silva

March/2012

Advisor: Claudio Esperança

Department: Systems Engineering and Computer Science

Recently, Katz et al. [1] have shown how visibility information can be extracted from a point cloud by the so-called HPR operator (Hidden Point Removal). In a nutshell, the operator consists of a simple transformation of the cloud followed by a convex hull computation. Since convex hulls take O(n log n) time to compute in the worst case, this method has been considered impractical for real-time applications using medium to large point clouds. In this work, a GPU implementation of an approximate convex-hull algorithm based on the Kavan et al. [2] algorithm is presented. Moreover, it's described a way to compute a partial surface reconstruction from a simple triangulation of the visible points. Experiments show that the method is suitable for real-time rendering and partial reconstruction of point clouds interactively.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

CUDA	Compute Unified Device Architecture, p. 25
DOP	Discrete Oriented Polytope, p. 16
GPU	Graphics Processing Unit, p. 4
HPR	Hidden Point Removal, p. 3

Capítulo 1

Introdução

Malhas poligonais ainda são a representação de superfícies mais comum em diversas aplicações de computação gráfica. Entretanto, nos últimos anos, representações baseadas em pontos ressurgiram como alternativa às malhas poligonais.

A necessidade de se representar objetos cada vez mais detalhados levaram a um aumento da complexidade poligonal dos modelos 3D. O custo adicional para armazenar e processar a informação topológica de malhas polinomiais grandes é uma das razões para o interesse em uma geometria baseada em pontos. Além disso, a resolução dos dispositivos de saída (monitores) não acompanham a resolução dos modelos. Desta forma a rasterização de triângulos tornou-se ineficiente, pois muitos triângulos são projetados em menos de um *pixel* na imagem renderizada.

Por outro lado, a evolução de sistemas de escaneamento 3D, que realizam a aquisição da geometria e da aparência de objetos reais complexos, permitiu a obtenção de amostras mais precisas e mais numerosas. Para obter resultados de alta qualidade, geralmente é preciso uma amostragem mais densa para as nuvem de pontos do que para malhas poligonais. Apesar disso, a simplicidade da representação baseada em pontos compensa o maior número de primitivas. A nuvem de pontos resultante pode ser considerada uma amostragem da superfície processada. Uma amostra da nuvem de pontos é formada obrigatoriamente por sua posição 3D e possivelmente outras propriedades tais como normal, cor ou propriedades de material.

1.1 Motivação

A noção de visibilidade é bem definida para as representações de superfícies tradicionais. O mesmo não pode ser dito para nuvens de pontos, pois pontos não possuem área e assim não podem ocluir um outro ponto.

Uma maneira de resolver o problema de visibilidade a partir de um ponto de vista é realizar a reconstrução de superfície. O problema de converter uma representação de um objeto baseada em pontos em uma representação de superfície (malha poligonal, superfícies paramétricas ou implícitas) é conhecido como reconstrução de superfície. Inúmeros trabalhos [5–16] estudaram este problema que não é simples tanto na teoria quanto na prática. A densidade da nuvem de pontos é um fator fundamental para se obter a reconstrução. A reconstrução de superfície a partir de pontos e normais é um problema mais simples, sendo assim, algumas abordagens exigem que a nuvem venha acompanhada com essa informação e outras tentam estimá-la. Uma vez definida a superfície, a visibilidade pode ser computada utilizando os métodos tradicionais de determinação de superfícies ocultas. Isto significa que a nuvem de pontos contém informações inerentes à visibilidade.

O operador HPR(*Hidden Point Removal*) proposto recentemente por Katz et al. [1] é um algoritmo simples que determina a visibilidade de nuvem de pontos sem estimar as normais e sem realizar a reconstrução. As Figuras 1.1 (a) e (c) mostram as nuvens de pontos do modelo do *Armadillo* e *Bimba*, respectivamente. Com todos os pontos renderizados, fica difícil determinar se os objetos estão de frente ou de costas. Após a aplicação do operador HPR, pode ser visto nas Figuras 1.1 (c) e (d) que os objetos estão de costas.

O operador HPR computa o fecho convexo dos pontos obtidos a partir de uma transformação da nuvem original para computar a visibilidade. O algoritmo não depende da resolução da tela, pois opera no espaço do objeto e funciona bem para nuvens esparsas e densas. Além disso, o operador permite computar uma reconstrução parcial de superfície sem nenhum custo adicional. O método, no entanto, não funciona bem com nuvens de pontos ruidosas e em regiões que possuem uma

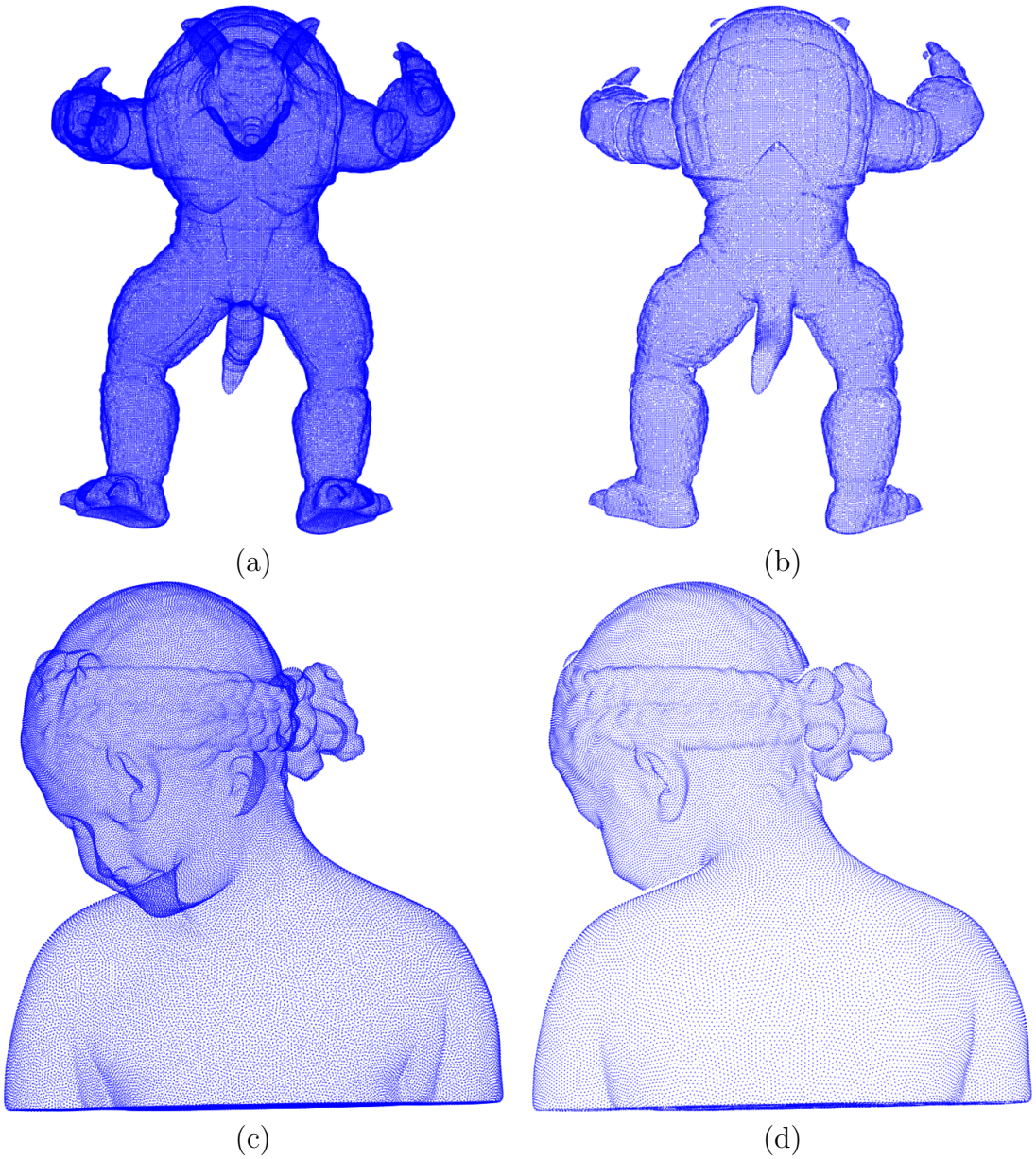


Figura 1.1: Exemplo.

alta curvatura [17].

1.2 Objetivos

O objetivo deste trabalho é pesquisar e desenvolver maneiras de tornar o operador HPR mais eficiente, obtendo taxas interativas. Apesar de sua simplicidade, o operador HPR depende da computação do fecho convexo, que no pior caso não pode ser computado em tempo mais rápido que $O(n \log n)$ para uma nuvem de n pontos em três dimensões, impedindo sua utilização em aplicações em tempo real.

Mais especificamente, este trabalho analisou a utilização de algoritmos de fecho convexo aproximado. A primeira e óbvia motivação para computar um fecho convexo aproximado é o tempo de execução. A segunda motivação reside no fato do operador original apresentar uma taxa de erro que depende da densidade de amostragem da nuvem e de um parâmetro de entrada, como será mostrado mais adiante. Como o resultado do operador original já é aproximado, o método para se chegar até ele também pode ser aproximado. Além disso, este trabalho buscou algoritmos que fossem adequados ao modelo de computação paralela das placas gráficas atuais. Embora o problema de computar o fecho convexo em 2D possa ser resolvido de forma eficiente na GPU (*Graphics Processing Unit*), não há nenhuma solução paralela que funcione bem na GPU para o problema em 3D. Durante o desenvolvimento deste trabalho, Gao et al. [18] apresentaram uma nova abordagem, denominada *gHull*, para computar o fecho convexo para um conjunto de pontos em 3D usando a GPU. Os autores exploraram a computação rápida do diagrama de Voronoi digital e a sua relação com o fecho convexo para computar a resposta do anterior em vez de diretamente. O algoritmo proposto é robusto, com capacidade para maximizar o paralelismo disponível na GPU e obter ganhos de velocidade significativos. Testes realizados mostraram ganhos de desempenho de $3x$ a $10x$ em relação ao *Quickhull* [19].

1.3 Contribuições

Neste trabalho, foram propostas modificações ao operador HPR. A segunda etapa do operador que computa o fecho convexo é custosa, então são empregados algoritmos de fecho convexo aproximado para obter uma solução mais rápida, mesmo à custa de precisão. Primeiramente, foi utilizado o algoritmo proposto Bentley et al. [3]. A ideia do algoritmo é usar um subconjunto dos pontos e em seguida user o fecho convexo dessa amostra como um aproximação do fecho convexo do conjunto completo de pontos. A quantidade de pontos amostrada é controlada por um parâmetro. Com essa abordagem, foi possível reduzir o tempo de execução, porém não foi possível

obter taxas interativas. Como pode ser notado, esta abordagem apenas filtra um conjunto de pontos, mas ainda depende de um algoritmo de fecho convexo exato.

A segunda abordagem se baseia na propriedade explorada por Kavan et al. [2] de que um ponto extremo em uma dada direção pertence ao fecho convexo. O método proposto possui características adequadas para uma implementação em GPU. Assim como a abordagem anterior, é possível definir uma taxa de amostragem que controla a qualidade do resultado final. Além disso, é apresentado como uma reconstrução parcial de superfície dependente do ponto de vista pode ser obtida. Uma malha poligonal é extraída a partir de uma triangulação dos pontos visíveis, tornando possível empregar técnicas tradicionais de malhas para renderização de nuvem de pontos. O nível de aproximação pode ser calibrado a fim de obter uma visualização fiel do objeto para uma dada resolução de tela.

Os resultados mostram que a partir de uma determinada taxa de amostragem, a taxa de acerto dos pontos classificados como visíveis é muito próxima à do operador exato. O método proposto foi utilizado para visualização de nuvem de pontos e reconstrução parcial de superfície em diversos modelos com tamanhos diferentes. Com a implementação em GPU, foi possível obter taxas interativas para modelos de tamanhos moderados. Para nuvens da ordem de milhões de pontos, é possível obter taxas interativas com uma amostragem apropriada para a resolução da tela e que produz resultados visuais bons.

1.4 Organização do Texto

Este trabalho está organizado da seguinte forma. O Capítulo 2 apresenta o problema visibilidade em nuvem de pontos juntamente com uma breve revisão de métodos de renderização baseada em pontos e reconstrução de superfícies; dois problemas com objetivos diferentes, porém capazes de tratar o problema de visibilidade. Além disso, é descrito o operador HPR e suas propriedades. Em seguida, o Capítulo 3 apresenta o problema de fecho convexo de um conjunto de pontos e a descrição de diversos algoritmos de fecho convexo aproximado. Além disso é proposto um novo algoritmo,

obtido pela combinação de ideias de dois métodos detalhados. O Capítulo 4 descreve a principal contribuição do trabalho, uma maneira de computar o operador HPR de forma aproximada, porém com um ganho de velocidade que permite computar a visibilidade de nuvens de pontos com taxas interativas. Além disso, é apresentada uma maneira de obter uma reconstrução parcial de superfície a partir dos pontos visíveis, produzindo uma malha poligonal, também a taxas interativas. O Capítulo 5 apresenta os resultados obtidos para o problema de visibilidade em si e para as aplicações de visualização de nuvens de pontos e reconstrução parcial de superfície. Finalmente, no Capítulo 6, são relatadas as conclusões do trabalho, as limitações e os trabalhos futuros.

Capítulo 2

Trabalhos Relacionados

Visibilidade é um dos principais problemas da Computação Gráfica. Normalmente, busca-se a ordenação das primitivas de acordo com a sua profundidade. Há técnicas que operam no espaço do objeto, como a técnica de traçado de raios, que computa a interseção entre um raio que parte do ponto de vista e a primitiva, tipicamente um polígono. Essa técnica apresenta os melhores resultados, porém é muito cara computacionalmente. Diversos trabalhos estudaram maneiras de otimizar este método [20]. A técnica mais empregada até hoje é o *z-buffer* [21], que opera no espaço da imagem e se aproveita da aceleração por *hardware* para obter resultados mais rápidos. O objetivo deste trabalho é extrair a visibilidade a partir de um conjunto de pontos, porém a maioria dos métodos citados computa a visibilidade de polígonos.

Os métodos de renderização de nuvens de pontos foram os primeiros a tratar o problema de visibilidade de pontos. A qualidade do resultado destes métodos dependem do cálculo da visibilidade. Analisando tanto o método de traçado de raios quanto o de *z-buffer* é possível perceber a dificuldade trazida pelos modelos baseados em pontos. Como pontos e raios são primitivas singulares, o traçado de raios não é uma boa abordagem. Uma solução para este problema é traçar raios com espessura, por exemplo cilindros [22] ou cones [23]. Estes métodos, entretanto, não operam a taxas interativas. Uma outra solução é associar ao ponto uma primitiva com área positiva [24]. Em contraste aos métodos de traçado de raios, há os métodos baseados em *splatting*. *Splatting* é uma técnica eficiente para a renderização baseada

em pontos que usa o *z-buffer* para resolver o problema da visibilidade. Essa técnica rasteriza cada ponto associando ao pixel mais próximo a cor do ponto. A densidade, normalmente, não é suficiente para evitar buracos na imagem final. Neste caso, também é associado ao ponto uma primitiva com área positiva, por exemplo, quadrados, círculos ou elipses. Para obter imagens realistas, é necessário utilizar um filtro no espaço da imagem.

Como os pontos da nuvem são amostras de uma superfície, uma outra maneira para computar a visibilidade é reconstruir a superfície. A abordagem mais comum considera informação topológica extraída dos mapas de altura [7, 8] produzidos pelos *scanners* 3D. Outros métodos não consideram conectividade, mas necessitam da normal do ponto [16]. Em contraste, métodos como [5] não consideram a conectividade nem a normal dos pontos.

Os algoritmos de superfícies implícitas buscam criar uma função $f : \Re^3 \rightarrow \Re$ a partir da entrada amostrada, definida de modo que seja positiva fora e negativa dentro do objeto. Sendo assim, a superfície pode ser extraída como a superfície de nível zero de $f()$. Uma malha poligonal pode ser então obtida utilizando um algoritmo de poligonização tal como *marching cubes* [25].

2.1 Operador HPR

O operador HPR [1] computa visibilidade independentemente da renderização e sem realizar a reconstrução da superfície. Ele trata nuvens sem normal por ponto e não precisa estimá-las. O método consiste de dois passos: inversão e construção do fecho convexo.

O passo da inversão mapeia os pontos em um espaço dual. Seja P uma amostragem de superfície S e um ponto de vista de C. Primeiro, sem perda de generalidade, P é movido para um sistema de coordenadas com origem em C. Inversão é uma função que mapeia um ponto $p_i \in P$ ao longo do raio de C a p_i e é monotonicamente decrescente em $||p_i||$. Há diversas maneiras de realizar o passo de inversão. Neste trabalho, foi utilizado *spherical flipping*, como sugerido em [1].

Considere uma esfera d-dimensional com raio R centrada na origem C e que contenha todos os pontos de P. *Spherical flipping* reflete um ponto $p_i \in P$ em relação à esfera de acordo com a seguinte equação:

$$\widehat{p_i} = f(p_i) = p_i + 2(R - ||p_i||)\frac{p_i}{||p_i||} \tag{2.1}$$

Spherical flipping reflete todo ponto p_i interno à esfera, para um ponto fora da esfera, ao longo do raio de C a p_i, de acordo com a Figura 2.1.

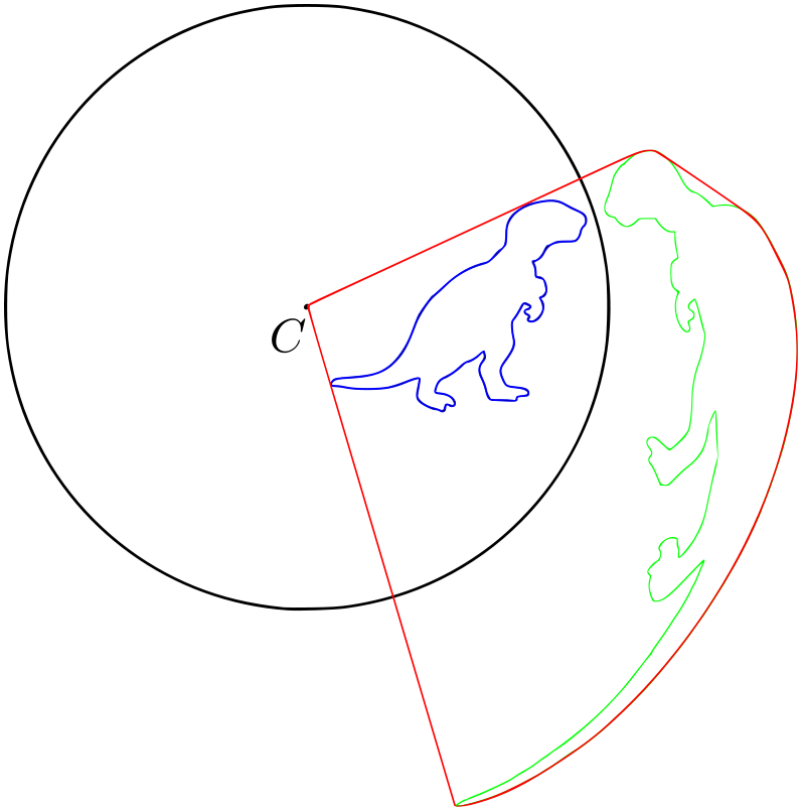


Figura 2.1: A curva em verde é obtida através da inversão (*spherical flipping*) da curva 2D em azul, usando o círculo de raio R centrado em C. A curva em vermelho é o fecho convexo do conjunto de pontos composto da curva invertida e do centro C. Esta imagem é usada apenas para ilustração, na prática, R é muito maior.

Seja $\widehat{P} = \{\widehat{p_i} = f(p_i) \mid p_i \in P\}$ a nuvem de pontos obtida a partir dos pontos transormados de P. O operador HPR estima que um ponto p_i é visível a partir de C se o seu ponto invertido $\widehat{p_i}$ reside no fecho convexo de $\widehat{P} \cup \{C\}$.

O operador pode ser aplicado em nuvens de pontos em qualquer número de dimensões. O passo da inversão tem complexidade assintótica $O(n)$, independente-

mente da dimensão, para uma nuvem de pontos com n pontos. O fecho convexo pode ser computado em $O(n \log n)$ em 2D e 3D. Entretanto, para dimensões maiores, a complexidade do fecho convexo não é mais linear no número de pontos. O chamado Teorema do Limite Superior determina que complexidade de pior caso do fecho convexo de n pontos em d-dimensões é $\Theta(n^{\lfloor d/2 \rfloor})$. Este trabalho analisa somente o caso em 3 dimensões.

A corretude do método é provada quando a entrada é a própria superfície. Neste caso, todo ponto marcado pelo operador como visível é realmente visível. Repare que ainda pode haver falsos negativos, ou seja, pontos visíveis marcados como não visíveis. No limite, quando $R \to \infty$, todo ponto visível será classificado como visível. O aumento do valor de R serve para tratar as regiões com alta curvatura. Na prática, porém, a entrada é uma amostragem da superfície. Neste caso, pode haver falsos positivos, ou seja, pontos não visíveis marcados como visíveis. À medida que R cresce, mais pontos são considerados visíveis. Isto acontece pois os pontos ao serem transformados por *spherical flipping* com uma esfera de raio infinito, a distância deles para o centro da esfera (ponto de vista) fica desprezível e portanto eles tendem a se encontrar no fecho convexo. Desta maneira, são utilizados valores de R grandes para nuvens densas e valores pequenos para nuvens esparsas.

Além da ineficiência em regiões de alta curvatura, Mehra et al. [17] também mostraram que o operador é suscetível a ruídos e propuseram um algoritmo robusto. Apesar da simplicidade do algoritmo HPR, não é possível obter taxas interativas para nuvens grandes. Tavares et al. [26] computaram a visibilidade de nuvem de pontos combinando a técnica de *splatting* com operadores morfológicos no *z-buffer* para reduzir os efeitos de silhueta na imagem final. O método foi implementado na GPU e obteve taxas interativas para visualização de nuvem de pontos. Por outro lado, este método apresenta uma taxa de erro superior ao operador HPR e depende da fase de renderização.

2.2 Reconstrução de Superfícies

A visualização da nuvem de pontos é realizada exibindo os pontos do fecho convexo. $\widehat{P} \cup \{C\}$. O operador HPR pode ser usado para computar uma reconstrução parcial de superfície a partir de um ponto de vista. A reconstrução é computada considerando os triângulos que compõem o fecho convexo. Neste processo são criados triângulos, normalmente próximos a fronteira da nuvem, que claramente não fazem parte da superfície esperada. Em [1], Katz et al. removeram os triângulos cujas arestas são maiores que um limiar. Neste trabalho, foram removidos os triângulos cujas arestas projetadas (espaço de imagem) são maiores que um limiar. Note que a reconstrução não aumenta a complexidade do algoritmo, pois o fecho convexo é sempre computado.

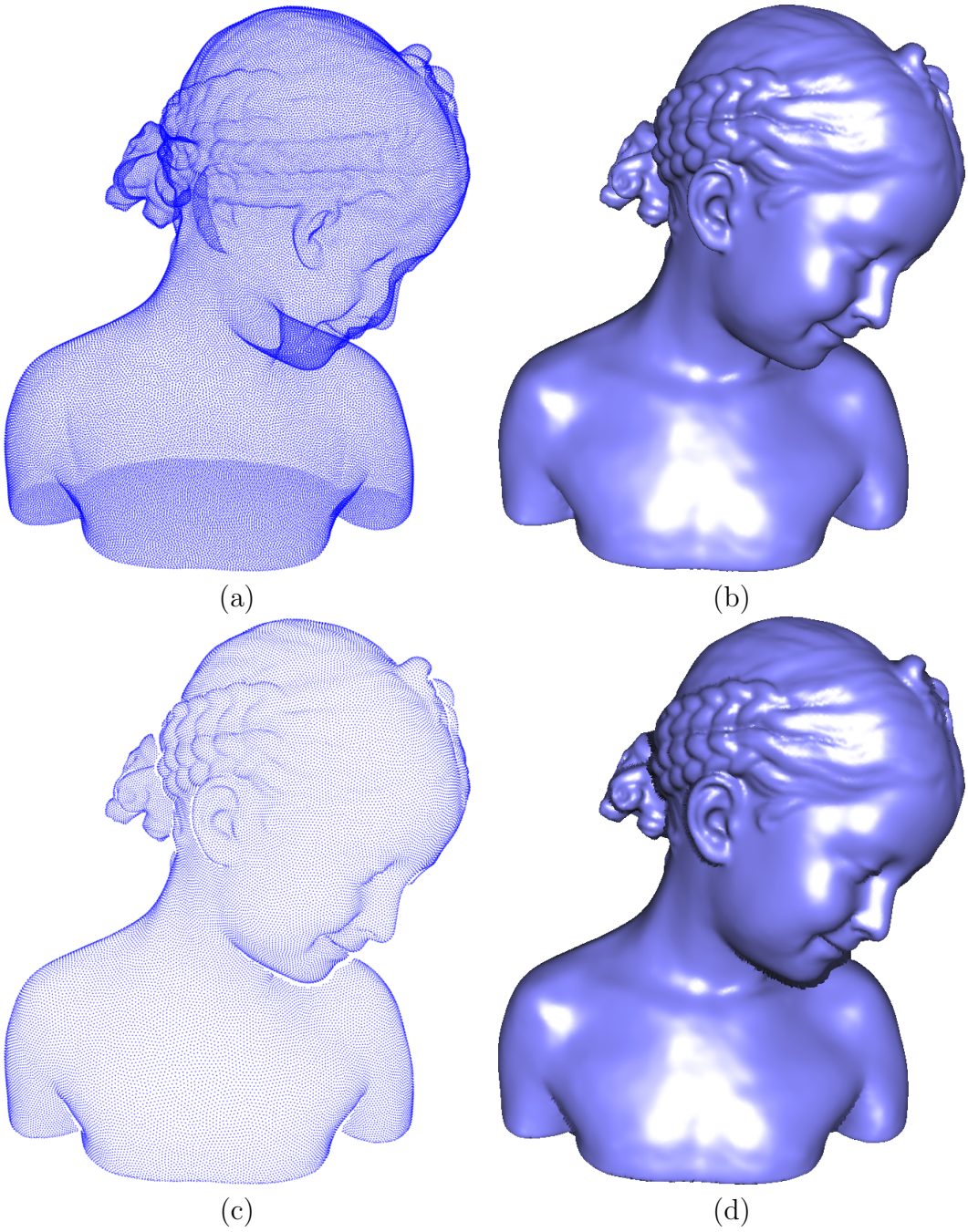


Figura 2.2: Exemplo.

Capítulo 3

Fecho Convexo Aproximado

O fecho convexo de um conjunto S é menor conjunto convexo que contenha S. Para um conjunto finito de pontos $P \in \mathbb{R}^d$, o fecho convexo forma uma politopo convexo em $\mathbb{R}^d$. O problema de computar o fecho convexo é um tópico clássico em geometria computacional e apresenta uma vasta literatura. O fecho convexo em e 2 e 3 dimensões pode ser computado em $O(n \log n)$, no entanto, para dimensões maiores que 3, a complexidade do fecho convexo não é mais linear no número de pontos. O chamado Teorema do Limite Superior determina que complexidade combinatorial de pior caso do fecho convexo de n pontos em d-dimensões é $\Theta(n^{\lfloor d/2 \rfloor})$.

Como discutido anteriormente, apesar da simplicidade, o operador HPR depende da computação do fecho convexo que tem complexidade assintótica $O(n \log n)$ para uma nuvem de n pontos em três dimensões. Em [1], os autores empregaram o algoritmo *Quickhull* [19], que computa o fecho convexo em 3D em tempo $O(n \log n)$ para entradas favoráveis, porém é quadrático no pior caso, tornando-o ineficiente para lidar com nuvens de pontos grandes.

Uma maneira de aumentar a velocidade da técnica é usar um algoritmo de fecho convexo aproximado. O fato do operador HPR ser aproximado para entradas que são amostragens de uma superfície, reforça esta ideia, uma vez que os erros introduzidos por uma técnica e a outra são independentes. Além disso, quando usado para renderização, o operador poderia trabalhar com uma amostra pequena, porém suficientemente boa para a resolução da tela.

Neste capítulo serão apresentados alguns algoritmos de fecho convexo aproximado.

3.1 Algoritmo de Bentley, Faust e Preparata

A principal ideia do algoritmo proposto por Bentley et al. [3] é obter um sub-conjunto dos pontos e então usar o fecho convexo dessa amostragem como uma aproximação do fecho convexo do conjunto completo de pontos.

A Figura 3.1 ilustra os passos do algoritmo que será descrito em duas dimensões, porém é possível generalizá-lo para qualquer dimensão. No primeiro passo do algoritmo, são achados os valores extremos na dimensão x. A região entre estes dois valores é dividida em k faixas igualmente espaçadas; veja a Figura 3.1(a). Em seguida, para cada faixa são achados os pontos com valores extremos de y; veja a Figura 3.1(b). Estes pontos, juntamente com os pontos com valores extremos de x formam o subconjunto. Havendo diversos pontos com valores extremos de x, são escolhidos os pontos com valores extremos na dimensão y. Sendo assim, há no máximo $2k + 4$ pontos amostrados. Finalmente, o fecho convexo dos pontos amostrados é computado e usado como uma aproximação do fecho do conjunto; veja a Figura 3.1(c). Note que o fecho resultante é apenas uma aproximação, pois há pontos do conjunto original que estão fora do fecho. Note também que como o método utiliza um algoritmo exato em um subconjunto dos pontos, então o fecho obtido é uma aproximação interna do fecho exato, isto é, todos os pontos do fecho aproximado estão dentro do fecho exato; veja a Figura 3.1(d).

Este algoritmo tem complexidade de tempo $O(n + k)$. Computar os valores máximo e mínimo de x e os extremos em cada faixa requer tempo $O(n)$. Usando ingenuamente um dos diversos algoritmos que computam o fecho convexo de um conjunto de pontos no plano em tempo $O(n \log n)$, é possível computar o fecho convexo em tempo $O(k \log k)$, pois há no máximo $2k + 4$ pontos amostrados. No entanto, repare que ao distribuir os pontos nas faixas, eles passam a estar ordenados na direção x. Desta forma é possível utilizar uma variante [27] do *Graham's Scan*

[28] que realiza uma varredura da esquerda para direita em vez de sentido anti-horário e computa o fecho em tempo $O(n)$ para um conjunto previamente ordenado. Com isso, computar o fecho passar a levar tempo $O(k)$ e a complexidade total do algoritmo é $O(n + k)$.

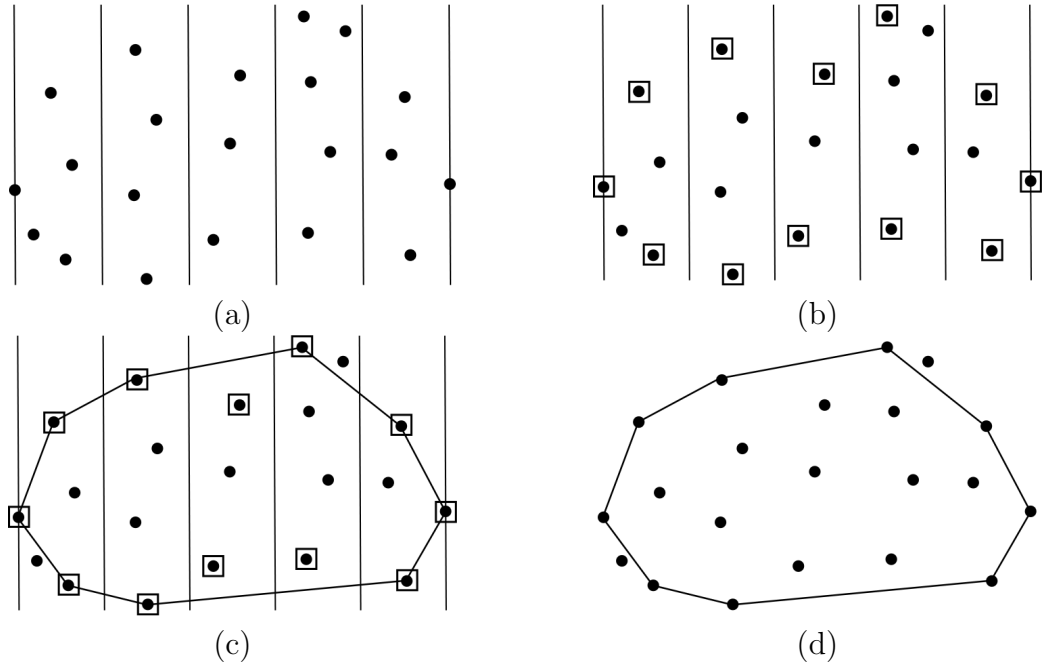


Figura 3.1: Ilustração do algoritmo de Bentley et al. [3] em 2D com $k = 5$. Primeiro (a), a região delimitada pelos valores extremos na dimensão x é dividida em k faixas igualmente espaçadas. Em seguida (b), são achados os pontos com valores extremos de y para cada faixa. Finalmente (c), é computado o fecho convexo dos pontos selecionados. O fecho obtido é uma aproximação interna do fecho exato (d).

O algoritmo descrito pode ser facilmente generalizado para outras dimensões. No caso 3D, interesse deste trabalho, o primeiro passo computa os valores extremos nas dimensões x e y. O retângulo resultante é então particionado em uma grade com $(k+2) \times (k+2)$ quadrados com linhas paralelas aos eixos x e y. Em cada quadrado são achados os pontos com valores extremos de z, resultando em um conjunto com no máximo $2(k+2)^2$ pontos. Finalmente, o fecho convexo dos pontos amostrados é computado.

Diferentemente do caso bidimensional em que foi possível tirar proveito da ordenação dos pontos, neste caso não é possível fazer o mesmo com a grade para obter um algoritmo mais rápido. Desta forma, usando um dos diversos algoritmos que computam o fecho convexo em tempo $O(n \log n)$ em três dimensões, a complexidade

de tempo do algoritmo aproximado é $O(n + k^2 \log k)$.

3.2 Algoritmo de Kavan, Kolingerova e Zara

Intuitivamente, um ponto p_i pertence ao fecho convexo de P se é um ponto extremo para uma dada direção $\vec{d}$. Em outras palavras, se q é um ponto de origem, então $(p_i - q) \cdot \vec{d}$ é máximo para todos os pontos da nuvem.

Kavan et al. [2] exploraram essa propriedade em um algoritmo que computa o fecho convexo aproximado de um conjunto de pontos. Embora o método seja descrito em duas dimensões, ele pode ser generalizado para outras dimensões. O fecho convexo de P forma um politopo convexo que pode ser definido pela interseção de um número finito de semiespaços. Uma ideia inicial é aproximar o fecho convexo por um k-DOP (*Discrete Oriented Polytope*). O algoritmo é dividido em dois passos:

1. Primeiro, um ponto de origem q dentro do fecho é selecionado em $O(n)$. Isso pode ser facilmente obtido escolhendo o centróide ou o centro da caixa envolvente da nuvem;

2. O plano é então dividido em k setores centrados em q e igualmente espaçados, cada um cobrindo um ângulo de $\frac{2\pi}{k}$. Para cada setor i, é definida a direção $\vec{d_i}$ que aponta na direção da bissetriz do ângulo setor. A direção $\vec{d_i}$ é a direção da normal do i-ésimo semi-espaço. O deslocamento dos semi-espaços é definido pelo maior valor de $(p_i - q) \cdot \vec{d_i}$ dentro todos os pontos de P. Claramente, este passo pode ser computado em $O(nk)$.

A complexidade final do algoritmo é $O(nk)$ para uma nuvem com n pontos e produz uma aproximação externa do fecho exato. É questionável se este método é superior aos algoritmos exatos que computam o fecho em $O(n \log n)$ em duas e três dimensões.

Um exemplo de um fecho convexo aproximado obtido pelo algoritmo descrito em 2D é ilustrado na Figura 3.2. Note que o fecho resultante é apenas uma aproximação, pois há pontos do fecho que não fazem parte do conjunto de pontos original. Note

também que o fecho resultante é uma aproximação externa do fecho exato, pois todos os pontos do fecho aproximado se encontram fora do fecho exato; veja a Figura 3.2(d).

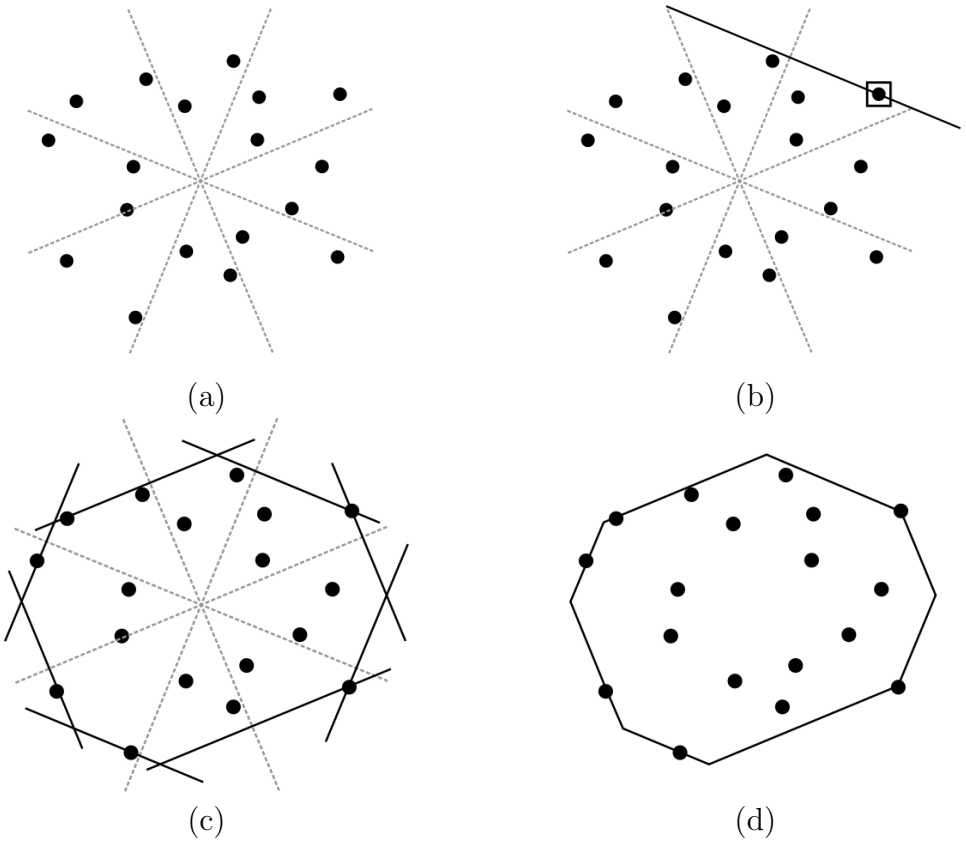


Figura 3.2: Ilustração de um algoritmo discutido por Kavan et al. [2] em 2D com $k = 8$. A direção dos k setores é computada (a). O ponto extremo e o semi-plano de para uma das direções é obtido (b). A interseção dos semi-planos definidos por cada direção e o respectivo ponto extremo (c) define o fecho convexo (d).

A partir do método anterior, os autores propuseram uma melhoria. O algoritmo é dividido em três passos:

1. Idêntico ao primeiro passo do método anterior;

2. Assim como no método anterior, o plano é dividido em k setores centrados em q e igualmente espaçados, cada um cobrindo um ângulo de $\frac{2\pi}{k}$ e a direção $\vec{d_i}$ é definida da mesma maneira. Porém, neste momento, os pontos são distribuídos em seus respectivos setores. Para cada setor i, computa-se o ponto p_i dentre os pontos do setor que maximize $(p_i - q) \cdot \vec{d_i}$. A ideia é que o ponto selecionado p_i seja uma boa estimativa do ponto extremo para direção d_i e

e, assim, provavelmente, um ponto do fecho convexo. Este passo pode ser computado em $O(n)$;

3. Finalmente, a estimativa de cada setor i é refinada comparando cada ponto p_i originalmente selecionado com os pontos selecionados para todos os outros setores. Se um outro ponto $p_j, j \neq i$ que seja uma melhor estimativa para a direção d_i for achado, então p_i é atualizado. Observe que este procedimento pode remover, mas nunca adicionar pontos ao fecho aproximado. Este passo leva tempo $O(k^2)$, pois cada ponto selecionado deve ser comparado com todos outros pontos selecionados.

Este algoritmo tem complexidade assintótica $O(n+k^2)$. A partir deste momento, quando for citado o método de Kavan et al., será considerado o segundo algoritmo, pois este consiste uma melhoria em relação ao primeiro. O erro da aproximação depende do raio do círculo que contém P. Claramente, quanto menor o raio, menor será o erro. Para uma explicação detalhada, veja [2].

Um exemplo de um fecho convexo aproximado obtido pelo algoritmo descrito em 2D é ilustrado na Figura 3.3. Note que o fecho resultante é apenas uma aproximação, pois há pontos do fecho que não fazem parte do conjunto de pontos original e há pontos do conjunto original que estão fora do fecho; veja a Figura 3.2(d).

Apesar do artigo não ser explícito a respeito, a extensão destes algoritmos para três dimensões é direta, embora alguns cuidados devam ser tomados para particionar a nuvem em setores de mesmo tamanho. Este trabalho sugere a utilização do algoritmo proposto por Leopardi [4], que particiona hiperesferas em qualquer número de setores com a mesma medida de Lebesgue, que corresponde ao perímetro em $\mathbb{R}^2$ e à área em $\mathbb{R}^3$. Repare que em $\mathbb{R}^2$, o algoritmo produz a mesma partição descrita anteriormente. A Figura 3.4 ilustra um exemplo da partição da esfera em $\mathbb{R}^3$ em 200 setores.

Como o algoritmo tem complexidade de tempo $O(n+k^2)$, ele será mais eficiente que o algoritmo exato, somente se $k \in o(\sqrt{n \log n})$. Entretanto, nuvens de pontos obtidas a partir de *scanners* 3D contêm somente pontos da superfície do modelo. Se

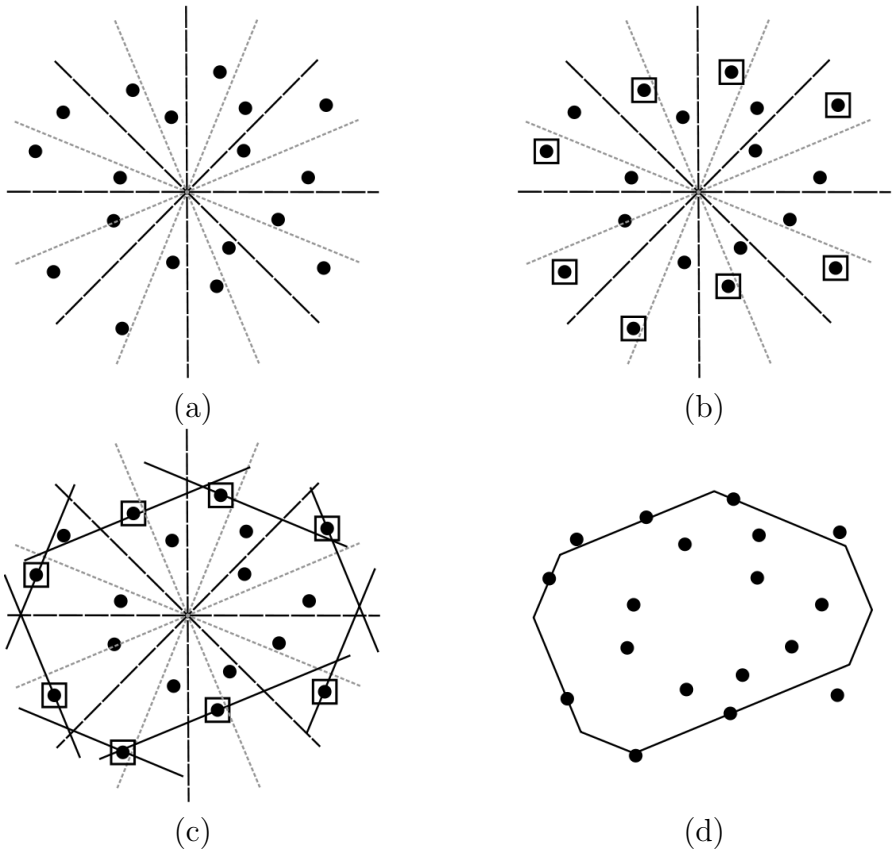


Figura 3.3: Ilustração do algoritmo de Kavan et al. [2] em 2D com $k = 8$. O plano é subdividido em k setores e os pontos são distribuídos entre eles (a). O ponto extremo de casa setor é computado (b). A interseção dos semi-planos definidos pela direção e o ponto extremo de cada setor (c) define o fecho convexo (d).

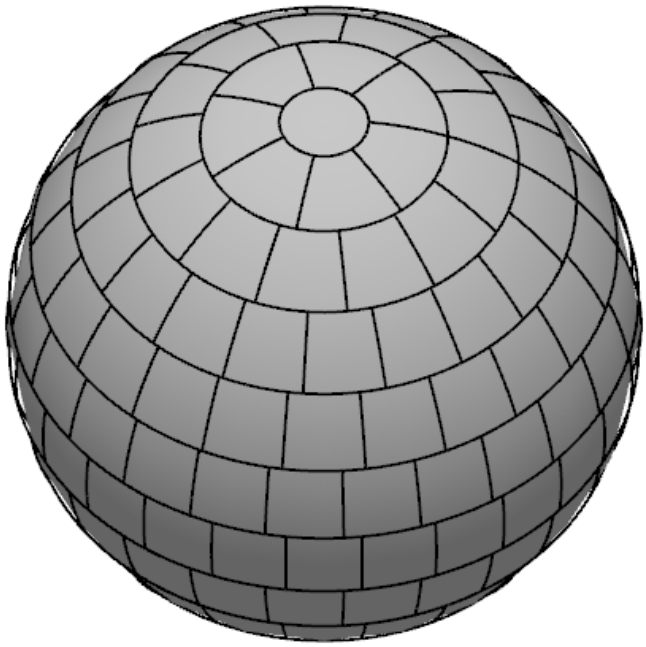


Figura 3.4: Partição da esfera em $\mathbb{R}^3$ em 200 setores obtida usando o algoritmo [4].

o modelo é convexo ou possui um número pequeno de concavidades, espera-se que, aproximadamente, metade dos pontos sejam visíveis. Com isso, a complexidade do algoritmo aproximado torna-se $O(n^2)$, menos eficiente que o exato.

3.3 Algoritmo Híbrido

Este trabalho propõe um novo algoritmo de fecho convexo aproximado que é uma combinação dos dois métodos apresentados anteriormente. Assim como o algoritmo de Bentley et al. [3], o algoritmo proposto utiliza o fecho convexo de uma amostragem dos pontos como uma aproximação do fecho. Porém, a amostragem de pontos é obtida utilizando a abordagem do método de Kavan et al. [2]. Observe que os passos de 1 e 2 produzem uma amostragem dos pontos originais. Finalmente, o fecho convexo dos pontos amostrados é computado e usado como uma aproximação do fecho do conjunto. A Figura 3.5 ilustra o método descrito em duas dimensões, porém o método pode ser generalizado para dimensões maiores. Note que o fecho resultante é apenas uma aproximação, pois há um ponto do conjunto que está fora do fecho. Note também que assim como o método de Bentley et al. [3], o método utiliza um algoritmo exato em um subconjunto dos pontos, então o fecho obtido é uma aproximação interna do fecho exato; veja a Figura 3.5(d).

Este algoritmo tem complexidade de tempo $O(n+k)$ em 2D. Os passos 1 e 2 podem ser computados em $O(n)$, como demonstrado na seção anterior. Considerando que pode haver um ponto por setor, ao fim do passo 2 haverá no máximo k pontos amostrados. Assim como em [3], no caso bidimensional, os pontos amostrados já estão ordenados, só que neste caso eles se encontram ordenados no sentido (anti-)horário. Desta forma, é possível utilizar o algoritmo de Graham [28] e computar o fecho convexo em $O(k)$, totalizando $O(n+k)$.

A extensão do algoritmo para outras dimensões é direta e como no algoritmo de Kavan et al. [2], sugere-se utilizar o algoritmo de particionamento de Leopardi [4]. Novamente, diferentemente do caso bidimensional, em que foi possível tirar proveito da ordenação dos pontos, para dimensões maiores não é possível fazer o mesmo.

Desta forma, usando um dos diversos algoritmos que computam o fecho convexo em tempo $O(n \log n)$ em três dimensões, a complexidade de tempo do algoritmo aproximado é $O(n + k \log k)$.

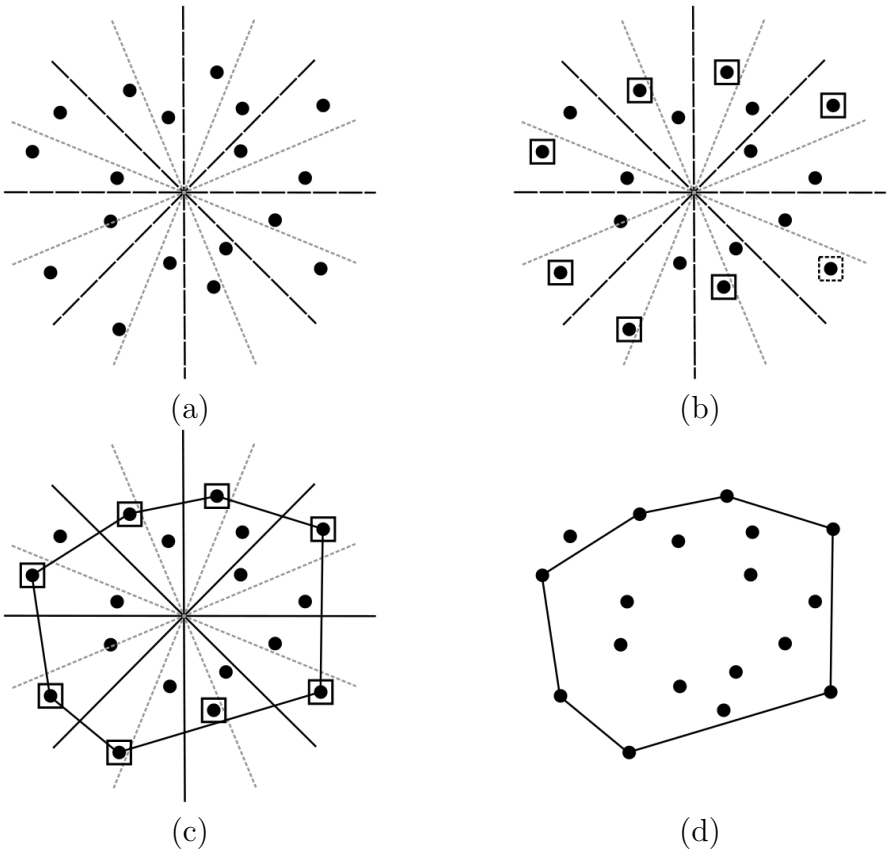


Figura 3.5: Ilustração do algoritmo propostos que combina ideias dos algoritmos de Bentley et al. [3] e Kavan et al. [2] com $k = 8$. O plano é subdividido em k setores e os pontos são distribuídos entre eles (a). O ponto extremo de casa setor é computado (b). Finalmente (c), é computado o fecho convexo dos pontos selecionados. O fecho obtido é uma aproximação interna do fecho exato (d).

Capítulo 4

Computação eficiente do operador HPR

Neste capítulo será descrita a principal contribuição desta dissertação. O algoritmo de Kavan et al. [2] é um ponto a partir do qual o operador HPR pode ser calculado de forma eficiente. Uma observação importante é que, embora o algoritmo aproximado descrito vise produzir um politopo, neste trabalho só há necessidade de selecionar pontos da nuvem que fazem parte do fecho. Não havendo necessidade de se computar a topologia do fecho, é possível usar estruturas de dados simples que possam ser percorridas simultaneamente, usando um modelo de computação paralela, tal como programação em GPU. Desta maneira, se a computação for dividida igualmente entre p processadores, os passos (1) e (2) (veja seção 3.2) podem ser computados em $O(n/p)$. Outra observação importante é que o passo (3) pode ser calculado de forma mais eficiente, examinando uma vizinhança limitada de cada setor em vez de todos os k setores. A idéia é que a melhor estimativa para cada setor pode ser obtida usando um esquema para a propagação de pontos extremos. Assim, o passo (3) pode ser reescrito como:

3) A estimativa de cada setor i é refinada comparando cada ponto originalmente selecionado p_i com os pontos p_j selecionados para todos setores *vizinhos* $j \in \mathcal{N}(i)$. A propagação pode ser feita de duas maneiras:

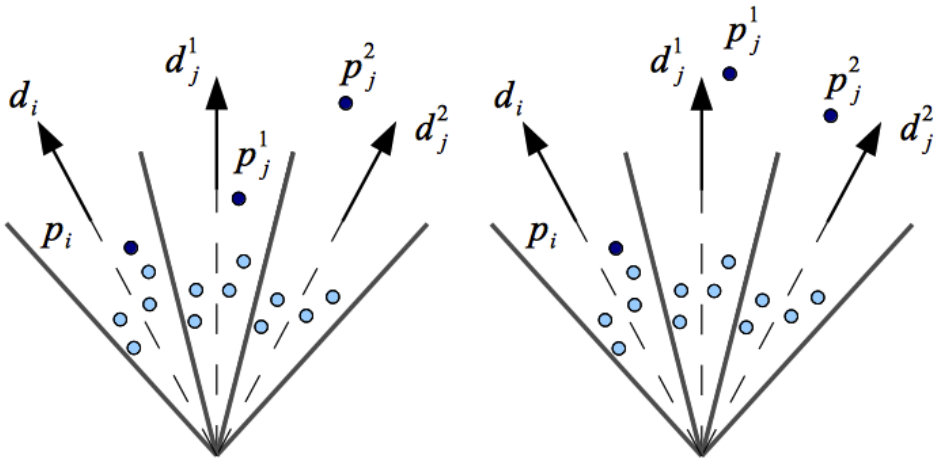


Figura 4.1: Em (a), o candidato p_j^2 é uma melhor estimativa para ambos os setores j^1 e i, enquanto que em (b), p_j^2 é uma estimativa melhor para o setor i, mas não para j^1; neste caso, p_i será substituído por p_j^1.

(a) Propagação adaptaiva: caso alguma estimativa inicial seja alterada, este passo é repetido até que nenhuma estimativa melhor seja descoberta para qualquer setor, ou;

(b) Propagação fixa: o passo é realizado uma única vez para uma vizinhança de tamanho $m \times m$.

A justificativa para esta alteração é que é mais provável que o ponto selecionado p_i do setor i com o direção d_i seja substituído pelo candidato p_j correspondente a direção d_j se o ângulo entre d_i e d_j é pequeno. Além disso, suponha que o candidato p_j^2 de um setor j^2 que não é um vizinho imediato do setor i seja selecionado para substituir p_i. Há um setor j^1, então, que é vizinho de i tal que (1) p_j^2 é também um substituto para p_j^1, ou (2) p_j^2 não é um substituto para p_j^1, porém, neste caso, p_j^1 é um substituto melhor para p_i (veja Figura 4.1).

A complexidade de tempo do passo (3), utilizando o processo adaptativo, depende do número de iterações k' necessárias para terminar a propagação. Se cada setor possui um número constante de vizinhos, então o custo total será $O(kk')$. No pior caso, $k' \approx k$, produzindo a mesma complexidade do algoritmo original. Observe, entretanto, que um valor grande de k' significa que os pontos candidatos são atribuídos a grandes intervalos angulares, fazendo com que o fecho convexo tenha

faces grandes. É razoável assumir que esta situação não é comum para nuvem de pontos densas obtidas a partir de scanners 3D, e assim, k' é significativamente inferior a k. No caso da propagação fixa, a complexidade é $O(km^2)$. Além disso, ambos esquemas de propagação podem ser facilmente implementados em arquiteturas paralelas, melhorando o desempenho do processo como um todo.

4.1 Definição dos Setores

O primeiro passo consiste em estabelecer um sistema de coordenadas apropriado que permite definir os setores onde os pontos da nuvem serão distribuídos. Para este propósito, uma esfera envolvente centrada em C_e e com raio r é computada. Neste trabalho, C_e é o centróide da nuvem e r é a maior distância entre C_e e um ponto da nuvem. Desta forma, um sistema de coordenadas é definido de forma que a posição do observador C é a origem e o eixo x passa por C_e (veja Figura 4.2).

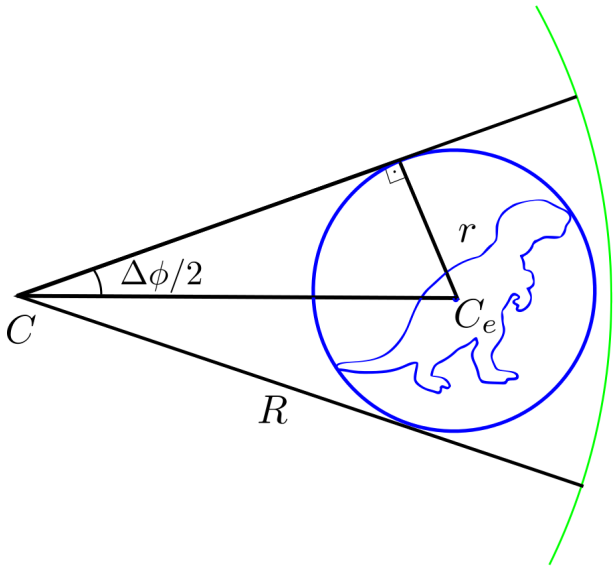


Figura 4.2: Sistema de coordenadas para definir os setores. O ponto de vista C está sobre a origem, com o eixo x passando pelo centro da esfera envolvente à nuvem de pontos C_e.

Os setores são definidos dividindo os ângulos horizontal e vertical do *frustum* de visualização de maneira a criar uma grade regular. A região do *frustum* que contém

a esfera envolvente será simétrica, cobrindo um ângulo $\Delta\phi$ dado por

$$\Delta\phi = 2\sin^{-1}\frac{r}{|C - C_e|}.$$

Usando coordenadas esféricas, o *frustum* irá corresponder a intervalos de φ and θ dados por

$$\begin{aligned} \varphi &\in [-\tfrac{\Delta\phi}{2}, +\tfrac{\Delta\phi}{2}], \\ \theta &\in [\tfrac{\pi}{2} - \tfrac{\Delta\phi}{2}, \tfrac{\pi}{2} + \tfrac{\Delta\phi}{2}]. \end{aligned}$$

A fim de produzir k setores, estes intervalos angulares são regularmente divididos $\sqrt{k}$ vezes em cada direção. Os setores assim formados terão uma forma de pirâmide e as direções $\vec{d_i}$ serão alinhadas com os bissetores de φ and θ. Note que os setores não serão idênticos, pois o tamanho dos setores será menor perto dos polos, isto é, para θ próximo de zero ou π. Assim, esta maneira particular de definir os setores, só é adequada quando o ponto de vista não se encontra muito próximo da nuvem, produzindo um *frustum* relativamente estreito. Apesar de haver métodos que não impõem esta restrição e ainda assim produzem setores mais uniformes – por exemplo, o método proposto por Leopardi [4], citado anteriormente – este esquema tem a vantagem de facilitar a visita aos oito vizinhos de um dado setor. Esta característica será fundamental para o passo de propagação de pontos extremos (veja Section4.3).

Este passo do algoritmo foi implementado por dois *kernels* de CUDA (*Compute Unified Device Architecture*) que processam todos os pontos da nuvem. O primeiro *kernel* computa o centróide C_e, enquanto que o segundo computa r. Ambos os *kernels* são implementados utilizando soma de prefixos em paralelo [29] que leva tempo $O(n/p + \log n)$, usando p processadores. Este passo é realizado uma única vez, pois não depende do ponto de vista.

4.2 Computação dos Pontos Extremos dos Setores

Uma vez computados C_e e r, e sendo k definido de alguma maneira, o intervalo angular $\Delta\phi$ pode ser computado, permitindo assim definir a geometria de todos os setores. Neste momento, um outro *kernel* realiza uma varredura simples de todos os pontos da nuvem com os seguintes objetivos:

1. aplicar uma transformação afim na nuvem, movendo o ponto de vista C para a origem do sistema de coordenadas e centro da esfera C_e para algum ponto no eixo x,

2. computar a inversão (*spherical flipping*) de cada ponto, armazenando estes valores em um *array* P de tamanho n, e

3. atribuir um identificador do setor para cada ponto, armazenando estes valores em um *array* $SETOR$ de tamanho n.

O identificador do setor de um ponto é um número inteiro entre 0 e $k-1$. Este número pode ser determinado computando as coordenadas esféricas do ponto e identificando o intervalo angular apropriado em φ and θ, onde o ponto se encontra. Por exemplo, se um ponto se encontra no i'-ésimo intervalo na direção φ e no j'-ésimo intervalo na direção θ, então o número identificador do setor é $i\sqrt{k}+j$. Repare que a computação realizada neste *kernel* é realizada independentemente para cada ponto e assim executa em tempo $O(n/p)$, para p processadores.

Para o próximo passo, é necessário saber a a direção $\vec{d_i}$, que aponta para o centro de cada setor. Para isso, um *kernel* computa as direções e armazena os valores em um *array* de tamanho k chamado *DIR*. Este *kernel* executa em tempo $O(k/p)$ e assim como o passo inicial, este passo é computado uma única vez. Tendo as direções computadas, um outro *kernel* computa a projeção de cada ponto invertido na direção do seu respectivo setor. Em suma, um *array* chamado *DIST* de tamanho

n é computado com uma varredura em paralelo de todos os pontos de forma que

$$DIST[i] = P[i] \cdot DIR[SETOR[i]].$$

Finalmente, um ponto extremo para cada setor deve ser achado, examinando somente os pontos atribuídos ao setor. Isto requer uma reordenação do *array* P que contém a nuvem invertida tal que os pontos de um mesmo setor se encontrem em posição contígua de memória. Isto é obtido por meio de uma operação de ordenação que utiliza os valores em $SETOR$ como chaves. Neste trabalho, foi utilizada a implementação otimizada para GPU descrita por Merril e Grimshaw [30] e disponível na biblioteca *Thrust* [31] do algoritmo de ordenação *radix sort*. O algoritmo *radix sort* tem complexidade $O(n/p)$, para p processadores. Após a ordenação de P, os pontos extremos de cada setor são computados com uma soma de prefixos segmentados [32] em tempo $O(n/p + \log n)$. O resultado desta etapa é armazenada em um *array* chamado MAX de tamanho k que contém os índices dos pontos extremos.

4.3 Propagação dos Pontos Extremos

4.3.1 Propagação Adaptativa

Neste passo, o ponto considerado inicialmente como um ponto extremo para um dado setor pode ser substituído por um ponto extremo de um dos oito setores que compartilham uma aresta ou um vértice na grade angular. Esta é uma frase crítica do algoritmo, pois ela deve ser repetida uma quantidade de vezes até que nenhum ponto extremo seja substituído.

Infelizmente, contar o número de pontos extremos substituídos é uma tarefa complicada devido a ocorrência de setores vazios, isto é, setores para os quais nenhum ponto extremo tenha sido computado em iterações anteriores. Setores vazios podem acontecer por duas razões diferentes: (1) o setor corresponde a uma região fora da projeção do objeto, ou (2) o setor está dentro da projeção do objeto, no entanto

não há nenhum ponto dentro deste setor. A Figura 4.3 ilustra estas situações. Claramente, o processo de propagação deve ignorar os setores do primeiro caso, mas não aqueles do segundo caso. Assim, o algoritmo faz uso de um *array* auxiliar de valores booleanos chamado *VAZIO*, de forma que *VAZIO*[i] é *verdadeiro* se o setor i é vazio, e provavelmente do tipo (1). Saber se o setor é do tipo (1) não é uma tarefa trivial. A fim de ter uma maior certeza de que não há setores vazios do tipo (2), observa-se que setores vazios deste tipo tornam-se mais prováveis quanto k aumenta. Desta forma, para valores de k/n acima de um limiar, em vez de usar um *array VAZIO* de tamanho k, utiliza-se uma amostragem mais grosseira. Assim, por exemplo, se o *array VAZIO* possui $k/4$ elementos, cada um dos seus elementos será *falso*, somente se nenhum ponto da nuvem cai em uma vizinhança de setores 2×2 da grade angular de k elementos.

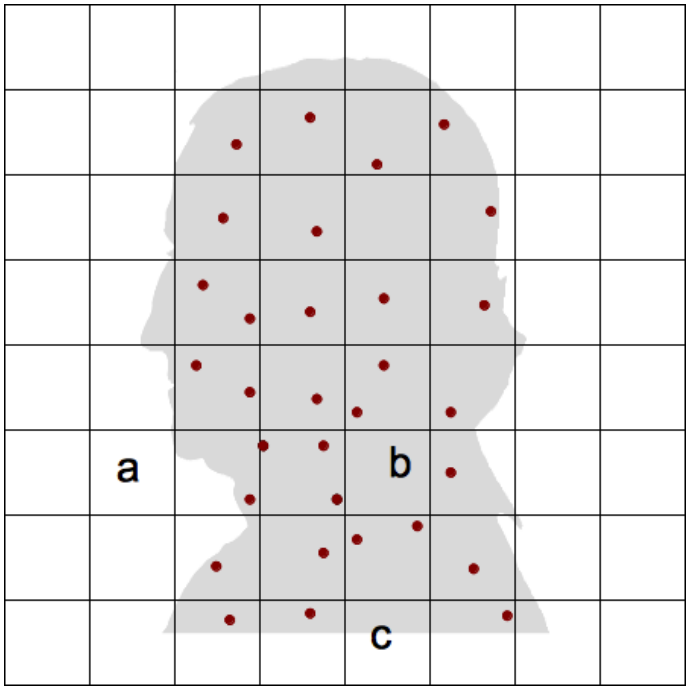


Figura 4.3: Setores que se encontram fora da projeção do objeto como (a) não possuem pontos extremos e não precisam ser visitados pelo passo de propagação. Setores como (b) que estão dentro da projeção do objeto, mas que não possuem amostras da nuvem devem ser visitados. Setores como (c) correspondem a borda da projeção do objeto.

4.3.2 Propagação Fixa

Diferentemente da propagação adaptativa em que o processo é repetido até que nenhum ponto extremo seja substituído, a propagação fixa visita uma região de tamanho fixo definida por um parâmetro. A região é definida por uma máscara de tamanho $m \times m$. Surpreendentemente, para valores de m pequenos, é possível obter resultados satisfatórios, como será mostrado no Capítulo 5.

4.3.3 Abordagem *Gather versus Scatter*

Uma consideração importante foi escolher entre uma estratégia de *gather* ou *scatter* para a implementação de ambos passos de propagação. Na estratégia de *gather*, a *thread* examinando o setor s visita os seus vizinhos à procura de um substituto para seu candidato atual, enquanto que em uma estratégia *scatter*, o candidato no setor s é considerado como um possível substituto para cada um de seus vizinhos. Na primeira abordagem, cada *thread* pode alterar um único setor, enquanto que na segunda, as modificações podem ocorrer de maneira concorrente. Neste trabalho, somente a propagação fixa foi implementada em GPU e foi utilizada a estratégia *gather*. Este passo foi implementado em um kernel de CUDA com complexidade $O((k\ m^2)/p)$.

No entanto, uma implementação em CPU, desenvolvida para fins de de comparação, usa a estratégia *scatter*, de modo que cada iteração sucessiva visita somente setores que tiveram seus candidatos alterados na iteração anterior. Isto reduz consideravelmente o número de setores visitados em cada passo, especialmente quando muitos passos de propagação são necessários devido a um valor de k elevado.

4.4 Reconstrução Parcial de Superfície

Katz et al. [1] usaram o operador HPR para realizar uma reconstrução parcial da superfície a partir de um ponto de vista. A reconstrução é obtida considerando não só os vértices do fecho convexo, mas também as faces que o compõem. Apesar do

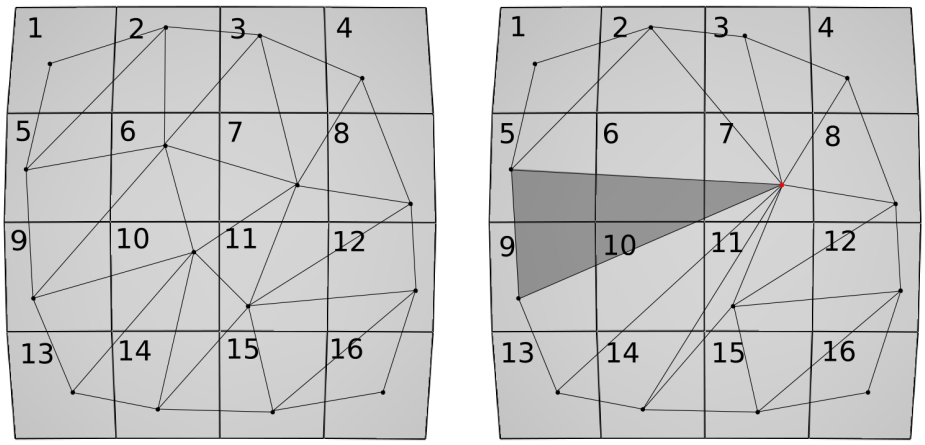
método computar o fecho convexo da nuvem invertida, a inversa da função *spherical flipping* é computada para obter uma triangulação dos pontos no espaço primal. A reconstrução pode ser obtida sem nenhum custo adicional, uma vez que a topologia do fecho (triangulação) já é computada para computar os pontos visíveis.

Neste trabalho, também é proposta uma reconstrução parcial da superfície. Apesar da topologia da malha não ser computada, ainda é possível realizar uma triangulação a partir da grade angular. Para cada vizinhança de 2×2 setores, são criados dois triângulos. Ao visitar o setor i, quatro pontos podem ser usados para formar os dois triângulos. São eles: os pontos cujos índices são $MAX[i]$, $MAX[i+1]$, $MAX[i+\sqrt{k}]$, and $MAX[i+\sqrt{k}+1]$.

A Figura 4.4 ilustra o processo de triangulação descrito. Na Figura 4.4(a) vê-se uma a triangulação dos pontos quando todos os setores possuem pontos distintos. Repare, no entanto, que a etapa de propagação pode ter atribuído o mesmo ponto extremo para diversos setores vizinhos. No caso da Figura 4.4(b), o ponto localizado no setor 7 (em vermelho) é também o ponto extremo dos setores 6 e 10. Note que o triângulo em destaque é formado quando o setor 5 é visitado e é formado juntamente com pontos dos setores 6 e 9. Triângulos inválidos são trivialmente eliminados exigindo que os três vértices de cada triângulo sejam distintos. Observe que nenhum triângulo é gerado quando o setor 6 é visitado, enquanto que somente um triângulo é gerado pelo setor 2.

Esta etapa tem complexidade $O(k)$. Cada setor produz, no máximo, dois triângulos, à exceção dos setores da última linha e da última coluna grade, que não produzem nenhum triângulo, pois não possuem vizinhos suficientes. Desta forma, são formados até $2(\sqrt{k}-1)^2$ triângulos.

Assim como em [1], alguns triângulos, cujo o tamanho das arestas seja superior a um certo limiar, podem ser eliminados.



(a) (b)

Figura 4.4: Em (a), os pontos extremos de cada setor encontram-se em seus respectivos setores, enquanto que em (b), os pontos extremos dos setores 6 e 10 encontram-se no setor 7.

Capítulo 5

Resultados

A fim de avaliar a exatidão e a utilidade das técnicas descritas neste trabalho, uma série de experimentos foram conduzidos. Os resultados obtidos substituindo o algoritmo de fecho convexo exato pelos algoritmos de Bentley et al. [3], Kavan et al. [2] e o algoritmo híbrido proposto não apresentaram ganhos significativos. Desta forma, este capítulo mostrará em detalhe os resultados obtidos utilizando o método proposto descrito no Capítulo 4.

A primeira bateria de experimentos computa a visibilidade em nuvens de pontos e visa demonstrar que mesmo com a utilização do método aproximado é possível obter uma taxa de acerto muito próxima à obtida pelo HPR com o fecho convexo exato. Além disso, são comparados os desempenhos obtidos com a implementação em CPU e em GPU do método proposto e o operador HPR com uma implementação eficiente do algoritmo [19]. A implementação em CPU utiliza o passo de propagação adaptativo, enquanto que a implementação em GPU utiliza o passo de propagação fixo.

A segunda bateria de experimentos computa a reconstrução parcial de superfície em nuvens de pontos e visa demonstrar que mesmo nos casos em que a taxa de pontos classificados como visíveis é consideravelmente inferior, as triangulações obtidas apresentam resultados visuais satisfatórios.

Todos experimentos foram conduzidos em um processador Intel i7 (2.4 GHz) e 8GB de memória e uma placa gráfica Nvidia GTX 470 com 1GB de memória.

Todos os protótipos foram escritos em C++ e OpenGL. O operador HPR exato foi implementado com a biblioteca Qhull [19]. Os algoritmos em GPU foram escritos usando *C for CUDA* (4.0) e a biblioteca Thrust [31].

A Tabela 5.1 exibe informações relevantes dos modelos usados nos experimentos. Apesar dos modelos serem malhas poligonais, somente os vértices são usados como entrada para os algoritmos testados.

Modelo	Número de Pontos	Exemplos
Bimba	74.764	Figura 5.2 e 5.5
Armadillo	172.974	Figura 5.3 e 5.6
Dragon	437.645	Figura 5.4
Happy Buddha	543.652	Figure
Buddha	719.560	Figura 5.7
Asian Dragon	3.609.455	Figura 5.8

Tabela 5.1: Modelos utilizados nos experimentos.

5.1 Determinação de Visibilidade

Nesta seção, serão apresentados os resultados obtidos para medir a capacidade do método em computar a visibilidade de nuvens de pontos. Claramente, a precisão do método proposto depende do tamanho da grade angular, definida pelo parâmetro k. À medida que k cresce, mais pontos visíveis são detectados a um custo maior de processamento. Uma questão crucial, então, é quão densa uma grade deve ser para produzir aproximadamente o mesmo número de pontos visíveis que o algoritmo exato. O gráfico da Figura 5.1 mostra a quantidade de pontos visíveis em função de k para o modelo *Armadillo*, que contém 172.974 pontos. Para a pose em particular utilizada nos experimentos, o número máximo de pontos visíveis foi de aproximadamente 57.000, obtido para um valor de k próximo de 1 milhão. Isto significa que aumentar o valor de k acima deste valor é desnecessário.

Finalmente, a fim de comparar os resultados obtidos com o método proposto e o algoritmo HPR original, testes foram conduzidos para nuvens de tamanhos diferentes. Os resultados são apresentados na Tabela 5.2. A fim de proporcionar uma

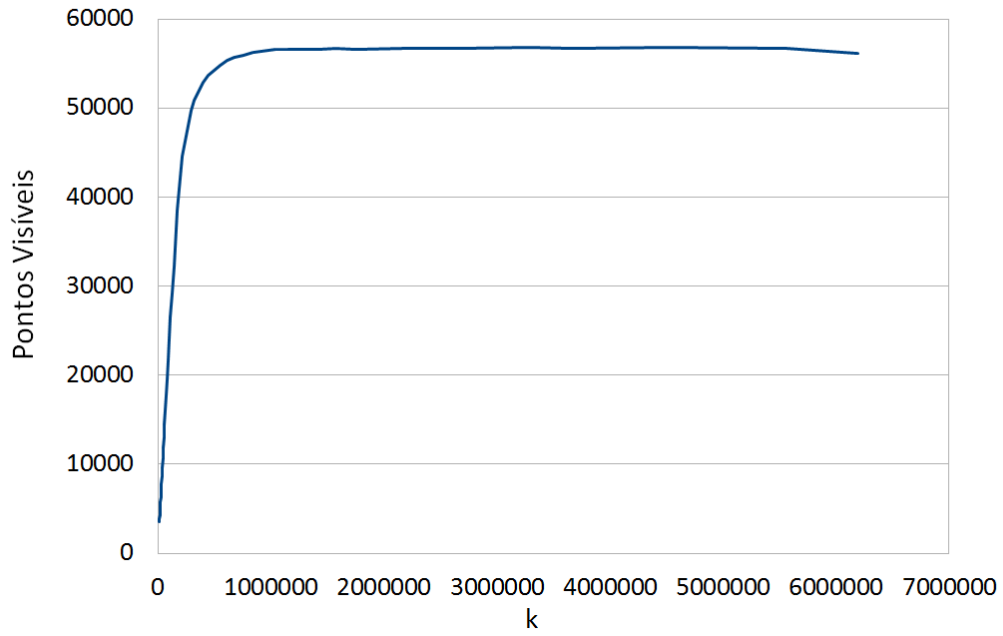


Figura 5.1: Número de pontos visíveis em função de k para o modelo *Armadillo*.

comparação justa, os valores utilizados para k nestes experimentos foram estabelecidos de modo a produzir aproximadamente o mesmo número de pontos visíveis que a implementação exata, assim o número de pontos visíveis mostrados na tabela são apenas aproximados.

Claramente, para nuvens mais densas, o método proposto torna-se menos competitivo, uma vez que tem de lidar com um grande número de setores vazios. Em particular, a implementação em CPU apresentou um desempenho inferior ao método exato para o modelo *Buddha*. Neste caso, menos de 5% dos setores estão ocupados com pontos extremos distintos.

Vale ressaltar que a implementação em CPU que utiliza a propagação adaptativa, realiza diversos passos de propagação até que todos setores tenham a melhor estimativa para o ponto extremo. Já a implementação em GPU utiliza a propagação fixa, que realiza um único passo de propagação, porém para um vizinhança de tamanho $m \times m$. A última coluna da Tabela 5.2 além de apresentar o desempenho obtido com a implementação em GPU, mostra os valores de m utilizados. Os resultados mostram que mesmo para valores pequenos de m é possível obter resultados com a

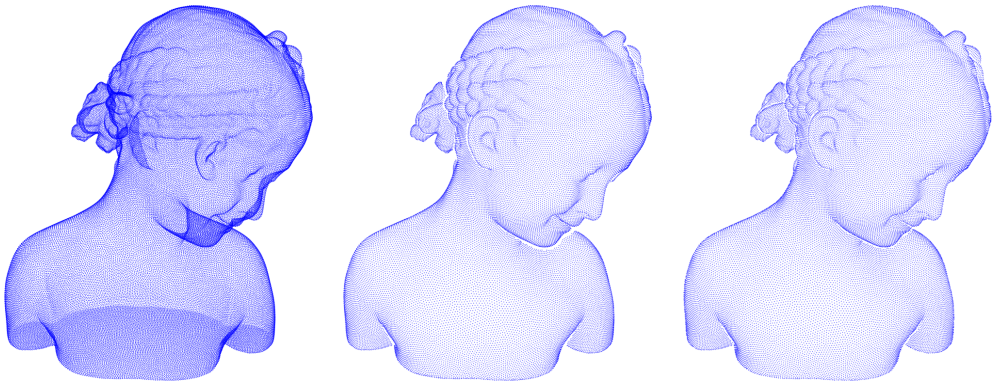
mesma taxa de erro. O modelo *Dragon* apresenta uma distribuição não uniforme dos pontos, com isso é necessário percorrer uma região de vizinhança maior para fechar buracos em em algumas regiões da nuvem, o que explica o valor de m elevado em relação ao usado com os demais modelos.

Deve ser enfatizado que os resultados da Tabela 5.2 foram obtidos com valores de k que são desnecessariamente grandes para algumas aplicações. No caso da reconstrução parcial de superfície de nuvens de pontos, o valor de k será escolhido de forma a obter um bom resultado visual, dependendo da resolução da tela.

Modelo	Total de Pontos	Pontos Visíveis	QPS Exato	QPS - Approximado	
				CPU	GPU
Bimba	74.764	$\sim$ 27.000	2	5	50 (m = 3)
Armadillo	172.974	$\sim$ 63.000	1	2	25 (m = 3)
Dragon	437.645	$\sim$ 150.000	0.42	0.28	2.3 (m = 10)
Buddha	719.560	$\sim$ 250.000	0.30	0.18	1.7 (m = 6)

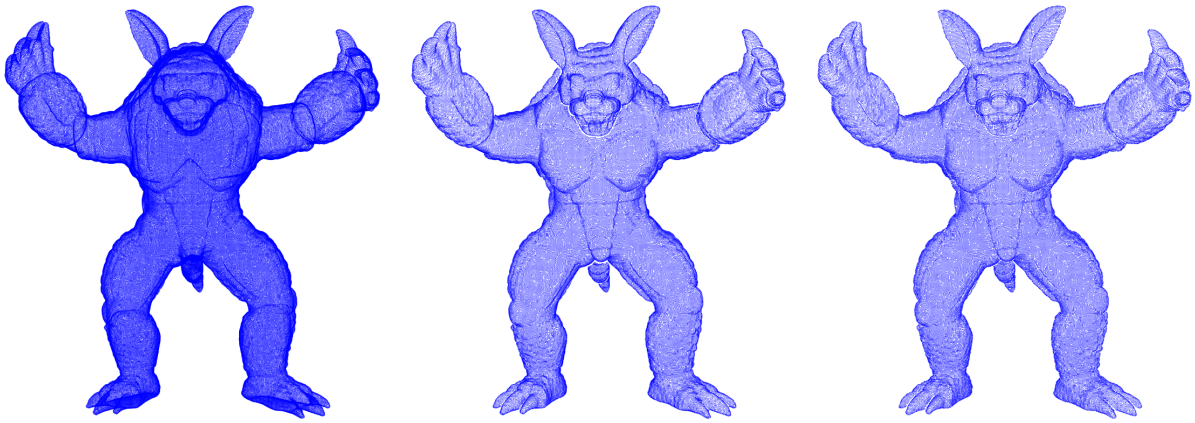
Tabela 5.2: Número de pontos classificados como visíveis e tempo de renderização da nuvem de pontos obtida em quadros por segundo (QPS).

As Figuras 5.2, 5.3 e 5.4 apresentam os resultados obtidos para os modelos *Bimba*, *Armadillo* e *Dragon*, respectivamente. As figuras do modelo *Buddha* não foram mostrados, pois a quantidade de pontos do modelo requer uma resolução de imagem grande para distinguir os detalhes visuais.



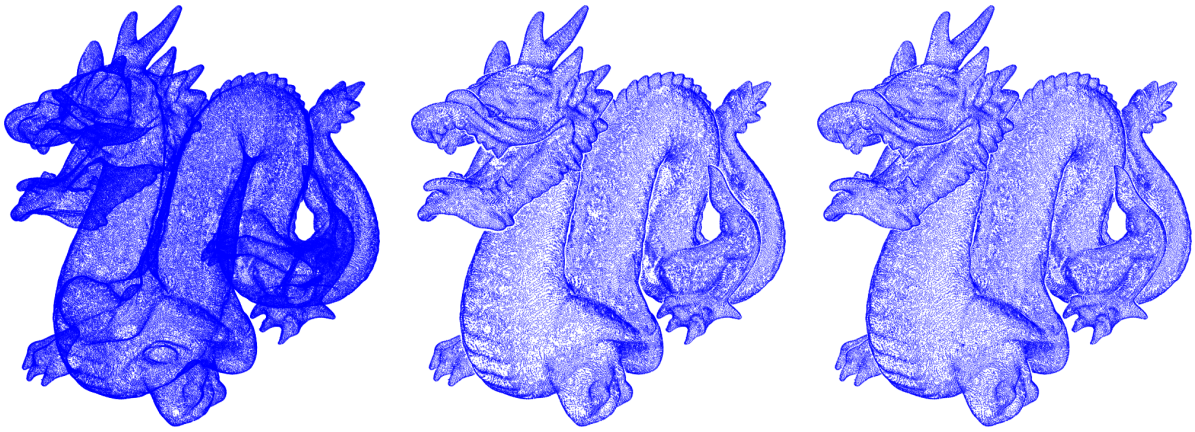
(a) (b) (c)

Figura 5.2: Visualização da nuvem de pontos do modelo *Bimba* (a) original, (b) após HPR e (c) após HPR modificado.



(a) (b) (c)

Figura 5.3: Visualização da nuvem de pontos do modelo *Armadillo* (a) original, (b) após HPR e (c) após HPR modificado.



(a) (b) (c)

Figura 5.4: Visualização da nuvem de pontos do modelo *Dragon* (a) original, (b) após HPR e (c) após HPR modificado.

5.2 Reconstrução Parcial de Superfície

Nesta seção, serão apresentados os resultados obtidos para medir a capacidade do método em computar a reconstrução parcial de superfície a partir de nuvens de pontos. É preciso ser enfatizado que os resultados da Tabela 5.2 foram obtidos para valores de k que são desnecessariamente grandes para a aplicação de renderização da reconstrução parcial da superfície em tempo real. Como já mencionado, o valor de k pode ser escolhido de forma a apresentar um bom resultado para uma dada resolução de tela. Todos os resultados apresentados nesta seção foram obtidos para uma imagem de 900×900 pixels.

Uma vez que o algoritmo proposto depende do número de setores, é útil reali-

zar uma inspeção visual dos resultados obtidos para diferentes valores de k. As Figuras 5.5(a) e 5.5(b) mostram a renderização dos pontos e das faces do modelo *Bimba*. As Figuras 5.5(c)/(d) e 5.5(e)/(f) mostram renderizações semelhantes obtidas utilizando o método HPR eficiente nos pontos da nuvem para valores de $k = 25.000$ e $k = 850.000$, respectivamente. Como referência para comparação, as Figuras 5.5(g)/(h) ilustram os resultados obtidos com o operador HPR original e , isto é, utilizando um algoritmo de fecho convexo exato.

Como esperado, uma grade angular mais fina revela mais detalhes sobre a geometria do objeto. Não existem diferenças significativas visuais entre as Figuras 5.5(e)/(f) e 5.5(g)/(h), o que indica que os resultados com uma grade suficientemente densa são indistinguíveis daqueles obtidos com uma implementação exata do fecho convexo. Realmente, para o modelo e a pose mostrados na Figura 5.5, as renderizações para $k = 850.000$ classificaram 26.598 pontos como visíveis, enquanto que o operador HPR original obteve 27.351 pontos visíveis para o mesmo valor de R.

A Tabela 5.3 mostra a taxa de renderização em quadros por segundo para diversos modelos em função dos parâmetros k e m. As Figuras 5.6, 5.7 e 5.8 mostram a renderização para os modelos *Bimba*, *Buddha* e *Asian Dragon*, respectivamente. É possível notar artefatos, principalmente próximos das bordas e das silhuetas dos modelos.

Modelo	Número de Pontos	$\sqrt{k}$	m	QPS
Bimba	74.764	923	3	50
Armadillo	172.974	1.269	3	25
Buddha	719.560	678	3	25
Asian Dragon	3.609.455	906	1	11

Tabela 5.3: Reconstrução

5.3 Limitações

Um dos principais pontos fortes do método proposto é a possibilidade de calibrar a quantidade de detalhe visual escolhendo um valor apropriado de k. Essa dependência

com a quantidade de setores da grade angular é também a razão de uma das suas principais limitações. À medida que k aumenta, os setores se tornam cada vez menos ocupados, isto é, são produzidos setores vazios de ambos os tipos (1) e (2) (veja Seção 4.3). Uma vez que os setores vazios são representados na estrutura de dados, isso leva a um desperdício de memória. Um gráfico demonstrando esse comportamento é mostrado na Figura 5.9.

Nos experimentos realizados neste trabalho, esta limitação não foi um problema para modelos com até um milhão de pontos. Para estes modelos foi possível computar a visibilidade com uma taxa de acerto muito próxima do operador HPR exato, como pode ser visto na Tabela 5.2. Entretanto, para modelos maiores, como o *Asian Dragon* que contém mais de $3,6$ milhões de pontos, mostrado na Figura 5.8, o método proposto detectou aproximadamente metade dos pontos visíveis. Por outro lado, o operador HPR exato leva mais de 16 segundos para computar a visibilidade, enquanto que o método proposto faz a metade do trabalho a 2 quadros por segundo. Além disso, a Figura 5.8(b), mostra que com apenas 98.097 pontos foi possível obter uma reconstrução parcial da superfície bastante satisfatória a 11 quadros por segundo.

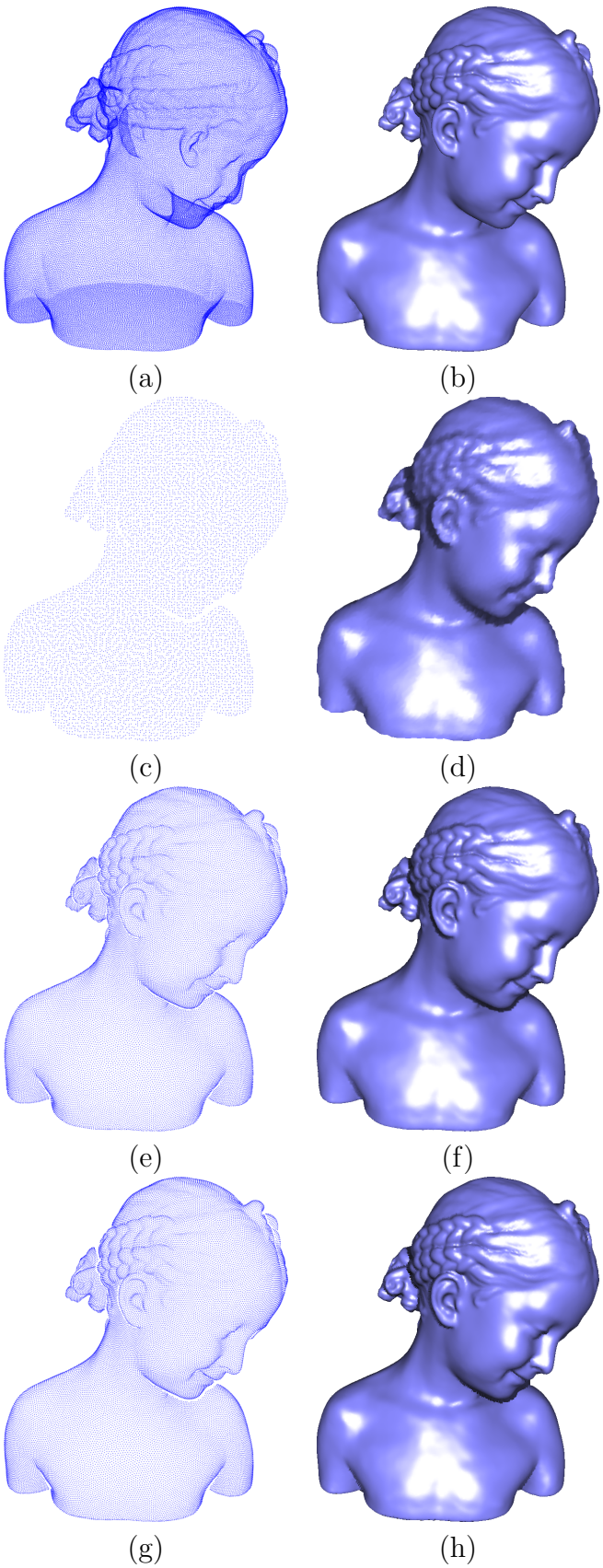


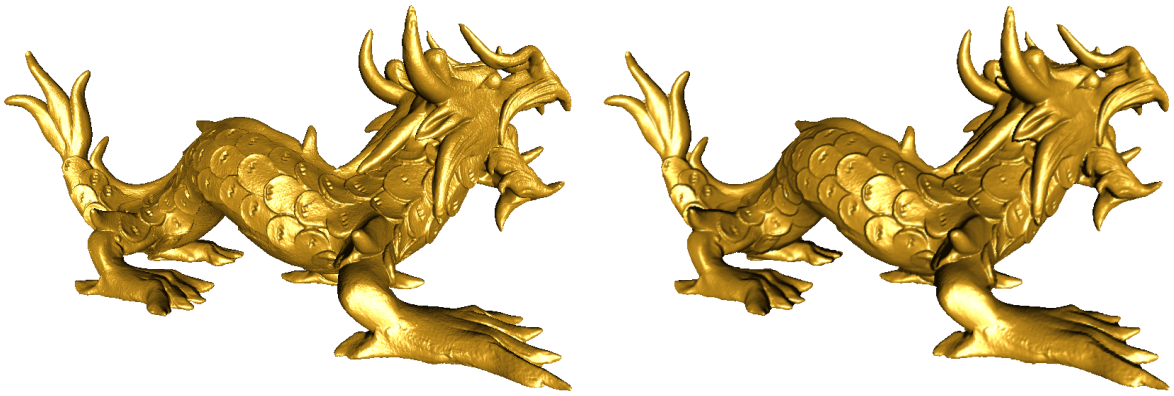
Figura 5.5: Resultados visuais. Renderização do modelo *Bimba* original (a)/(b), após HPR aproximado com $k = 25.000$ (c)/(d) e com $k = 850.000$ (e)/(f), e após HPR exato (g)/(h).

(a) (b)

Figura 5.6: Renderização do modelo *Armadillo* original (a) e da reconstrução de superfície parcial do HPR aproximado com $\sqrt{k} = 1.269$ a 50 quadros por segundo (b).

(a) (b)

Figura 5.7: Renderização do modelo *Buddha* original (a) e após reconstrução de superfície parcial do HPR aproximado com $\sqrt{k} = 678$ a 25 quadros por segundo (b).



(a) (b)

Figura 5.8: Renderização do modelo *Asian Dragon* original (a) e após reconstrução de superfície parcial do HPR aproximado com $\sqrt{k} = 906$ a 11 quadros por segundo (b).

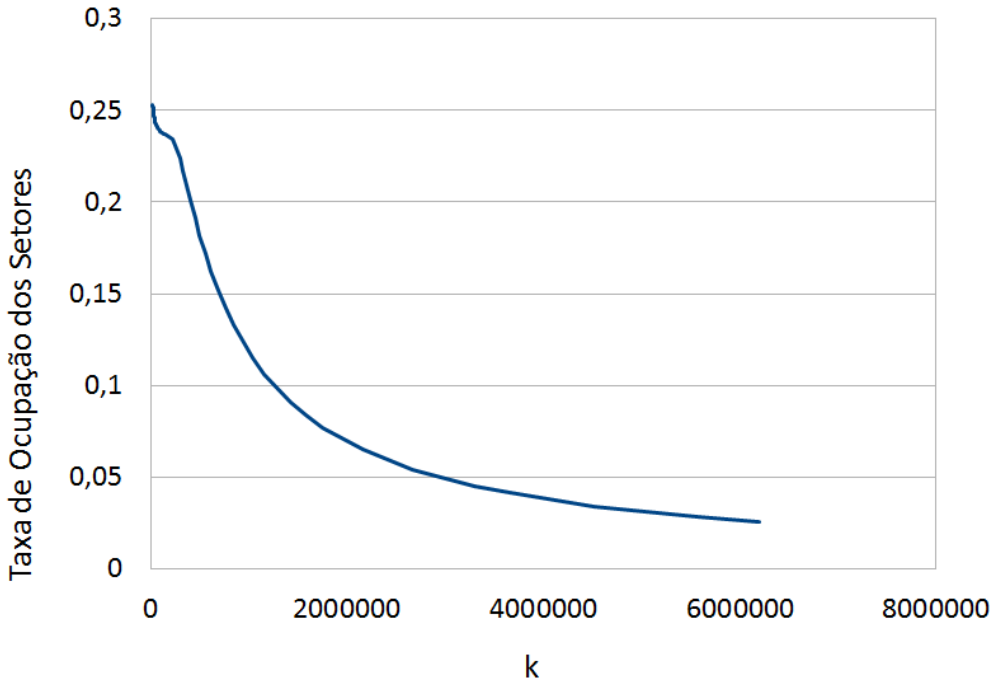


Figura 5.9: Taxa de ocupação dos setores em função de k para o modelo *Armadillo*.

Capítulo 6

Conclusões

Representações de superfícies baseadas em pontos se tornaram bastante populares nos últimos anos, graças a sua simplicidade, que elimina o custo de armazenar e processar a informação topológica de uma malha poligonal, e também graças à evolução dos sistemas que realizam a aquisição da geometria de objetos 3D, permitindo a obtenção de amostras mais precisas.

Neste trabalho foi apresentado o problema de extrair a visibilidade de uma nuvem de pontos a partir de um ponto de vista. Para este problema foi apresentado o inovador operador HPR que computa a visibilidade, independentemente da renderização e sem realizar a reconstrução da superfície. A visibilidade é extraída a partir do fecho convexo dos pontos em um espaço dual. O fecho convexo de um conjunto de n pontos em 3D tem complexidade assintótica $O(n \log n)$ o que impossibilita obter taxas interativas para valores de n grandes.

Neste trabalho, foram apresentada uma maneira tornar o operador HPR mais eficiente. O método proposto no Capítulo 4 permitiu computar o operador HPR de forma aproximada a taxas interativas. A partir de um valor apropriado do parâmetro k é possível ter um compromisso entre tempo de execução e qualidade do resultado. Além disso, graças à sua simplicidade, o método proposto pode ser facilmente implementado em GPU, permitindo um ganho adicional de desempenho. A Seção 4.4 mostrou como é possível obter uma reconstrução parcial da superfície a partir da grade angular. Além de computar a visibilidade a partir de um ponto de vista,

foram mostrados duas aplicações: visualização da nuvem de pontos (sem a reconstrução) e reconstrução parcial de superfície em tempo real. Como a reconstrução parcial é feita em tempo real, este método pode ser comparado aos métodos de renderização de nuvem de pontos, tais como os métodos baseados em *splatting* e os métodos baseados em traçados de raios, porém com a diferença de que no caso método apresentado neste trabalho, uma malha poligonal é extraída.

6.1 Trabalhos Futuros

Alguns trabalhos futuros incluem a implementação do processo de propagação adaptativa em GPU e extensões ou melhorias do método apresentado. Algumas extensões já estão previstas; em primeiro lugar, a partir do *array EMPTY* deve ser possível implementar um gerenciamento de memória mais eficiente, onde apenas setores não vazios precisam ser armazenados e processados. Em segundo lugar, busca-se uma maneira para se estimar os parâmetros k e R que produzam a melhor saída possível. Além disso, é necessário realizar um tratamento de *anti-aliasing*. É possível notar artefatos de *flickering* durante a manipulação do modelo, especialmente nas silhuetas e nas bordas. Esse *aliasing* é causado pela natureza regular da grade angular.

Apesar de apresentar resultados satisfatórios para nuvens de pontos de tamanhos variados, as nuvens utilizadas são na verdade malhas poligonais em que somente os vértices são considerados. Desta forma, uma outra sugestão de trabalho futuro seria a utilização de dados obtidos de *scanners* 3D ou de câmeras RGBZ.

Além da visualização de nuvens de pontos, uma aplicação imediata do método proposto seria a computação de sombras. Uma outra aplicação seria uma reconstrução global do modelo que pode ser obtida combinando a reconstrução parcial a partir de um conjunto de pontos de vista. Utilizando uma versão robusta a ruídos do operador HRP, Mehra et al. [17] apresentaram uma abordagem baseada em grafos para colar reconstruções parciais e obter uma reconstrução global consistente. Um trabalho futuro consistiria em investigar abordagens mais simples para este problema. Como a superfície da reconstrução parcial é modelada por malhas poligonais,

uma ideia inicial seria usar o técnica de *zippering* [7].

Referências Bibliográficas

[1] KATZ, S., TAL, A., BASRI, R. "Direct visibility of point sets". In: *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, p. 24, New York, NY, USA, 2007. ACM. doi: http://doi.acm.org/10.1145/1275808.1276407.

[2] KAVAN, L., KOLINGEROVA, I., ZARA, J. "Fast approximation of convex hull". In: *Proceedings of the 2nd IASTED international conference on Advances in computer science and technology*, pp. 101–104, Anaheim, CA, USA, 2006. ACTA Press. ISBN: 0-88986-545-0. Disponível em: <`http://portal.acm.org/citation.cfm?id=1166444.1166462`>.

[3] BENTLEY, J. L., PREPARATA, F. P., FAUST, M. G. "Approximation algorithms for convex hulls", *Commun. ACM*, v. 25, n. 1, pp. 64–68, 1982. ISSN: 0001-0782. doi: http://doi.acm.org/10.1145/358315.358392.

[4] LEOPARDI, P. "A partition of the unit sphere into regions of equal area and small diameter". In: *Electronic Transactions on Numerical Analysis*, v. 25, pp. 309–327, 2006.

[5] HOPPE, H., DEROSE, T., DUCHAMP, T., et al. "Surface reconstruction from unorganized points". In: *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pp. 71–78, New York, NY, USA, 1992. ACM. ISBN: 0-89791-479-1. doi: http://doi.acm.org/10.1145/133994.134011.

[6] EDELSBRUNNER, H., MÜCKE, E. P. "Three-dimensional alpha shapes", *ACM Trans. Graph.*, v. 13, n. 1, pp. 43–72, 1994. ISSN: 0730-0301. doi: http://doi.acm.org/10.1145/174462.156635.

[7] TURK, G., LEVOY, M. "Zippered polygon meshes from range images". In: *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pp. 311–318, New York, NY, USA, 1994. ACM. ISBN: 0-89791-667-0. doi: http://doi.acm.org/10.1145/192161.192241.

[8] CURLESS, B., LEVOY, M. "A volumetric method for building complex models from range images". In: *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pp. 303–312, New York, NY, USA, 1996. ACM. ISBN: 0-89791-746-4. doi: http://doi.acm.org/10.1145/237170.237269.

[9] AMENTA, N., BERN, M., KAMVYSSELIS, M. "A new Voronoi-based surface reconstruction algorithm". In: *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pp. 415–421, New York, NY, USA, 1998. ACM. ISBN: 0-89791-999-8. doi: http://doi.acm.org/10.1145/280814.280947.

[10] BERNARDINI, F., MITTLEMAN, J., RUSHMEIER, H., et al. "The Ball-Pivoting Algorithm for Surface Reconstruction", *IEEE Transactions on Visualization and Computer Graphics*, v. 5, n. 4, pp. 349–359, 1999. ISSN: 1077-2626. doi: http://dx.doi.org/10.1109/2945.817351.

[11] AMENTA, N., CHOI, S., DEY, T. K., et al. "A simple algorithm for homeomorphic surface reconstruction". In: *SCG '00: Proceedings of the sixteenth annual symposium on Computational geometry*, pp. 213–222, New York, NY, USA, 2000. ACM. ISBN: 1-58113-224-7. doi: http://doi.acm.org/10.1145/336154.336207.

[12] AMENTA, N., CHOI, S., KOLLURI, R. K. "The power crust". In: *SMA '01: Proceedings of the sixth ACM symposium on Solid modeling and applications*, pp. 249–266, New York, NY, USA, 2001. ACM. ISBN: 1-58113-366-9. doi: http://doi.acm.org/10.1145/376957.376986.

[13] DEY, T. K., GOSWAMI, S. "Tight cocone: a water-tight surface reconstructor". In: *SM '03: Proceedings of the eighth ACM symposium on Solid modeling and applications*, pp. 127–134, New York, NY, USA, 2003. ACM. ISBN: 1-58113-706-0. doi: http://doi.acm.org/10.1145/781606.781627.

[14] DEY, T. K., GOSWAMI, S. "Provable surface reconstruction from noisy samples". In: *SCG '04: Proceedings of the twentieth annual symposium on Computational geometry*, pp. 330–339, New York, NY, USA, 2004. ACM. ISBN: 1-58113-885-7. doi: http://doi.acm.org/10.1145/997817.997867.

[15] KAZHDAN, M. "Reconstruction of solid models from oriented point sets". In: *SGP '05: Proceedings of the third Eurographics symposium on Geometry processing*, p. 73, Aire-la-Ville, Switzerland, Switzerland, 2005. Eurographics Association. ISBN: 3-905673-24-X.

[16] KAZHDAN, M., BOLITHO, M., HOPPE, H. "Poisson surface reconstruction". In: *SGP '06: Proceedings of the fourth Eurographics symposium on Geometry processing*, pp. 61–70, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association. ISBN: 30905673-36-3.

[17] MEHRA, R., TRIPATHI, P., SHEFFER, A., et al. "Visibility of noisy point cloud data", *Computers and Graphics*, v. In Press, Accepted Manuscript, pp. –, 2010. Disponível em: <http://www.sciencedirect.com/science/article/B6TYG-4YMB660-1/2/eb083cf7f95ad4248c925f87e7348706>.

[18] GAO, M., CAO, T.-T., TAN, T.-S., et al. "gHull: a three-dimensional convex hull algorithm for graphics hardware". In: *Symposium on Interactive 3D Graphics and Games*, I3D '11, pp. 204–204, New York, NY, USA, 2011. ACM. ISBN: 978-1-4503-0565-5. doi: http://doi.acm.org/10.1145/1944745.1944784. Disponível em: <http://doi.acm.org/10.1145/1944745.1944784>.

[19] BARBER, C. B., DOBKIN, D. P., HUHDANPAA, H. "The quickhull algorithm for convex hulls", *ACM Trans. Math. Softw.*, v. 22, n. 4, pp. 469–483, 1996. ISSN: 0098-3500. doi: http://doi.acm.org/10.1145/235815.235821.

[20] ARVO, J., KIRK, D. "A survey of ray tracing acceleration techniques". pp. 201–262, London, UK, UK, Academic Press Ltd., 1989. ISBN: 0-12-286160-4. Disponível em: <http://dl.acm.org/citation.cfm?id=94788.94794>.

[21] CATMULL, E. E. *A subdivision algorithm for computer display of curved surfaces.* Tese de Doutorado, 1974. AAI7504786.

[22] SCHAUFLER, G., JENSEN, H. W. "Ray Tracing Point Sampled Geometry". In: *Proceedings of the Eurographics Workshop on Rendering Techniques 2000*, pp. 319–328, London, UK, 2000. Springer-Verlag. ISBN: 3-211-83535-0. Disponível em: <http://portal.acm.org/citation.cfm?id=647652.732143>.

[23] WAND, M., STRASSER, W. "Multi-Resolution Point-Sample Raytracing". In: *Graphics Interface 2003 Conference Proceedings*, 2003.

[24] WALD, I., SEIDEL, H.-P. "Interactive Ray Tracing of Point Based Models". In: *Proceedings of 2005 Symposium on Point Based Graphics*, 2005.

[25] LORENSEN, W. E., CLINE, H. E. "Marching cubes: A high resolution 3D surface construction algorithm", *SIGGRAPH Comput. Graph.*, v. 21, pp. 163–169, August 1987. ISSN: 0097-8930. doi: http://doi.acm.org/10.1145/37402.37422. Disponível em: <http://doi.acm.org/10.1145/37402.37422>.

[26] TAVARES, D., COMBA, J. "Efficient Approximate Visibility of Point Sets on the GPU". In: *Graphics, Patterns and Images (SIBGRAPI), 2010 23rd SIBGRAPI Conference on*, pp. 239 –246, 30 2010-sept. 3 2010. doi: 10.1109/SIBGRAPI.2010.40.

[27] ANDREW, A. M. "Another Efficient Algorithm for Convex Hulls in Two Dimensions", *Information Processing Letters*, v. 9, n. 5, pp. 216–219, 1979.

[28] GRAHAM, R. "An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set", *Information Processing Letters*, v. 1, pp. 132–133, 1972.

[29] SENGUPTA, S., HARRIS, M., ZHANG, Y., et al. "Scan primitives for GPU computing". In: *Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware*, GH '07, pp. 97–106, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association. ISBN: 978-1-59593-625-7. Disponível em: <http://dl.acm.org/citation.cfm?id=1280094.1280110>.

[30] MERRILL, D., GRIMSHAW, A. "High Performance and Scalable Radix Sorting: A case study of implementing dynamic parallelism for GPU computing", *Parallel Processing Letters*, v. 21, n. 02, pp. 245–272, 2011. ISSN: 0129-6264. doi: 10.1142/S0129626411000187. Disponível em: <http://www.worldscinet.com/ppl/21/2102/S0129626411000187.html>.

[31] HOBEROCK, J., BELL, N. "Thrust: A Parallel Template Library". 2010. Disponível em: <http://www.meganewtons.com/>. Version 1.4.0.

[32] SENGUPTA, S., HARRIS, M., GARLAND, M., et al. "Efficient Parallel Scan Algorithms for many-core GPUs". In: *Scientific Computing with Multicore and Accelerators*, Taylor & Francis, cap. 19, pp. 413–442, 2011.