

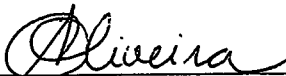
VISUALIZAÇÃO INTERATIVA DE MODELOS BASEADOS EM PONTOS

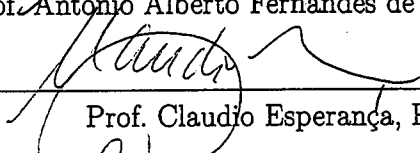
Ricardo Guerra Marroquim

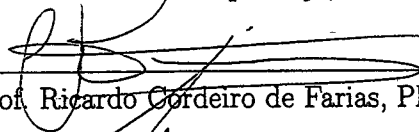
TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

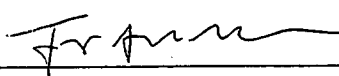
  
Prof. Paulo Roma Cavalcanti, D.Sc.

  
Prof. Antonio Alberto Fernandes de Oliveira, D.Sc.

  
Prof. Claudio Esperança, Ph.D.

  
Prof. Ricardo Cordeiro de Farias, Ph.D.

  
Prof. Luiz Carlos Pacheco Rodrigues Velho, Ph.D.

  
Prof. João Luiz Dohl Comba, Ph.D.

RIO DE JANEIRO - RJ, BRASIL

SETEMBRO DE 2008

MARROQUIM, RICARDO GUERRA

Visualização Interativa de Modelos Baseados em Pontos [Rio de Janeiro] 2008

XVII, 96 p. 29,7cm (COPPE/UFRJ, D.Sc., Engenharia de Sistemas e Computação, 2008)

Tese – Universidade Federal do Rio de Janeiro, COPPE

1. Representação por pontos
2. Visualização interativa
3. Renderização baseada em imagem
4. Renderização baseada em pontos

I. COPPE/UFRJ      II. Título (série)

*Aos meus pais que sempre me  
deram todo apoio imaginável,  
amo vocês.*

*À minha avó Gicélia  
Marroquim (in memoriam).*

# Agradecimentos

Durante esses seis anos de mestrando, doutorando, trabalhando e convivendo no LCG tive a sorte de fazer muitos amigos e o prazer de trabalhar ao lado de todos professores e vários alunos do laboratório. Agradecimentos especiais ao meu orientador Paulo Roma que me acompanha desde o início junto ao professor Claudio Esperança, e sempre me deram confiança para seguir não apenas nesta pesquisa de tese mas em todos os caminhos da ciência durante este tempo; ao professor Antonio Oliveira por todo empenho e conversas inspiradoras na hora do almoço; e ao professor Ricardo Farias pela amizade e motivação.

Afora das paredes do LCG também gostaria de agradecer o imenso apoio do professor Luiz Velho que acompanhou toda a minha trajetória acadêmica desde o mestrado até este momento. Suas conversas e idéias entusiasmantes são de valor imensurável.

Essa tese não seria possível sem a colaboração de uma pessoa que acima de tudo se tornou um bom amigo, Martin Kraus. O seu empenho e dedicação à pesquisa são fontes inspiradoras para qualquer cientista.

Meia dúzia de anos em um lugar cria laços que ficam para uma eternidade, por isso agradeço todos os membros da família LCG que se tornaram verdadeiros irmãos. Ao amigo Alvaro, ingressamos juntos no mestrado e terminamos o doutorado ao mesmo tempo, agora partimos para destinos diferentes mas com certeza os caminhos neste intercâmbio Brasil-Peru ainda se cruzarão; ao amigo André, que apesar de proporcionar discussões interminavelmente longas sobre pesquisa e vida, tem uma dedicação enorme e um coração maior ainda; ao grande Saulo, que nunca deixa de nos impressionar com sua jovialidade; ao amigo Disney, pra quem nunca tem tempo ruim e nunca deixa ninguém desanimar; ao Yalmar, a quem nunca falta tempo para ajudar os colega; e finalmente à todos outros companheiros que estão chegando,

ficando e partindo: Felipe, Alberto, Leandro, Zezim, Guina, Carlos e Flávio. E sempre ao irmão Vitor, estando longe ou perto é sempre um grande companheiro de todas horas.

À todos colegas e funcionários do Pesc, é um enorme prazer poder compartilhar essa conquista com vocês. Um abraço especialmente grande pra Mercedes, ao lado dela não existem pessoas tristes.

Como nem tudo na vida é trabalho, mais uma vez agradeço todos grandes amigos, eu me considero uma pessoa de extrema sorte por ter tantas pessoas tão queridas do meu lado e que apesar de não terem a mínima idéia do que eu faço acham tudo impressionante: todos eles de Santa Tereza, de trilhas, de bares, do colégio, da banda, da faculdade e de todos os cantos do Brasil e do mundo.

À minha família, o que mais me faz falta nessa minha cidade maravilhosa, mas se eles não fossem pernambucanos talvez não fossem tão perfeitos assim.

À minha irmã e minha sobrinha que continua peste.

Aos Escravos da Mauá e ao Carnaval de Olinda!!

Agradeço ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) pelo suporte financeiro.

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

## VISUALIZAÇÃO INTERATIVA DE MODELOS BASEADOS EM PONTOS

Ricardo Guerra Marroquim

Setembro/2008

Orientador: Paulo Roma Cavalcanti

Programa: Engenharia de Sistemas e Computação

Métodos baseados em imagem possuem complexidade proporcional à resolução da imagem e não ao número de primitivas, o que muitas vezes permite renderizar cenas mais eficientemente do que métodos baseados na geometria. Adicionalmente, por causa de sua natureza paralela, esses métodos são altamente adaptáveis para execução na placa gráfica. Por outro lado, durante os últimos anos a área de gráficos baseados em pontos emergiu como um complemento promissor para outros tipos de representação. No entanto, com o aumento contínuo da complexidade da cena é preciso desenvolver soluções para renderizar diretamente grandes nuvens de pontos. Nesta tese propomos algoritmos para renderizar eficientemente modelos de pontos volumosos utilizando técnicas de reconstrução de imagem. O custo de reconstrução é associado apenas à resolução de saída, exceto pela fase de projeção das amostras. O método também é estendido para interpolar conjuntamente outros tipos de primitivas, como linhas e triângulos. Por fim, a estratégia também é eficiente quanto ao uso de memória, já que não requer nenhuma estrutura de dados auxiliar.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

## INTERACTIVE VISUALIZATION OF POINT MODELS

Ricardo Guerra Marroquim

September/2008

Advisor: Paulo Roma Cavalcanti

Department: Systems Engineering and Computer Science

Image based methods have proved to efficiently render scenes with a higher efficiency than geometry based approaches, mainly because one of their most important advantages: the bounded complexity by the image resolution, instead of by the number of primitives. Furthermore, due to their parallel and discrete nature, they are highly suitable for GPU implementations. On the other hand, during the last few years point-based graphics has emerged as a promising complement to other representations. However, with the continuous increase of scene complexity, solutions for directly processing and rendering point clouds are in demand. In this thesis, algorithms for efficiently rendering large point models using image reconstruction techniques are proposed. Except for the projection of samples onto screen space, the reconstruction time is bounded only by the screen resolution. The method is also extended to interpolate other primitives, such as lines and triangles. In addition, no extra data-structure is required, making the strategy memory efficient.

# Sumário

<b>Lista de Figuras</b>	<b>xi</b>
<b>Lista de Tabelas</b>	<b>xvii</b>
<b>1 Introdução</b>	<b>1</b>
1.1 Contextualização do Trabalho . . . . .	1
1.1.1 Aquisição . . . . .	2
1.1.2 Reconstrução da Superfície . . . . .	2
1.1.3 Processamento e Modelagem . . . . .	3
1.1.4 Visualização . . . . .	3
1.2 Motivação . . . . .	3
1.3 Proposta . . . . .	5
1.4 Contribuições . . . . .	6
1.5 Organização da Tese . . . . .	8
<b>2 Trabalhos Relacionados</b>	<b>10</b>
2.1 Renderização Baseada em Imagem . . . . .	10
2.1.1 Renderização sem Geometria . . . . .	11
2.1.2 Renderização com Geometria Explícita . . . . .	13
2.2 Renderização Baseada em Pontos . . . . .	15
2.2.1 Visualizando Pontos . . . . .	15
2.2.2 Adicionando Informação à Reconstrução . . . . .	20
2.2.3 Renderização na GPU . . . . .	21
2.2.4 Alternativas ao Splatting . . . . .	23
2.3 Discussão . . . . .	24



<b>3</b>	<b>Surface Splatting</b>	<b>26</b>
3.1	Teoria do Surface Splatting . . . . .	26
3.1.1	Parametrização Local . . . . .	27
3.1.2	Equação de Renderização . . . . .	28
3.1.3	Filtros Gaussianos . . . . .	29
3.1.4	Sessões Cônicas . . . . .	32
3.1.5	Mapeamento 2D-2D Composto . . . . .	35
3.2	Implementando um Algoritmo de Surface Splatting Interativo . . . . .	37
3.2.1	Núcleo de Reconstrução . . . . .	37
3.2.2	Filtro de Reamostragem . . . . .	38
3.2.3	Compondo a Matriz de Pixels . . . . .	38
<b>4</b>	<b>Reconstrução de Superfícies em Espaço de Imagem para Nuvens de Pontos Não-Estruturadas</b>	<b>40</b>
4.1	Algoritmo em Pirâmide . . . . .	40
4.1.1	Algoritmo de Análise-Síntese . . . . .	42
4.2	Algoritmo em Pirâmide para Renderização de Pontos . . . . .	43
4.2.1	Projeção dos Pontos . . . . .	44
4.2.2	Interpolação com Algoritmo em Pirâmide . . . . .	45
4.2.3	Iluminação por Pixel . . . . .	48
4.2.4	Implementação na GPU . . . . .	49
4.2.5	Resultados . . . . .	51
4.3	Discussão . . . . .	52
<b>5</b>	<b>Renderização de Linhas e Triângulos em Espaço de Imagem</b>	<b>58</b>
5.1	Renderização de Linhas com Reconstrução de Imagens . . . . .	58
5.1.1	Fitas . . . . .	58
5.1.2	Tubos . . . . .	59
5.1.3	Exemplo: Folhas e Galhos . . . . .	61
5.2	Combinando Pontos, Linhas e Triângulos . . . . .	63
5.3	Discussão . . . . .	65
<b>6</b>	<b>Pirâmide com Núcleos de Busca</b>	<b>67</b>
6.1	Criando a Hierarquia . . . . .	68

6.2	Interpolando os Splats . . . . .	69
6.3	Discussão . . . . .	71
<b>7</b>	<b>Níveis de Detalhe para Pontos</b>	<b>76</b>
7.1	Texturas de Objeto . . . . .	76
7.1.1	Estrutura Hierárquica . . . . .	77
7.1.2	Métrica de Erro . . . . .	79
7.2	Projeção dos Pontos . . . . .	80
7.3	Discussão . . . . .	80
<b>8</b>	<b>Conclusões</b>	<b>84</b>
8.1	Trabalhos Futuros . . . . .	88
	<b>Referências Bibliográficas</b>	<b>90</b>

# Lista de Figuras

2.1	O valor do pixel $(x, y)$ , intersectado pelo raio partindo do observador, é computado utilizando as parametrizações dos planos $(u, v)$ e $(s, t)$ . . . . .	12
2.2	(a) uma fatia $(s, u, v)$ do modelo de uma fruteira e (b) o mesmo modelo renderizado, ambos gerados pelo Lumigraph. (Retiradas de [1])	13
2.3	Duas imagens geradas pelo Light Field. (Retiradas de [2]) . . . . .	14
2.4	(a) uma imagem de referência e (b) uma imagem gerada com uma Árvore LDI a partir de 36 referências. (Retiradas de [3]) . . . . .	15
2.5	A superfície escura é renderizada utilizando um nível de menor resolução na pirâmide onde não há buracos. Este hierarquia é utilizada para preencher os pixels vazios na imagem. (Retirada de [4]) . . . . .	17
2.6	Duas imagens geradas pelo algoritmo de Grossman e Dally. (Retiradas de [4]) . . . . .	17
2.7	Duas imagens geradas pelo algoritmo <i>Surfels</i> . (Retiradas de [5]) . . . . .	18
2.8	Um detalhe do modelo digital da cabeça de Davi renderizado com o <i>QSplat</i> . (Retirada de [6]) . . . . .	19
2.9	Imagem gerada com o algoritmo <i>Surface Splatting</i> . (Retirada de [7]) . . . . .	20
2.10	Imagens geradas com os <i>Pontos Diferenciais</i> de Kalaiiah e Varshney. (Retiradas de [8]) . . . . .	21
2.11	Imagens geradas com os <i>Phong Splatting</i> de Botsch et al. (Retiradas de [9]) . . . . .	22
2.12	(a) imagem gerada com os <i>c-surfels</i> de Luz et al. e (b) a representação poligonal de um <i>c-surfel</i> . (Retiradas de [10]) . . . . .	23
2.13	Imagens geradas pelos algoritmo de (a) Botsch e Kobbelt e (b) pelo algoritmo de Guennebaud et al. (Retiradas de [11] e [12], respectivamente) . . . . .	24

3.1	Na esquerda, o retalho colocado sobre a amostra no espaço de objeto. Na direita, o sistema de coordenadas locais em volta de um ponto $p'_i$ , com a vizinhança $\{p'_a, p'_b, p'_c \dots\}$ definida neste sistema. . . . .	28
3.2	Uma Gaussiana comparada com um filtro caixa (filtro passa-baixa ideal), ambos no espaço de frequência. A amplitude da Gaussiana dita o balanço entre <i>aliasing</i> e borramento. . . . .	31
3.3	Os três possíveis casos não degenerados das cônicas: elipse, parábola e hipérbole. . . . .	32
3.4	Na esquerda o isocontorno projetado corretamente como uma cônica, mas com centro deslocado. Na direita o centro projetado corretamente mas com isocontorno incorreto. (Retirada de [13]) . . . . .	34
4.1	(a) Uma pirâmide construída a partir de uma imagem com resolução 16x16. (b) Um pixel de uma resolução mais baixa é computado pela média dos quatro diretamente abaixo. . . . .	41
4.2	(a) Imagem original e (b) o primeiro nível da decomposição utilizando a base de Haar. Note que o canto superior esquerdo da decomposição corresponde a uma resolução mais baixa da imagem original ao reduzir cada dimensão pela metade, enquanto que os coeficientes de detalhe são armazenados nos outros quadrantes. (Retiradas de [14]) . . . . .	42
4.3	Os pesos utilizados para construir um novo pixel durante a fase de (a) <i>análise</i> e (b) <i>síntese</i> . . . . .	43
4.4	Resultados do algoritmo de reconstrução de imagens em GPU. Na linha de cima as duas imagens de entrada com pixels vazios, em baixo as respectivas reconstruções. (Retiradas de [15]) . . . . .	44
4.5	Resumo do algoritmo de renderização de pontos: primeiro, os pontos são projetados para o espaço de imagem; em seguida, o algoritmo de <i>análise-síntese</i> é aplicado para interpolar os valores das amostras; por último, iluminação por pixel é computada. . . . .	45
4.6	A elipse é definida pelo centro de projeção $p_i$ e a normal projetada $\vec{n}$ . O eixo maior perpendicular à normal tem tamanho $a$ igual ao raio projetado, enquanto o eixo menor tem tamanho $b = a \cdot n_z$ . . . . .	46

4.7	O vetor de deslocamento é ilustrado para os cinco primeiros níveis. A seta indica o deslocamento do centro do pixel correspondente (em verde) ao centro da elipse (círculo preto). Apenas no nível 0 o centro da elipse corresponde ao centro do pixel. Note que somente uma elipse foi propagada, não havendo necessidade de realizar médias de elipses.	47
4.8	Durante a propagação da elipse seu centro é mantido pelo vetor de deslocamento. O pixel em verde no nível 0 corresponde ao centro da elipse.	48
4.9	A seqüência demonstra a evolução de duas elipses na pirâmide até o momento ( <i>d</i> ) em que são combinadas ( <i>e</i> ) gerando uma única elipse (vermelha) com novo centro e eixos.	49
4.10	( <i>a</i> ) Elipses renderizadas com 20% do raio para ilustrar a distribuição dos pontos e a eliminação de pontos oclusos (e.g., parte traseira da cabeça) e ( <i>b</i> ) o mesmo modelo renderizado sem buracos e iluminado.	50
4.11	( <i>a</i> ) O mapa de normais gerado pela interpolação em pirâmide e ( <i>b</i> ) a imagem resultante após iluminação por pixel.	51
4.12	Diagramação dos 10 níveis gerados para uma imagem de $512 \times 512$ . (Retirada de [15])	52
4.13	Esquema de <i>ping-pong</i> entre as duas texturas: ( <i>a</i> ) a fase de análise constrói a pirâmide de baixo para cima; ( <i>b</i> ) a fase de cópia preenche os níveis ausentes; ( <i>c</i> ) a fase de síntese computa os pixels inválidos de cima para baixo.	54
4.14	A primeira textura armazena as componentes da normal e o raio projetado, enquanto que a segunda armazena a profundidade, o intervalo de profundidade e as componentes do vetor de deslocamento.	54
4.15	O mesmo modelo renderizado ( <i>a</i> ) sem atributos de cor e ( <i>b</i> ) com atributos de cor por amostra.	55
4.16	Modelos renderizados com o algoritmo em pirâmide. De cima para baixo: Armadillo, Happy Buddha, Asian Dragon e Thai Statue.	56

5.1	(a) Representação de uma fita por uma polilinha contendo cinco vértices com normais e raios. (b) Ilustração da renderização da polilinha correspondendo ao centro da fita. Normais e raios são interpolados para cada pixel coberto. . . . .	59
5.2	Uma fita torcida com os dois lados renderizados e iluminados. . . . .	60
5.3	(a) Representação de um tubo por uma polilinha. A cada vértice é atribuído o vetor tangente e o raio do tubo. (b) Ilustração da rasterização da polilinha onde os vetores tangentes e raios são linearmente interpolados. . . . .	60
5.4	Iluminação de tubos: a normal $n$ da superfície é computada pela projeção do vetor de iluminação $l$ no plano ortogonal à tangente $t$ . . .	61
5.5	Ilustração da conversão de (a) folhas e (b) galhos para polilinhas. . .	62
5.6	Uma magnificação de uma folha renderizada como uma fita. . . . .	62
5.7	Comparação entre (a) o modelo original representado por triângulos e (b) a representação por linhas. As áreas retangulares marcadas representam as magnificações da Figura 5.8. . . . .	63
5.8	(a) Magnificação da imagem gerada com triângulos na Figura 5.7a e (b) . . . . .	63
5.9	Renderização de dez árvores com (a) o modelo triangular e (b) o modelo baseado em linhas. . . . .	64
5.10	Renderização combinando dois tipos de primitivas: pontos (Dragão) e triângulos (Buddha). . . . .	65
5.11	(a) Um modelo híbrido com triângulos (azul) e pontos (vermelho). (b) Um detalhe do modelo e (c) o mesmo detalhe com as arestas dos triângulos e pontos com apenas uma fração dos raios. . . . .	66
6.1	O pixel deve ser grande suficiente para que o núcleo cubra toda a projeção da amostra. Utilizando um tamanho de $5 \times 5$ o nível da figura da esquerda não é suficiente, pois somente os pixels cinzas são capazes de encontrar o centro da projeção em preto. Na figura da direita, o nível possui a cobertura mínima para que todos pixels dentro da elipse encontrem o seu centro. . . . .	68

6.2	As buscas efetuadas por um pixel (nível 0) com um núcleo $3 \times 3$ . Em preto o pixel correspondente em cada nível e em cinza escuro a cobertura da núcleo. Note que o nível 4 contendo apenas um pixel não é necessário, uma vez que o nível 3 é totalmente coberto. . . . .	70
6.3	A mesma cena renderizada com três tamanhos diferentes de núcleo; da esquerda para direita: $3^2$ , $5^2$ e $7^2$ . Note que o ganho qualitativo entre os núcleos $5^2$ e $7^2$ é imperceptível, porém a performance reduz-se em 38%. . . . .	71
6.4	Um detalhe do rabo do dragão renderizado com um núcleo $5 \times 5$ (esquerda) e o algoritmo PPR (direita). Note como a estratégia com núcleos produz silhuetas mais suaves e com menos artefatos. . . . .	72
6.5	Um detalhe da barba do Netuno renderizado com um núcleo $5 \times 5$ (esquerda) e o algoritmo PPR (direita). . . . .	73
6.6	Da esquerda para direita: Buddha, Netuno e a Statuette renderizados com o algoritmo de núcleos. . . . .	73
6.7	Detalhe do Netuno. . . . .	74
6.8	Detalhe das esculturas na parte inferior do modelo Statuette. . . . .	75
7.1	Durante a renderização, o vértice representativo é capaz de recuperar um retalho da superfície armazenado na Textura de Objeto, utilizando os recursos do <i>shader</i> de geometria. . . . .	77
7.2	(a) Os <i>splats</i> são agrupados para formar os níveis de resolução mais baixos. (b) Representação dos níveis de detalhe na hierarquia: o nível mais alto contém os vértices representativos capazes de recuperar os demais níveis armazenados na Textura de Objeto. . . . .	78
7.3	O erro perpendicular é a distância máxima entre o plano tangente ao <i>splat</i> pai com centro em $c$ e as extensões dos filhos com centros em $c_i$ . . . . .	79

7.4	(a) Armadillo renderizado utilizando os níveis de detalhe, onde as cores cinza, amarelo, azul e vermelho correspondem respectivamente aos níveis 0, 1, 2 e 3. (b) O mesmo modelo sem níveis de detalhe, ou seja, apenas os splats do nível 0. Note como as regiões suaves, como o peitoral do modelo, são renderizadas com splats de menor resolução. Para efeitos de ilustração os splats são renderizados com apenas uma fração do raio. . . . .	81
7.5	Armadillo renderizado com níveis de detalhe (a) de um ponto de vista próximo e (b) de um ponto de vista distante. O grafo em barras indica a porcentagem de amostras renderizadas de cada nível, onde as cores cinza, amarelo, azul e vermelho correspondem respectivamente aos níveis 0, 1, 2 e 3. . . . .	82
7.6	Uma cena contendo três modelos (Armadillo, Netuno e Dragão) renderizada com níveis de detalhe. O grafo indica a porcentagem de pontos renderizados em cada nível, onde as cores cinza, amarelo, azul e vermelho correspondem respectivamente aos níveis 0, 1, 2 e 3. . . .	83
8.1	Cada tipo de primitiva é mais vantajosa em uma faixa em relação a distância do observador. Pontos são incluídos em um faixa entre os polígonos e estruturas regulares baseada em imagem, e.g., LDIs. . . .	87



# Lista de Tabelas

- 4.1 Tempo de renderização, em quadros por segundo *qps*, com uma placa 7800 GTX para duas resoluções de saída diferentes. \*Cada tempo total por quadro é seguido por dois tempos em parênteses: tempo de projeção e tempo de interpolação. Todos os tempos são descritos em milisegundos. . . . . 53
- 4.2 Tempo de renderização com uma placa 8800 GTS com e sem interpolação dos atributos de cor. \*Cada tempo total por quadro é seguido por dois tempos em parênteses: tempo de projeção e tempo de interpolação. Todos os tempos são descritos em milisegundos. . . . . 53

# Capítulo 1

## Introdução

A representação de superfícies é um problema fundamental da computação gráfica. Normalmente, o tipo de aplicação define qual representação é mais apropriada, entre as quais podem-se citar: superfícies paramétricas (NURBS, Bézier), superfícies implícitas (*Level Sets*, RBFs) e malhas poligonais. Introduzir uma nova representação pode ser um processo demorado, pois exige um grande esforço até que seus benefícios sejam reconhecidos. Por este motivo, a representação baseada em pontos precisou de alguns anos até ser amplamente aceita. Há mais de vinte anos, Levoy e Whitted [16] publicaram o primeiro artigo sobre o uso de pontos como primitivas gráficas, porém foi apenas durante a última década que a área recebeu uma atenção crescente. Não é por menos que já foram realizados cinco edições de um simpósio específico da área (*Symposium on Point-Based Graphics*), alguns dos principais congressos de computação gráfica reservam sessões ao tema e o primeiro livro dedicado exclusivamente ao assunto foi lançado recentemente [17].

### 1.1 Contextualização do Trabalho

Para que o trabalho apresentado seja melhor enquadrado dentro de um escopo maior, e sua motivação melhor compreendida, a seguir, serão descritos brevemente as etapas de criação de conteúdo. Este processo começa com a digitalização de um objeto real, passando por processos de modelagem e transformações, até a sua visualização.

### 1.1.1 Aquisição

Esta etapa envolve um processo físico para registrar objetos tridimensionais. *Scanners* 3D não só se tornaram ferramentas mais populares e poderosas nos últimos anos, como também mais acessíveis. Serviços especializados para registrar objetos reais como modelos tridimensionais são cada vez mais comuns. A resolução dos equipamentos modernos é suficiente para capturar fielmente os objetos, podendo chegar a décimos de milímetro e gerar registros na ordem de bilhões de amostras.

Para capturar integralmente o objeto o processo de aquisição de modelos 3D geralmente se realiza através de múltiplos registros pelo equipamento. O resultado é uma amostragem discreta do objeto físico. Além de dados geométricos, alguns equipamentos também possibilitam a captura de informações adicionais, e.g., cor, propriedade do material e valor de confiança da medida. Cada registro, dependendo do equipamento, tem forma de um conjunto de pontos não estruturado com coordenadas tridimensionais, ou de uma *Range Image*, onde é retornada uma imagem com informação de profundidade por pixel. Apesar do segundo método ser mais popular, de uma forma ou de outra é necessário que os registros sejam alinhados em uma etapa conseguinte. Num último momento, um pós-processamento é realizado para remover ruídos e amostras que estão numericamente distantes das demais (*outliers*). As vezes, no caso de superfícies convolutas ou com muitas concavidades, também se faz necessário o preenchimento de buracos não registrados pelo equipamento.

### 1.1.2 Reconstrução da Superfície

A conversão de uma nuvem de pontos para alguma outra representação onde a superfície é representada continuamente, é chamada de reconstrução de superfície. Entre as opções possíveis estão as malhas poligonais, retalhos paramétricos ou a curva de nível zero de uma função. A escolha depende tanto do tipo de aplicação quanto do tipo de entrada de dados e de restrições impostas, como, por exemplo, de que a superfície seja uma variedade. No entanto, a reconstrução mais comum é gerar malhas poligonais, como descrito no trabalho de Hoppe [18, 19].

### 1.1.3 Processamento e Modelagem

Esta fase abrange uma série de transformações para analisar ou modificar o modelo. O processamento inclui operações tais como filtragem, compressão, segmentação, extração de características e reamostragem (refinamento ou redução), entre outras. A modelagem visa editar a aparência e forma do modelo, como, por exemplo, com deformações a mão livre, pintura ou texturização da superfície, e operações de extrusão.

### 1.1.4 Visualização

A última etapa refere-se à visualização do modelo. Este processo envolve amostrar a representação tridimensional em um plano discreto bidimensional, geralmente a matriz de pixels de um monitor. Todo algoritmo de renderização compromete-se com um balanço entre qualidade e desempenho. No entanto, para visualizações interativas, as malhas poligonais obtiveram uma grande vantagem sobre as demais representações. Isto se deve principalmente pela especialização do hardware gráfico em processar triângulos. Comumente, outras representações são transformadas em malhas poligonais para alcançar desempenhos interativos.

Por outro lado, com o advento de programação em placa gráfica, as modernas GPUs (Graphics Processor Unit), é cada vez mais possível gerar visualizações em tempo real de alta qualidade mesmo para representações que não sejam malhas. Por exemplo, métodos para renderizar nuvens de pontos diretamente alcançam taxas interativas da ordem de dezenas de milhões de amostras por segundo. É neste tópico que esta tese propõe suas contribuições.

## 1.2 Motivação

Pontos, por serem a primitiva geométrica mais simples, oferecem vantagens sobre outras representações. Um dos motivos impulsionadores da área, e que talvez explique seu reconhecimento tardio, é a constante evolução dos equipamentos de aquisição. Ao trabalhar diretamente com pontos, é possível evitar a etapa de reconstrução da superfície, que geralmente é custosa e complexa.

Outro motivo que torna a área atrativa, é o fato da resolução dos dispositivos de

saída, como monitores, não acompanhar o crescimento da resolução dos dados adquiridos. Desta forma, triângulos de uma malha muito densa tendem a ser projetados com tamanhos menores do que ao de um pixel, gerando uma sobrecarga desnecessária no processo de rasterização devido ao custo de montagem dos polígonos. Além disso, o custo de manutenção e processamento de malhas é um problema para aplicações que lidam com grandes volumes de dados.

Pode-se destacar como principal vantagem da representação por pontos a ausência de informação de conectividade. Malhas triangulares possuem um custo adicional para preservar conectividade e respeitar restrições topológicas dos elementos. Seguramente, esta vantagem não é gratuita, pois são necessários mecanismos alternativos para determinar vizinhanças locais nas nuvens de pontos. Ainda mais, para obter resultados de alta qualidade, geralmente é preciso uma amostragem mais densa para pontos do que para malhas poligonais. Entretanto, as estruturas espaciais e de dados são naturalmente mais simples para conjuntos de pontos, muitas vezes compensando o maior número de primitivas.

Embora hajam pesquisas importantes sobre renderização de pontos, os algoritmos propostos ainda não são tão eficientes quanto os de malhas poligonais, principalmente devido a especialização dos hardware gráficos em renderizar triângulos. Por este motivo, algumas aplicações que processam e modelam diretamente nuvens de pontos ainda recorrem à malha poligonal para visualização. Simulações de fluidos e fumaça que utilizam partículas, por exemplo, usam tradicionalmente pontos; porém, a visualização destas simulações são realizadas com malhas poligonais, geralmente adquiridas por métodos de extração de iso-superfícies, e.g., *Marching Cubes* [20].

Em contrapartida, mesmo não aproveitando inteiramente as otimizações de hardware, algumas aplicações são beneficiadas ao utilizar uma representação mais simples sem conectividade explícita. Em particular, pode-se ressaltar a área de objetos dinâmicos ou deformáveis: as primitivas dos objetos podem ser modificadas sem preocupação em manter condições de variedade. Adicionalmente, operações de reamostragem são mais simples por não haver necessidade de reestruturar a representação. De fato, a estrutura de pontos já possui uma hierarquia de níveis de detalhes implícita, onde é possível, por exemplo, visualizar ou enviar dados progressivamente.

Ainda mais, existem aplicações onde a importância da manipulação direta dos pontos é evidente, especialmente quando os dados são obtidos por amostragens densas. Especificamente, na área de renderização de pontos, são necessários algoritmos que permitam uma visualização interativa e de alta qualidade, permitindo uma fácil investigação do modelo virtual. No caso de aplicações para a área de *Herança Cultural*, por exemplo, o modelo digital auxilia na restauração e reconstrução de estátuas, artefatos e construções arquitetônicas. Neste caso a preservação dos detalhes em alta resolução é imprescindível. No entanto, o custo para gerar uma malha do modelo que resultará em triângulos muito pequenos não traz nenhuma vantagem aparente. Ademais, a reconstrução da superfície implica em heurísticas e aproximações, podendo levar a interpretações falsas sobre o modelo real.

Em suma, o uso da representação baseada em pontos não tem como objetivo substituir outras, mas sim complementá-las. A sua natureza simples e versátil carrega vantagens para algumas aplicações, mas é preciso discernir em que situações ela é realmente útil.

### 1.3 Proposta

O objetivo deste trabalho é pesquisar e desenvolver algoritmos para visualização interativa baseada em pontos. Mais especificamente, é abordado o problema de visualização de dados volumosos, à vista que os métodos existentes dependem de um processo custoso para gerar imagens de alta qualidade, i.e., possuem complexidade diretamente relacionada ao número de amostras.

Visando reduzir este custo associado à complexidade geométrica, recorreremos ao uso de técnicas de renderização baseada em imagens. Neste âmbito, a complexidade pode ser reduzida apenas à resolução da saída, e não ao número de amostras: “Uma grande vantagem de se representar um objeto com uma imagem é que a renderização é proporcional ao número de pixels renderizados, e não ao número de vértices do modelo geométrico.” [21]. Como mencionado anteriormente, a resolução dos equipamentos de aquisição evolui mais rapidamente do que a dos dispositivos de saída. Por este motivo, o desvinculo com a complexidade da geometria traz uma grande vantagem por refletir melhor esta evolução.

Nesta tese abordaremos a reconstrução de superfícies em um contexto apenas de visualização, diferentemente do citado no processo de criação de conteúdo. Aqui, a reconstrução é dependente das condições de camera, cena e iluminação, e não ocorre uma transformação do modelo para uma nova representação. Em outras palavras, a reconstrução é gerada na forma de uma imagem, realizando uma interpolação da superfície para preencher os espaços (pixels) vazios entre as amostras. Certamente, para modelos densos, não haverá muitos pixels a serem preenchidos, porém, sob ampliações para inspeção de detalhes o problema volta a surgir. Por isso priorizamos um método robusto, capaz de tratar modelos de diversas densidades sem requerimento de uniformidade.

Para suportar e contextualizar o trabalho apresentado, discussões sobre a programação em placas gráficas (GPUs) são incluídas nesta tese. Esta técnica se tornou muito popular, e até quase indispensável, para aplicações interativas. As GPUs modernas são processadores paralelos poderosos, capazes de executar programas genéricos. Os primeiros hardware gráficos programáveis surgiram por volta de 2002. Desde então, se tornou uma ferramenta que permite ao desenvolvedor sobre-escrever funcionalidades fixas do *pipeline* gráfico, aproveitando o poder de paralelismo das GPUs.

Como será mostrado, a representação por pontos também oferece uma série de vantagens que permitem explorar os recursos das placas gráficas eficientemente. Pelo fato de um ponto carregar todos os atributos necessários para processamento e renderização, a sua estrutura pode ser mapeada de uma maneira simples e direta na GPU: a primitiva geométrica (ponto) se aproxima da primitiva de imagem (pixel).

## 1.4 Contribuições

De uma forma geral, não só foram pesquisados e desenvolvidos novos métodos para visualizar interativamente nuvens de pontos densas, mas o resgate das técnicas de renderização baseada em imagem possibilitou uma discussão sobre o seguinte paradigma: primitiva geométrica *vs* primitiva de imagem. O processo de renderização é fundamentalmente um processo de reamostragem, onde o modelo é adequado à matriz de pixels. Tendo em vista que os dados são capturados em uma resolução

muito maior do que aquela em que são visualizados, a eficiência da renderização está associada a um meio de compatibilizar a complexidade geométrica com a da imagem. Este processo deve ser realizado de forma que haja a menor perda de informação possível.

A seguir são descritas as principais contribuições deste trabalho:

- *Reconstrução de superfícies em espaço de imagem a partir de nuvens de pontos não estruturadas*

É apresentado um algoritmo para reconstrução de superfícies, a partir de nuvens de pontos, executado inteiramente no espaço de imagem. Por não depender do número de amostras projetadas, a complexidade da reconstrução é reduzida à resolução de saída. Esta parte do trabalho foi publicada no **Symposium on Point-Based Graphics 2007** [22].

- *Extensão do algoritmo anterior para modelos híbridos contendo linhas e triângulos*

Para abranger o escopo da proposta, o método em espaço de imagem foi estendido para renderizar linhas e triângulos junto aos pontos. Adicionalmente, foi criado um algoritmo específico para renderizar árvores usando linhas para representar galhos e folhas. Esta extensão foi publicada no **Computer & Graphics Journal** [23].

- *Nova proposta de interpolação de superfícies para gerar reconstruções de alta qualidade*

Uma investigação mais profunda sobre a interpolação no espaço de imagem resultou em uma nova técnica para respeitar de maneira mais fidedigna as bordas e silhuetas do modelo. A reconstrução ainda é executada inteiramente no espaço de imagem, porém é realizada de uma forma mais precisa usando padrões de busca na hierarquia. Este trabalho foi aceito para publicação no **SIBGRAPI 2008**.

- *Níveis de detalhe para nuvens de pontos*

Finalmente, foram exploradas algumas técnicas para gerar uma estrutura de níveis de detalhe para os modelos de pontos. Um método utilizando recursos modernos das GPUs será apresentado como poster no **SIGGRAPH 2008** (semi-finalista no **ACM Student Research Competition**).



## 1.5 Organização da Tese

Esta tese está organizada da forma descrita abaixo:

- *Capítulo 2*

Revisão dos trabalhos relacionados na literatura de Renderização Baseada em Imagem e Renderização Baseada em Pontos.

- *Capítulo 3*

Apresentação da teoria do algoritmo *Surface Splatting*, atualmente o mais popular renderizador de pontos. Também é descrita uma abordagem do algoritmo para renderização interativa.

- *Capítulo 4*

Apresentação do algoritmo *Pyramid Point Renderer* para visualização interativa de nuvens de pontos. A técnica propõe a reconstrução de superfícies em espaço de imagem utilizando estruturas hierárquicas.

- *Capítulo 5*

Seguindo a proposta do capítulo anterior, extensões são apresentadas para incluir modelos híbridos, e/ou renderizar primitivas como linhas e triângulos dentro da mesma metodologia.

- *Capítulo 6*

Uma nova abordagem para reconstrução em espaço de imagem, onde o objetivo é gerar imagens de alta qualidade respeitando ao máximo as fronteiras do modelo. Algumas idéias e algoritmos são apresentados junto a uma discussão sobre suas vantagens e desvantagens.

- *Capítulo 7*

Uma maneira de aumentar o desempenho dos algoritmos propostos é diminuir o custo da projeção das amostras, a única que ainda depende da resolução de entrada. Para tanto, alguns métodos e estruturas de níveis de detalhes para pontos são propostos neste capítulo.

- *Capítulo 8*

Finalizamos a tese com uma discussão geral sobre métodos em espaço de ima-

gem para renderizar diretamente nuvens de ponto. Também são apresentadas sugestões para trabalhos futuros.

# Capítulo 2

## Trabalhos Relacionados

Esta tese está fundamentada essencialmente em duas áreas da computação e processamento gráfico: **Renderização Baseada em Imagem (RBI)** e **Renderização Baseada em Pontos (RBP)**. Principalmente nas primeiras propostas de RBP, houve uma grande influência dos algoritmos baseados em imagem. Nestes trabalhos o objetivo era transferir a rasterização e texturização do processo de visualização para uma fase de pré-processamento. No entanto, os dados originais não eram necessariamente conjuntos de pontos. Com o surgimento de novas motivações, e.g., equipamentos de aquisição de alta resolução, soluções para renderizar diretamente nuvens de pontos foram propostas, porém, distanciando-se dos conceitos dos algoritmos baseados em imagem.

Na primeira seção deste capítulo é apresentado o tópico Renderização Baseada em Imagem, enquanto que na segunda são brevemente descritos alguns trabalhos de Renderização Baseada em Pontos. No próximo capítulo um algoritmo específico de RBP, o *Surface Splatting*, é descrito em mais detalhes.

### 2.1 Renderização Baseada em Imagem

O termo renderização baseada em imagem surgiu com vigor nos meados da década de 90 [24]. As primeiras propostas objetivavam atingir renderizações fotorealísticas utilizando a imagem como primitiva gráfica. De fato, quando se pergunta: “Quantas primitivas geométricas são necessárias para convencer de que a imagem resultante é real?”, podemos usar a citação de Alvy Smith [25] como resposta: “Realidade

são 80 milhões de polígonos.”. No entanto, é preciso refletir sobre a restrição de serem primitivas geométricas. Em outras palavras, ou ainda melhor, nas palavras de McMillan e Bishop [26]: “Ironicamente, a principal medida subjetiva da qualidade da imagem utilizada pelos proponentes de sistemas geométricos de renderização, é o grau com o qual as imagens resultantes são indistinguíveis de uma fotografia”. A questão colocada é: se o resultado desejado é gerar uma renderização com qualidade fotográfica, porque não utilizar a própria imagem como primitiva?

De toda maneira, enquanto uma fotografia oferece uma quantidade enorme de informação sobre a estrutura e aparência da cena, ela ainda é uma imagem bidimensional estática. Sendo assim, não é possível manipular o ambiente de maneira trivial. A fim de superar esta limitação, surgiu a área de *Modelagem e Renderização Baseada em Imagem*, com o propósito de usufruir da facilidade de aquisição e eficiência de visualização das imagens, geralmente fotografias.

Lengyel [24] classificou os algoritmos de renderização baseada em imagem em uma escala de necessidade de geometria, onde menos significa um baseamento na aparência, enquanto mais geometria aponta para uma abordagem baseada em física. Ele situa ainda a área de gráficos baseados em imagem como uma sobreposição entre as áreas de computação gráfica e visão computacional. A seguir, serão descritos alguns algoritmos situando-os em duas classes quanto ao uso de geometria.

### 2.1.1 Renderização sem Geometria

No extremo da linha relativo à aparência, Lengyel coloca dois algoritmos que foram desenvolvidos praticamente em simultâneo: o *Light Field* [2] e o *Lumigraph* [1]. Ambos descrevem um sistema para caracterizar a propagação da luz em uma cena como uma função 4D. Uma das principais vantagens desta classe de algoritmos é o seu custo de visualização, que é praticamente independente da complexidade da cena.

Para entender melhor como a função é gerada a partir de imagens, é preciso descrever a referência destes trabalhos: a *função plenóptica*. Esta função é definida como a intensidade dos raios de luz passando pela camera de cada posição 3D  $(V_x, V_y, V_z)$ , com cada ângulo possível  $(\theta, \phi)$ , para cada comprimento de onda  $\lambda$ , em cada tempo  $t$ :  $(V_x, V_y, V_z, \theta, \phi, \lambda, t)$ . Em outras palavras, a *função plenóptica*

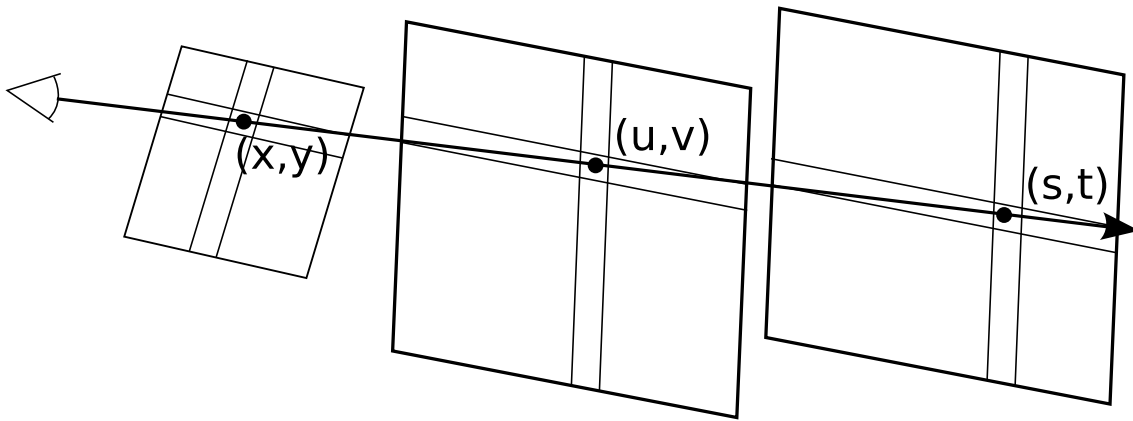


Figura 2.1: O valor do pixel  $(x, y)$ , intersectado pelo raio partindo do observador, é computado utilizando as parametrizações dos planos  $(u, v)$  e  $(s, t)$ .

7D define um conjunto de todos os possíveis mapas de ambiente para uma dada cena [27].

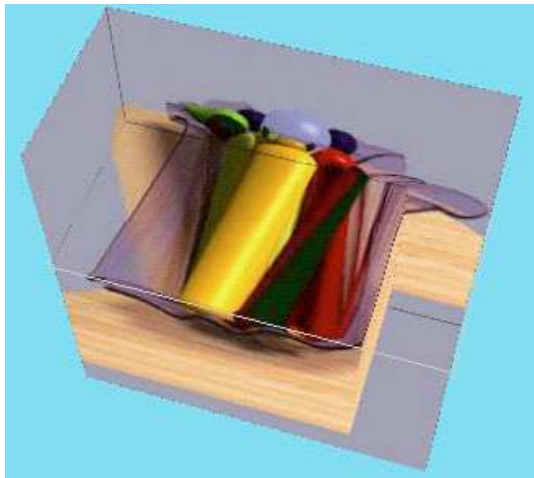
A *modelagem plenóptica* [26], por sua vez, reduz a função para cinco dimensões ao remover  $t$  e  $\lambda$ , ou seja, ao limitar o problema para ambientes estáticos. Os autores colocam que toda abordagem de renderização baseada em imagem pode ser interpretada como uma tentativa de reconstruir a *função plenóptica* a partir de um conjunto de amostras.

Por sua vez, os trabalhos *Light Field* e *Lumigraph* reduziram ainda mais a complexidade da função. Ao constatar que, enquanto a posição do observador estiver fora do fecho convexo do objeto, a função pode ser representada em quatro dimensões:

$$P_4(u, v, s, t), \quad (2.1)$$

onde  $(u, v)$  e  $(s, t)$  são parâmetros de dois planos. No caso do *Light Field*, por exemplo, um plano é o da camera e o outro o da cena (plano focal). Desta forma, a função 4D é na realidade uma parametrização por coordenadas de interseção de raios passando pelos dois planos. Para gerar uma imagem, a função é amostrada utilizando os raios de visão do observador passando pelo plano da imagem. A Figura 2.1 ilustra o raio partindo do observador e atravessando os planos. Uma fatia  $(s, u, v)$  do *Lumigraph* é ilustrada na Figura 2.2a.

Pode-se compreender esses algoritmos como uma amostragem de todos os possíveis raios de luz passando pela cena e intersectando os planos de referência. Renderizar uma cena a partir de um novo ponto de vista, significa extrair uma fatia 2D do



(a)



(b)

Figura 2.2: (a) uma fatia  $(s, u, v)$  do modelo de uma fruteira e (b) o mesmo modelo renderizado, ambos gerados pelo Lumigraph. (Retiradas de [1])

volume 4D. Para tanto, devem ser computadas as coordenadas  $(u, v, s, t)$  para cada raio lançado do observador atravessando um pixel  $(x, y)$  no plano da imagem. Uma maneira simples de realizar esta tarefa é utilizar um algoritmo de traçado de raios, onde é verificada a interseção de cada raio com os planos. Porém, uma maneira mais eficiente é projetar os planos no espaço de imagem como quadriláteros, para que o hardware de rasterização faça o trabalho de interpolar as coordenadas no interior. Num segundo passo, uma reamostragem da radiância é realizada utilizando interpolação linear, bilinear ou quadrilinear. Exemplos de renderizações podem ser vistos nas Figuras 2.2b (Lumigraph) e 2.3 (Light Field).

Na realidade, o *Lumigraph* poderia estar situado na classe de algoritmos descrita a seguir, pois, diferentemente do *Light Field*, utiliza uma aproximação da geometria para guiar o teste de profundidade. Porém, como a base do algoritmo está inteiramente na imagem, ele foi classificado como pertencendo à classe sem geometria.

### 2.1.2 Renderização com Geometria Explícita

Diferentemente dos algoritmos anteriores, algumas técnicas armazenam informações tridimensionais da imagem. Um exemplo é a técnica de *3D Warping*, onde são geradas várias imagens de referência de uma cena contendo um valor de profundidade



(a)



(b)

Figura 2.3: Duas imagens geradas pelo Light Field. (Retiradas de [2])

por pixel. Para renderizar a cena a partir de um novo ponto de vista, os pixels de todas as imagens são reprojados no espaço tridimensional, e depois projetados no plano da nova imagem. Uma desvantagem desta técnica é que nem sempre é possível cobrir toda superfície no novo ponto de vista, resultando em buracos na visualização. Esta limitação pode ser causada por dois motivos: primeiro, pela diferença entre a resolução da amostragem e a resolução da imagem final; segundo, pela possibilidade de algumas regiões não terem sido capturadas nas imagens de referência.

Em sequência, Shade et al. [28] apresentaram Imagens com Camadas de Profundidade (*Layered Depth Images* ou *LDI*). Nesta abordagem, em contraste com o *3D Warping*, é proposto um armazenamento menos dispendioso dos dados ao se juntar todas as imagens de referência em um único centro de projeção. Para lidar com o problema de oclusão, cada pixel da referência contém uma lista de profundidades e cores, onde o raio passando pelo pixel intersecta a cena. Para renderizar a partir de um ponto de vista diferente, os pixels da LDI são projetados para a nova referência. Todavia, por causa da resolução fixa da LDI, a densidade de amostragem não é adequada para todos os pontos de vista.

Para superar esta desvantagem, Chang et al. [3] propuseram *Árvores de LDIs*, combinando uma partição hierárquica do espaço com as LDIs. Nesta abordagem, uma *octree* é utilizada onde cada nó armazena uma LDI. Cada pixel de uma imagem de referência é adaptativamente colocado em um nó da árvore e, depois, filtrado para todos nós ancestrais. Esta representação oferece a vantagem das amostras filtradas

poderem ser utilizadas para diminuir o tempo de renderização, por exemplo, ao escolher um nível da árvore de acordo com a distância ao observador.

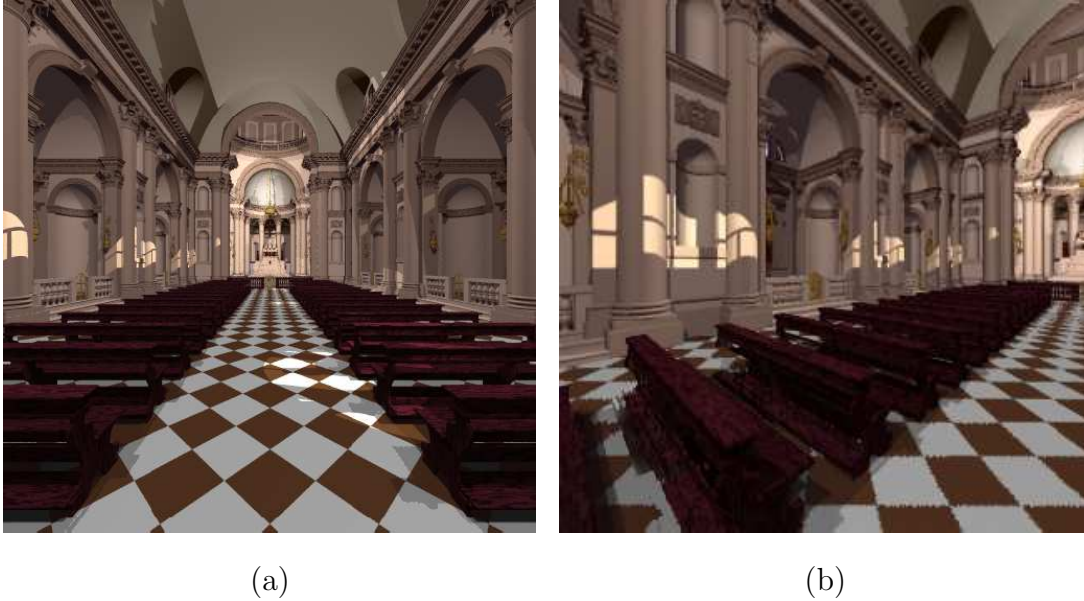


Figura 2.4: (a) uma imagem de referência e (b) uma imagem gerada com uma Árvore LDI a partir de 36 referências. (Retiradas de [3])

## 2.2 Renderização Baseada em Pontos

Os primeiros algoritmos para renderizar pontos utilizaram amplamente técnicas baseadas em imagem. Nestas propostas, o modelo é reamostrado para gerar imagens de referências, permitindo o uso de técnicas de renderização com geometria implícita. A evolução da área levou a algoritmos que renderizam diretamente a nuvem de pontos, porém, perdendo a característica de possuir uma complexidade relativa à resolução da imagem.

### 2.2.1 Visualizando Pontos

A proposta de utilizar pontos na área de visualização não é recente, em 1985 Levoy e Whitted [16] propuseram o seu uso como primitiva gráfica. De fato, o uso de pontos em geral é ainda mais antigo, já que os próprios autores citaram em 1985: “O uso de pontos não é uma novidade”, colocando como exemplo a modelagem de fumaça, fogo e árvores. De uma certa maneira, eles anteciparam uma das razões



pelas quais pontos ganharam interesse: o crescimento contínuo da complexidade visual das cenas. A proposta foi separar a primitiva de modelagem do *pipeline* gráfico introduzindo pontos como *meta-primitivas*. Ao converter qualquer tipo de primitiva para a representação por pontos, um algoritmo padrão de renderização poderia ser firmado.

Contudo, foi somente mais de uma década depois que esta pesquisa obteve continuidade. Em 1998 Grossman e Dally [4] introduziram um algoritmo baseado em espaço de imagem para conjuntos de pontos. Da mesma forma que o *3D Warping*, esta técnica trabalha com várias imagens de referência do objeto, mas, neste caso, os dados armazenados contêm profundidade e a normal da superfície, além de propriedades de cor. Para obter as imagens de entrada, a malha poligonal do modelo é amostrada a partir de 32 vistas ortogonais. Porém, como colocado por Grossman e Dally, as amostras com normais são independentes do ponto de vista, diferentemente das imagens de referência comumente utilizadas em renderização baseada em imagem (RBI). Desta forma, é possível eliminar consideravelmente a redundância dos dados e, por conseguinte, reduzir o espaço de armazenamento. Outra vantagem sobre os métodos até então publicados de RBI é a possibilidade de computar iluminação dinâmica. Para renderizar o objeto a partir de um novo ponto de vista, as amostras são projetadas para o espaço de imagem. Porém, sob magnificação, por exemplo, a resolução das amostras pode não ser suficiente para reconstruir uma superfície sem buracos. Para contornar esta limitação, é gerada uma pirâmide de imagens com as profundidades das amostras projetadas, onde um algoritmo do tipo *pull-push* [29] é utilizado para lidar com o preenchimento dos buracos (Figura 2.5). Ao incluir primitivas geométricas amostradas em partições regulares, a proposta leva traços da principal vantagem dos métodos de RBI, à vista que sua complexidade é quase independente do número de amostras. Duas imagens renderizadas com este algoritmo podem ser observadas na Figura 2.6.

Dois anos depois, Pfister et al. [5] introduziram o termo *Surfel*, ou *Surface Element*, para descrever uma n-tupla de dimensão zero com atributos de forma e aparência. Um surfel é uma aproximação local da superfície de um objeto. Os autores posicionam a renderização de surfels entre os métodos de RBI e os métodos convencionais baseados em geometria. Assim como em Grossman e Dally, são utili-

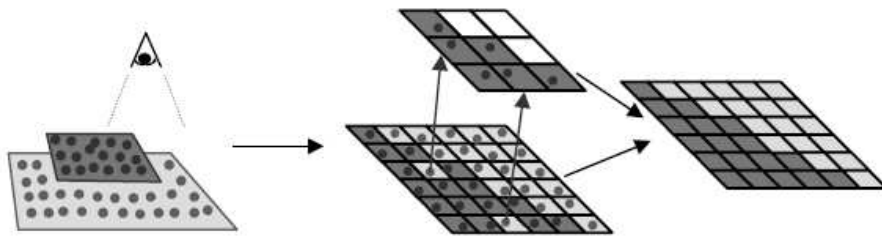


Figura 2.5: A superfície escura é renderizada utilizando um nível de menor resolução na pirâmide onde não há buracos. Esta hierarquia é utilizada para preencher os pixels vazios na imagem. (Retirada de [4])

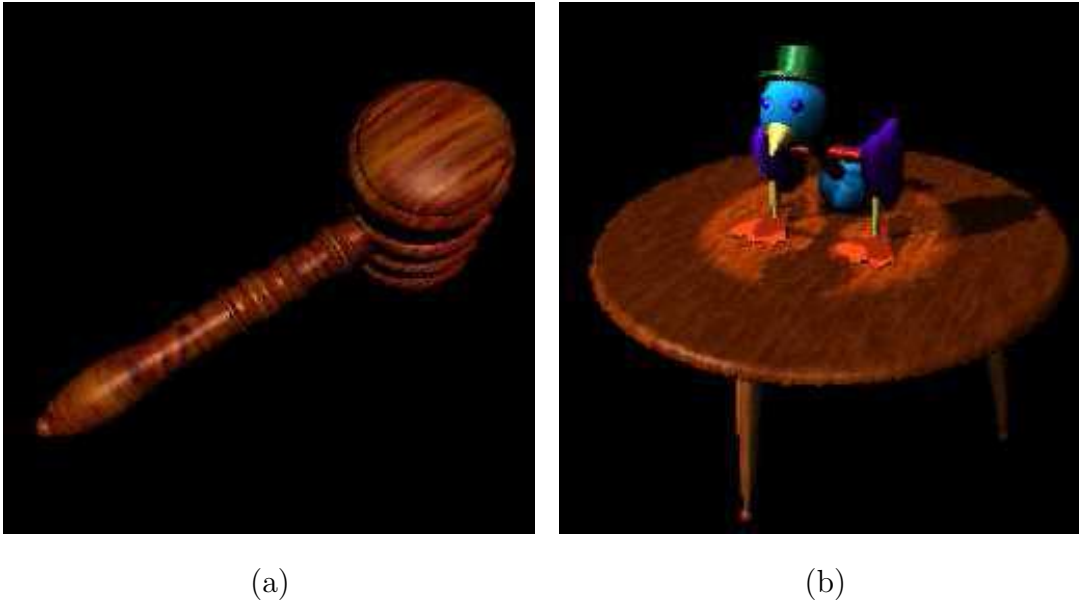


Figura 2.6: Duas imagens geradas pelo algoritmo de Grossman e Dally. (Retiradas de [4])

zadas técnicas de reconstrução baseada em imagem para renderizar os objetos. Uma hierarquia similar às árvores de LDI é criada, no entanto, uma LDC (*Layered Depth Cube*) é armazenada em cada nó ao invés de uma LDI. As LDCs são formadas por três LDIs ortogonais (três lados de um cubo). A *octree* é construída de baixo para cima, onde o nível mais baixo contém a LDC de maior resolução. Diferentemente do método anterior, o teste de visibilidade é realizado através da rasterização das projeções das aproximações locais, i.e., projeções elípticas do círculo. A Figura 2.7 ilustra dois exemplos renderizados com este algoritmo.

No mesmo ano, o algoritmo *QSplat* [6] foi desenvolvido, durante o Projeto Michelangelo Digital [30], para renderizar nuvens de pontos massivas. Este é o primeiro



(a)



(b)

Figura 2.7: Duas imagens geradas pelo algoritmo *Surfels*. (Retiradas de [5])

trabalho de renderização de pontos a fugir do paradigma de renderização baseada em imagem. É utilizado um método simples baseado no percorrimento de uma hierarquia de esferas envolventes. Uma representação compacta dos dados, seja em disco ou em memória, permite uma renderização progressiva do modelo. Para manter o desempenho interativo é utilizado um ajuste dinâmico de nível de detalhe. Os *splats* podem ser projetados como quadrados, círculos, elipses, ou funções Gaussianas. Apesar de não focar em técnicas para garantir imagens de alta qualidade, a reconstrução final, além de eficiente, não apresenta artefatos significativos por trabalhar apenas com nuvens densas de pontos (entre cem milhões e um bilhão de pontos). No entanto, o método foi desenvolvido especificamente para distribuições uniformes de pontos, ou pelo menos, quase uniformes. Um exemplo pode ser observado na Figura 2.8.

O primeiro trabalho a apresentar um método para renderizar diretamente nuvens de pontos não uniformes, sem a necessidade de estruturas de dados auxiliares, foi o *Surface Splatting* [31, 7, 32]. Assim como os *surfels*, as amostras representam aproximações da superfície, com informação de raio e normal. Porém, a proposta é projetar esta aproximação local no espaço de imagem utilizando filtros de alta qualidade. Para tanto, Zwicker et al. estenderam a metodologia de reamostragem de texturas, descrita por Heckbert [33], para formular Filtros Elípticos com Média Ponderada (EWA ou *Elliptical Weighted Average*) no espaço de imagem ao invés



Figura 2.8: Um detalhe do modelo digital da cabeça de Davi renderizado com o *QSplat*. (Retirada de [6])

de no espaço de objeto. Esses filtros representam a projeção da área de influência de cada amostra, geralmente representada por uma função Gaussiana. Os filtros, ou núcleos de reconstrução, são ponderados para reconstruir a superfície onde a contribuição de uma amostra projetada é computada para cada pixel coberto.

Com a metodologia do *Surface Splatting*, é possível renderizar pontos com texturas detalhadas e ter suporte a transparência, como ilustrado na Figura 2.9. Filtros *anti-aliasing* são incluídos para garantir uma reconstrução sem buracos e evitar a fase de reconstrução de imagens. Por um lado, esta nova metodologia acrescenta importantes inovações como os filtros de alta qualidade e a renderização direta de nuvens não uniformes; por outro lado, ela se distancia da principal vantagem dos métodos baseados em imagem: complexidade de reconstrução associada à resolução de saída, e não ao número de amostras.

A teoria de *Surface Splatting* foi estendida por Räsänen [13] e Zwicker et al. [34] para computar uma projeção perspectivamente correta dos *splats*, que até então era aproximada por uma projeção ortogonal. O contorno do *splat* é representado por uma seção cônica, evitando artefatos gerados nas implementações anteriores devido à aproximação da projeção. O algoritmo *Surface Splatting*, contendo a projeção perspectiva das seções cônicas, é descrito em maiores detalhes no Capítulo 3.



Figura 2.9: Imagem gerada com o algoritmo *Surface Splatting*. (Retirada de [7])

## 2.2.2 Adicionando Informação à Reconstrução

Informações a respeito da geometria diferencial local da superfície foram incluídas por Kalaiyah e Varshney [35, 8]. A extensão de uma amostra da superfície é determinada pela curvatura local. *Pontos Diferenciais* não só permitem reduzir a quantidade de amostras por aproximarem melhor a superfície, como resultam em renderizações de alta qualidade ao variar a normal no interior do *splat* (Figura 2.10). Para projetar as amostras na tela é utilizado um retalho retangular contendo um mapa de normais, simulando a curvatura local.

Ainda na mesma linha, Botsch et al. [9] introduziram o *Phong Splatting*, incluindo um campo de normal linearmente variante por *splat*. Assim como no *Pontos Diferenciais*, são necessárias menos amostras em comparação com o algoritmo tradicional de *splatting*. Contudo, a metodologia anterior amostrava superfícies NURBS suaves e contínuas, enquanto o *Phong Splatting* permite extrair as informações de curvatura diretamente de nuvens de pontos, e depois simplificá-las. Com o campo de normais é possível gerar um modelo de iluminação similar ao de *Phong*, comparado com as propostas de *Surface Splatting* anteriores, que geravam um modelo com-

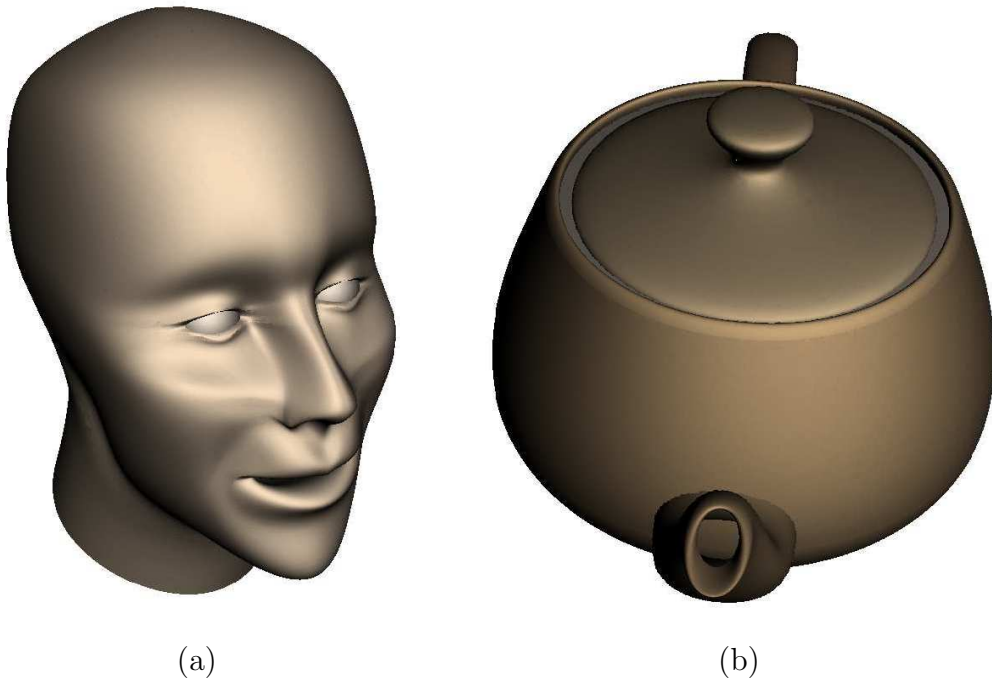


Figura 2.10: Imagens geradas com os *Pontos Diferenciais* de Kalaiyah e Varshney. (Retiradas de [8])

patível com o de *Gouraud*. Imagens ilustrativas do algoritmo podem ser observadas na Figura 2.11.

A idéia de *splats* curvos também foi utilizada por Luz et al. [10, 36] com os *c-surfels*. No entanto, ao invés de definir as curvaturas principais, os *splats* são fisicamente curvados formando conchas poligonais (Figura 2.12b). Esses retalhos poligonais adaptam melhor a curvatura local, especialmente ao longo das silhuetas. Cada *c-surfel* possui uma textura monocromática e opacidade decaindo de acordo com uma aproximação Gaussiana. No entanto, a técnica assume uma distribuição uniforme ao atribuir um raio constante para todas amostras. A Figura 2.12b ilustra uma renderização com os *c-surfels*.

### 2.2.3 Renderização na GPU

O uso de GPUs programáveis na área de Renderização Baseada em Pontos foi explorada por diversos autores nos últimos anos. Para mapear o algoritmo de *Surface Splatting* na GPU, o método foi primeiro revertido para o espaço de objeto por Ren et al. [37]. O algoritmo, assim como muitos subsequentes, consiste em duas passadas: na primeira é aplicado um teste de visibilidade, enquanto que na segunda é

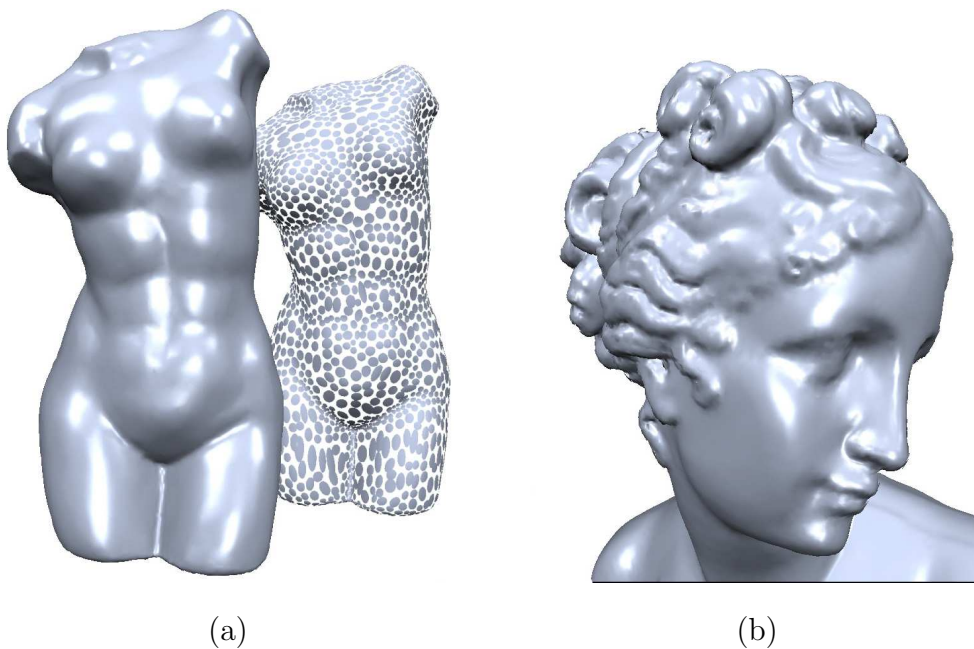


Figura 2.11: Imagens geradas com os *Phong Splatting* de Botsch et al. (Retiradas de [9])

realizada a filtragem EWA. No entanto, os *splats* são projetados como retângulos texturizados, aumentando o fluxo de vértices em quatro vezes, e, conseqüentemente, diminuindo o desempenho.

Coconu et al. [38] propuseram uma implementação com apenas uma passada. Porém, para evitar a primeira, relativa ao teste de visibilidade, as amostras são projetadas ordenadamente de trás para frente, necessitando de estrutura de dados auxiliares. Ainda mais, o uso de filtros como o EWA não é possível, resultando em artefatos visíveis.

Botsch e Kobbelt [39, 11] apresentaram uma implementação mais fiel ao EWA *Surface Splatting* em espaço de imagem, capaz de gerar *splats* de alta qualidade usando uma abordagem em dois passos novamente (Figura 2.13a). Guennebaud et al. [12] apresentaram melhorias ao algoritmo e também incorporaram transparência na renderização (Figura 2.13b).

Para superar a desvantagem da especialização do hardware em renderizar triângulos, Weyrich et al. [40] propuseram uma arquitetura dedica à renderização de pontos. O mecanismo evita multi-passadas e inclui um teste de profundidade ternário para compor os *splats*. O seu desempenho máximo de 17.5M *splats* por segundo ainda é comparável com os algoritmos em GPU. No entanto, como o protótipo não foi



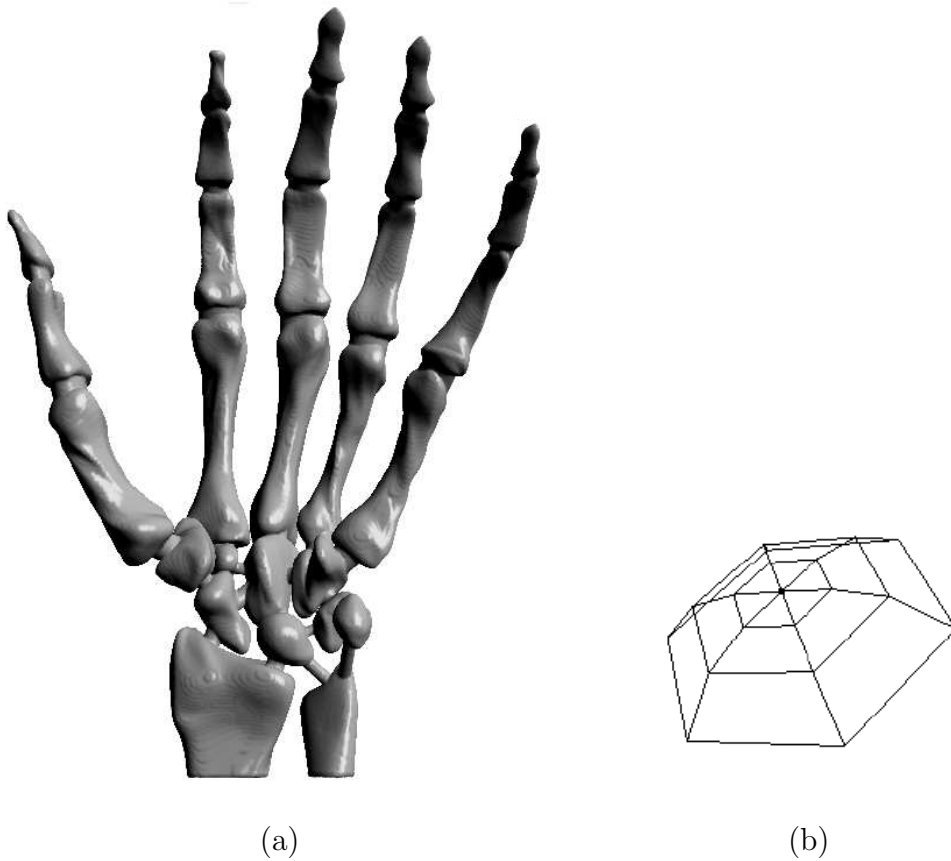


Figura 2.12: (a) imagem gerada com os *c-surfels* de Luz et al. e (b) a representação poligonal de um *c-surfel*. (Retiradas de [10])

montado utilizando materiais de ponta, os autores esperam chegar a 200M *splats* por segundo utilizando componentes mais modernos.

## 2.2.4 Alternativas ao Splatting

Finalmente, como alternativas para os métodos de *Splatting*, alguns trabalhos sobre traçado de raios para nuvens de pontos foram propostos [41, 42, 43]. Porém, para computar a interseção do raio com o modelo de forma eficiente, é preciso extrair a superfície da nuvens de pontos. Uma maneira é aproximar a superfícies polinomiais. Contudo, essas técnicas se distanciam um pouco do escopo deste trabalho, e não serão descritas em mais detalhes.



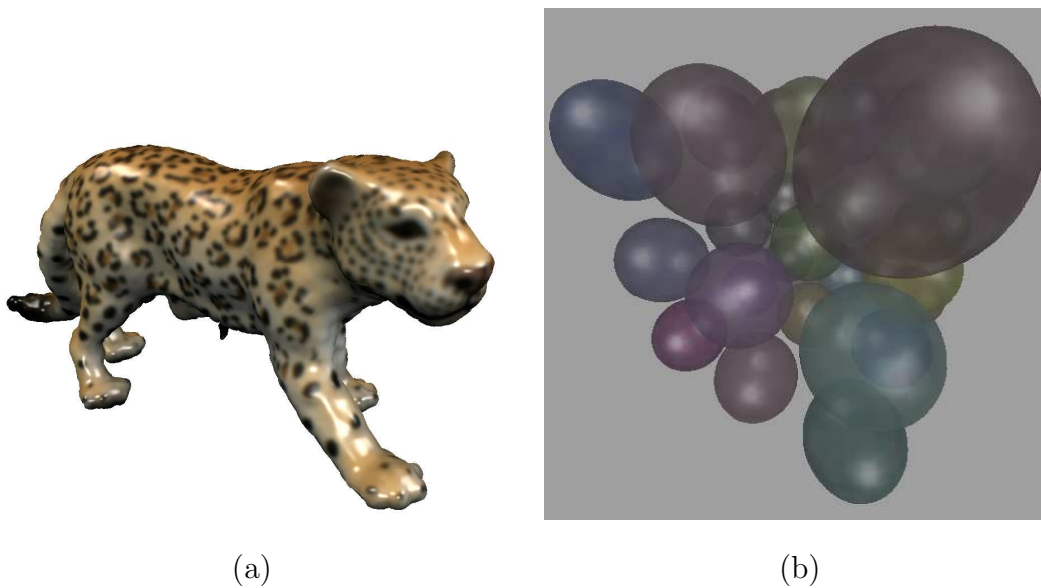


Figura 2.13: Imagens geradas pelos algoritmo de (a) Botsch e Kobbelt e (b) pelo algoritmo de Guennebaud et al. (Retiradas de [11] e [12], respectivamente)

## 2.3 Discussão

Os primeiros algoritmos classificados como Renderização Baseada em Pontos, podem ser vistos, na verdade, como uma transição das técnicas baseadas em imagem para baseadas em pontos. Naquele momento o objetivo não era especificamente renderizar nuvens de pontos oriundas de dados digitalizados, mas sim, amostrar modelos como estruturas regulares de pontos para que fosse possível transportar algumas etapas da visualização para um estágio de pré-processamento. De fato, no trabalho *Surfels*, Pfister et al. [5] colocam como objetivo superar algumas limitações dos algoritmos de RBI, como por exemplo, possibilitar iluminação dinâmica.

No entanto, com o avanço da tecnologia dos equipamentos de digitalização 3D, junto ao aumento da complexidade da cena, surgiu uma nova preocupação em lidar diretamente com a representação por pontos, a fim de evitar processos custosos de reconstrução (transformar em malhas poligonais) e manutenção. Neste momento a área de gráficos baseados em pontos se afastou um pouco das técnicas baseadas em imagem, pois o objetivo principal não mais era obter independência da complexidade da cena, mas sim renderizar interativamente e diretamente nuvens de pontos.

De uma certa forma, o propósito desta tese resgata esta ligação entre os algoritmos baseados em pontos e aqueles baseados em imagem. Ao mesmo tempo que

desacopla a fase de reconstrução do modelo de sua complexidade, também cumpre com o objetivo de renderizar diretamente nuvens de pontos eficientemente.

# Capítulo 3

## Surface Splatting

Atualmente, *Surface Splatting* é a técnica mais popular para renderização direta de nuvens de pontos não uniformes. A entrada de dados é um conjunto não estruturado de amostras  $S_i$  adimensionais, e a saída uma representação contínua da superfície no espaço de imagem para um determinado ponto de vista. Na próxima sessão é apresentada a teoria de projeção dos filtros EWA em espaço de imagem, enquanto que na Sessão 3.2 é descrita uma implementação prática do algoritmo.

### 3.1 Teoria do Surface Splatting

Uma amostra  $P_k$  contém atributos espaciais e de aparência, e pode ser expressa pela n-tupla  $\{p_x, p_y, p_z, n_x, n_y, n_z, w_r, w_g, w_b\}$ . Os primeiros três elementos definem a posição espacial, os três seguintes a normal da superfície naquele ponto, e os últimos três os atributos de cor (RGB).

Ao renderizar uma nuvem de pontos de forma simplória, onde cada amostra é projetada em um pixel, normalmente a densidade não é suficiente para evitar buracos na imagem final. Para preencher os buracos entre as amostras projetadas a superfície é localmente reconstruída no espaço de imagem. Para tanto, uma parametrização local da superfície a ser representada, em torno de cada amostra no espaço de objeto, é projetada para o espaço de imagem, onde são compostas para computar o valor de cada pixel e, desta forma, reconstruir a superfície.

Geralmente, a aproximação local, ou núcleo de reconstrução, é definida como um retalho circular ou elíptico da superfície. No entanto, projetar essas primitivas

para o espaço de imagem, seguindo o *pipeline* gráfico tradicional para triângulos, é um processo custoso e proibitivo quando se quer alcançar taxas interativas. A solução proposta pelo algoritmo de *Surface Splatting* é computar a contribuição do núcleo de reconstrução diretamente no espaço de imagem. Este núcleo é associado a um filtro *anti-aliasing* para formar um filtro de reamostragem centralizado na projeção da amostra. O filtro de reamostragem é rasterizado e sua contribuição por pixel acumulada na matriz da imagem. Em uma segunda passada, a superfície é reconstruída ponderando as contribuições armazenadas em cada pixel.

O *Surface Splatting* é essencialmente um mecanismo de reconstrução de superfície, que assim como diversos problemas na computação gráfica envolve a conversão de sinais contínuos para discretos e vice-versa. O processo completo de reamostragem possui como entrada um sinal discreto, que por sua vez é reconstruído e finalmente amostrado novamente para gerar uma saída discreta. No caso de renderização de modelos de pontos, a entrada é uma nuvem de pontos que é reconstruída como uma superfície contínua e então discretizada como pixels.

### 3.1.1 Parametrização Local

Uma aproximação local da superfície é construída através de uma parametrização no entorno de uma amostra. Um sistema de coordenadas locais é definido pelo plano tangente de uma amostra e seu vetor normal, como ilustrado na Figura 3.1a. Desta forma, a superfície é aproximada por uma função contínua reconstruída a partir da representação discreta. Esta função pode ser avaliada em uma posição  $u$  arbitrária, em coordenadas locais, por uma média ponderada de todas as posições das amostras  $u_k$ , também em coordenadas locais:

$$f_c(u) = (f \otimes r)(u) = \sum_{k \in \mathbb{N}} f(u_k) r(u - u_k), \quad (3.1)$$

onde  $\otimes$  é um operador de convolução, e  $f(u_k)$  é a cor da amostra  $k$ . Por último, define-se o núcleo de reconstrução local  $r(u)$  por funções de base com suporte local, isto é,  $f_c(u)$  só é avaliada em uma pequena vizinhança em volta de  $u$ , como esclarecido na Figura 3.1b.

Para gerar uma reconstrução contínua da superfície, todas as aproximações locais são somadas. Contudo, como cada uma é definida em um sistema de coordenadas

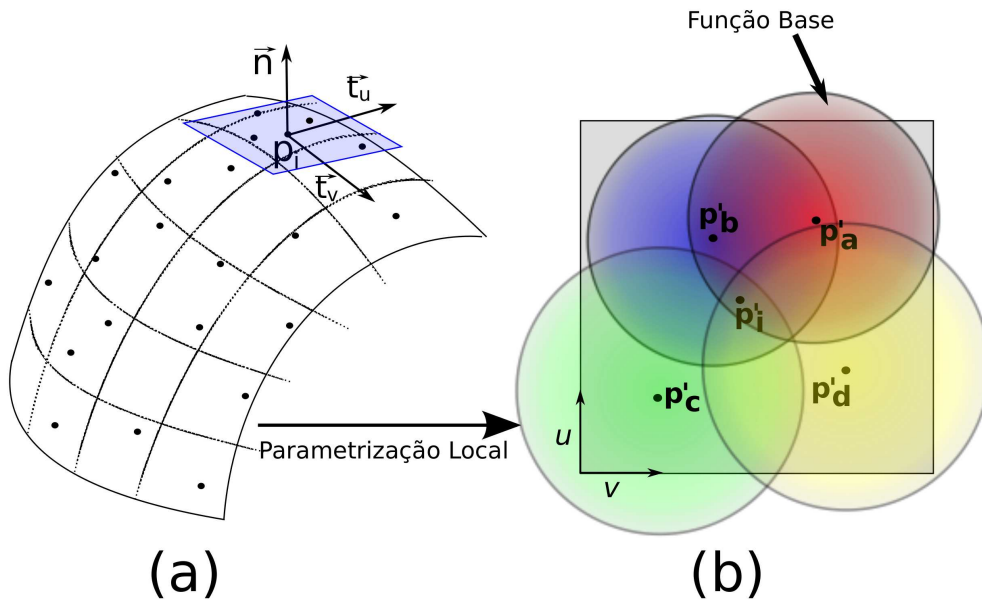


Figura 3.1: Na esquerda, o retalho colocado sobre a amostra no espaço de objeto. Na direita, o sistema de coordenadas locais em volta de um ponto  $p'_i$ , com a vizinhança  $\{p'_a, p'_b, p'_c, \dots\}$  definida neste sistema.

diferente, elas devem ser mapeadas para um espaço comum. Isto é alcançado projetando todos núcleos de reconstrução para o espaço de imagem e, depois, compondo-os por meio de médias ponderadas.

### 3.1.2 Equação de Renderização

Para renderizar uma amostra projetada, ou *splat*, seu núcleo de reconstrução é primeiro projetado para o espaço de imagem:

$$g_c(x) = f_c(m^{-1}(x)), \quad (3.2)$$

onde  $x = m(u) : \mathbb{R}^2 \rightarrow \mathbb{R}^2$  define um mapeamento do espaço local para o espaço de imagem. Neste domínio a função deve ser limitada, geralmente utilizando um filtro passa-baixa, para evitar efeitos de *aliasing*. Um pré-filtro  $h$  é utilizado resultando em:

$$g'_c(x) = (g_c \otimes h)(x) = \int_{\mathbb{R}^2} g_c(t)h(x - t)dt. \quad (3.3)$$

A função contínua e limitada é então amostrada multiplicando-a por uma função

pende  $i_x$ , gerando uma representação discreta:

$$g(x) = g'_c(x)i(x) = \begin{cases} g'_c(x) & \text{se } x \in X, \\ 0 & \text{em caso contrário,} \end{cases} \quad (3.4)$$

onde  $X$  é o conjunto de posições das amostras de saída, i.e., pixels.

Substituindo  $f_c$  e  $c_c$  por suas expressões nas Equações 3.1 e 3.2 pode-se escrever a Equação 3.3 da seguinte forma:

$$\begin{aligned} g(x) &= g'_c(x) \\ &= \int_{\mathbb{R}^2} g_c(t)h(x-t)dt \\ &= \int_{\mathbb{R}^2} f_c(m^{-1}(t))h(x-t)dt \\ &= \int_{\mathbb{R}^2} \sum_{k \in \mathbb{N}} f(u_k)r(m^{-1}(t) - u_k)h(x-t)dt \\ &= \sum_{k \in \mathbb{N}} f(u_k)\rho_k(x), \end{aligned} \quad (3.5)$$

onde

$$\rho_k(x) = \int_{\mathbb{R}^2} r(m^{-1}(t) - u_k)h(x-t)dt. \quad (3.6)$$

A Equação 3.5 revela que o valor final de um pixel, posicionado em  $x$ , é a soma de todas as amostras  $f(u_k)$  ponderadas pelo filtro de reamostragem  $\rho_k(x)$ . Ainda mais, é possível primeiro realizar a projeção de cada filtro separadamente, antes de somar as contribuições no espaço de imagem. Zwicker et al. [31] nomearam esta abordagem de *Surface Splatting*.

### 3.1.3 Filtros Gaussianos

Os filtros Gaussianos são uma escolha popular para modelar o pré-filtro  $h$  e o filtro de reconstrução  $r$  [31, 33]. Apesar de ser um filtro não-ideal, o seu suporte compacto permite um truncamento limitando significativamente os efeitos de *aliasing*. A seguir algumas propriedades das funções Gaussianas são definidas.

Uma função Gaussiana centrada na origem é determinada por:

$$g_\sigma(x) = \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{1}{2}\frac{x^2}{\sigma^2}}, \quad (3.7)$$

onde  $\sigma$  é o desvio padrão e  $\sigma^2$  a variância.

A integral de  $g_\sigma$  na reta é 1 e quando se aplica a ela uma transformada de Fourier, o espectro é outra Gaussiana [44]:

$$g_\sigma(x) \rightarrow G_\sigma(\omega) = e^{-\frac{\sigma^2 \omega^2}{2}} = \frac{\sqrt{2\pi}}{\sigma} \cdot g_{\frac{1}{\sigma}}(\omega). \quad (3.8)$$

Analogamente, uma Gaussiana circular em duas dimensões com variância  $\sigma^2$  é definida por:

$$g_I(u) = \frac{1}{2\pi} e^{-\frac{x \cdot x^T}{2\sigma^2}}, \quad (3.9)$$

com  $x = \{x_x, x_y\}$ .

Como o interesse é utilizar a Gaussiana  $g_I(u)$  como filtro de projeção, é preciso considerar a transformação  $u = xM^{-1}$ :

$$|M^{-1}|g_I(xM^{-1}) = \frac{|M^{-1}|}{2\pi} e^{-\frac{xM^{-1}(xM^{-1})^T}{2\sigma^2}}, \quad (3.10)$$

onde o fator de normalização  $|M^{-1}|$  garante que a integral da função, depois da transformação, se mantém unitária.

Uma Gaussiana bidimensional:

$$\frac{1}{2\pi|M|} e^{-\frac{1}{2}x(M^T M)^{-1}x^T} \quad (3.11)$$

é definida por uma matriz de covariância:

$$V = \begin{bmatrix} A & B \\ B & C \end{bmatrix} = \frac{1}{\sigma^2}(M^T M). \quad (3.12)$$

Observe que  $V$  é uma matriz simétrica não-singular, cujo determinante é positivo dado que:

$$|V| = AC - B^2 = \frac{1}{\sigma^4}|M||M^T| = \frac{|M|^2}{\sigma^4} > 0. \quad (3.13)$$

Além disso,  $A = \frac{\|M_1\|^2}{\sigma^4}$  e  $C = \frac{\|M_2\|^2}{\sigma^4}$  são positivos ( $M_i, i = 1, 2$  é a  $i^a$  coluna de  $M$ ). Essas três condições são necessárias e suficientes para que  $V$  possa ser considerada como uma matriz de co-variância.

Assim a passagem de uma Gaussiana de um domínio para outro resulta em outra Gaussiana. Porém, quando a função projetada é discretizada por reamostragem, é preciso escolher  $\sigma$  de forma a resolver adequadamente o compromisso entre evitar artefatos de *aliasing* e o borramento excessivo.

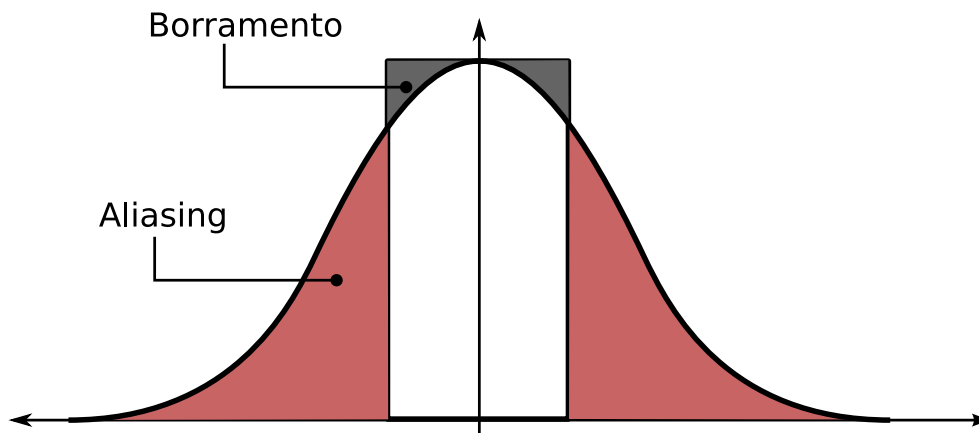


Figura 3.2: Uma Gaussiana comparada com um filtro caixa (filtro passa-baixa ideal), ambos no espaço de frequência. A amplitude da Gaussiana dita o balanço entre *aliasing* e borrarmento.

Ainda mais, acrescenta-se que pelo fato das Gaussianas serem FRI (*filtros de resposta infinita*), elas devem ser substituídas por aproximações de suporte compacto. A dimensão deste suporte também precisa ser levada em conta, tendo em vista o compromisso mencionado acima.

O borrarmento é introduzido ao se cortar as altas-frequências para eliminar o efeito de *aliasing*. Mais precisamente, o borrarmento é caracterizado pelas frequências altas que foram indevidamente cortadas pelo filtro passa-baixa, o *aliasing* é identificado como as frequências altas que passaram indevidamente pela faixa de corte do filtro.

Uma propriedade da transformada de Fourier expressa na Equação 3.8 é que quanto mais estreito o gráfico de um filtro Gaussiano no espaço de domínio, mais largo ele será no espaço de frequência. Filtros estreitos no espaço de domínio deixam passar a maior parte do sinal relativo às frequências altas, resultando em pouco borrarmento, porém, muito *aliasing*; em contrapartida, filtros largos resultam no efeito contrário. Normalmente, valores entre  $1.0 < \sigma < 2.0$  são usados para o filtro de reconstrução e o pré-filtro [32]. Uma Gaussiana no espaço de frequência é comparada a um filtro ideal *caixa* na Figura 3.2.



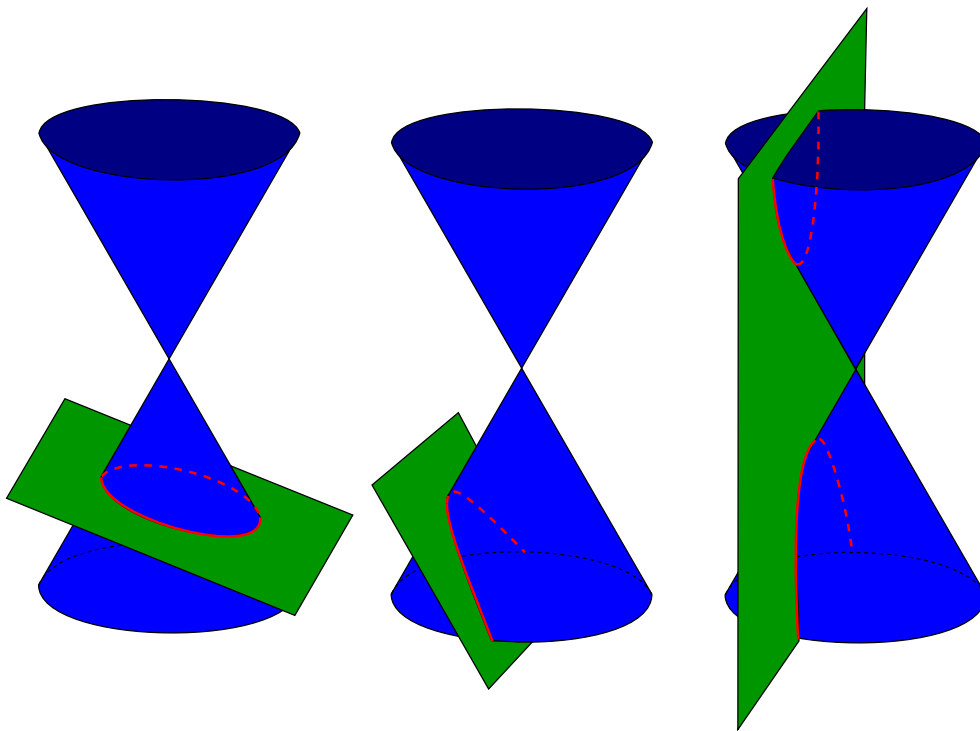


Figura 3.3: Os três possíveis casos não degenerados das cônicas: elipse, parábola e hipérbole.

### 3.1.4 Sessões Cônicas

Como mencionado no capítulo anterior, Räsänen [13] e Zwicker et al. [34] introduziram um método para a projeção perspectiva correta dos splats. Para tanto, utilizaram cônicas para mapear as funções no espaço de imagem. Nesta seção serão descritas algumas propriedades das cônicas, seguindo o mesmo raciocínio das funções Gaussianas. Como poderá ser observado, existe uma relação direta entre as Gaussianas e as cônicas, possibilitando esta proposta.

Uma seção cônica é uma curva planar obtida pela interseção de um plano com um cone. Descartando os casos degenerados, é possível obter três tipos de resultados: elipses, parábolas ou hipérbolas. Estes casos são ilustrados na Figura 3.3. A equação implícita da cônica é definida por:

$$f(x, y) = Ax^2 + 2Bxy + Cy^2 + Dx + Ey - F = 0, \quad (3.14)$$

com pelo menos um valor diferente de zero entre  $A$ ,  $B$  e  $C$ . O tipo de interseção pode ser facilmente avaliado ao analisar  $\Delta = 4AC - B^2$ :

- $\Delta > 0 \rightarrow$  elipse;

- $\Delta = 0 \rightarrow$  parábola;
- $\Delta < 0 \rightarrow$  hipérbole.

Por motivos de simplificação, os casos degenerados não estão sendo levados em consideração. Eles ocorrem quando o plano de interseção cruza o vértice do cone, resultando em uma linha ou um ponto.

A equação da cônica pode ser simplificada considerando que seu centro está na origem, isto é,  $D = E = 0$ :

$$Ax^2 + Bxy + Cy^2 = F. \quad (3.15)$$

Esta equação pode ser escrita em forma de matriz  $\bar{x}Q\bar{x}^T = F$ , onde  $Q$  é matriz da cônica:

$$Q = \begin{bmatrix} A & B \\ B & C \end{bmatrix}. \quad (3.16)$$

Assumindo que  $A > 0$ , então se  $\Delta = 4AC - B^2 > 0$ ,  $Q$  se torna positiva definida e pode ser tomada como uma matriz de co-variância. A cônica associada é uma elipse e no caso especial onde  $A = C$ , ela assume a forma de um círculo.

Analogamente à seção anterior, pode-se analisar o mapeamento da cônica utilizando  $x = uM$  e  $u = xM^{-1}$ :

$$uQu^T = F \quad (3.17)$$

$$xM^{-1}Q(xM^{-1})^T = xM^{-1}QM^{-1T}x^T = F \quad (3.18)$$

$$xQ'x^T = F, \text{ onde} \quad (3.19)$$

$$Q' = M^{-1}QM^{-1T}. \quad (3.20)$$

Novamente, existe uma semelhança do que acontece com os filtros Gaussianos onde o mapeamento  $M$  leva uma cônica em outra cônica. Isto não é, obviamente, uma coincidência, pois os isocontornos de uma Gaussiana são cônicas:

$$g_v(x) = e^{-\frac{1}{2}xV^{-1}x^T} = K \Leftrightarrow xV^{-1}x^T = -2\ln(K). \quad (3.21)$$

O filtro de reconstrução  $V_r$  é projetado no espaço de imagem de uma forma similar à projeção de uma cônica:

$$J_{u_k}V_rJ_{u_k}^T = M^{-1}Q_rM^{-1T} = Q'_r, \quad (3.22)$$

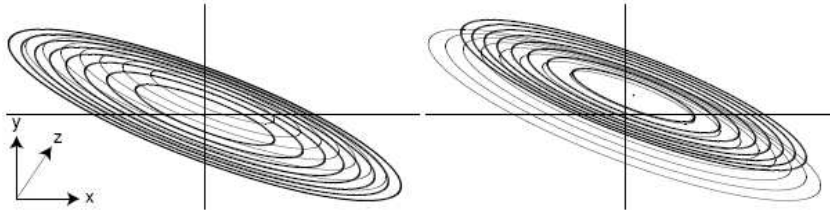


Figura 3.4: Na esquerda o isocontorno projetado corretamente como uma cônica, mas com centro deslocado. Na direita o centro projetado corretamente mas com isocontorno incorreto. (Retirada de [13])

onde  $J_{u_k}$  é o Jacobiano de  $M^{-1}$ , que no caso de uma projeção é a própria matriz  $M^{-1}$ .

Seguindo os passos do *Surface Splatting*, depois da transformação para o espaço de imagem, um filtro *anti-aliasing* é aplicado. Como o filtro passa-baixa  $h(x)$  também é definido como um Gaussiana, pode-se pensar também em uma cônica  $Q_h$  como variância  $V_h$ . Finalmente, o filtro de reamostragem é formado pela junção do filtro de reconstrução e do pré-filtro. O resultado é uma Gaussiana cuja variância é:

$$V_{\rho_k} = J_{u_k} V_r J_{u_k}^T + V_h = Q_{\rho_k}, \quad (3.23)$$

dado que a variância de uma convolução de funções Gaussianas é a soma das suas variâncias. Em vista disto os isocontornos desta Gaussiana composta serão também elipses associadas à matriz:

$$Q_{\rho_k} = M^{-1} Q_r M^{-1T} + Q_h. \quad (3.24)$$

Uma função Gaussiana elíptica possui infinitos isocontornos elípticos concêntricos. Por este motivo, para efeitos de rasterização,  $\rho_k$  é truncado e a cônica  $Q_{\rho_r}$  é associada ao seu isocontorno de corte, como detalhado na Seção 3.2.2.

A diferença entre projetar a função Gaussiana e a cônica pode ser observada na Figura 3.4. Enquanto que a projeção da cônica leva a renderização exata do isocontorno do filtro Gaussiano, seu centro é deslocado. Porém a projeção afim do filtro, apesar de ter centro correto, possui isocontornos diferentes.

### 3.1.5 Mapeamento 2D-2D Composto

Das últimas sessões, pode-se concluir que ao projetar um núcleo de reconstrução para cada amostra, todas contribuições podem ser somadas diretamente no espaço de imagem gerando uma superfície contínua e suave. Porém, como definir eficientemente  $\rho_k(x)$  no espaço de imagem?

Na metodologia de mapeamento de textura, a transformação completa de coordenadas envolve primeiro um mapeamento do espaço de textura 2D para o espaço de objeto 3D, e depois um segundo mapeamento do espaço de objeto 3D para o espaço de imagem 2D. O primeiro mapeamento é chamado de *parametrização* da superfície, enquanto que o segundo é a *projeção*. Felizmente, os dois podem ser combinados em um único mapeamento composto 2D-2D.

Assim como no mapeamento de texturas, no *Surface Splatting* as aproximações locais da superfície também são mapeadas de coordenadas 2D locais para coordenadas de tela 2D, através de um mapeamento composto  $x = m(u)$ . Este mapeamento inclui todas as seguintes transformações de sistemas de coordenadas:

- 2D-3D parametrização das coordenadas locais para espaço de objeto;
- 3D-3D transformação para o espaço de camera;
- 3D-2D projeção sobre o plano de imagem;
- 2D-2D transformação de *viewport*.

Como demonstrado na Seção 3.1.3, filtros Gaussianos são fechados sob mapeamento lineares. Entretanto, essa propriedade não se mantém para projeções em perspectiva, incluída na sequência acima. Sendo assim, não é garantido que a projeção de uma função Gaussiana venha a ser outra Gaussiana; em outras palavras, uma elipse não será necessariamente transformada em outra elipse. O tratamento de um *splat* parabólico ou hiperbólico não é cogitado devido a sua extensão não finita. Para evitar este problema,  $m(u)$  é aproximado por um mapeamento afim local:

$$m_{u_k}(u) = x_k + J_{u_k} \cdot (u - u_k), \quad (3.25)$$

onde  $x_k = m(u_k)$  define a posição na imagem da amostra projetada  $u_k$ . Definimos ainda  $J_{u_k}$  como sendo o Jacobiano de  $m(u)$  em  $u_k$ , i.e.,  $J_{u_k} = \frac{\partial m}{\partial u}(u_k)$ . Esta

aproximação é mais precisa na vizinhança de  $u_k$ : a matriz Jacobiana representa a melhor **aproximação linear** de uma função diferenciável na vizinhança de um ponto. Como é esperado que os modelos sejam razoavelmente densos, esta aproximação não deve produzir artefatos visuais no *Surface Splatting*.

Neste momento, a Equação 3.6 pode ser modificada para incluir a aproximação afim local  $m_{u_k}(u)$ :

$$\rho_k(x) = \int_{\mathbb{R}^2} r(m_{u_k}^{-1}(t) - u_k)h(x - t)dt. \quad (3.26)$$

A expressão  $m_{u_k}^{-1}(t)$  pode ser substituída por  $J_{u_k}^{-1} \cdot (t - m_{u_k}(u_k))$  usando a Equação 3.25, como demonstrado a seguir:

$$\begin{aligned} m_{u_k}(u) &= x_k + J_{u_k}(u - u_k) \text{ e } x_k = m_{u_k}(u_k), \\ m_{u_k}(u) &= m_{u_k}(u_k) + J_{u_k}(u - u_k), \\ J_{u_k}(u - u_k) &= m_{u_k}(u) - m_{u_k}(u_k), \\ (u - u_k) &= J_{u_k}^{-1}(m_{u_k}(u) - m_{u_k}(u_k)). \end{aligned} \quad (3.27)$$

Fazendo então  $u = m_{u_k}^{-1}(t)$  tem-se finalmente que:

$$(m_{u_k}^{-1}(t) - u_k) = J_{u_k}^{-1}(t - m_{u_k}(u_k)). \quad (3.28)$$

Substituindo a Equação 3.28 na Equação 3.26:

$$\begin{aligned} \rho_k(x) &= \int_{\mathbb{R}^2} r(J_{u_k}^{-1} \cdot (t - m_{u_k}(u_k)))h(x - t)dt, \\ &\quad \text{substituindo } r'_k(x) = r_k(J_{u_k}^{-1} \cdot x), \\ \rho_k(x) &= \int_{\mathbb{R}^2} r'(t - m_{u_k}(u_k))h(x - t)dt, \\ &\quad \text{e substituindo } \tau = t - m_{u_k}(u_k), \\ \rho_k(x) &= \int_{\mathbb{R}^2} r'_k(\tau)h(x - m_{u_k}(u_k) - \tau)d\tau = (r'_k \otimes h)(x - m_{u_k}(u_k)). \end{aligned} \quad (3.29)$$

Esta equação indica que feita a aproximação local indicada na Equação 3.25, o filtro de reamostragem no espaço de imagem é uma convolução entre o pré-filtro  $h$  com o núcleo de reconstrução  $r_k$ .

## 3.2 Implementando um Algoritmo de Surface Splatting Interativo

Resumidamente, o algoritmo consiste em projetar funções Gaussianas truncadas para o espaço de imagem, já no espaço de imagem limitar a banda utilizando um filtro passa-baixa e acumular as contribuições em uma matriz de pixels. Os detalhes de implementação desses três passos são descritos a seguir.

### 3.2.1 Núcleo de Reconstrução

O núcleo de reconstrução em coordenadas locais é determinado por uma função base como descrito na Seção 3.1.1. Esta função é definida como uma Gaussiana associada à matriz de co-variância  $V_{r_k}$ . A matriz de co-variância deve refletir a densidade de amostragem em volta do ponto. Para distribuições não uniformes, a Gaussiana é dimensionada por um fator  $r$  associado à densidade local:

$$V_{r_k}^{-1} = \begin{bmatrix} \frac{1}{r^2} & 0 \\ 0 & \frac{1}{r^2} \end{bmatrix} \quad (3.30)$$

A escolha de  $r$  como uma função da distância ao  $k$ -vizinho mais distante é uma boa estimativa para a maioria das aplicações, onde  $k = 10$  geralmente rende uma boa aproximação. No entanto, como esta estimativa leva em consideração apenas a distribuição local, deve-se assumir que a superfície foi corretamente amostrada.

Para o pré-filtro é utilizada uma Gaussiana associada à matriz de co-variância  $V_h = S^2$ , onde  $S$  é o fator de escala determinado pelas dimensões da tela:

$$S^{-1} = \begin{bmatrix} \frac{1}{largura} & 0 \\ 0 & \frac{1}{altura} \end{bmatrix} \quad (3.31)$$

Para projetar a função base no espaço de imagem, é utilizada a aproximação afim descrita na Seção 3.1.5. Primeiro, a matriz cônica  $Q = V_{r_k}$  é transformada em

coordenadas homogêneas:

$$Q_h = \begin{bmatrix} Q_{00} & Q_{10} & 0 & 0 \\ Q_{01} & Q_{11} & 0 & 0 \\ 0 & 0 & -F & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad (3.32)$$

A projeção da cônica para o espaço de imagem é então computada da seguinte forma:

$$Q_{h_s} = M^{-1}Q_hM^{-1T}, \quad (3.33)$$

onde a matriz  $M$  é definida pelos vetores tangentes  $\{t_u, t_v\}$  e centro  $p$  projetados no espaço de imagem:

$$M = \begin{bmatrix} t_u \\ t_v \\ p \end{bmatrix}. \quad (3.34)$$

Finalmente, a cônica  $Q_{h_s}$  é transladada para obter a forma de uma cônica central  $Q_s$ .

### 3.2.2 Filtro de Reamostragem

O pré-filtro é somado ao núcleo de reconstrução, formando o filtro de reamostragem.

Por sua vez, a matriz do filtro de reamostragem é definida por:

$$Q_p = Q_s + Q_h. \quad (3.35)$$

O filtro é rasterizado avaliando os pixels dentro de uma caixa envolvente alinhada com os eixos da elipse. Em outras palavras, a elipse representando um isocontorno do filtro é rasterizada, onde um peso de acordo com a distância ao centro é atribuído à contribuição de cada pixel.

### 3.2.3 Compondo a Matriz de Pixels

Para superfícies opacas, a composição da matriz de pixels é trivial. O novo *fragmento* (contribuição do pixel) contém informação sobre um pedaço de uma superfície, e deve

ser avaliado considerando se o valor corrente da matriz de pixels. A comparação é realizada utilizando os valores de profundidade do fragmento e do pixel corrente. Para tanto, é efetuado um teste de profundidade ternário, isto é, além das duas opções do teste padrão, uma terceira também é possível dado um intervalo de confiança de profundidade da superfície:

- se o valor de profundidade do fragmento é maior que todo o intervalo, e portanto pertence a outra superfície, ele é descartado;
- se o valor do fragmento é menor que todo o intervalo, o seu valor substitui integralmente o valor armazenado no pixel;
- e finalmente, se a profundidade do fragmento estiver dentro do intervalo, sua contribuição é adicionada a superfície corrente.

O valor final de um pixel é computado através de uma média ponderada de todas as contribuições:

$$g(x) = \sum_{k \in \mathbb{N}} w_k \frac{\rho_k(x)}{\sum_{j \in \mathbb{N}} \rho_j(x)}. \quad (3.36)$$

Apesar desta metodologia funcionar bem para a maioria dos casos, ela peca ao estabelecer um intervalo fixo de confiança  $\varepsilon$  para toda superfície, especialmente nos casos de nuvens de pontos não uniformes:

$$|\text{new fragment}_z - \text{frame buffer pixel}_z| < \varepsilon. \quad (3.37)$$

Räsänen [13] propôs um intervalo adaptativo baseado nas propriedades locais da superfície. Neste caso o intervalo é determinado utilizando valores máximos e mínimos de profundidade para a amostra projetada. Assim, *splats* são considerados como pertencentes a mesma superfície, em caso de sobreposição de seus intervalos.



# Capítulo 4

## Reconstrução de Superfícies em Espaço de Imagem para Nuvens de Pontos Não-Estruturadas

Na Sessão 2.1 a independência entre o custo computacional e a complexidade geométrica foi definida como a principal vantagem dos algoritmos de renderização baseada em imagem. Por sua vez, uma das maiores contribuições dos algoritmos baseado em pontos é o desvinculo da conectividade. O algoritmo apresentado neste capítulo propõe reunir as vantagens das duas áreas, integrando os núcleos elípticos do Surface Splatting com uma variação do método *análise-síntese* para reconstrução em espaço de imagem.

### 4.1 Algoritmo em Pirâmide

Os algoritmos em pirâmide foram extensivamente utilizados no campo de processamento de imagens e computação gráfica. Um dos exemplos mais conhecidos é o uso das pirâmides Gaussianas para gerar *mip-maps* [45], i.e., diferentes níveis de resoluções para texturas. Ogden et. al. [46] utilizaram as pirâmides para solucionar problemas em compressão de dados, análise de imagens, realce de características, remoção de ruído, entre outros. Em um trabalho subsequente [29], demonstraram a aplicação do algoritmo para preenchimento de informações ausentes em imagens, isto é, um método para pixels vazios.

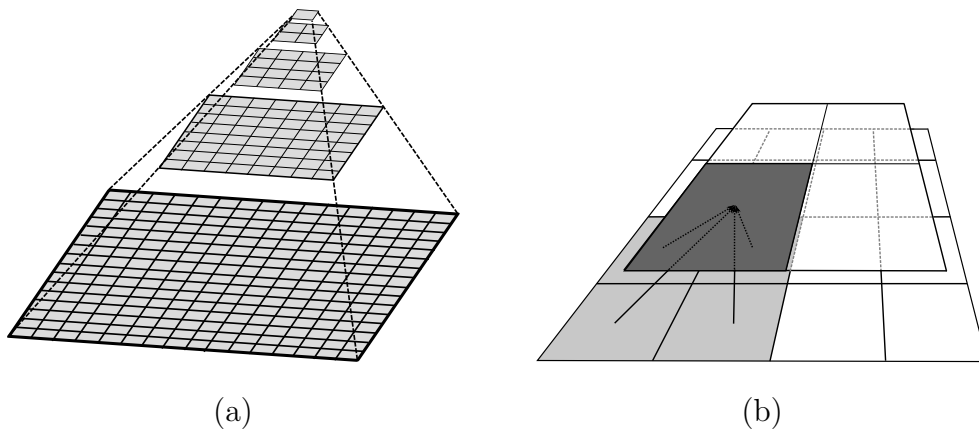


Figura 4.1: (a) Uma pirâmide construída a partir de uma imagem com resolução 16x16. (b) Um pixel de uma resolução mais baixa é computado pela média dos quatro diretamente abaixo.

A pirâmide é geralmente computada como uma hierarquia de níveis de resolução de uma imagem, como ilustrado na Figura 4.1a. O nível mais baixo da pirâmide (maior resolução) armazena a imagem de entrada e, a cada novo nível, uma nova imagem é criada com metade da resolução. Cada nível é gerado por médias ou interpolações de pixels do nível diretamente abaixo. No caso de pirâmides Laplacianas [47], por exemplo, as diferenças entre cada nível de resolução também são armazenadas.

A representação em multiresolução foi estudada também por meio da teoria de *waveletes* [48]. A hierarquia em pirâmide é similar à decomposição de imagens utilizando a base de Haar, uma das bases *wavelets* mais simples [49, 50, 14]. Neste caso as dimensões da imagem também são reduzidas pela metade em cada interação, porém, as diferenças entre os níveis, ou coeficientes de detalhe, são mantidas para possibilitar a descompressão ou transmissão progressiva. A Figura 4.2 ilustra um exemplo do primeiro nível de decomposição utilizando a base de Haar. Diferentemente do algoritmo em pirâmide utilizado neste trabalho, a decomposição *wavelet* gera uma estrutura com mesmo tamanho da imagem original. No caso da pirâmide hierárquica, todos os níveis de resolução são mantidos integralmente ocupando um espaço maior de armazenamento.

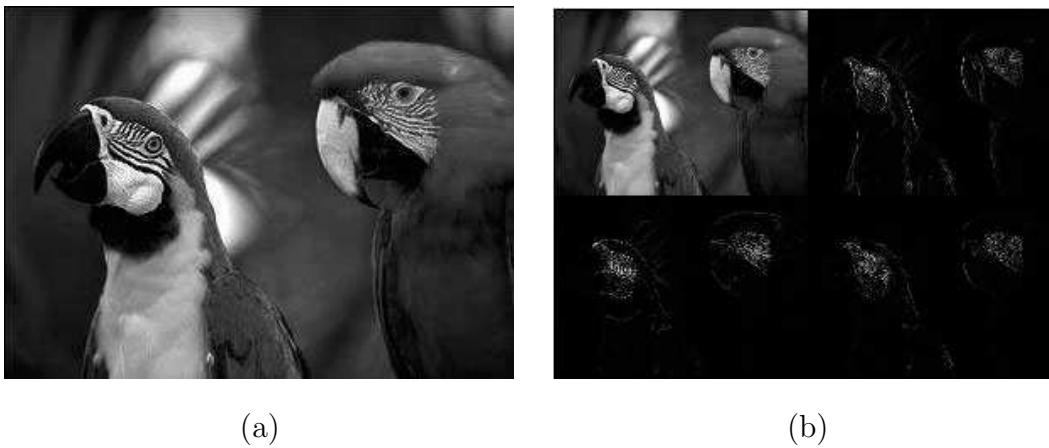


Figura 4.2: (a) Imagem original e (b) o primeiro nível da decomposição utilizando a base de Haar. Note que o canto superior esquerdo da decomposição corresponde a uma resolução mais baixa da imagem original ao reduzir cada dimensão pela metade, enquanto que os coeficientes de detalhe são armazenados nos outros quadrantes. (Retiradas de [14])

#### 4.1.1 Algoritmo de Análise-Síntese

No entanto, o nosso interesse é aplicar a pirâmide no contexto de reconstrução de imagem. Mais especificamente, uma variante do algoritmo *análise-síntese* é utilizada para interpolar dados esparsos. Este método é separado em duas partes: *análise*, onde a pirâmide é criada de baixo para cima; e *síntese*, onde os pixels são interpolados de cima para baixo para reconstruir as informações ausentes.

A construção da pirâmide parte do nível mais baixo, i.e., de maior resolução. A cada nível superior, um pixel é construído pela média dos quatro pixels diretamente abaixo (Figura 4.1b). Neste caso, como todos pixels de maior resolução estão equidistantes do pixel acima, um peso igual ( $\frac{1}{4}$ ) é atribuído aos quatro, como ilustrado na Figura 4.3a. Porém, se o enfoque é reconstrução, deve-se assumir que alguns pixels estão vazios, ou seja, são inválidos. Por isso, a média é realizada somente com os pixels válidos (contendo informação) e o valor renormalizado de acordo. No caso dos quatro pixels serem inválidos o novo pixel também o será.

A fase de síntese, por sua vez, reconstrói os pixels inválidos partindo do topo da pirâmide. Cada pixel inválido é computado através da interpolação dos quatro pixels acima mais próximos, como ilustrado na Figura 4.3b. Neste caso, não é necessário renormalizar os pesos, já que o nível acima já foi reconstruído e seus pixels marcados

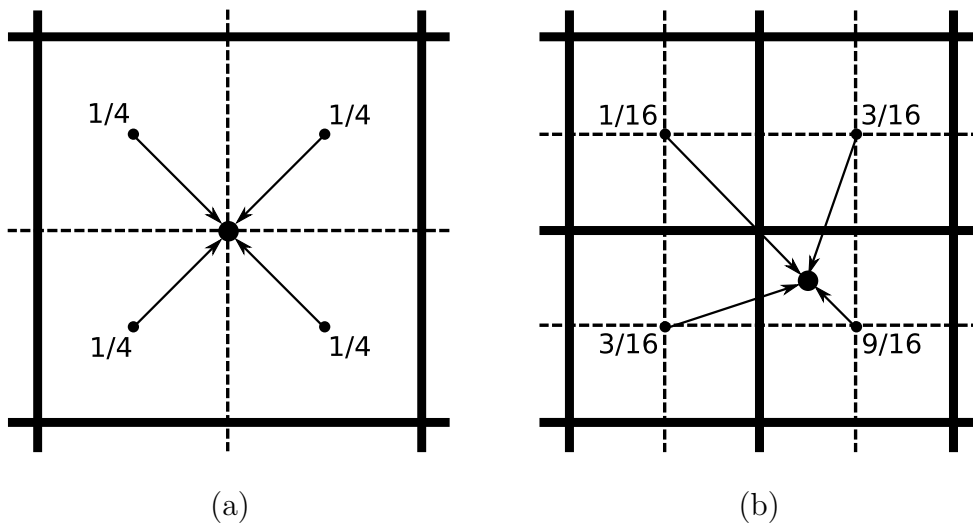


Figura 4.3: Os pesos utilizados para construir um novo pixel durante a fase de (a) *análise* e (b) *síntese*.

como válidos.

Strengert et al. [15] desenvolveram filtros em pirâmide utilizando os recursos da programação em GPU. O método para interpolação de dados esparsos, descrito acima, é executado em 1.4 milissegundos em uma placa nVidia GeForce 7800 GTX 512MB, para uma imagem de tamanho  $1024^2$ . Na Figura 4.4 podem ser observados alguns resultados deste método em GPU.

## 4.2 Algoritmo em Pirâmide para Renderização de Pontos

Nesta seção é apresentada uma variação do algoritmo de *análise-síntese* para interpolar dados esparsos providos de uma nuvem de pontos. Um resumo do algoritmo é ilustrado na Figura 4.5. Na primeira etapa as amostras do modelo são projetados para tela, onde cada uma ocupa o espaço de um pixel. Em seguida o algoritmo em pirâmide de *análise-síntese* é aplicado para interpolar os valores das amostras entre os espaços vazios e, assim, reconstruir a superfície. Por último, a iluminação é computada por pixel utilizando os valores de normal e cor interpolados. Cada etapa é detalhada nas próximas sessões.



Figura 4.4: Resultados do algoritmo de reconstrução de imagens em GPU. Na linha de cima as duas imagens de entrada com pixels vazios, em baixo as respectivas reconstruções. (Retiradas de [15])

### 4.2.1 Projeção dos Pontos

Os pontos são projetados para o plano de imagem utilizando uma matriz padrão de *model-view*. O vetor normal é utilizado para determinar os pontos visíveis de acordo com o vetor de visão, ou seja, descartar pontos cuja normal aponte para trás do plano de imagem. Além da normal, também é requerido um atributo de raio por ponto. Este raio define a área de extensão do ponto e é proporcional à distância de amostragem local em torno do ponto.

Uma forma de estimar a densidade da amostragem local  $\rho_i$  é computando a esfera de raio  $r$  mínimo, centrada na amostra  $p$ , que contém os  $k$ -vizinhos mais próximos. Em outras palavras,  $r$  corresponde à distância de  $p$  ao  $k$ -vizinho mais distante. O

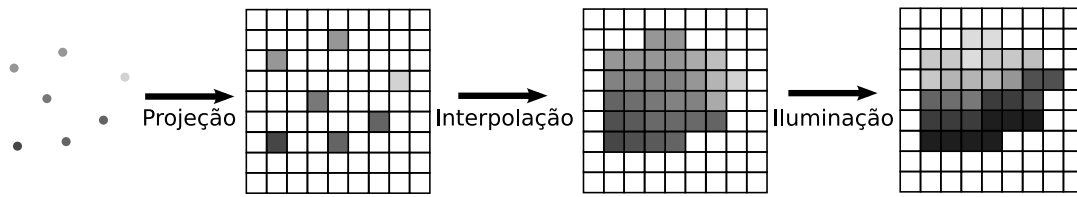


Figura 4.5: Resumo do algoritmo de renderização de pontos: primeiro, os pontos são projetados para o espaço de imagem; em seguida, o algoritmo de *análise-síntese* é aplicado para interpolar os valores das amostras; por último, iluminação por pixel é computada.

valor de  $\rho_i$  na posição  $p_i$  é avaliado da seguinte forma [32]:

$$\rho_i = \frac{k}{\pi r_i^2}. \quad (4.1)$$

A distância de amostragem local pode ser derivada da densidade  $\rho_i$ :

$$\eta(x) = \frac{1}{\sqrt{\rho(x)}}. \quad (4.2)$$

Cada ponto é projetado com tamanho de um pixel e, portanto, seus atributos são armazenados em no máximo um pixel da matriz de imagem. Um teste de oclusão é acionado para lidar com múltiplas projeções em um mesmo pixel, onde somente o mais à frente é mantido.

## 4.2.2 Interpolação com Algoritmo em Pirâmide

Diferentemente do algoritmo original de *análise-síntese*, é necessário interpolar mais do que a cor para reconstruir a superfície. Além dos atributos projetados, i.e., as normais e raios, são necessários ainda um valor de profundidade e um vetor de deslocamento por pixel, como será detalhado adiante. Atributos adicionais também podem ser interpolados, como cor ou coordenadas de textura por exemplo. Os atributos no espaço de imagem definem uma elipse em duas dimensões limitando a área de influência de cada amostra projetada, análogo ao *Surface Splatting*. Por isso, deste ponto em diante, os pixels são também referenciados como elipses durante a interpolação.

A elipse é derivada dos atributos do pixel considerando uma projeção ortogonal de um círculo no espaço de objeto para o espaço de imagem. O eixo maior da elipse

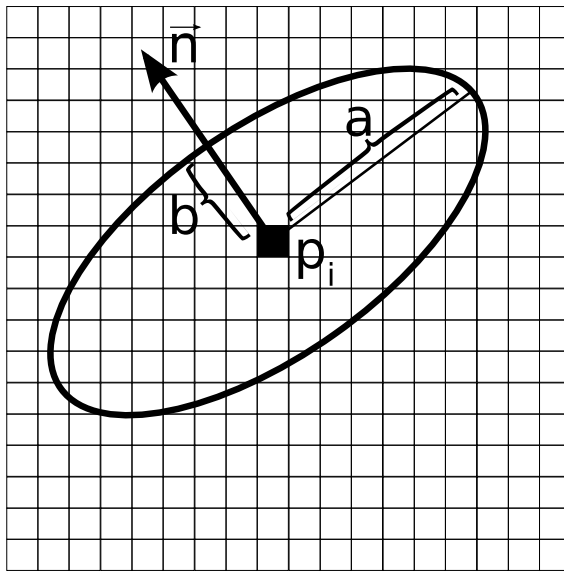


Figura 4.6: A elipse é definida pelo centro de projeção  $p_i$  e a normal projetada  $\vec{n}$ . O eixo maior perpendicular à normal tem tamanho  $a$  igual ao raio projetado, enquanto o eixo menor tem tamanho  $b = a \cdot n_z$ .

é definido como perpendicular à projeção da normal no plano de imagem e com tamanho igual ao raio projetado. Por sua vez, o eixo menor é definido na direção da normal e com tamanho do raio projetado multiplicado pelo coordenada  $z$  da normal. A projeção de uma amostra no plano da imagem é ilustrada na Figura 4.6.

Durante a fase de análise, os níveis da pirâmide são computados como descrito na Sessão 4.1.1. Porém, um teste adicional de oclusão é realizado para descartar amostras de superfícies não visíveis. A média dos atributos é realizada com no máximo quatro pixels (elipses) do nível abaixo, considerando que alguns podem ser inválidos. A elipse com menor valor de profundidade (mais à frente) é utilizada como base de comparação para o teste. As demais só são incluídas na média se seus intervalos de profundidade intersectam o intervalo da elipse mais à frente. O intervalo de profundidade é definido pelo valor de profundidade mais/menos o raio projetado da amostra.

O vetor de deslocamento é necessário para que o centro das elipses não sejam perdidos durante o processo de interpolação. A cada novo pixel criado, seus atributos definem uma nova elipse em uma resolução mais baixa. Entretanto, o centro da elipse não corresponde necessariamente ao centro do novo pixel, sendo necessário armazenar um vetor de deslocamento. Este vetor indica o deslocamento do centro

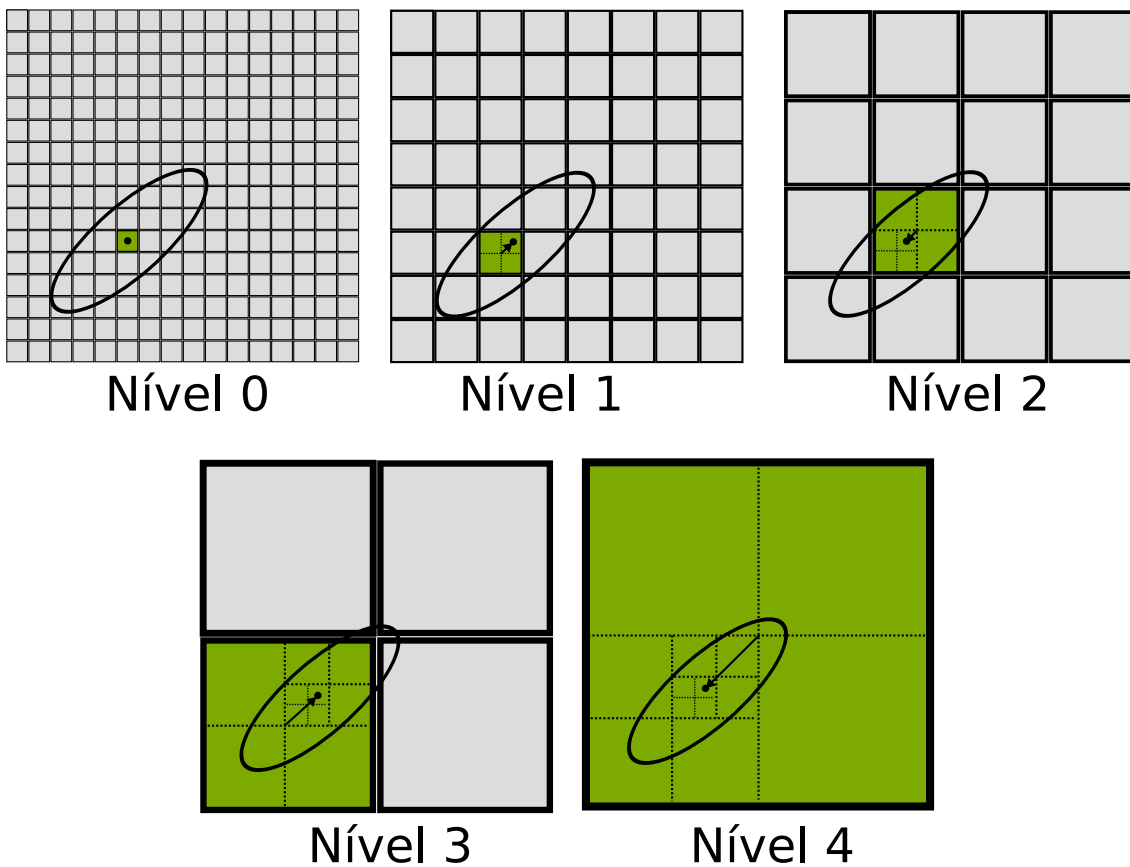


Figura 4.7: O vetor de deslocamento é ilustrado para os cinco primeiros níveis. A seta indica o deslocamento do centro do pixel correspondente (em verde) ao centro da elipse (círculo preto). Apenas no nível 0 o centro da elipse corresponde ao centro do pixel. Note que somente uma elipse foi propagada, não havendo necessidade de realizar médias de elipses.

do pixel ao centro da elipse, como ilustrado na Figura 4.7. O centro da nova elipse é definido pela média dos vetores de deslocamento. Na Figura 4.8 a mesma propagação da elipse é demonstrada com uma visão vertical.

Na fase seguinte de síntese, todos os pixels que permaneceram inválidos são interpolados. Os mesmos quatro pixels do algoritmo original são utilizados para computar os atributos de um pixel inválido. Porém, alguns testes são realizados para decidir se uma elipse será ou não utilizada na interpolação. Primeiro, ela deve cobrir o pixel a ser computado. Para tanto, um teste simples de ponto no interior/exterior da elipse é realizado utilizando o vetor de deslocamento para determinar o seu centro.

Das elipses restantes, o mesmo teste de oclusão da fase de análise é realizado. Os atributos do pixel são então computados pela média de todas elipses não oclusas



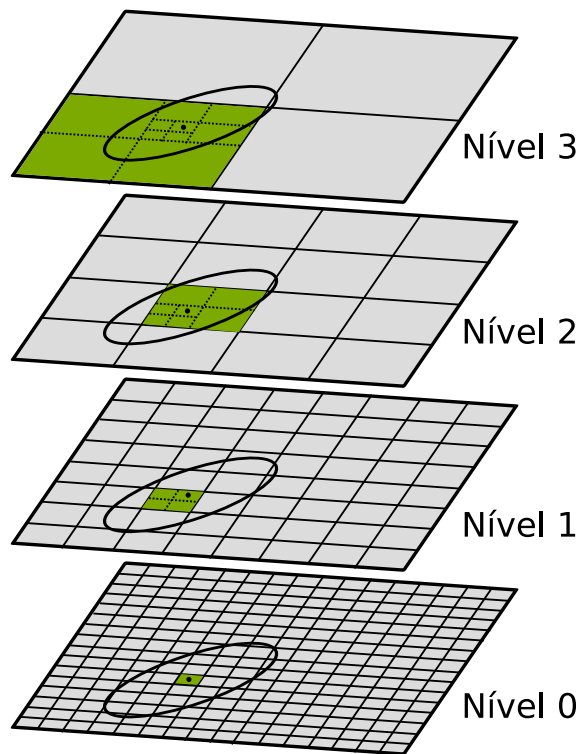


Figura 4.8: Durante a propagação da elipse seu centro é mantido pelo vetor de deslocamento. O pixel em verde no nível 0 corresponde ao centro da elipse.

que o cobre. Os pesos utilizados são diferentes daqueles mostrados na Figura 4.3. Ao invés de valores fixos, um peso Gaussiano proporcional à distância do pixel ao centro da elipse é aplicado. A Figura 4.9 ilustra a distribuição e a renderização das amostras com a interpolação respeitando os limites das elipses no plano de imagem.

Além dos pixels inválidos, aqueles que representam pedaços oclusos de uma superfície também são recomputados. Isto é realizado com um teste de profundidade entre o pixel corrente e o pixel correspondente um nível acima. Se o intervalo de profundidade estiver atrás do intervalo do pixel acima, ele é considerado como inválido e recomputado. A Figura 4.10 ilustra a distribuição das elipses e o descarte de amostras não visíveis.

### 4.2.3 Iluminação por Pixel

Após o processo de interpolação, todos pixels considerados como interiores à silhueta do modelo possuem atributos válidos e, conseqüentemente, um vetor normal. Em outras palavras, a imagem interpolada pode ser interpretada como um mapa de normais do modelo. Para os pixels válidos, iluminação de Phong é computada

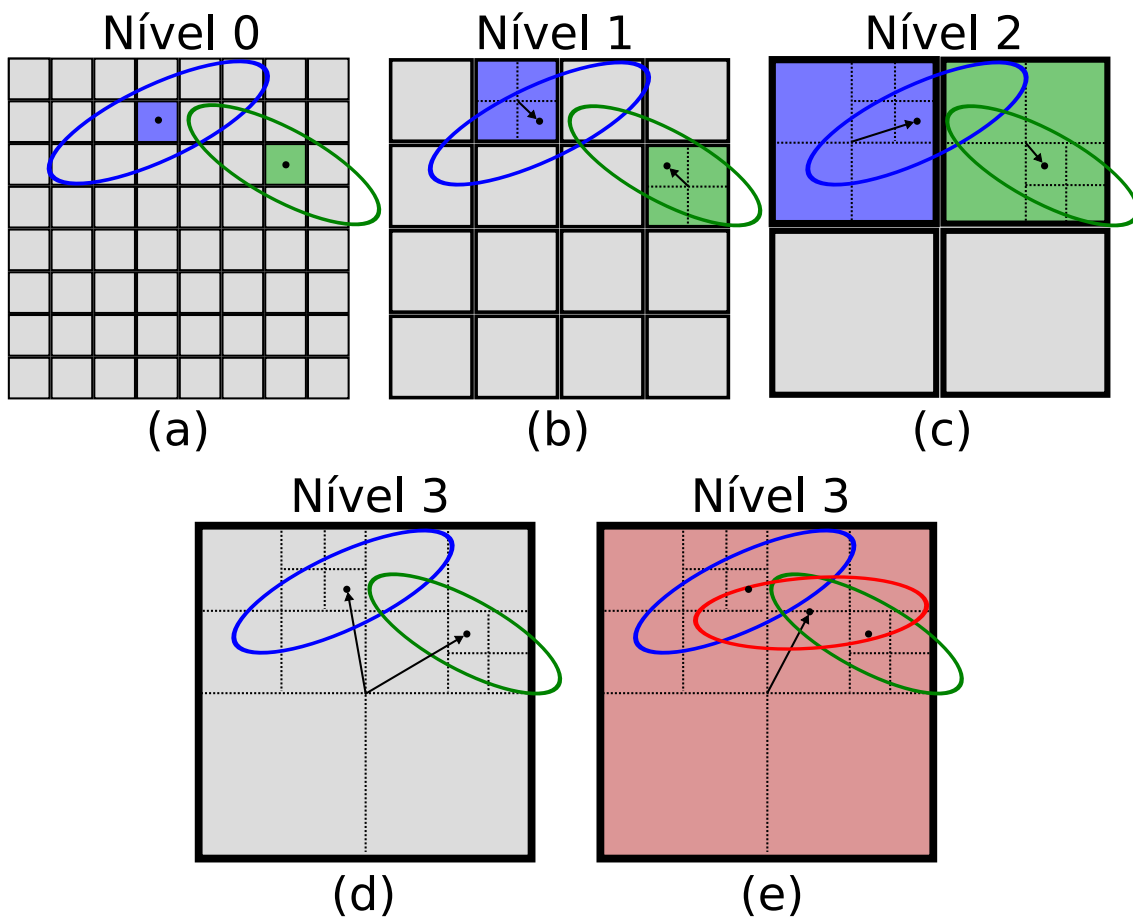


Figura 4.9: A sequência demonstra a evolução de duas elipses na pirâmide até o momento (d) em que são combinadas (e) gerando uma única elipse (vermelha) com novo centro e eixos.

utilizando, adicionalmente, informação de cor ou textura. A Figura 4.11 ilustra o mapa de normais e o modelo após a fase de iluminação.

#### 4.2.4 Implementação na GPU

Enquanto que a implementação na GPU da projeção dos pontos e da iluminação por pixel é trivial, a implementação do algoritmo em pirâmide é menos óbvia. Os níveis da pirâmide são armazenados em uma textura para serem acessados pelos *shaders*<sup>1</sup>. De forma a não limitar o número de níveis ao número máximo de unidades de texturas disponível pelo hardware gráfico, todos são armazenados em uma única

<sup>1</sup>Programas executados na GPU que substituem trechos das funcionalidades fixas do pipeline gráfico. Nas placas atuais existem três níveis: *shader* de vértice, *shader* de geometria e *shader* de fragmento.

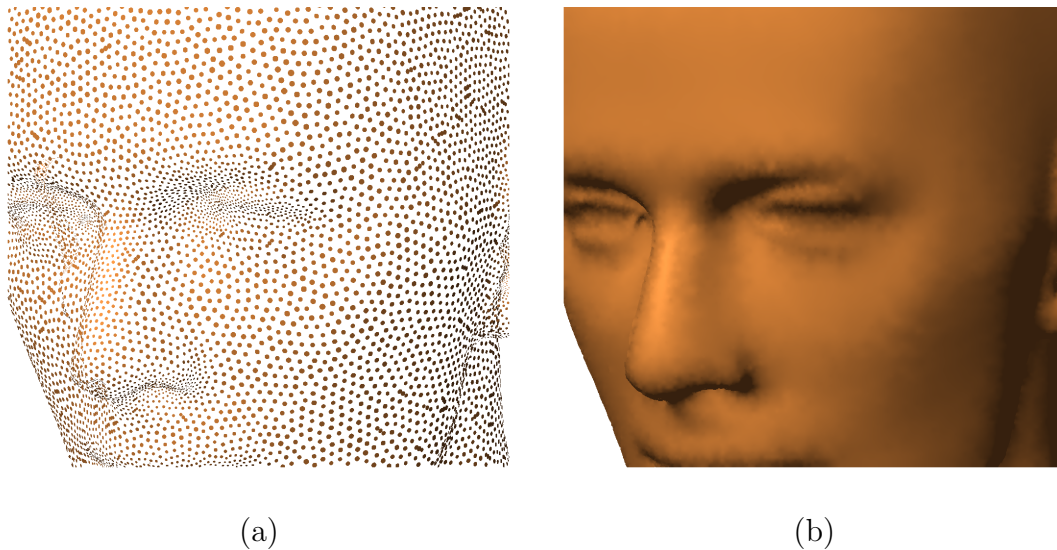


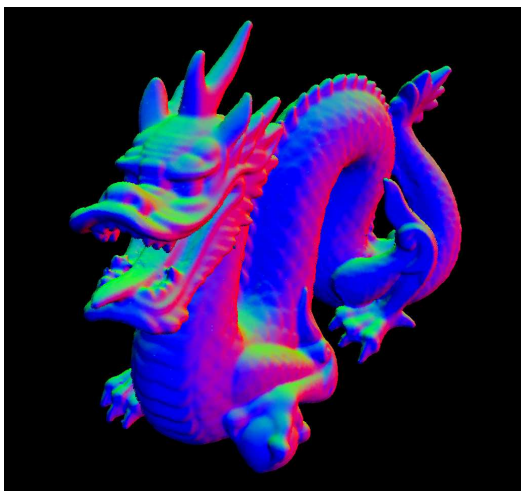
Figura 4.10: (a) Elipses renderizadas com 20% do raio para ilustrar a distribuição dos pontos e a eliminação de pontos oclusos (e.g., parte traseira da cabeça) e (b) o mesmo modelo renderizado sem buracos e iluminado.

textura. O esquema de empacotamento é similar ao utilizado por Strengert et al. [15], como pode ser observado na Figura 4.12.

Contudo, existe uma limitação atual de hardware que não permite acessar uma textura para leitura e escrita ao mesmo tempo. Por isso, é necessário recorrer a um esquema do tipo *ping-pong* para interpolar os dados. O primeiro nível da pirâmide é criado a partir da projeção dos pontos no espaço de imagem e, conseqüentemente, em uma textura. Para criar o segundo nível, esta textura é utilizada para leitura e uma segunda de mesmo formato para escrita. Para a criação do terceiro nível os papéis são invertidos, e assim por diante. O mesmo esquema é utilizado na fase de síntese.

Ao final da fase de análise, uma textura mantém os níveis de número ímpar e a outra os níveis de número par. Porém, na fase de síntese é preciso ler dois níveis consecutivos - o nível corrente e o imediatamente acima, ou seja, um nível ímpar e outro par - para computar o valor de um novo pixel. Em vista disto, uma passada adicional de cópia é necessária para preencher os níveis ausentes nas duas texturas. Este esquema é ilustrado na Figura 4.13, no entanto com um empacotamento ineficiente das texturas por motivos ilustrativos.

Cada pixel de uma textura *RGBA* possui apenas 4 elementos de ponto flutuante. Para armazenar todos os atributos de cada pixel são necessárias pelo menos duas



(a)



(b)

Figura 4.11: (a) O mapa de normais gerado pela interpolação em pirâmide e (b) a imagem resultante após iluminação por pixel.

texturas organizadas da forma ilustrada na Figura 4.14. Para interpolar atributos adicionais, como cor por exemplo, são necessárias texturas adicionais. As múltiplas texturas são manuseadas em grupos no esquema ping-pong. A Figura 4.15 apresenta um mesmo modelo renderizado com e sem a textura adicional armazenando atributos de cor.

## 4.2.5 Resultados

Os resultados da Tabela 4.1 foram gerados com um Pentium 4 (3 GHz) com 2 GB RAM e uma GeForce 7800 GTX com 512 MB de memória. Todos os modelos foram pré-processados para computar os vetores normais e os raios em cada ponto. A quarta coluna da tabela apresenta tempos em milisegundos para uma resolução de  $512 \times 512$ , enquanto que a sexta coluna apresenta tempos para uma resolução de  $1024 \times 1024$ . Além do tempo total por quadro, dois tempos em parênteses são detalhados: tempo para projetar os pontos e tempo para interpolar os dados com o algoritmo de *análise-síntese*. Como esperado, essas duas operações requerem quase todo o tempo de renderização, enquanto as outras operações, como iluminação por exemplo, são quase desprezíveis. Renderizações de alguns modelos incluídos na Tabela 4.1 são apresentados na Figura 4.16

Para o mesmo hardware gráfico e resolução de  $1024 \times 1024$ , Zhang e Pajarola



Figura 4.12: Diagramação dos 10 níveis gerados para uma imagem de  $512 \times 512$ . (Retirada de [15])

[51] atingiram uma marca de 24.9 M por segundo, enquanto Guennebaud et al. [12] alcançaram 37.5 M splats por segundo. Comparativamente, o algoritmo proposto atinge entre 50 M e 60 M splats por segundo. Além disso, os algoritmos de Splatting apenas atingem o pico de performance para splats de tamanho pequeno, enquanto o algoritmo proposto não depende do tamanho dos splats.

A Tabela 4.2 apresenta resultados para modelos renderizados com e sem interpolação de atributos de cor. Esses resultados foram gerados em um Intel Core Duo 6600 (2.4GHz) com 2GB RAM e uma placa GeForce 8800 GTS com 640MB de memória. Com esta configuração a performance chega a 130 M splats por segundo, isto é, equivalente a 65% do desempenho 200 M splats/seg esperado do hardware dedicado[40], porém utilizando apenas hardware genérico. De qualquer forma, o protótipo do hardware alcançou apenas 17.5 M splats/seg.

### 4.3 Discussão

Como se pode notar a partir dos resultados apresentados, para uma dada resolução de saída e quantidade de atributos, o tempo de reconstrução é praticamente constante para todos modelos. Isto era esperado, já que o algoritmo de reconstrução não depende do número de amostras projetadas e sim do número de pixels. Esta qua-

Tabela 4.1: Tempo de renderização, em quadros por segundo *qps*, com uma placa 7800 GTX para duas resoluções de saída diferentes. \*Cada tempo total por quadro é seguido por dois tempos em parênteses: tempo de projeção e tempo de interpolação. Todos os tempos são descritos em milisegundos.

modelo	# pontos	512 × 512 viewport		1024 × 1024 viewport	
		qps	tempo por quadro*	qps	tempo por quadro*
Head	25 K	82	12 ms (0.2, 12)	24	42 ms (0.3, 38)
Armadillo	173 K	71	14 ms (1.4, 13)	22	46 ms (1.7, 41)
Happy Buddha	544 K	61	16 ms (5.1, 11)	21	47 ms (4.9, 39)
Asian Dragon	3.610 K	26	39 ms (28, 10)	14	71 ms (29, 38)
Thai Statue	5.000 K	20	49 ms (38, 10)	12	82 ms (39, 38)

Tabela 4.2: Tempo de renderização com uma placa 8800 GTS com e sem interpolação dos atributos de cor. \*Cada tempo total por quadro é seguido por dois tempos em parênteses: tempo de projeção e tempo de interpolação. Todos os tempos são descritos em milisegundos.

modelo	# pontos	Sem atributos de cor		Com atributos de cor	
		qps	tempo por quadro*	qps	tempo por quadro*
Armadillo	173 K	89	11 ms (1.2 ms, 8.5 ms)	46	22 ms (1.5 ms, 18 ms)
Dragon	437 K	78	13 ms (2.2 ms, 8.9 ms)	44	23 ms (2.9 ms, 18 ms)
Happy Buddha	544 K	76	13 ms (2.6 ms, 8.8 ms)	42	24 ms (3.4 ms, 18 ms)
Asian Dragon	3610 K	36	28 ms (18 ms, 8.3 ms)	23	45 ms (25 ms, 17 ms)
Thai Statue	5000 K	29	35 ms (25 ms, 8.2 ms)	18	55 ms (34 ms, 18 ms)

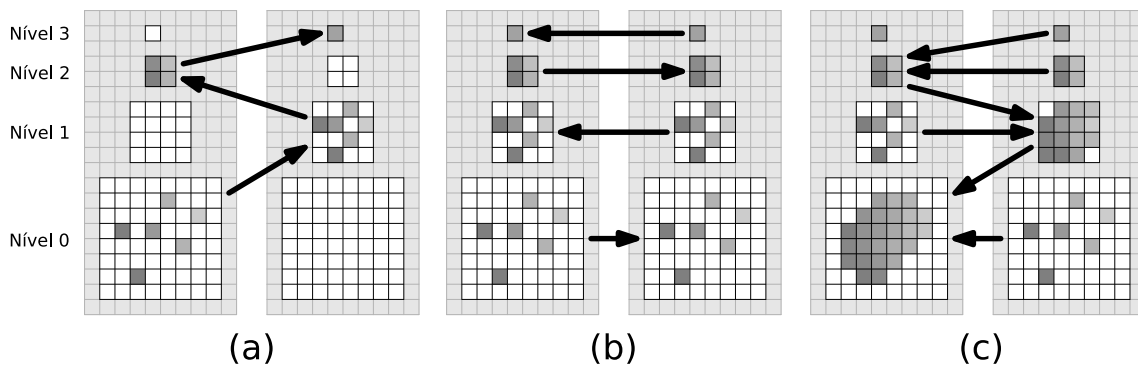


Figura 4.13: Esquema de *ping-pong* entre as duas texturas: (a) a fase de análise constrói a pirâmide de baixo para cima; (b) a fase de cópia preenche os níveis ausentes; (c) a fase de síntese computa os pixels inválidos de cima para baixo.

Textura 1:

$n_x$	$n_y$	$n_z$	$r$
-------	-------	-------	-----

Textura 2:

$z$	$z_{\min}$	$v_x$	$v_y$
-----	------------	-------	-------

Figura 4.14: A primeira textura armazena as componentes da normal e o raio projetado, enquanto que a segunda armazena a profundidade, o intervalo de profundidade e as componentes do vetor de deslocamento.

lidade aproxima este método daqueles baseados em imagem, porém, ainda mantém as características geométricas do Surface Splatting.

O algoritmo apresentado neste capítulo apresenta alguns pontos positivos que merecem destaque:

- a projeção é feita de maneira simples e direta, e para cada ponto são realizadas apenas duas multiplicações por matriz - uma para posição e outra para a normal - ou seja, possui complexidade  $O(n)$  porém com uma constante baixa;
- a interpolação tem custo independente do número de amostras, melhorando a complexidade em relação as outras propostas, mais especificamente  $O(n)$  contra  $O(n * a)$ , onde  $a$  é o tamanho médio do splat em pixels;
- o teste de profundidade é executado simultaneamente com a interpolação dispensando múltiplas passadas, como acontece na maioria dos algoritmos de



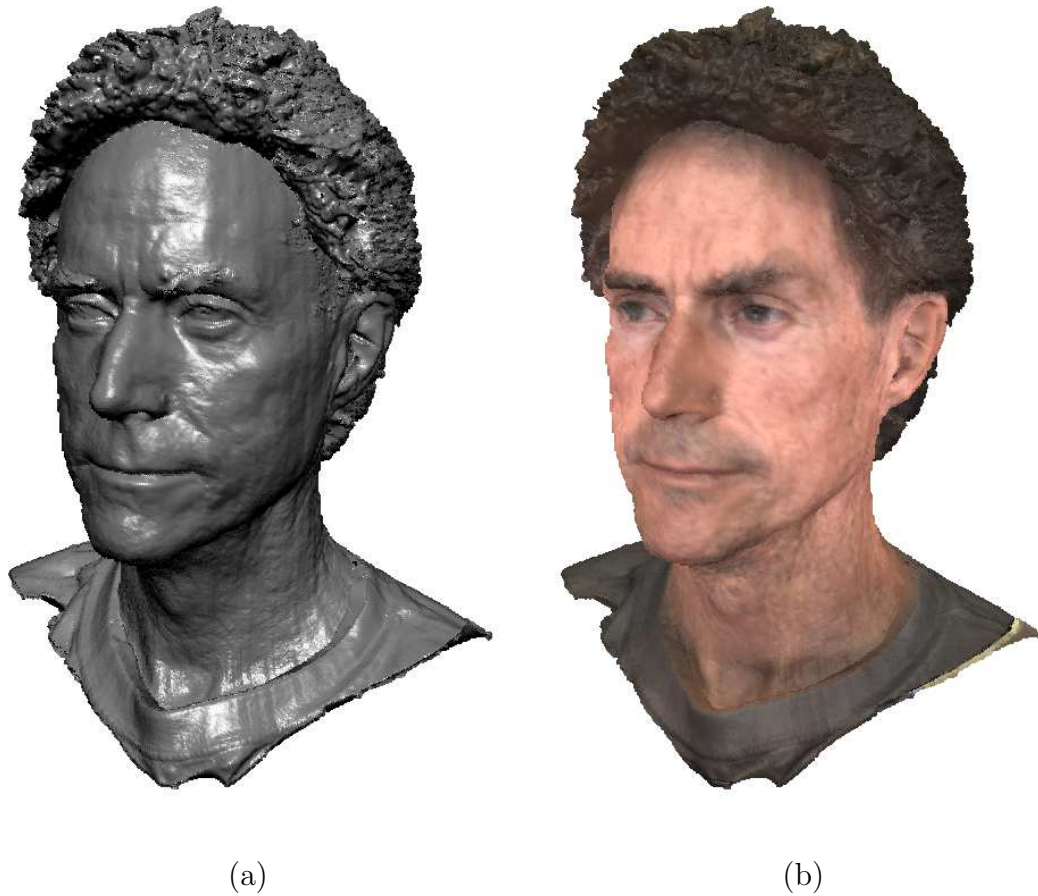


Figura 4.15: O mesmo modelo renderizado (a) sem atributos de cor e (b) com atributos de cor por amostra.

Surface Splatting;

- ao interpolar normais é possível realizar iluminação por pixel, o que não ocorria nas primeiras propostas de renderização por pontos baseado em imagens [4, 5].

Contudo, o algoritmo ainda possui algumas limitações:

- a projeção de pontos elimina amostras concorrentes no mesmo pixel, permanecendo apenas aquela mais à frente;
- não é realizado nenhum tratamento *anti-aliasing* dos splats;
- elipses são geradas na pirâmide durante a reconstrução mas nem sempre propagadas até o nível 0, gerando descontinuidades perceptíveis nas silhuetas;
- não é trivial adicionar transparência aos modelos já que não existe ordenação na interpolação.



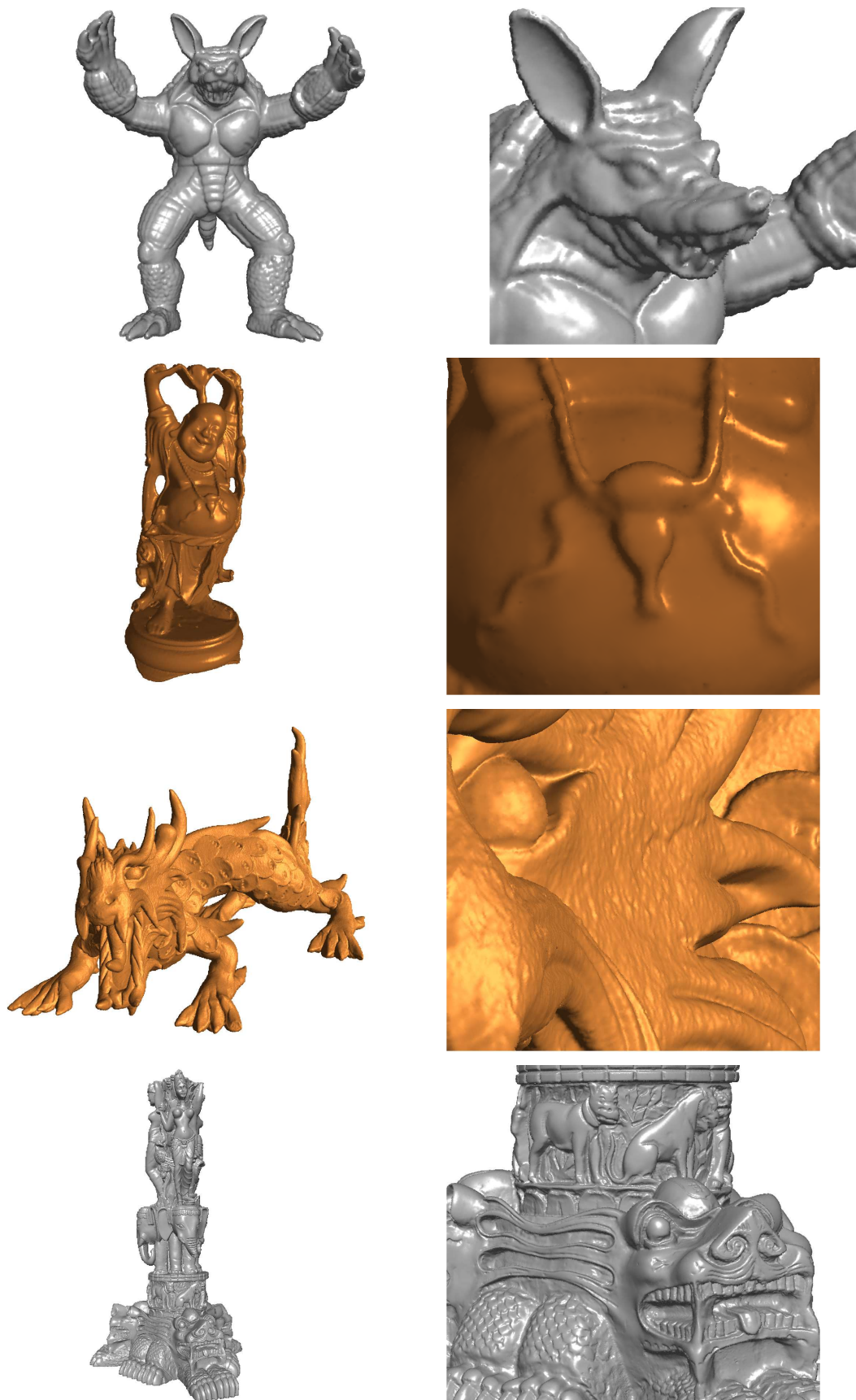


Figura 4.16: Modelos renderizados com o algoritmo em pirâmide. De cima para baixo: Armadillo, Happy Buddha, Asian Dragon e Thai Statue.

De uma forma geral, o algoritmo se destaca pelas altas taxas de renderização, porém, ainda não alcança a qualidade do Surface Splatting, especialmente por não aplicar nenhum tratamento *anti-aliasing*.

# Capítulo 5

## Renderização de Linhas e Triângulos em Espaço de Imagem

Como mencionado na Seção 1.2, o objetivo da representação por pontos não é substituir as demais, senão complementá-las. Para tanto, é importante que um renderizador de pontos também seja capaz de tratar outras primitivas. Neste capítulo é apresentada uma extensão do algoritmo proposto no capítulo anterior para tratar linhas e triângulos. Mais especificamente, é descrita na Sessão 5.1 uma abordagem para renderizar fitas e tubos representados por linhas, seguida de um prova de conceito onde um modelo de uma árvore é renderizado utilizando essas primitivas. Por fim, é descrita a integração de triângulos com o método na Sessão 5.2.

### 5.1 Renderização de Linhas com Reconstrução de Imagens

Neste trabalho, linhas são utilizadas para renderizar dois tipos de primitivas: fitas e tubos. Essas primitivas diferem não apenas no processo de interpolação, mas também na iluminação.

#### 5.1.1 Fitas

Fitas são sessões planares de comprimento e largura variados e limitados, como demonstrado na Figura 5.1a. Para acoplar esta primitiva ao algoritmo em pirâmide

ela deve ser rasterizada como pixels representando pontos. Para tanto, o centro da fita é aproximada por uma polilinha com espessura de um pixel, onde a cada vértice é atribuído a normal no ponto e metade da largura como raio. Estes atributos são interpolados no processo de rasterização do *pipeline* gráfico gerando uma sequência de pixels, como ilustrado na Figura 5.1b.

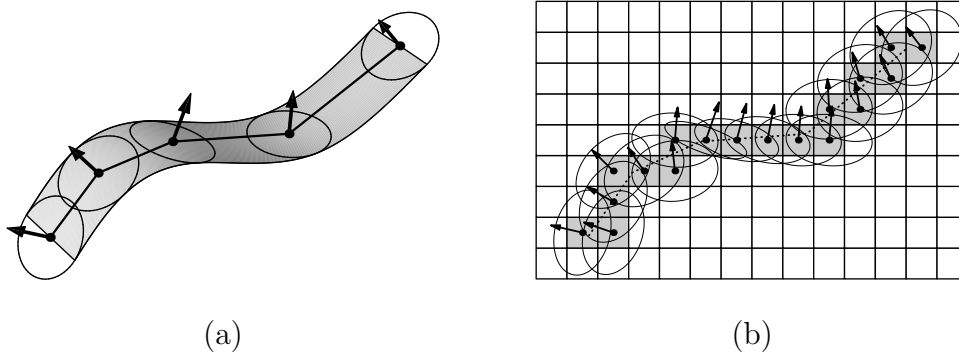


Figura 5.1: (a) Representação de uma fita por uma polilinha contendo cinco vértices com normais e raios. (b) Ilustração da renderização da polilinha correspondendo ao centro da fita. Normais e raios são interpolados para cada pixel coberto.

Existem algumas diferenças no processamento de pontos e fitas. Por exemplo, a remoção de pontos não visíveis durante a projeção não é aplicada, tendo em vista que a fita é visível de ambos lados, como exemplificado na Figura 5.2. Portanto, a iluminação também deve considerar as normais em ambos os lados. Todavia, pixels pertencentes a fitas são indistinguíveis daqueles pertencentes a pontos durante a interpolação.

### 5.1.2 Tubos

O segundo tipo de primitiva derivado das linhas são os tubos, i.e., cilindros curvos de espessura variada, como ilustrado na Figura 5.3a. Assim como as fitas, tubos são representados por uma polilinha aproximando seu centro. A cada vértice é atribuído metade do diâmetro do tubo e, diferentemente das fitas, o vetor tangente local ao invés da normal. As polilinhas são novamente rasterizadas com espessura de um pixel e os atributos linearmente interpolados entre os vértices, como demonstrado na Figura 5.3b.

Em contraste com a interpolação de pontos e fitas, o algoritmo de *análise-síntese*

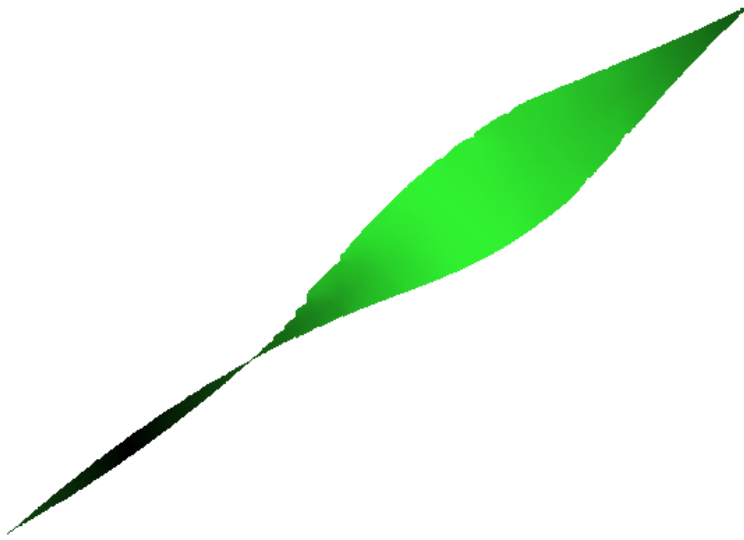


Figura 5.2: Uma fita torcida com os dois lados renderizados e iluminados.

sempre considera o vetor para o centro da camera como normal da superfície. Desta forma, são renderizados discos paralelos ao plano de imagem, aproximando a espessura do tubo projetado.

A iluminação dos tubos também difere das fitas e se aproxima das propostas na literatura para iluminação de linhas [52, 53]. O método mais simples para iluminação difusa por um único ponto de luz determina a normal da superfície  $n$ , pela projeção do vetor de iluminação  $l$ , no plano ortogonal ao vetor tangente  $t$  da linha, como ilustrado na Figura 5.4. Contudo, é importante realçar que esta aproximação só é válida para estruturas finas; alternativas para níveis de detalhes maiores são

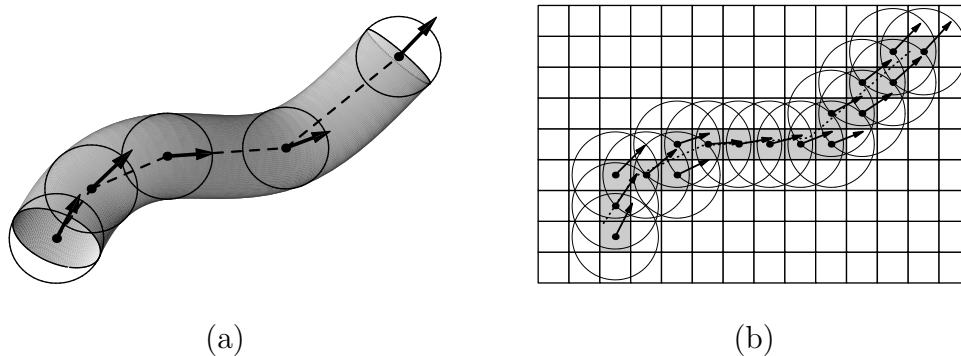


Figura 5.3: (a) Representação de um tubo por uma polilinha. A cada vértice é atribuído o vetor tangente e o raio do tubo. (b) Ilustração da rasterização da polilinha onde os vetores tangentes e raios são linearmente interpolados.

discutidos na Sessão 5.2.

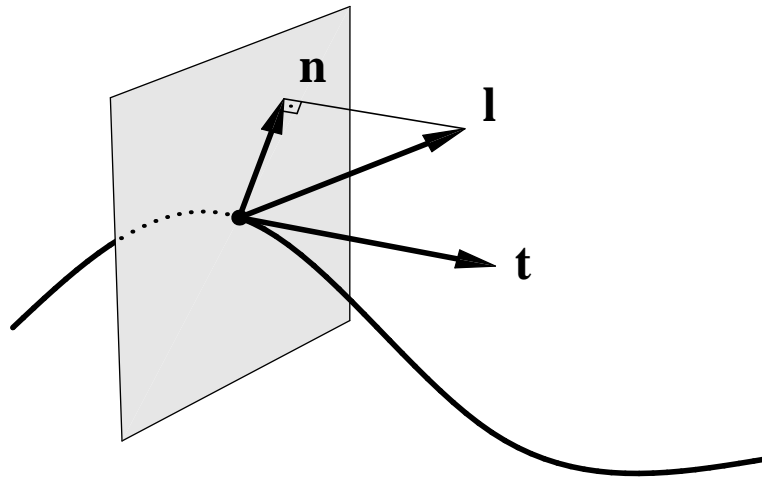


Figura 5.4: Iluminação de tubos: a normal  $n$  da superfície é computada pela projeção do vetor de iluminação  $l$  no plano ortogonal à tangente  $t$ .

### 5.1.3 Exemplo: Folhas e Galhos

A renderização de tubos e fitas pelo algoritmo em pirâmide é demonstrada pela conversão de um modelo poligonal para uma representação baseada em linhas. Mais especificamente, convertamos o modelo de uma macieira utilizado por Deussen e colaboradores em vários trabalhos [54, 55, 56], que é inclusive ilustrado na capa do livro *Physically Based Rendering* [56]. Esta malha contém 200.496 triângulos e 451.215 vértices para representar as folhas, enquanto os troncos são modelados com 351.184 triângulos e 371.583 vértices.

Cada folha do modelo original é modelada com quatro triângulos e seis vértices. Elas foram convertidas para fitas com quatro vértices: os dois extremos e as duas medianas das arestas internas, como ilustrado na Figura 5.5a. Aos raios dos extremos são atribuídos valores próximos de zero, enquanto aos dois vértices interiores são atribuídos metade do comprimento das arestas correspondentes. Desta maneira, foram gerados 150.372 segmentos de fitas e 200.496 vértices. Uma visão magnificada de uma folha é mostrada na Figura 5.6.

Os galhos da macieira, por sua vez, foram convertidos para um conjunto de tubos. A malha poligonal de cada galho consiste em sequências de anéis de vértices conectados por triângulos alongados. A Figura 5.5b demonstra um exemplo com

anéis de cinco vértices. A média dos vértices de cada anel é computada gerando um único vértice na representação por linhas. Os centros de anéis adjacentes, i.e., ligados por triângulos, são conectados para formar as polilinhas. O raio do vértice da polilinha é determinado pela distância máxima a todos os vértices do anel. Os vetores tangentes são facilmente computados pela média das direções das arestas conectadas. Os galhos foram convertidos para 63.887 segmentos e 69.604 vértices.

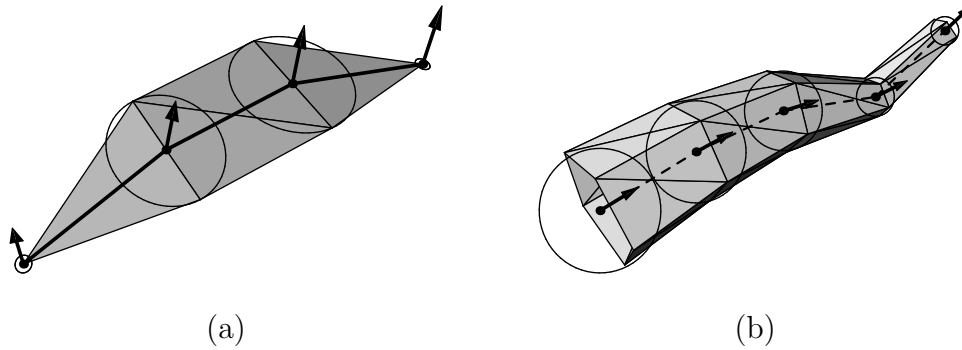


Figura 5.5: Ilustração da conversão de (a) folhas e (b) galhos para polilinhas.

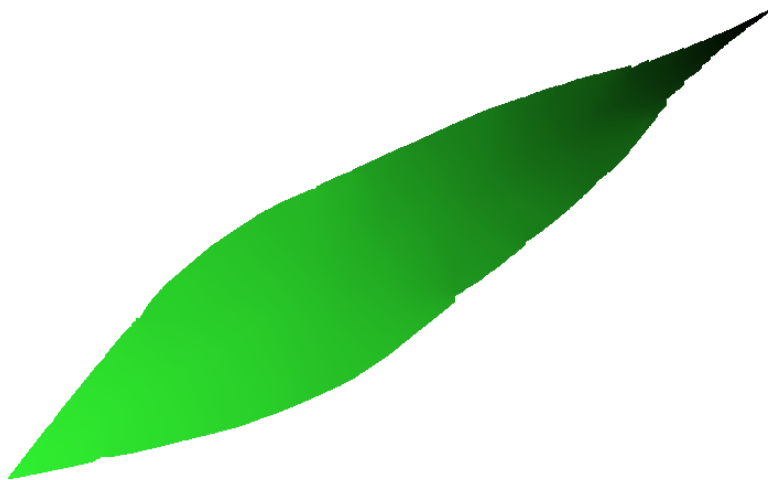


Figura 5.6: Uma magnificação de uma folha renderizada como uma fita.

No entanto, os triângulos do tronco principal não foram convertidos, já que tubos não aproximam bem estruturas grossas. Portanto, os 227 triângulos e 681 vértices correspondentes ao tronco foram mantidos.

O modelo original e o convertido são apresentados na Figura 5.7. A implementação proposta atinge 31qps contra 28qps do modelo triangular. Entretanto, quando o número de primitivas cresce, a diferença também aumenta, e.g., o algoritmo em pirâmide renderiza dez cópias do modelo a 15qps contra 9qps do algoritmo

para triângulos (Figura 5.9). No mais, apenas um terço dos vértices são necessários para a representação por tubos e fitas.

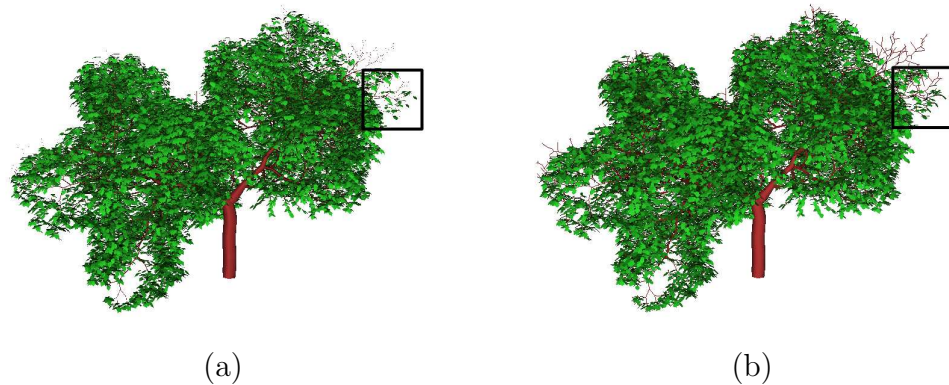


Figura 5.7: Comparação entre (a) o modelo original representado por triângulos e (b) a representação por linhas. As áreas retangulares marcadas representam as magnificações da Figura 5.8.

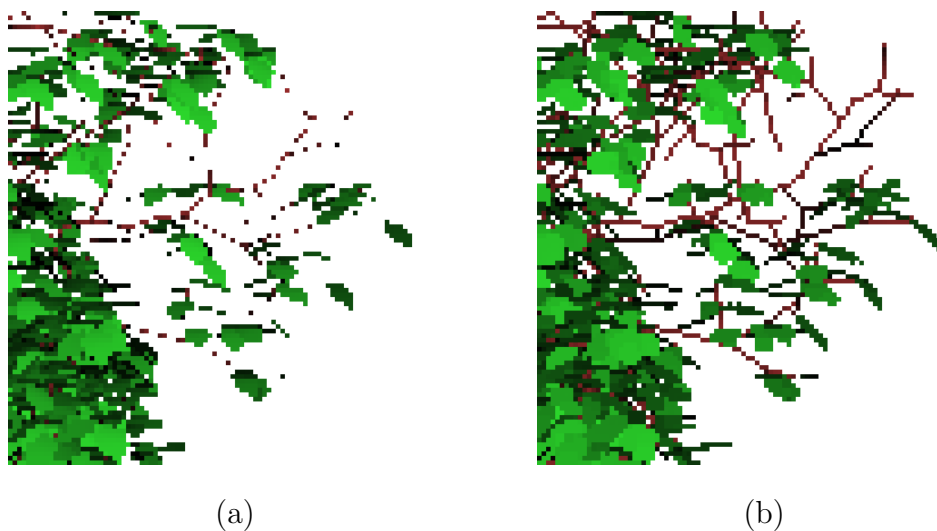


Figura 5.8: (a) Magnificação da imagem gerada com triângulos na Figura 5.7a e (b) da renderização com linhas na Figura 5.7b.

## 5.2 Combinando Pontos, Linhas e Triângulos

É improvável que uma primitiva seja ótima para todos elementos de uma cena. Ainda mais, ao utilizar técnicas de nível de detalhe dinâmicos, o mesmo objeto pode ser representado por primitivas diferentes dependendo do nível requerido. Por



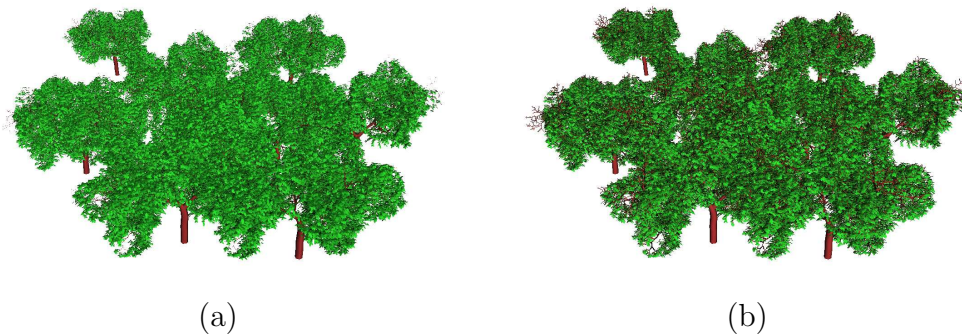


Figura 5.9: Renderização de dez árvores com (a) o modelo triangular e (b) o modelo baseado em linhas.

consequente, é importante que o algoritmo seja capaz de tratar diferentes tipos de primitivas dentro da mesma metodologia.

As fitas e tubos podem ser facilmente renderizados junto aos pontos no algoritmo em pirâmide, bastando um identificador por pixel para diferenciar o tipo de primitiva. Todavia, a renderização de polígonos apresenta desafios adicionais. Uma solução é realizar um método em duas passadas, onde os modelos poligonais e por pontos são renderizados separadamente e com valores de profundidade. As imagens podem ser então combinadas de acordo com as profundidades e valores de transparência. Enquanto este método oferece algumas vantagens por ser capaz de renderizar objetos semi-transparentes, seu custo é mais elevado por necessitar de duas passadas diferentes de renderização.

Felizmente, é possível renderizar polígonos de maneira trivial com o algoritmo proposto. Para tanto, basta atribuir raios próximos de zero para cada vértice do polígono. Desta forma, os pixels interpolados pelo processo de rasterização representam pontos com raios menores do que o de um pixel, i.e., não são expandidos na interpolação da pirâmide. Como o conteúdo da matriz de pixels não afeta a performance do algoritmo, esta técnica não impõe um custo adicional ao método. Um exemplo de uma cena com um modelo poligonal e outro baseado em pontos pode ser observado na Figura 5.10.

Adicionalmente, modelos híbridos podem ser renderizados facilitando a passagem entre tipos de primitivas nas estruturas de multi-resolução ou níveis de detalhe. A Figura 5.11 ilustra um exemplo de um modelo renderizado com dois tipos de primitivas diferentes. De forma similar, modelos representados por linhas podem

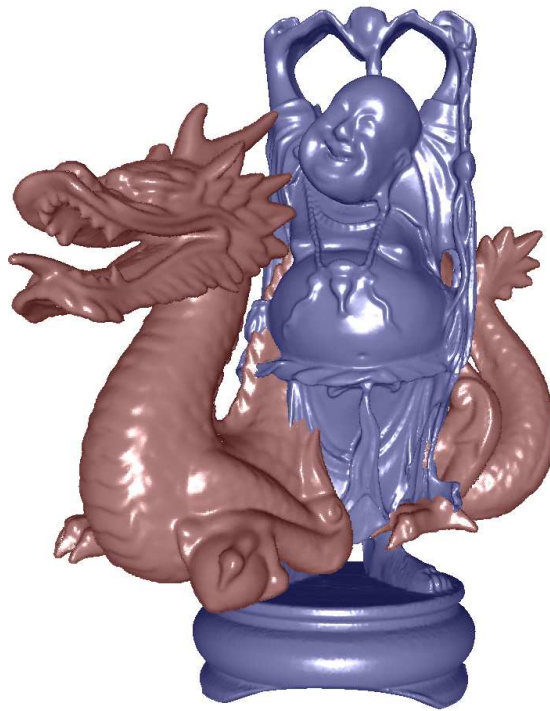


Figura 5.10: Renderização combinando dois tipos de primitivas: pontos (Dragão) e triângulos (Buddha).

ser adicionados à cena.

### 5.3 Discussão

A utilização de triângulos para representar estruturas finas, como fios de cabelo, pêlos, grama, etc., geralmente requer mais vértices do que uma representação por linhas de qualidade similar. Além disso, a simplificação de um modelo baseado em linhas geralmente é mais trivial do que para triângulos. Mesmo que os hardware gráficos sejam especializados em tratar triângulos, a renderização de linhas também é bem suportada, i.e., renderizar a aresta mais longa de um triângulo não é mais custoso do que renderizar o triângulo. A abordagem por linhas também requer menos fragmentos e portanto tende a projetar menos vértices. Por estes motivos, e por possuir custo de interpolação constante, o método baseado em pirâmide oferece uma grande vantagem, especialmente para modelos que ocupam uma grande porção da tela.

Um característica importante deste algoritmo é que, por não aplicar técnicas de

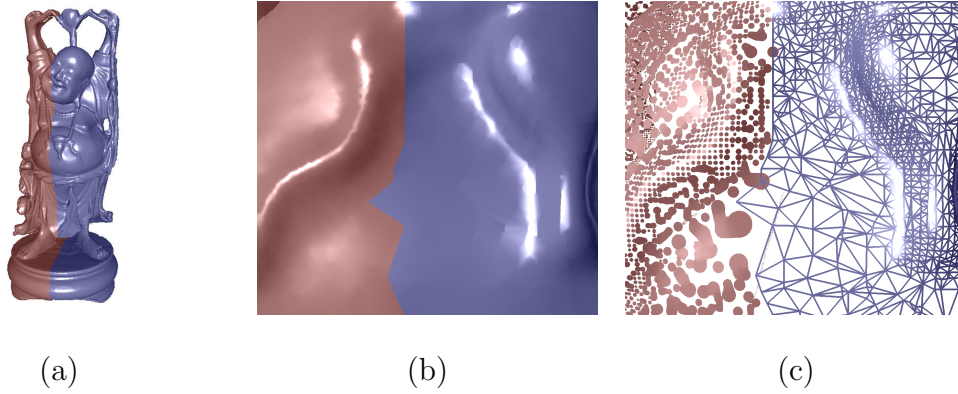


Figura 5.11: (a) Um modelo híbrido com triângulos (azul) e pontos (vermelho). (b) Um detalhe do modelo e (c) o mesmo detalhe com as arestas dos triângulos e pontos com apenas uma fração dos raios.

*anti-aliasing*, as fitas e tubos são renderizados com pelo menos um pixel de espessura. Sendo assim, estruturas finas apareceram mais grossas com a renderização por linhas comparada com a renderização por triângulos. Este efeito pode ser observado em um detalhe magnificado da macieira na Figura 5.8. Note que a magnificação é feita na imagem renderizada sem utilizar métodos de interpolação, e não aproximando a camera do modelo.

De um modo geral, o algoritmo proposto para renderizar linhas no espaço de imagem oferece taxas altas de interatividade. Porém, por não ser apropriado para estruturas grossas, é ideal para representações de baixa resolução para níveis de detalhe dinâmicos, por exemplo.

# Capítulo 6

## Pirâmide com Núcleos de Busca

Um problema do algoritmo descrito no Capítulo 4 é a criação de novos splats na hierarquia. Ao combinar mais de uma elipse em um nível de menor resolução, alguns pixels inválidos são sintetizados a partir das originais e outros a partir da nova elipse. Isto gera descontinuidades na interpolação, uma vez que nem todas as elipses são integralmente propagadas de volta ao nível 0. Esta inconsistência é gerada pelo fato de uma nova elipse não cobrir as originais e muito menos corresponder à união delas.

Por um lado, ao utilizar estruturas de multi-resolução é inevitável que informação seja perdida em níveis mais altos da pirâmide. Por outro lado, ao tratar dados esparsos, muitos espaços (pixels) não são utilizados em cada nível da estrutura. Sendo assim, para gerar um método mais eficiente é necessário evitar a combinação de elipses. Uma solução para este problema é distribuí-las da melhor forma possível na hierarquia.

O método apresentado a seguir oferece duas vantagens em relação ao método original: primeiro, arranja as elipses na pirâmide de acordo com o tamanho de suas projeções, diminuindo a criação de novos splats durante a fase de construção; segundo, rasteriza fielmente as elipses da hierarquia ao evitar combinações na fase de interpolação.

Brevemente, o algoritmo proposto projeta os pontos de forma idêntica ao algoritmo do Capítulo 4; realiza a fase de análise de forma análoga, porém propagando as elipses somente até o primeiro nível onde possam ser cobertas por uma matriz de  $k \times k$  pixels; reconstrói a superfície realizando uma busca com a mesma matriz em todos os níveis a partir de cada pixel do nível 0; e, finalmente, realiza iluminação

por pixel de forma idêntica ao algoritmo PPR.

## 6.1 Criando a Hierarquia

A pirâmide é gerada de forma similar ao algoritmo do Capítulo 4. No entanto, o conceito da pirâmide é um pouco diferente; neste caso, o pixel de cada nível cresce mantendo a imagem com o mesmo tamanho. As elipses são propagadas até o primeiro nível onde o splat é coberto por uma matriz de  $k \times k$  pixels. Esta matriz também é utilizada como núcleo de busca, garantindo que todos pixels do nível zero são capazes de encontrar os splats na fase de interpolação. Ainda mais, cada pixel do nível zero só deve encontrar um splat uma única vez, garantindo que não hajam múltiplas contribuições do mesmo splat na interpolação. Para tanto, a projeção da amostra é armazenada apenas em um único nível computado da seguinte forma:

$$l = \lceil \log_2 \frac{D}{t} \rceil, \quad (6.1)$$

onde  $D$  e  $t$  são, respectivamente, o raio projetado e o tamanho do núcleo em pixels. A Figura 6.1 demonstra um exemplo da propagação de uma elipse para o seu nível correto.

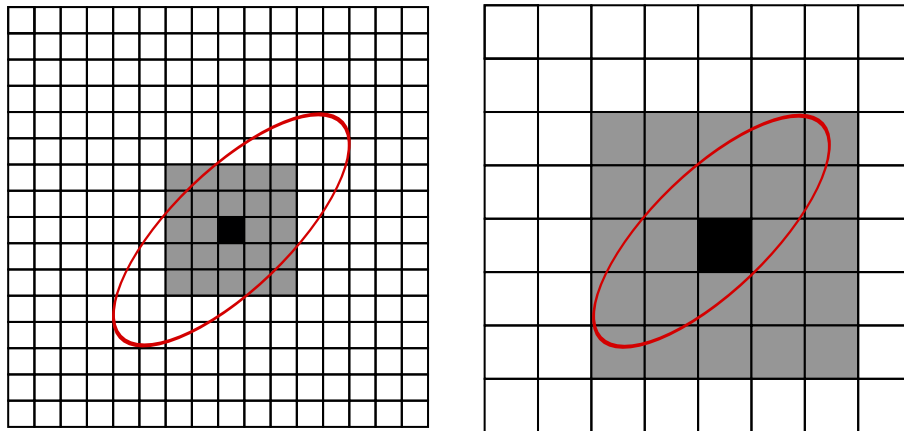


Figura 6.1: O pixel deve ser grande suficiente para que o núcleo cubra toda a projeção da amostra. Utilizando um tamanho de  $5 \times 5$  o nível da figura da esquerda não é suficiente, pois somente os pixels cinzas são capazes de encontrar o centro da projeção em preto. Na figura da direita, o nível possui a cobertura mínima para que todos pixels dentro da elipse encontrem o seu centro.

Mesmo que seja possível projetar a amostra diretamente no nível correto, a

propagação na construção da pirâmide traz algumas vantagens: primeiro, splats propagados a um mesmo pixel ainda devem ser combinados nesta fase; segundo, projetar um pixel para outro nível que não seja o zero significa alterar a posição projetada de um ponto antes que ele passe pelo processador de vértices, o que não é trivial; e finalmente, a fase de construção tem um custo muito inferior à fase de interpolação, e por isso o ganho de performance não é significativo.

Os efeitos de *aliasing* e *flickering* são reduzidos consideravelmente ao armazenar um splat em um único nível. Na implementação original não é garantido que as informações da estrutura sejam propagadas de volta ao nível 0. Como mencionado anteriormente, esta inconsistência provoca rasterizações incompletas de algumas elipses.

Em resumo, uma amostra projetada é propagada na pirâmide até um determinado nível onde é armazenada e registrada como válida. Quando duas ou mais elipses atingem um mesmo pixel elas são combinadas criando uma nova elipse. Neste momento, o mesmo teste de profundidade do algoritmo original é realizado. Note que, quanto maior o tamanho do núcleo menos as projeções são propagadas e, consequentemente, menos elipses novas são criadas.

## 6.2 Interpolando os Splats

A fase de interpolação possui um custo maior do que a fase de construção da pirâmide. Para cada pixel do nível 0 é realizada uma busca em todos níveis com um núcleo de tamanho fixo. Em outras palavras, cada pixel busca informações nos vizinhos do seu pixel correspondente em cada nível.

A única diferença no acúmulo das contribuições é o peso da média ponderada. Ao invés do peso exponencial, é utilizado uma medida linear:

$$w = 1.0 - d. \tag{6.2}$$

Este peso gera uma interpolação mais suave, enquanto o peso exponencial deixa traços claros dos contornos das elipses resultando em uma reconstrução artificial.

Quanto maior o núcleo de busca, menos níveis precisam ser construídos e, por consequência, incluídos na busca. Para um tamanho  $3 \times 3$ , por exemplo, o nível mais alto da pirâmide pode ser excluído, já que o núcleo no nível diretamente abaixo é

suficiente para cobrir todos pixels. A propagação de um núcleo  $3 \times 3$  a partir de uma resolução de  $16^2$  é ilustrada na Figura 6.2. Quanto maior o núcleo, mais acessos à textura por nível são necessários para recuperar os pixels. Por outro lado, núcleos maiores tendem a combinar menos elipses, gerando reconstruções de melhor qualidade.

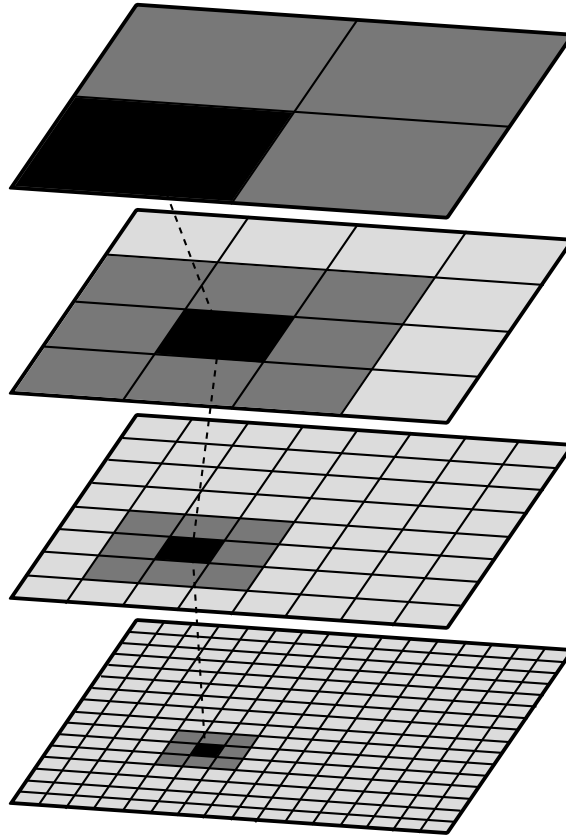


Figura 6.2: As buscas efetuadas por um pixel (nível 0) com um núcleo  $3 \times 3$ . Em preto o pixel correspondente em cada nível e em cinza escuro a cobertura da núcleo. Note que o nível 4 contendo apenas um pixel não é necessário, uma vez que o nível 3 é totalmente coberto.

Para uma resolução típica de  $1024^2$  onze níveis são criados, onde o nível 0 contém a resolução de saída e o nível 10 contém apenas um pixel. Para um núcleo  $3 \times 3$  o último nível não é necessário, por isso cada pixel realiza  $10 \times 3^2 = 90$  buscas. A busca de cada nível é executada em uma passada diferente na GPU e, para cada, uma mais um acesso é necessário para recuperar os atributos acumulados no pixel, somando um total de 100 acessos por pixel nesta configuração. Para efeito de comparação, um algoritmo ingênuo, onde cada pixel busca todos outros do nível zero, realiza  $1024^2$

acessos por pixel.

## 6.3 Discussão

O algoritmo apresentado neste capítulo oferece um compromisso entre qualidade e performance de acordo com o tamanho do núcleo. Um tamanho de  $3^2$  atinge taxas similares ao algoritmo PPR, enquanto o núcleo  $5^2$  atinge renderizações de alta qualidade. Contudo, ao aumentar o tamanho de  $3^2$  para  $5^2$  ocorre uma redução de performance de quase 40%. Por outro lado, núcleos maiores do que  $5^2$  não resultaram em nenhum aumento qualitativo expressivo. Na Figura 6.3 pode-se observar um detalhe da perna do Armadillo renderizado com três núcleos diferentes.

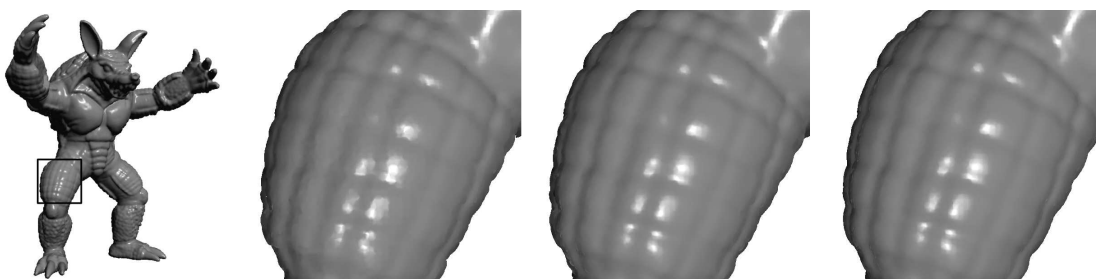


Figura 6.3: A mesma cena renderizada com três tamanhos diferentes de núcleo; da esquerda para direita:  $3^2$ ,  $5^2$  e  $7^2$ . Note que o ganho qualitativo entre os núcleos  $5^2$  e  $7^2$  é imperceptível, porém a performance reduz-se em 38%.

A diferença entre a performance do algoritmo com núcleos e o PPR diminui com o aumento do número de amostras do modelo. Isto é esperado, já que para modelos grandes o tempo é dominado pela fase de projeção. Todavia, para modelos pequenos (Armadillo, Dragão) a diferença entre os algoritmos é de aproximadamente 15%, podendo chegar a 5% para modelos maiores (Netuno, Statuette). Desta forma, as taxas de renderização ainda se mantêm altas em relação ao Surface Splatting, pois o PPR alcançou uma melhoria de 40% em relação ao mesmo. Nas Figuras 6.4 e 6.5 podem ser observadas comparações qualitativas entre o PPR e a interpolação com um núcleo  $5^2$ . Outros modelos são ilustrados nas Figuras 6.6, 6.7 e 6.8.

Pelo fato de criar menos elipses durante a fase de construção da pirâmide e não criar, remover ou alterar as elipses durante a interpolação, o algoritmo apresenta uma robustez maior em relação ao PPR. Apesar de não estar livre de artefatos de *aliasing*



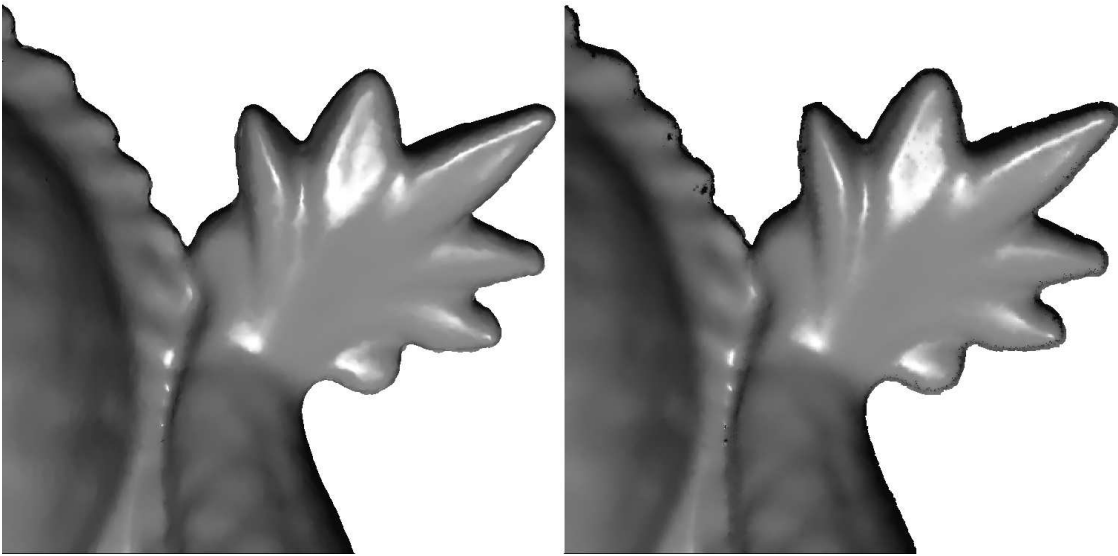


Figura 6.4: Um detalhe do rabo do dragão renderizado com um núcleo  $5 \times 5$  (esquerda) e o algoritmo PPR (direita). Note como a estratégia com núcleos produz silhuetas mais suaves e com menos artefatos.

por falta de tratamento adequado, é observada uma quantidade expressivamente menor, especialmente durante animações.

Uma desvantagem deste algoritmo é o aumento da complexidade. Ao fazer uma busca a partir do nível 0 em todos outros níveis, a complexidade de reconstrução atinge  $O(n \log n)$  contra  $O(n)$  do algoritmo original. Para reverter este aumento do custo é necessário que cada nível seja capaz de reconstruir suas elipses e interpolar os pixels sem recorrer sempre ao nível de maior resolução. Realizando a reconstrução por nível é pelo menos possível realizar uma busca com núcleo de tamanho unitário a partir do nível 0 e, desta forma, reduzir consideravelmente a constante de complexidade.

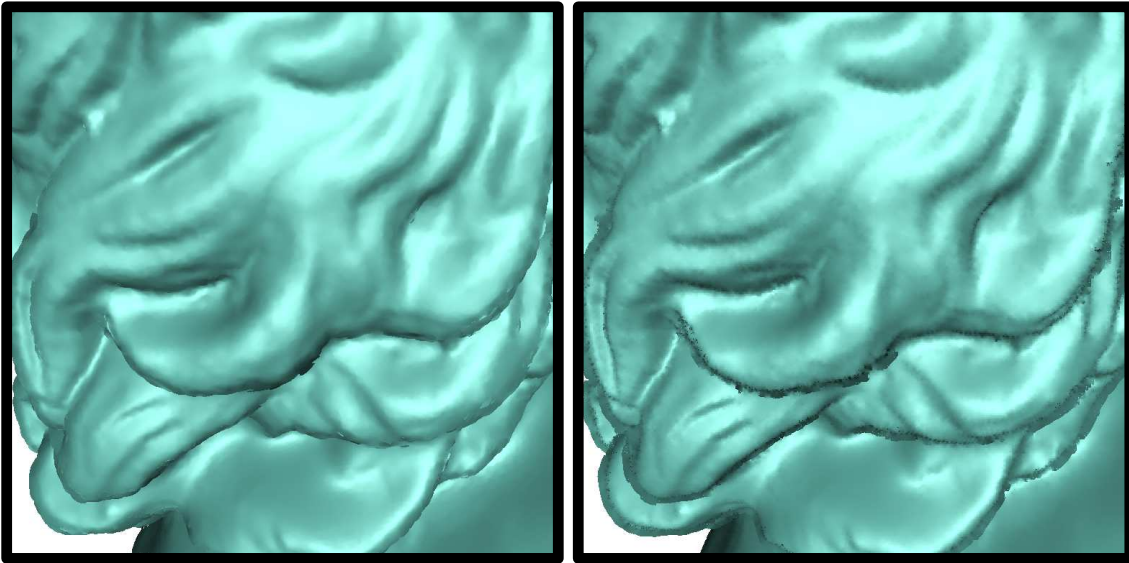


Figura 6.5: Um detalhe da barba do Netuno renderizado com um núcleo  $5 \times 5$  (esquerda) e o algoritmo PPR (direita).



Figura 6.6: Da esquerda para direita: Buddha, Netuno e a Statuette renderizados com o algoritmo de núcleos.

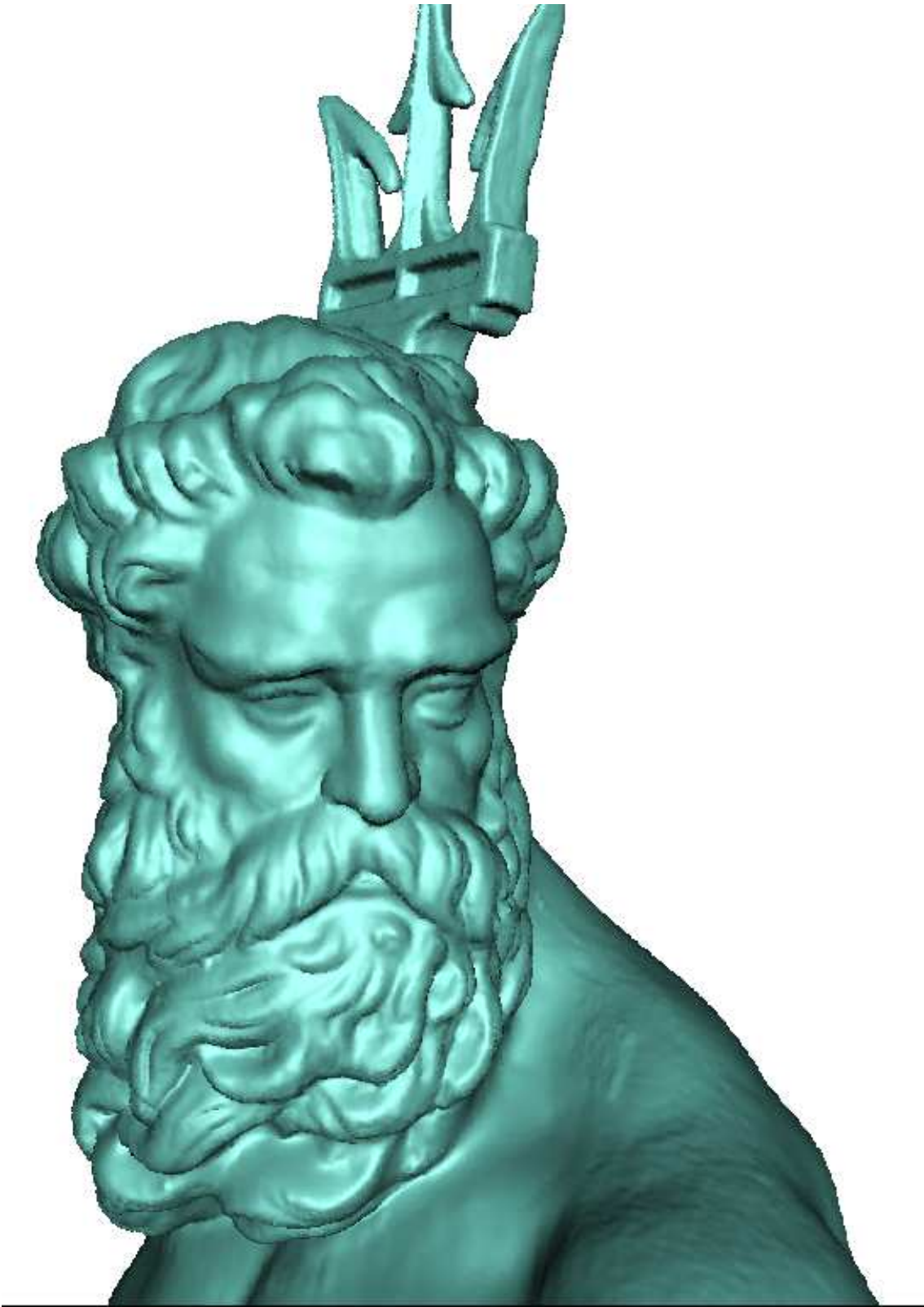


Figura 6.7: Detalhe do Netuno.



Figura 6.8: Detalhe das esculturas na parte inferior do modelo Statuette.

# Capítulo 7

## Níveis de Detalhe para Pontos

Enquanto que o custo da fase de reconstrução da superfície é independente do número de amostras, o mesmo não é verdade para a fase de projeção. Como descrito no Capítulo 4, o tempo de reconstrução pode chegar a poucos milisegundos, tendendo a diminuir mais com as novas gerações de placas gráficas. Por este motivo, para alcançar taxas de renderização mais altas é preciso diminuir o tempo da primeira fase. Como o custo de projeção por vértice é baixo, basicamente de duas multiplicações de vetor-matriz, é necessário diminuir o número de amostras projetadas. Uma maneira popular é utilizar técnicas de níveis de detalhe para determinar um conjunto mínimo de vértices a serem projetados dado um ponto de vista.

Na Sessão 7.1 é introduzida a **Textura de Objeto** para armazenar a estrutura hierárquica de pontos, enquanto que uma técnica para selecionar o nível de detalhe apropriado utilizando recursos recentes de programação em placa gráfica é descrita na Seção 7.2.

### 7.1 Texturas de Objeto

Para criar a estrutura de multi-resolução do modelo introduzimos o conceito de **Texturas de Objeto**. Esta textura armazena as primitivas agrupadas de forma a representar um retalho da superfície. Cada retalho possui um vértice representativo responsável por acessar os demais durante o processo de rasterização. Este vértice possui informações de quantas primitivas o retalho contém e a posição da primeira delas na Textura de Objeto.

O objetivo desta técnica é limitar o fluxo de dados entre a CPU e a GPU, um dos gargalos em muitas aplicações interativas. Ao armazenar a textura na memória da placa gráfica é necessário enviar apenas os vértices representativos para a GPU, onde os demais são recuperados sob demanda.

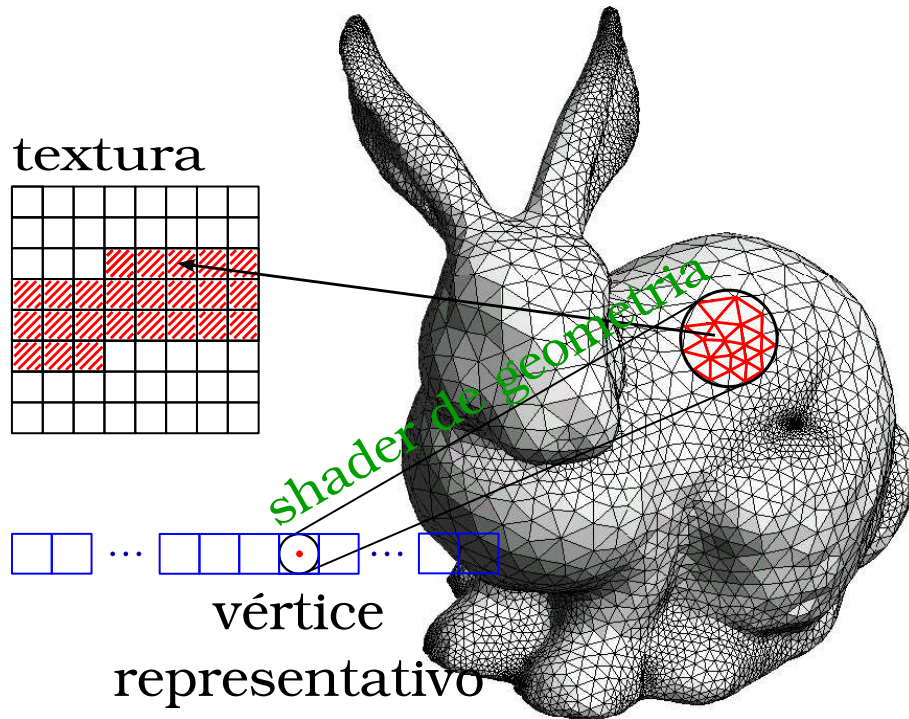


Figura 7.1: Durante a renderização, o vértice representativo é capaz de recuperar um retalho da superfície armazenado na Textura de Objeto, utilizando os recursos do *shader* de geometria.

### 7.1.1 Estrutura Hierárquica

Uma das principais vantagens da representação por pontos é que estruturas de multi-resolução são intuitivas e naturais: primeiro, pontos são adimensionais e, portanto, são facilmente limitados a ocupar apenas um nó da estrutura; segundo, por não haver preocupação em manter conectividade e respeitar restrições topológicas, a decimação da representação é geralmente simples.

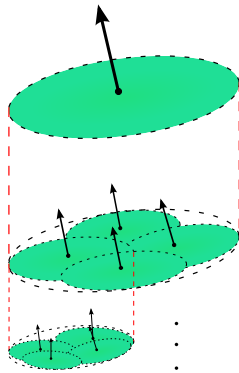
A estrutura proposta neste trabalho é construída de forma trivial com o objetivo de ilustrar o conceito das Texturas de Objeto e suportar a proposta de diminuir o custo de projeção. Uma *kd-tree* é utilizada para realizar a subdivisão espacial. O centro de massa dos pontos contidos em um nó é utilizado para realizar a subdivisão,



enquanto os eixos  $x$ ,  $y$  e  $z$  são alternados como eixo de subdivisão.

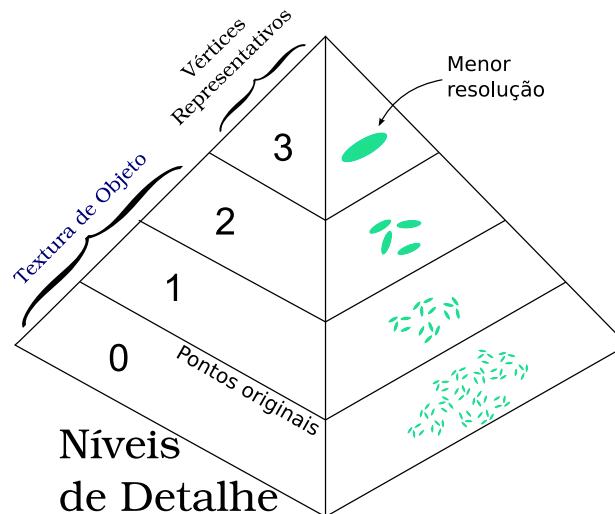
Os nós folha contêm no máximo um ponto do modelo, enquanto cada nó interno possui quatro filhos; portanto, o segundo nível da hierarquia é construído combinando as amostras de quatro folhas em um único splat e assim sucessivamente, como ilustrado na Figura 7.2a. Nesta implementação são criados quatro níveis, onde o nível de menor resolução armazena os vértices representativos da textura de objeto. No entanto, o número de níveis pode ser facilmente estendido. Desta forma, um vértice representativo contém quatro níveis de detalhes do retalho da superfície coberto pela sua extensão. Como podem existir nós folhas vazios, um retalho contém no máximo 4 vértices do nível dois, 16 do nível um, 64 do nível zero e exatamente um vértice do nível três (representativo). Para cada retalho, os vértices são ordenados decrescentemente por nível antes de serem armazenados na textura, ou seja, primeiro o nível 2, depois o 1 e finalmente o nível 0. A posição do primeiro vértice e a quantidade de vértices por nível são armazenadas junto ao vértice representativo. A Figura 7.2b ilustra a representação da hierarquia.

## Resolução Baixa



## Resolução Alta

(a)



## Níveis de Detalhe

(b)

Figura 7.2: (a) Os *splats* são agrupados para formar os níveis de resolução mais baixos. (b) Representação dos níveis de detalhe na hierarquia: o nível mais alto contém os vértices representativos capazes de recuperar os demais níveis armazenados na Textura de Objeto.

### 7.1.2 Métrica de Erro

Dado um ponto de vista, o nível de um retalho a ser projetado é determinado por uma métrica baseada na distância do vértice representativo ao observador e na projeção de seu vetor normal. Em outras palavras, quanto mais longe estiver o vértice representativo do observador, menor o nível de resolução requerido. Por outro lado, para evitar efeitos de *aliasing* nas silhuetas e bordas, quanto menor o ângulo entre a normal e o plano de imagem, maior o nível de resolução. Para tanto, utilizamos a métrica de erro perpendicular proposta por Dachsbacher et al. [57]:

$$e_p = \max\{((c_i - p) \cdot n) + d_i\} - \min\{((c_i - p) \cdot n) - d_i\}$$

$$\text{com } d_i = r_i \sqrt{1 - (n_i \cdot n)^2}.$$

O erro perpendicular  $e_p$  é a distância entre o plano tangente do splat pai aos dois planos paralelos que cobrem as extensões dos filhos, como pode ser observado na Figura 7.3.

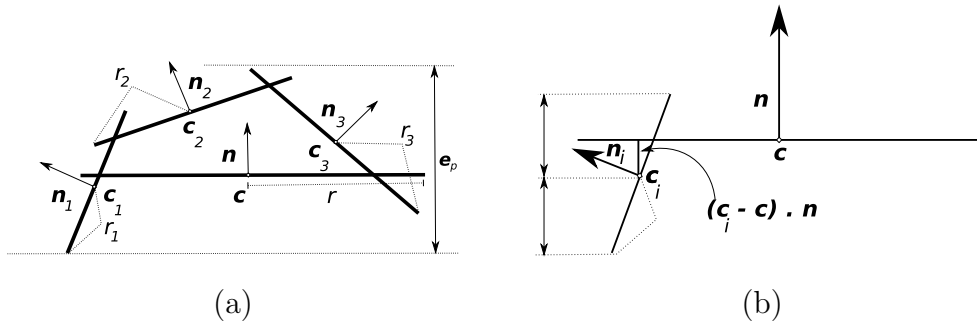


Figura 7.3: O erro perpendicular é a distância máxima entre o plano tangente ao splat pai com centro em  $c$  e as extensões dos filhos com centros em  $c_i$ .

Para que o erro reflita a necessidade de mais detalhes ao longo das silhuetas, ele precisa ser projetado para o plano de imagem durante a renderização, ou seja, ele é dependente do ponto de visão. A projeção do erro perpendicular  $\tilde{e}_p$  é dada por:

$$\tilde{e}_p = e_p \frac{\sin(\alpha)}{r}, \quad \text{com } \alpha = \angle(v, n),$$

onde  $v$  é o vetor de visão e  $r = |v|$ .



## 7.2 Projeção dos Pontos

Os vértices representativos, i.e., nível de menor resolução, são enviados para a GPU da mesma maneira do que no algoritmo original. Porém, o erro associado à cada também é projetado e utilizado como critério determinante do nível de resolução. É realizada uma comparação simples, baseada em um parâmetro de erro mínimo  $\epsilon$ . Note que, se o nível escolhido for o de menor resolução, nenhum vértice é recuperado da textura de objeto.

Para recuperar os vértices durante a projeção é necessário que a placa gráfica seja capaz de criar geometria dinamicamente. Felizmente, as placas que suportam programação no *Geometry Shader* são capazes de criar ou alterar as primitivas geométricas em tempo de renderização. Assim, após determinar o nível correto, as quantidades de vértices por nível são utilizadas para computar o deslocamento a partir da posição inicial na textura. Os vértices são então recuperados da Textura de Objeto e projetados.

## 7.3 Discussão

Neste primeiro protótipo não houve melhoria na performance, inclusive podendo as vezes perder para o algoritmo sem níveis de detalhe. Para habilitar o *Geometry Shader* é necessário indicar alguns parâmetros, como por exemplo, tipo e quantidade de primitiva de saída. Porém, o tempo de execução diminui expressivamente com o aumento da quantidade máxima de primitivas de saída, mesmo que na prática um número muito menor esteja sendo utilizado. Isto pode ser confirmado em uma situação onde o modelo é inteiramente renderizado com o menor nível de resolução, ou seja, não há acesso à textura de objeto. Mantendo esta configuração e modificando o parâmetro de número máximo de primitivas, a performance varia expressivamente mesmo que o algoritmo esteja sendo executado de forma idêntica. Devido a esta restrição a proposta não apresentou o ganho esperado, apesar da quantidade de vértices enviadas à GPU ser em torno de uma ordem de magnitude menor. No entanto, esperamos que com otimizações do *Geometry Shader* nas novas gerações de hardware gráficos esta limitação seja superada.

Alguns exemplos de modelos renderizados utilizando a estrutura de multi-resolução

podem ser observados nas Figuras 7.4, 7.5 e 7.6.

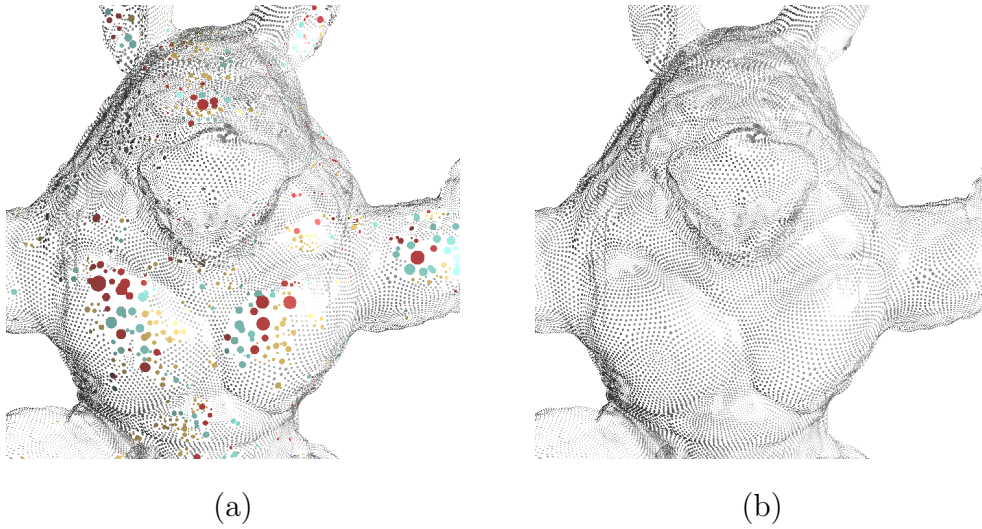


Figura 7.4: (a) Armadillo renderizado utilizando os níveis de detalhe, onde as cores cinza, amarelo, azul e vermelho correspondem respectivamente aos níveis 0, 1, 2 e 3. (b) O mesmo modelo sem níveis de detalhe, ou seja, apenas os splats do nível 0. Note como as regiões suaves, como o peitoral do modelo, são renderizadas com splats de menor resolução. Para efeitos de ilustração os splats são renderizados com apenas uma fração do raio.

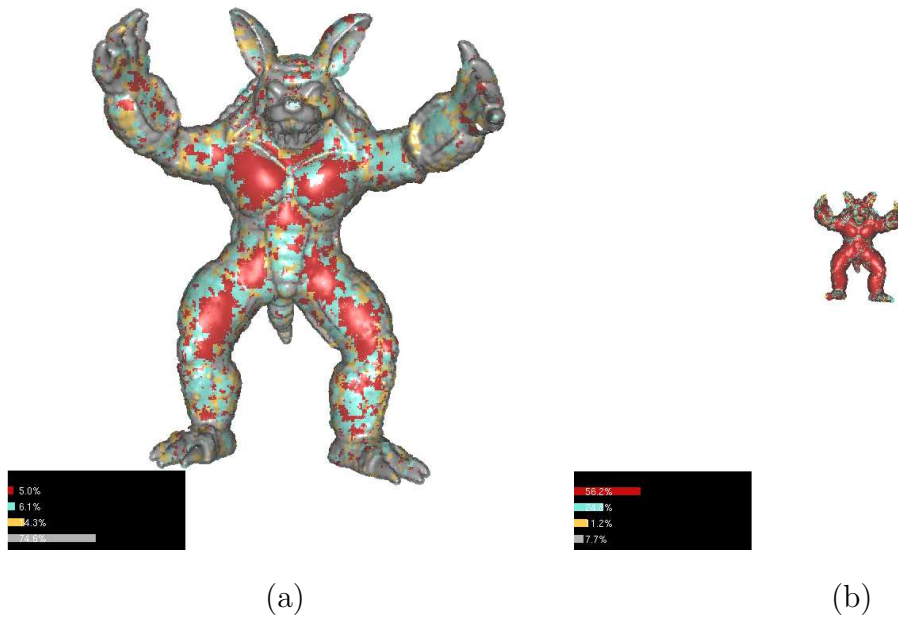


Figura 7.5: Armadillo renderizado com níveis de detalhe (a) de um ponto de vista próximo e (b) de um ponto de vista distante. O grafo em barras indica a porcentagem de amostras renderizadas de cada nível, onde as cores cinza, amarelo, azul e vermelho correspondem respectivamente aos níveis 0, 1, 2 e 3.

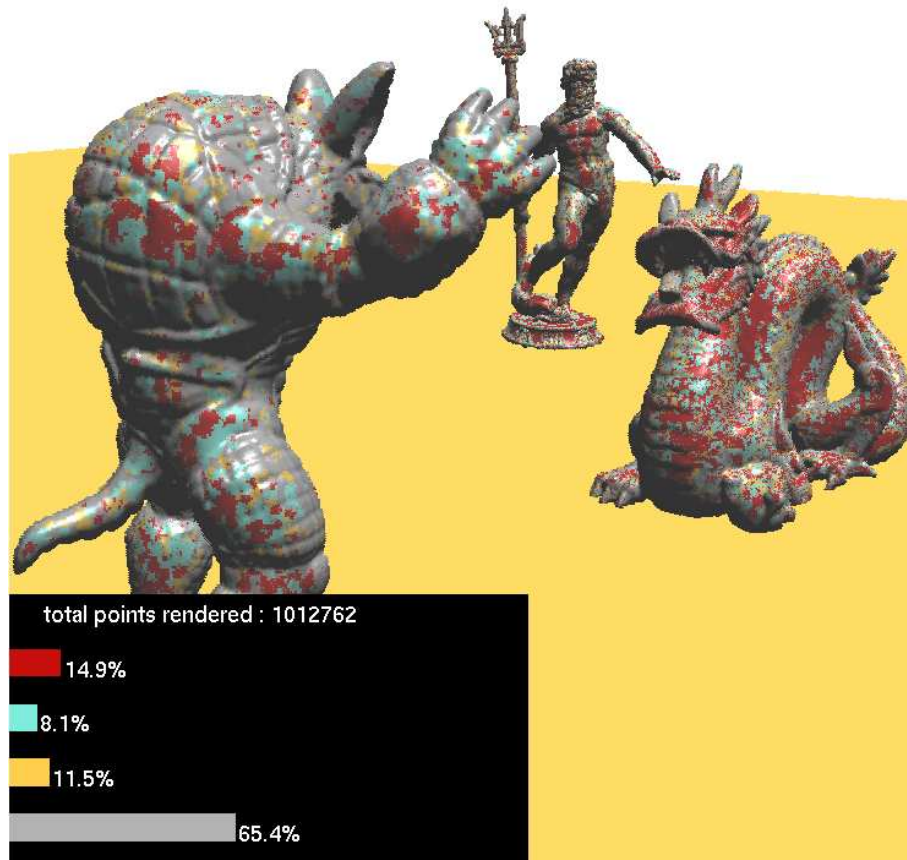


Figura 7.6: Uma cena contendo três modelos (Armadillo, Netuno e Dragão) renderizada com níveis de detalhe. O grafo indica a porcentagem de pontos renderizados em cada nível, onde as cores cinza, amarelo, azul e vermelho correspondem respectivamente aos níveis 0, 1, 2 e 3.

# Capítulo 8

## Conclusões

Nesta tese são apresentados métodos de renderização de modelos de pontos utilizando técnicas de renderização baseadas em imagem. Ao utilizar estruturas hierárquicas no espaço de imagem, como descrito no Capítulo 4, é possível atingir complexidade  $O(s + n)$ , onde  $s$  é o número de amostras e  $n$  o número de pixels. Mais especificamente, a primeira fase de projeção é executada em  $O(s)$  enquanto a fase de reconstrução possui custo  $O(n)$ . A simplicidade e flexibilidade da abordagem permite que outras primitivas, e.g., linhas e triângulos, sejam incorporadas ao método sem custos adicionais, como demonstrado no Capítulo 5.

Adicionalmente, é demonstrado que com variações mais fiéis à rasterização das elipses, como o algoritmo apresentado no Capítulo 6, é possível diminuir os artefatos criados pela reamostragem sem recorrer a complexidades associadas ao número de amostras. Um aumento qualitativo significativo é introduzido, apesar de elevar o tempo de reconstrução.

Diferentemente dos algoritmos que se baseiam exclusivamente na imagem como primitiva gráfica, como é o caso do *Light Field* [2], a fase de projeção ainda impede um desvinculo completo da complexidade com a geometria do modelo. No entanto, existem técnicas para diminuir expressivamente o custo de projeção, como a abordagem de níveis de detalhe apresentada no Capítulo 7.

A área de gráficos baseados em pontos se apresentou como uma linha robusta e de grande interesse, trazendo benefícios importantes para certas aplicações. Uma tendência atual é a junção das linhas de pontos e modelos volumétricos em uma área chamada *Gráficos Baseados em Amostras*. Muitas frentes nesta linha de pesquisa

ainda estão abertas para aprofundamento. Após um impulso inicial, onde naturalmente muitos métodos originalmente desenvolvidos para malhas foram repensados para pontos, uma concentração de esforços mais refinada começa a emergir. Em outras palavras, aplicações práticas surgem utilizando a teoria e protótipos desenvolvidos por pesquisadores nestes últimos anos.

Como exemplo, pode-se citar uma nova tecnologia de Video-3D desenvolvido pela LiberoVision associada ao grupo de computação gráfica do ETH Zurich. Esta aplicação, já utilizada por algumas emissoras de televisão (e.g., ESPN Axis), permite replays interativos em jogos de futebol. Outra empresa associada ao mesmo grupo de pesquisa, a Cyfex, também desenvolve aplicações para modelos baseados em pontos, como por exemplo, um plugin para web permitindo a visualização de modelos de pontos interativamente. Uma justificativa é que este tipo de representação pode ser eficientemente compactada e transmitida pela web por não conter informações de conectividade.

Outro exemplo é a colaboração do grupo Visual Computing Lab do ISTI-Pisa na concepção de ferramentas para restauração e documentação de herança cultural[58]. Uma característica desta área é a importância da fidelidade dos dados, ou seja, existe uma preferência por somente dados amostrados ao invés de interpolados, gerando uma demanda por modelos de alta resolução. Obviamente sempre haverá uma magnificação suficientemente grande onde as amostras serão projetadas de forma esparsa, porém este é um problema oriundo da discretização da amostragem.

Mais especificamente, a área de renderização a representação baseada em pontos tem se mostrado capaz de contornar as especificidades do hardware em rasterizar triângulos. Isto é demonstrado pela eficiência e qualidade das imagens geradas diretamente a partir de nuvens de pontos. A relação das propriedades entre pontos e pixels admite uma facilidade ao trabalhar em espaço de imagem e, mais ainda, fornece vantagens ao programar nas placas gráficas. A natureza discreta e matricial da imagem torna os algoritmos altamente adaptáveis ao modelo de programação em GPU.

A renderização baseada em imagem não permite apenas alcançar taxas interativas para um número grande de amostras, mas se adequa apropriadamente com a evolução dos equipamentos de aquisição e saída. Enquanto a resolução de aquisição

crece constantemente, permitindo capturar mais detalhes dos objetos reais, a resolução dos equipamentos de exposição evolui lentamente. No entanto, existe um limite físico de percepção visual humana onde aumentos de resolução passam a ser insignificantes para um mesmo tamanho de área de visualização a uma mesma distância de observação. Neste sentido, abordagens que possuem custo proporcional ao número de pixels permanecem eficientes ao longo da evolução das tecnologias gráficas.

Como enfatizado neste trabalho, a representação por pontos não objetiva substituir outras. Cada primitiva ou técnica de renderização possui vantagens e desvantagens, sendo necessário discernir quando utilizar cada uma. Pontos, por não aproximarem regiões planares tão bem quanto malhas poligonais, geralmente não são ideais para visualizações sujeitas a grande magnificações, i.e., quando a densidade dos pontos se afasta da densidade dos pixels. Embora sob grandes magnificações a maior parte da superfície visualizada seja reconstruída por interpolação, os triângulos geram resultados visualmente mais agradáveis. No entanto, esta vantagem é revertida quando a complexidade da cena ultrapassa a complexidade da imagem. Na Figura 8.1, retirada originalmente de [28], incluímos uma faixa onde pontos geralmente trazem mais benefícios.

Contudo, a renderização baseada em imagem ainda possui limitações, especialmente devido à problemas de reamostragem. Quando a complexidade da imagem é menor do que a complexidade do objeto, inevitavelmente haverá uma sub-amostragem no processo de renderização e, por isso, o método se torna mais propenso a *aliasing*. Ao projetar uma nuvem de pontos densa para o espaço de imagem é inevitável que surjam regiões com mais amostras do que pixels. A sub-amostragem implica não só em *aliasing* mas também em perda de detalhes. Certamente, este problema não depende somente se a abordagem é em espaço de imagem ou em espaço de objeto; no entanto, ao descartar pixels durante a fase de projeção, e.g., utilizando um teste de oclusão, a influência da amostra também é excluída do computo do valor de alguns pixels.

Em contrapartida, um paradigma fundamental da computação gráfica é o equilíbrio entre performance e qualidade. Até o momento, não existe um método ótimo nos dois sentidos. Sendo assim, acreditamos que a abordagem apresentada nesta tese seja um

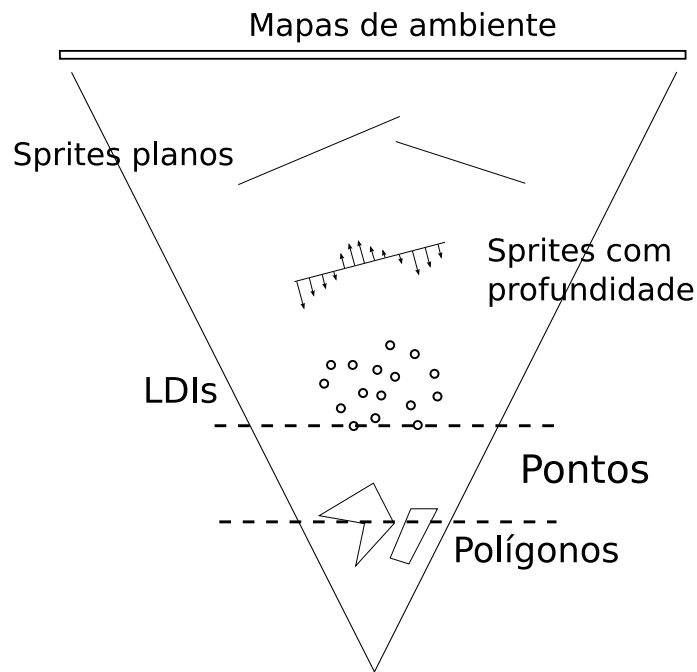


Figura 8.1: Cada tipo de primitiva é mais vantajosa em uma faixa em relação a distância do observador. Pontos são incluídos em uma faixa entre os polígonos e estruturas regulares baseadas em imagem, e.g., LDIs.

compromisso entre esses objetivos: atinge taxas altas de renderização mantendo a qualidade para uma grande faixa de distâncias de observação. Ainda que não alcance os resultados do Surface Splatting, especialmente das abordagens que incluem propriedades diferenciais, esta margem qualitativa pode ser estreitada incluindo filtros *anti-aliasing* e novas propriedades de superfície no esquema em pirâmide.

Um objetivo importante de uma representação é diminuir o tempo de criação de conteúdo. Para aplicações que exigem modelos de alta resolução, a aquisição, reconstrução, processamento e visualização se tornam muitas vezes demasiados custosos. Aonde possível, simplificar a representação pode trazer uma redução do custo total do processo. Isto não só ajuda a popularizar aplicações, como por exemplo, a digitalização e impressão 3D, mas também a promover e elevar a contribuição da área de computação gráfica como um todo ao resolver problemas práticos, entre os quais pode-se citar: rápidos diagnósticos de imagens médicas; liberdade e rapidez na criação de conteúdos de expressão humana, e.g., artes visuais; interatividade e realismo em aplicações de entretenimento; manufatura e reprodução de equipamentos; transmissão de dados; documentação e recuperação de artefatos culturais; são



apenas alguns exemplos onde contribuições são possíveis.

## 8.1 Trabalhos Futuros

Existem algumas extensões óbvias para os algoritmos apresentados. Certamente acrescentar filtros *anti-aliasing* é uma necessidade evidente. É possível notar artefatos de *flickering* durante a manipulação do modelo, especialmente nas silhuetas e bordas. Apesar dos artefatos terem sido consideravelmente reduzidos com o algoritmo descrito no Capítulo 6, ainda é preciso suavizar mais o efeito. Controlar a transparência das silhuetas é uma opção viável, porém identificar as bordas nem sempre é trivial.

Outro ponto essencial para enriquecer o trabalho é a inclusão de sombras. Existem métodos para computar sombras no espaço de imagem onde é possível manter a complexidade associada somente à resolução da imagem e ao número de fontes de luz, como por exemplo o algoritmo desenvolvido por Agrawala et al. [59]. Incrementar o sistema de iluminação permitindo novas fontes de luz e efeitos de iluminação global também é importante para aumentar o realismo e a qualidade da solução.

Para tornar a renderização mais eficiente é preciso depurar de forma minuciosa a relação entre a resolução do modelo e a resolução da imagem. Projetar com uma resolução baixa, i.e., dados muito esparsos no espaço de imagem, causa renderizações de menor qualidade por interpolar muitos pixels entre as amostras; por outro lado, dados muito densos provocam uma sobrecarga desnecessária na fase de projeção por haver muitas amostras concorrentes por pixel. Encontrar um balanço ótimo entre as resoluções não é trivial, porém, eleva a eficiência do algoritmo, especialmente ao utilizar níveis de detalhe.

Apesar de atingir taxas altas de renderização, o pré-processamento para o cálculo de raios e normais ainda é muito custoso. Técnicas para computar esses atributos diretamente no espaço de imagem são úteis para reprodução em tempo real dos dados adquiridos. Evidentemente, esta estratégia não atingirá a precisão de uma abordagem utilizando todas as amostras em espaço de objeto, mas permite caminhar em direção a uma solução de visualização de qualidade em tempo real do processo de registro. Outra área onde é interessante calcular os atributos das amostras em tempo

real é a de deformação, já que o raio muda dinamicamente com as transformações.

Video-3D também é uma área promissora para modelos baseados em pontos. Registros de cameras podem ser transformados rapidamente em nuvens de pontos para serem visualizadas e manipuladas interativamente. Porém, esta tarefa não consiste em apenas visualizar pontos como em uma cena estática, para evitar o envio de nova geometria para GPU a cada quadro é preciso computar as transformações das amostras na sequência do vídeo. Isto permite que a mesma geometria armazenada na placa gráfica seja utilizada a cada quadro e a cena renderizada eficientemente.

# Referências Bibliográficas

- [1] GORTLER, S. J., GRZESZCZUK, R., SZELISKI, R., et al., “The Lumigraph”. In: *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pp. 43–54, ACM Press: New York, NY, USA, 1996.
- [2] LEVOY, M., HANRAHAN, P., “Light field rendering”. In: *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pp. 31–42, ACM: New York, NY, USA, 1996.
- [3] CHANG, C.-F., BISHOP, G., LASTRA, A., “LDI tree: a hierarchical representation for image-based rendering”. In: *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pp. 291–298, ACM Press/Addison-Wesley Publishing Co.: New York, NY, USA, 1999.
- [4] GROSSMAN, J. P., DALLY, W. J., “Point Sample Rendering”. In: *Proceedings Eurographics Workshop on Rendering Techniques '98*, pp. 181–192, June 1998.
- [5] PFISTER, H., ZWICKER, M., VAN BAAR, J., et al., “Surfels: Surface Elements as Rendering Primitives”. In: *Siggraph 2000, Computer Graphics Proceedings*, pp. 335–342, ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.
- [6] RUSINKIEWICZ, S., LEVOY, M., “QSplat: a multiresolution point rendering system for large meshes”. In: *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pp.

343–352, ACM Press/Addison-Wesley Publishing Co.: New York, NY, USA, 2000.

- [7] ZWICKER, M., PFISTER, H., VAN BAAR, J., et al., “EWA Splatting”, *IEEE Transactions on Visualization and Computer Graphics*, v. 8, n. 3, pp. 223–238, 2002.
- [8] KALAIHAH, A., VARSHNEY, A., “Modeling and Rendering of Points with Local Geometry”, *IEEE Transactions on Visualization and Computer Graphics*, v. 9, n. 1, pp. 30–42, 2003.
- [9] BOTSCH, M., SPERNAT, M., KOBBELT, L., “Phong Splatting”. In: *PBG '04: Proceedings of the 1st Eurographics Symposium on Point-Based Graphics*, pp. 25–32, 2004.
- [10] LUZ, J. L. S., *Visualização de Nuvens de Pontos com Aproximações Quase Planares e Blending de Textura*, Master’s Thesis, Instituto Nacional de Matemática Pura e Aplicada, IMPA, September 2004.
- [11] BOTSCH, M., HORNING, A., ZWICKER, M., et al., “High-quality surface splatting on today’s GPUs”. In: *PBG '05: Proceedings of the Eurographics Symposium on Point-Based Graphics*, pp. 17–24, IEEE Computer Society: Los Alamitos, CA, USA, 2005.
- [12] GUENNEBAUD, G., BARTHE, L., PAULIN, M., “Splat/Mesh Blending, Perspective Rasterization and Transparency for Point-Based Rendering”. In: *IEEE/Eurographics/ACM Symposium on Point-Based Graphics, Boston, USA, 29/07/06-30/07/06*, pp. 49–58, Eurographics: <http://www.eg.org/>, 2006.
- [13] RÄSÄNEN, J., *Surface Splatting: Theory, Extensions and Implementation*, Master’s Thesis, Helsinki University of Technology, May 2002.
- [14] TALUKDER, K. H., HARADA, K., “Haar Wavelet Based Approach for Image Compression and Quality Assessment of Compressed Image”, *IAENG International Journal of Applied Mathematics*, v. 36, n. 1, 2007.

- [15] STRENGERT, M., KRAUS, M., ERTL, T., “Pyramid Methods in GPU-Based Image Processing”. In: *Workshop on Vision, Modelling, and Visualization VMV '06*, pp. 169–176, 2006.
- [16] LEVOY, M., WHITTED, T., *The Use of Points as a Display Primitive*, Tech. rep., University of North Carolina at Chapel Hill, January 1985.
- [17] GROSS, M., PFISTER, H., (eds), *Point-Based Graphics*. Morgan Kaufmann Publishers, 2007.
- [18] 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, ACM: New York, NY, USA, 1992.
- [19] HOPPE, H., *Surface reconstruction from unorganized points*, Ph.D. Thesis, University of Washington, Seattle, WA, USA, 1995.
- [20] LORENSEN, W. E., CLINE, H. E., “Marching cubes: A high resolution 3D surface construction algorithm”, *SIGGRAPH Comput. Graph.*, v. 21, n. 4, pp. 163–169, 1987.
- [21] AKENINE-MOLLER, T., MOLLER, T., HAINES, E., *Real-Time Rendering*. A. K. Peters, Ltd.: Natick, MA, USA, 2002.
- [22] MARROQUIM, R., KRAUS, M., CAVALCANTI, P. R., “Efficient point-based rendering using image reconstruction”. In: *PBG '07: Proceedings of the Eurographics Symposium on Point-Based Graphics*, pp. 101–108, September 2007.
- [23] MARROQUIM, R., KRAUS, M., CAVALCANTI, P. R., “Efficient Image Reconstruction for Point-Based and Line-Based Rendering”, *Computer & Graphics*, v. 32, pp. 189–203, 2008.
- [24] LENGYEL, J., “The Convergence of Graphics and Vision”, *Computer*, v. 31, n. 7, pp. 46–53, 1998.

- [25] SMITH, A. R., “The reality of simulated actors”, *Commun. ACM*, v. 45, n. 7, pp. 36–39, 2002.
- [26] MCMILLAN, L., BISHOP, G., “Plenoptic modeling: an image-based rendering system”. In: *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pp. 39–46, ACM: New York, NY, USA, 1995.
- [27] KANG, S. B., LI, Y., TONG, X., et al., “Image-Based Rendering”, *Foundations and Trends in Computer Graphics and Vision*, v. 2, n. 3, pp. 173–258, 2006.
- [28] SHADE, J., GORTLER, S., WEI HE, L., et al., “Layered depth images”. In: *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pp. 231–242, ACM: New York, NY, USA, 1998.
- [29] OGDEN, J., ADELSON, E., BERGEN, J., et al., “Pyramid Based Computer Graphics”, *RCA Engineer*, v. 30, pp. 4–15, 1985.
- [30] LEVOY, M., PULLI, K., CURLESS, B., et al., “The Digital Michelangelo Project: 3D Scanning of Large Statues”. In: *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pp. 131–144, ACM Press/Addison-Wesley Publishing Co.: New York, NY, USA, 2000.
- [31] ZWICKER, M., PFISTER, H., VAN BAAR, J., et al., “Surface splatting”. In: *Siggraph 2001, Computer graphics Proceedings*, pp. 371–378, ACM Press / ACM SIGGRAPH / Addison Wesley Longman: New York, NY, USA, 2001.
- [32] ZWICKER, M., *Continuous Reconstruction, Rendering, and Editing of Point-Sampled Surfaces*, Ph.D. Thesis, Swiss Federal Institute of Technology, ETH, Zurich, 2003.

- [33] HECKBERT, P. S., *Fundamentals of Texture Mapping and Image Warping*, Master's Thesis, University of California at Berkeley, Berkeley, CA, USA, 1989.
- [34] ZWICKER, M., RÄSÄNEN, J., BOTSCH, M., et al., "Perspective accurate splatting". In: *GI '04: Proceedings of the 2004 conference on Graphics interface*, pp. 247–254, Canadian Human-Computer Communications Society: School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 2004.
- [35] KALAIHAH, A., VARSHNEY, A., "Differential Point Rendering". In: *Proceedings of the 12th Eurographics Workshop on Rendering Techniques*, pp. 139–150, Springer-Verlag: London, UK, 2001.
- [36] LUZ, J. L., VELHO, L., CARVALHO, P. C. P., "Silhouette Enhanced Point-Based Rendering." In: *WSCG (Journal Papers)*, pp. 105–111, 2005.
- [37] REN, L., PFISTER, H., ZWICKER, M., "Object Space EWA Surface Splatting : A Hardware Accelerated Approach to High Quality Point Rendering", *Computer Graphics Forum*, v. 21, n. 3, pp. 461–470, 2002.
- [38] COCONU, L., HEGE, H.-C., "Hardware-accelerated point-based rendering of complex scenes". In: *EGRW '02: Proceedings of the 13th Eurographics workshop on Rendering*, pp. 43–52, Eurographics Association: Aire-la-Ville, Switzerland, Switzerland, 2002.
- [39] BOTSCH, M., KOBELT, L., "High-Quality Point-Based Rendering on Modern GPUs". In: *PG '03: Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, p. 335, IEEE Computer Society: Washington, DC, USA, 2003.
- [40] WEYRICH, T., HEINZLE, S., AILA, T., et al., "A Hardware Architecture for Surface Splatting", *ACM Transactions on Graphics (Proc. SIGGRAPH)*, v. 26, n. 3, Aug. 2007.
- [41] ADAMSON, A., ALEXA, M., "Approximating and intersecting surfaces from points". In: *SGP '03: Proceedings of the 2003 Eurographics/ACM SIG-*

*GRAPH symposium on Geometry processing*, pp. 230–239, Eurographics Association: Aire-la-Ville, Switzerland, Switzerland, 2003.

- [42] WALD, I., SEIDEL, H.-P., “Interactive ray tracing of point-based models”. In: *SIGGRAPH '05: ACM SIGGRAPH 2005 Sketches*, p. 54, ACM Press: New York, NY, USA, 2005.
- [43] ADAMS, B., WICKE, M., DUTRÉ, P., et al., “Interactive 3D Painting on Point-Sampled Objects”. In: *Proceedings of the Eurographics Symposium on Point-Based Graphics*, pp. 57–66, 2004.
- [44] GONZALEZ, R. C., WOODS, R. E., *Digital Image Processing*. Addison-Wesley Longman Publishing Co., Inc.: Boston, MA, USA, 2001.
- [45] WILLIAMS, L., “Pyramidal parametrics”, *SIGGRAPH Comput. Graph.*, v. 17, n. 3, pp. 1–11, 1983.
- [46] ADELSON, E. H., ANDERSON, C. H., BERGEN, J. R., et al., “Pyramid Methods in Image Processing”, *RCA Engineer*, v. 29, n. 6, 1984.
- [47] BURT, P. J., ADELSON, E. H., “The Laplacian Pyramid as a compact image code”, *IEEE Transactions on Communications*, v. COM-31,4, pp. 532–540, 1983.
- [48] MALLAT, S. G., “A theory for multiresolution signal decomposition: the wavelet representation”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 11, pp. 674–693, 1989.
- [49] STOLLNITZ, E. J., DEROSE, T. D., SALESIN, D. H., “Wavelets for Computer Graphics: A Primer, Part 1”, *IEEE Comput. Graph. Appl.*, v. 15, n. 3, pp. 76–84, 1995.
- [50] MULCAHY, C., “Image Compression Using the Haar Wavelet Transform”, *Spelman Science and Mathematics Journal*, v. 1, n. 1, pp. 22–31, April 1997.
- [51] ZHANG, Y., PAJAROLA, R., “Single-Pass Point Rendering and Transparent Shading”. In: *Proceedings of the Eurographics/IEEE VGTC Symposium on Point-Based Graphics '06*, pp. 37–48, 2006.



- [52] ZÖCKLER, M., STALLING, D., HEGE, H.-C., “Interactive visualization of 3D-vector fields using illuminated stream lines”. In: *VIS '96: Proceedings of the 7th conference on Visualization '96*, pp. 107–114, IEEE Computer Society Press: Los Alamitos, CA, USA, 1996.
- [53] MALLO, O., PEIKERT, R., SIGG, C., et al., “Illuminated lines revisited”. In: *In IEEE Visualization 2005 Conference Proceedings (2005)*, pp. 19–26, 2005.
- [54] DEUSSEN, O., HANRAHAN, P., LINTERMANN, B., et al., “Realistic modeling and rendering of plant ecosystems”. In: *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pp. 275–286, ACM: New York, NY, USA, 1998.
- [55] DEUSSEN, O., COLDITZ, C., STAMMINGER, M., et al., “Interactive visualization of complex plant ecosystems”. In: *VIS '02: Proceedings of the conference on Visualization '02*, pp. 219–226, IEEE Computer Society: Washington, DC, USA, 2002.
- [56] PHARR, M., HUMPHREYS, G., *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann Publishers Inc.: San Francisco, CA, USA, 2004.
- [57] DACHSBACHER, C., VOGELGSANG, C., STAMMINGER, M., “Sequential point trees”, *ACM Trans. Graph.*, v. 22, n. 3, pp. 657–662, 2003.
- [58] DELLEPIANE, M., CALLIERI, M., PONCHIO, F., et al., “Mapping highly detailed color information on extremely dense 3D models: the case of David’s restoration”, *Computer Graphics Forum*, v. 27, n. ?, pp. ??–??, 2008.
- [59] AGRAWALA, M., RAMAMOORTHY, R., HEIRICH, A., et al., “Efficient image-based methods for rendering soft shadows”. In: *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pp. 375–384, ACM Press/Addison-Wesley Publishing Co.: New York, NY, USA, 2000.