COPPE/UFRJ

RENDERIZAÇÃO DE MODELOS DE PONTOS USANDO MULTIRESOLUÇÃO

Felipe Moura de Carvalho

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

Orientadores: Antonio Alberto Fernandes de Oliveira
Ricardo Guerra Marroquim

Rio de Janeiro
Fevereiro de 2010

RENDERIZAÇÃO DE MODELOS DE PONTOS USANDO MULTIRESOLUÇÃO

Felipe Moura de Carvalho

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

Examinada por:

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

Prof. Ricardo Guerra Marroquim, D.Sc.

Prof. Paulo Roma Cavalcanti, D.Sc.

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

RIO DE JANEIRO, RJ – BRASIL
FEVEREIRO DE 2010

Carvalho, Felipe Moura de

Renderização de Modelos de Pontos usando Multiresolução/Felipe Moura de Carvalho. – Rio de Janeiro: UFRJ/COPPE, 2010.

XV, 52 p.: il.; 29, 7cm.

Orientadores: Antonio Alberto Fernandes de Oliveira
Ricardo Guerra Marroquim

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2010.

Referências Bibliográficas: p. 47 – 52.

1. *Splats*. 2. Simplificação. 3. Modelos de Pontos. I. Oliveira, Antonio Alberto Fernandes de *et al.*. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Aos meus pais

Valbert e Bernadete.

Agradecimentos

Quero agradecer aos meus Pais e família pela educação e o incentivo que me transmitiram, a confiança que sempre depositaram a mim e nas minhas decisões, e o carinho reconfortante e otimista que sempre me souberam dispensar.

Quero agradecer aos meus orientadores , agora Prof., Ricardo Marroquim pelas orientações e conselhos principalmente nesta reta final foi de suma importância. Ao Prof. Antonio Oliveira pela paciência e confiança nesta jornada.

Quero agradecer ao Prof. Paulo Roma, por ter aceito participar da minha banca examinadora e pelo suporte, ao Prof. Cláudio Esperança pelas sugestões e críticas ao meu trabalho, ao Prof. Luiz Velho por ter aceito participar da minha banca examinadora e pela sugestões dadas para enriquecer meu trabalho e ao Prof. Ricardo Farias pelo apoio.

Quero agradecer ao vários membros do LCG que aqui estão ou já se formaram: Yalmar, Alvaro, Guina, Alberto, Maximus, Flávio, Saulo, Leandro, Carlos Eduardo.

Quero agradecer aos amigos que fiz no PESC Jesus e Vinicius Ramos, e também as secretárias Sônia, Cláudia, Solange e Mercedes.

Quero agradecer ao Prof. Paulo Sérgio e por ter me guiado neste inicio de mestrado aqui no Rio, e ao José Ricardo (vulgo Zezim) por ter sido companheiro neste inicio tão difícil.

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

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

RENDERIZAÇÃO DE MODELOS DE PONTOS USANDO MULTIRESOLUÇÃO

Felipe Moura de Carvalho

Fevereiro/2010

Orientadores: Antonio Alberto Fernandes de Oliveira
Ricardo Guerra Marroquim

Programa: Engenharia de Sistemas e Computação

Representação digital de superfícies de modelos reais podem ser obtidas por métodos semi-automáticos usando vários dispositivos de registro, como $3D$ *scanners* e fotografia $3D$. Estes dispositivos são rápidos e precisos, o que resulta em modelos $3D$ com resoluções de ordem de magnitude elevada. Muitas aplicações em computação gráfica podem se beneficiar desses métodos de aquisição de geometria digital, incluindo: aplicações *CAD* (*Computer-Aided Design*), simulação física, realidade virtual, imagens médicas, arquitetura, arqueologia, efeitos especiais, animação e jogos eletrônicos.

Por outro lado, a quantidade de amostras geradas pela aquisição muitas vezes não pode ser tratada por métodos de processamento e visualização existentes, requerendo novas estruturas de dados e algoritmos para tratar estes objetos. Uma abordagem comum para atingir renderização interativa com qualidade é transformar a nuvem de ponto em uma representação por *splats*, que são discos ou elipses orientadas representando localmente a superfície; porém, outros tipos de operações, como simplificação, são geralmente realizadas na representação por pontos puros. Esta dissertação propõe um algoritmo de simplificação que trabalha diretamente com *splats*, suprindo a necessidade de uma conversão posterior após a simplificação da nuvem de pontos.

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

MULTIRESOLUTION RENDERING OF POINT SAMPLED SURFACE

Felipe Moura de Carvalho

February/2010

Advisors: Antonio Alberto Fernandes de Oliveira
Ricardo Guerra Marroquim

Department: Systems Engineering and Computer Science

Digital representations of real-world surfaces can now be obtained automatically using various acquisition devices such as $3D$ scanners and $3D$ photograph. These new fast and accurate data sources increase $3D$ surface resolution by several orders of magnitude. Many computer graphics applications can take benefit of this automatic process, including: CAD (Computer-Aided Design), physical simulation, virtual reality, medical imaging, architecture, archaeology, special effects, computer animation and video games.

Unfortunately, this geometry produced by these media comes at the price of a large, possibility gigantic, amount of data which requires new efficient data structure and algorithms offering scalability for processing such objects.

Aim to interactive and quality visualizations, models produced by these media, which is point clound, are converted in splat basead representations, which are oriented disc or ellipses. In this dissertation, we present a simplification algoritm that works directly with splats, supplying the need of simplification of pure point representation and a subsequent conversion of this point cloud simplified in a splat based represetation.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

CAD	*Computer-AidedDesign*, p. vi
CPU	*Central Processor Unit*, p. 21
EWA	*Elliptical Weighted Average*, p. 8
GPU	*Graphic Processor Unit*, p. 1
MLS	*Moving Least Squares*, p. 27
QEM	*Quadric Error Metric*, p. 29
SPT	*Sequential Point Trees*, p. 16
BVH	*Bounding Volume Hierarchies*, p. 12
BSP Tree	*Binary Space Partitioning Tree*, p. 14

Capítulo 1

Introdução

1.1 Motivação

Renderização Baseada em pontos vem se tornando popular nos últimos anos, especialmente devido à evolução do *hardware* gráfico 3D (*Graphic Processor Unit*) e à necessidade de processar modelos extremamente grandes, e.x. aqueles produzidos por *Scanners* $3D$ [9, 10]. Ao contrário do método tradicional de renderização aonde a superfície do modelo é aproximada por um conjunto de polígonos (malhas poligonais), técnicas de renderização baseadas em pontos usam apenas a nuvem de pontos sem conectividade explicita, como ilustrado na Figura 1.1.

Com o aumento da capacidade do hardware gráfico tornou-se possível tratar cenas com quantidades enormes de polígonos, porém a resolução dos dispositivos de saída não acompanharam esta taxa de crescimento; como consequência, muitas vezes polígonos são mapeados em apenas um *pixel* alocando tempo desnecessário ao processo de rasterização. No entanto, as GPUs são otimizadas para renderizar triângulos, assim como a maioria dos algoritmos para processar e visualizar modelos foram pensados para tratar malhas poligonais; desta forma, era de praxe a conversão de modelos de pontos para polígonos usando métodos como o algoritmo de *marching cubes* [11]. Por outro lado, com as GPUs modernas possuindo cada vez mais partes programáveis dentro do *pipeline* gráfico, é possível criar algoritmos de renderização de pontos que justifiquem o processamento direto nas nuvens de pontos, produzindo resultados eficientes e imagens de alta qualidade no caso da renderização. Especialmente quando para aplicação não é necessária a informação de topologia, ou

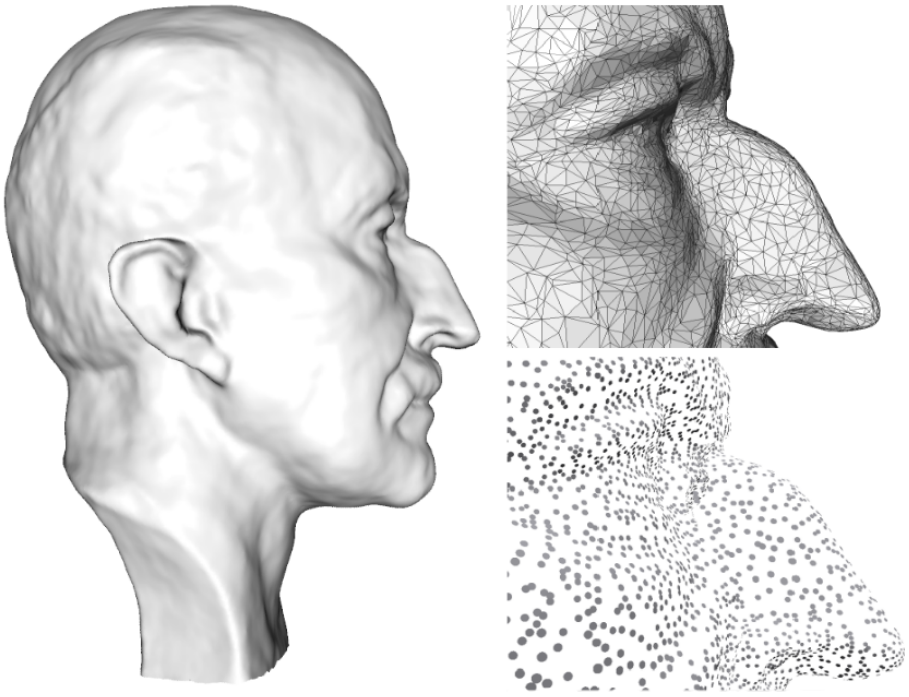


Figura 1.1: **Esquerda:** Modelo Max Planck. **Topo-Direita:** Renderização usando malhas poligonais **Baixo-Direita:** Renderização usando *Splats* renderizados com fração do tamanho para efeito de ilustração (Retirada de [1])

seja, a conectividade entre os pontos, é especialmente vantajoso o uso deste tipo de representação.

1.2 Proposta

Apesar de algoritmos eficientes para renderizar nuvens de pontos na GPU terem sido propostos [12, 13, 14], os modelos em resoluções cada vez maiores criam a necessidade de formas mais eficientes de visualizar a nuvem de pontos. O projeto de digitalizar as catacumbas de Roma por exemplo, gerou um modelo com mais de 1.2 bilhão de amostras [15]. Um método comum é a aplicação de níveis de detalhes (LOD), onde representações em resoluções mais baixas são criadas e escolhidas para renderização de acordo com uma métrica de visualização, como por exemplo distância ao observador.

Nesta tese consideramos o problema de simplificar o modelo de pontos para criar representações em resoluções mais baixas objetivando a renderização. O desafio é, dada uma nuvem de ponto, combinar amostras de forma a reduzir a complexidade do

modelo obtendo a melhor aproximação possível, isto é, com menor erro. Para tanto é proposto um modelo de clusterização baseado em *splats*, que são extensões de pontos, possuindo informação de densidade local e propriedades diferenciais da superfície no ponto. Diferentemente de outros métodos, a proposta é realizar toda a simplificação considerando *splats* elípticos, que provêm uma aproximação da superfície melhor do que os *splats* circulares, e ainda mais do que os pontos "puros". São abordados três problemas básicos para construir os diferentes níveis de resolução:

- quais pontos devem ser combinados para diminuir a resolução, o que normalmente envolve métricas de erros específicas para este tipo de representação;
- dado o conjunto de pontos a ser combinados, como formar um *splat* representativo a partir destes;
- como construir uma estrutura de dados eficientes para renderização considerando o suporte a algoritmos em GPU.

1.3 Organização da Dissertação

Esta dissertação está organizada da seguinte forma:

- **Capítulo 2** discutiremos o uso de pontos como primitiva para representar superfícies, sendo apresentado primeiro um breve histórico seguido por uma discussão quanto a primitiva em si. Por último serão apresentados os *splats* como extensão de pontos, focando na tarefa de visualização.
- **Capítulo 3** serão apresentadas as principais estruturas de dados para tratar nuvens de pontos.
- **Capítulo 4** apresentaremos os principais algoritmos de simplificação de nuvens de pontos. Este capítulo será divido em duas partes: simplificação de superfícies baseadas em pontos "puros"; e simplificação de superfícies baseadas em *splats*.
- **Capítulo 5** apresentaremos um método de simplificação de superfícies baseada em *splats* usando uma clusterização incremental.

- **Capítulo 6** faremos um breve revisão da dissertação, bem como conclusões e possíveis direções para trabalhos futuros.

Capítulo 2

Superfícies Baseada em Pontos

Superfícies baseadas em pontos ganharam recentemente a atenção da comunidade de computação gráfica [16]. Geralmente são provenientes de *scanners* $3D$ que amostram a superfície do objeto gerando uma nuvem de pontos $\mathcal{P}$. Operações de filtragem [17] são definidas para reduzir o ruído e preencher regiões com buracos. Nesta dissertação iremos assumir que o conjunto de pontos $\mathcal{P}$ é livre de ruídos com amostras adequadamente distribuídas na superfície.

Neste capítulo, discutiremos as origens e definições de pontos como primitiva geométrica de renderização. Na seção 2.1 será apresentado um breve histórico, enquanto que nas seções 2.2 e 2.3 a primitiva será descrita em mais detalhes. Por último discutiremos os *splats* na seção 2.4, que são uma extensão da representação pura por pontos.

2.1 Pontos como Primitiva Universal de Visualização

Visualização de modelos representados como uma coleção de pontos não é uma ideia recente, na década de 70 por exemplo, os primeiros vídeo games já representavam explosões de naves espaciais usando pontos luminosos. Pontos também foram uma representação popular para trabalhos de simulações baseadas em partículas, especialmente por existirem situações onde as representações clássicas encontram dificuldades em definir a superfície. Uma partícula pode ser vista como um ponto em $3D$ acrescido de alguns atributos, como tamanho, densidade, velocidade, entre

(a) (b)

Figura 2.1: Renderização Baseada em Pontos proposta por Levoy e Whitted (Retirada de [2]). (a) Ponto como primitiva universal de renderização. (b) *z-buffer* com tolerância para reconstruir uma superfície contínua.

outros. Em 1979 Charles Csuri et al. [18] usaram partículas estáticas para renderizar fumaça; em 1982 Jim Blinn [19] usou partículas para representar nuvens e poeira; no mesmo período, Reeves [20] apresentou seu famoso sistemas de partículas, mais genérico que os anteriores, que permitia simular fogo, explosões, etc..

Em 1985, Levoy e Whitted [2] foram os primeiros a considerar o uso de pontos como primitiva universal de modelagem e renderização. Eles propuseram representar superfícies genéricas como um conjunto de pontos $3D$ suficientemente denso, de modo a possibilitar a renderização da superfície contínua. De fato, a proposta era um pouco ambiciosa ao colocar os pontos como uma meta-primitiva, desta forma todas representações deveriam ser convertidas para pontos em um certo momento permitindo a unificação dos algoritmos de visualização. Para reconstruir uma superfície contínua, os pontos são incrementados com valores de normal, cor e um coeficiente de transparência, além de um valor que estima a densidade local, permitindo que o ponto projetado possua uma área de cobertura maior do que a de um pixel (Figura 2.1).

2.2 Pontos Puros

Inicialmente consideramos uma nuvem de pontos onde não haja nenhuma informação de topologia ou densidade da superfície. Na prática, esta representação dificilmente é utilizada diretamente, pois não é possível elaborar efeitos de iluminação sem ao menos a informação dos vetores normais. Xu et al. [21] apresentaram uma técnica de renderização sem informação de normal, porém tratavam apenas das silhuetas e não da superfície completa.

2.3 Pontos Orientados

Com uma nuvem de pontos suficientemente densa, é possível estimar as normais da superfície em cada ponto analisando sua vizinhança local. Assume-se o conjunto de pontos como uma coleção de posições $\mathcal{P} = \{\mathbf{c}_i\}$ no $\mathbf{R}^3$, $i \in \{1, \ldots, N\}$. Como não há informação de conectividade assumimos a utilização de algum método que nos forneça os k-vizinhos mais próximos do ponto, definidos como um subconjunto de k pontos com menor distância Euclidiana ao ponto dado. No capítulo 3 será apresentada uma estrutura eficiente para obter os k-vizinhos de uma nuvem de pontos; no entanto, assumindo-se que são dados os k vizinhos $\mathcal{N}_{\mathcal{P}}^k = \{\mathbf{c}_1, \ldots, \mathbf{c}_k\}$ de um ponto $\mathbf{c}_i$, a normal no ponto pode ser obtida pela análise da matriz de covariância, definida como:

$$\mathbf{C} = \sum_{j=1}^{k} (\mathbf{c}_j - \bar{\mathbf{c}})(\mathbf{c}_j - \bar{\mathbf{c}})^T, \tag{2.1}$$

onde $\bar{\mathbf{c}} = \frac{1}{k} \sum_{j=1}^{k} \mathbf{c}_j$ é a média de todos os vizinhos. Como a matriz 3×3 é semi-definida positiva e simétrica, ela possui todos os seus autovalores reais. Desta forma pode-se tomar o autovetor correspondente ao menor autovalor como uma estimativa da direção da normal da superfície no ponto $\mathbf{c}_i$.

Entretanto, a informação de normal geralmente não é suficiente para gerar uma visualização contínua da superfície, sendo necessário estimar o espaço entre pontos vizinhos, ou seja, uma valor local de densidade. Esta informação pode ser gerada implicitamente em superfícies amostradas de forma regular, porém algumas regiões podem ficar super-amostradas, já que áreas planares são consideradas com a mesma resolução do que áreas com grandes variações de curvatura. Desta maneira, *splats* (seção a seguir) constituem uma maneira simples de representar adaptativamente nuvens de pontos.

2.4 *Splat*

Os *splats* foram propostos primeiramente por Pfister et al.[22] que introduziram a ideia de "*surfel*" e interpolação local dos pontos projetados. Este trabalho foi estendido por Zwicker et al [23] em 2001, introduzindo o conceito de *Surface Splatting*, uma das técnicas mais populares de renderização de superfícies baseadas em pon-

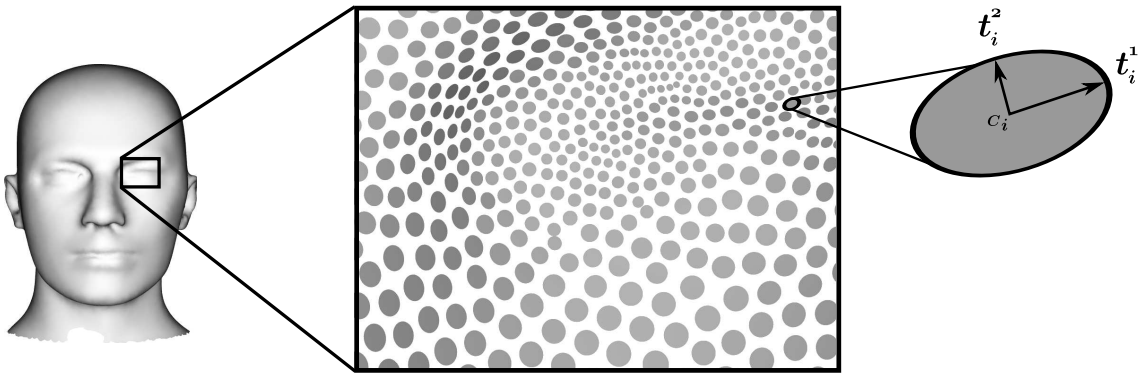


Figura 2.2: **Esquerda:** um modelo $3D$ representado por *splats*. **Centro:** visão ampliada da superfície com *splats* renderizados com metade dos valores dos raios originais. **Direita:** um *splat* elíptico é definido por uma posição $\mathbf{c}_i$ e dois eixos tangentes $\mathbf{t}_i^1$ e $\mathbf{t}_i^2$ (retirada de [3]).

tos. Neste trabalho foi proposto o uso do filtro EWA (*Elliptical Weighted Average*) para reconstruir a superfície em espaço de imagem utilizando os *splats* projetados. Um *splat* pode ser definido como circular ou elíptico como discutido por Kobbelt e Botsch [24]; nesta dissertação usaremos aqueles elípticos.

2.4.1 *Splat* Elíptico

Splats são definidos como pequenas elipses com centro definido pela posição do ponto, e vetor normal pela normal da superfície no ponto. Desta forma, cada *splat* pode ser definido unicamente por uma posição $\mathbf{c}_i$ e dois vetores $\mathbf{t}_i^1$ e $\mathbf{t}_i^2$ ortogonais e tangentes à superfície (Figura 2.2), onde os tamanhos dos vetores definem a extensão da elipse.

Existem alguns algoritmos para converter superfícies de pontos pura em representações baseada em *splats* [25, 26]. A partir dos vetores a normal por ser facilmente computada usando os eixos principais:

$$\mathbf{n}_i = \frac{\mathbf{t}_i^1 \times \mathbf{t}_i^2}{\|\mathbf{t}_i^1 \times \mathbf{t}_i^2\|}. \tag{2.2}$$

Um ponto $\mathbf{c}$ sobre o plano definido por $\mathbf{c}_i$, $\mathbf{t}_i^1$ e $\mathbf{t}_i^2$, também estará no interior do

splat se a seguinte condição for satisfeita:

$$t_1^2 + t_2^2 = \frac{({\mathbf{t}_i^1}^T(\mathbf{c} - \mathbf{c}_i))^2}{({\mathbf{t}_i^1}^T\mathbf{t}_i^1)^2} + \frac{({\mathbf{t}_i^2}^T(\mathbf{c} - \mathbf{c}_i))^2}{({\mathbf{t}_i^2}^T\mathbf{t}_i^2)^2} \leq 1, \tag{2.3}$$

onde $\mathbf{t}_1$ e $\mathbf{t}_2$ definem a parametrização dos pontos no espaço do *splat*. Esta formula pode ser simplificada usando-se $\mathbf{t}_i^{1'} = \mathbf{t}_i^1/({\mathbf{t}_i^1}^T\mathbf{t}_i^1)$ e $\mathbf{t}_i^{2'} = \mathbf{t}_i^1/({\mathbf{t}_i^2}^T\mathbf{t}_i^2)$:

$$t_1^2 + t_2^2 = ({\mathbf{t}_i^{1'}}^T(\mathbf{c} - \mathbf{c}_i))^2 + ({\mathbf{t}_i^{2'}}^T(\mathbf{c} - \mathbf{c}_i))^2 \leq 1. \tag{2.4}$$

Dada esta propriedade, uma definição formal de *splat* é exposta abaixo:

Definição 1.(*Splat* Elíptico) O *splat* s_i com centro $\mathbf{c}_i$ e eixos principais $\mathbf{t}_i^1$ e $\mathbf{t}_i^2$ e normal $\mathbf{n}_i$ é definido como:

$$s_i = \{\mathbf{c} \in \mathbf{R}^3 | \mathbf{n}_i^T(\mathbf{c} - \mathbf{c}_i) = 0 \wedge ({\mathbf{t}_i^{1'}}^T(\mathbf{c} - \mathbf{c}_i))^2 + ({\mathbf{t}_i^{2'}}^T(\mathbf{c} - \mathbf{c}_i))^2 \leq 1\}. \tag{2.5}$$

Como exposto por Kobbelt e Botsch [24], representar uma superfície como um conjunto de *splats* provê uma aproximação da mesma ordem das malhas poligonais. Ainda mais, como *splats* não possuem informação de conectividade, herdam a flexibilidade das primitivas de pontos.

2.4.2 *Surface Splatting*

A ideia de associar *splats* a pontos foi originada nos primeiros algoritmos de renderização baseada em pontos [22, 23]. O objetivo era criar um algoritmo de renderização de superfícies similar à rasterização de malhas poligonais. Entretanto, ao menos que a amostragem seja extremamente densa, a projeção simples de pontos no espaço de imagem resultará em buracos na imagem final (Figura 2.4 esquerda). Para evitar este problema, os *splats* são projetados e a renderização é realizada através da rasterização das elipses (Figura 2.4 direita). Ainda assim, esse procedimento resulta em imagens de baixa qualidade pois gera descontinuidades entre as elipses (ver figura 2.3 centro-esquerda). A fim de obter uma imagem de alta qualidade usando este algoritmo ingênuo, seria necessário uma quantidade muito grande de amostras, de modo a tornar as descontinuidades imperceptíveis.

Uma maneira de melhorar este algoritmo é tratar as regiões de interseção entre as elipses (Figura 2.3 centro-direita e direita). Porém, implementar este procedi-

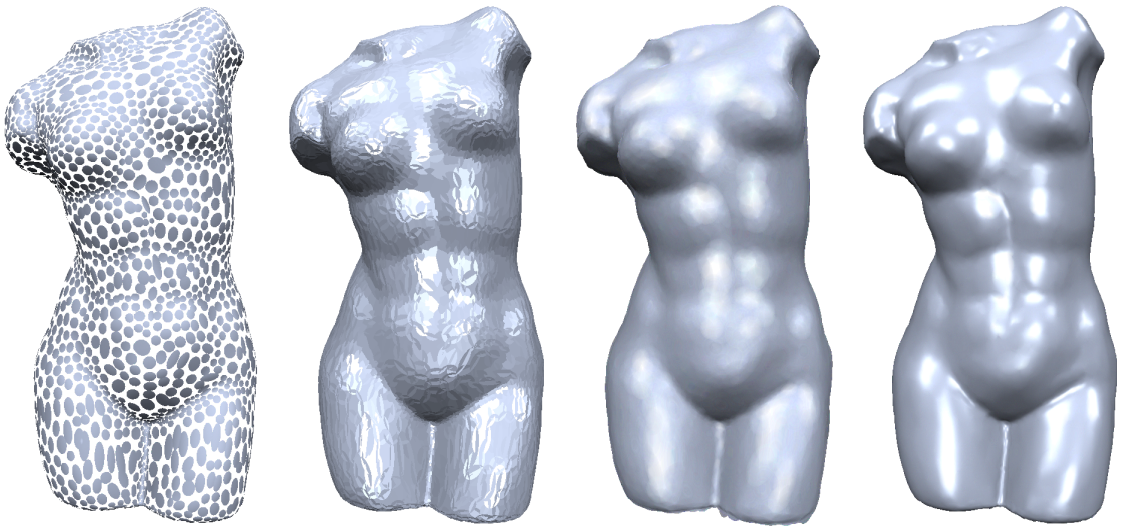


Figura 2.3: **Esquerda:** splats renderizados com metade dos valores dos raios para efeitos de ilustração. **Centro-esquerda:** splats renderizados sem interpolação. **Centro-direita:** interpolação das cores; **direita:** interpolação das normais. Retirada de [4]

mento diretamente no *pipeline* gráfico requer acesso aos valores de profundidade armazenados (*z-buffer*) para testar se duas elipses que se interseptam em espaço de imagem pertencem de fato à mesma superfície do objeto. Infelizmente, na GPU, estes valores de profundidade não são acessíveis em tempo de renderização, requerendo estratégias mais elaboradas que tratem o problema em duas passadas: na primeira as elipses são projetadas e rasterizadas preenchendo o buffer de profundidade de modo habitual; e na segunda as elipses são interpoladas utilizando o z-buffer da primeira passada como referência.

Para realizar a interpolação é feita uma média ponderada de todas as elipses que cobrem um dado pixel. Geralmente são utilizados filtros Gaussianos para atribuir um peso a cada elipse dependendo da distância ao seu centro de projeção.

Ainda mais, a interpolação pode ser realizada de duas formas distintas: interpolando as cores dos splats (Figura 2.3 centro-direita) ou as normais (Figura 2.3 direita). Na primeira, o computo de iluminação é realizado uma vez por elipse e as cores são interpoladas por pixel (análogo a tonalização *Gouraud*), enquanto que na segunda estratégia as normais são interpoladas e o cálculo de iluminação é realizado por pixel da imagem, uma técnica conhecida como *deffered shading*, ou *per-pixel*

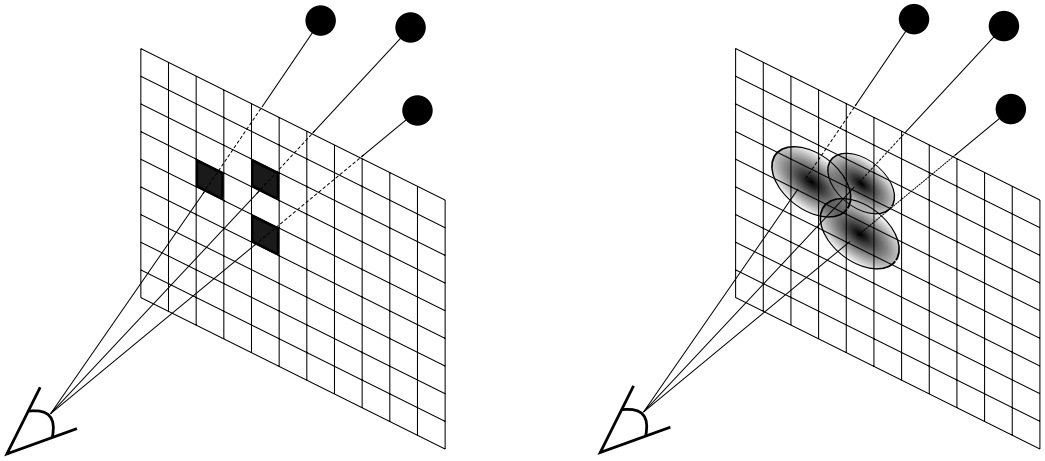


Figura 2.4: **Esquerda:** projeção de pontos com cobertura de um pixel. **Direita:** splats com área de cobertura são interpolados para gerar uma superfície contínua em espaço de imagem. Adaptada de [3]

shading.

Capítulo 3

Estruturas de Dados para Pontos

Neste capítulo apresentaremos as principais estrutura de dados utilizadas para tratar nuvem de pontos. Com ausência de conectividade, operações de modelagem podem ser realizadas com a adição ou remoção de apenas pontos, porém, não é explicitamente presente para outras operações informações de vizinhança onde se faz necessária.

Este capítulo apresenta algumas estruturas de dados para tratar nuvem de pontos além de algumas extensões propostas visando renderização interativa.

3.1 Estrutura de Partição do Espaço

Estruturas de Partição Espacial são comuns em computação gráfica, em particular quando se deseja processar modelos grandes. Diversos métodos de simplificação, reconstrução, compressão, visibilidade entre outros, são baseados neste tipo de estrutura. O objetivo é criar uma indexação do espaço, ou seja, dividi-lo em células e prover uma relação entre estas e o espaço ocupado pelo objeto [27]. No caso específico de nuvens de pontos, existem algumas classes de algoritmos disponíveis: baseados em representações hierárquicas, não hierárquicas ou em cluster de pontos. Nesta seção as estruturas mais comuns serão introduzidas: *Octree* (seção 3.1.1), *K-d Tree* (seção 3.1.2) e Hierarquia de Volumes Envolventes (BVH - *Bounding Volume Hierarchies*) (seção 3.1.3).

3.1.1 *Octrees*

A *octree* é uma das estruturas de partição espacial mais usadas para tratar modelos de pontos grandes, em especial quando o objetivo é renderização interativa [28, 29, 30, 31]. Dado o conjunto de pontos, sua caixa envolvente alinhada aos eixos é computada definindo o nó raiz. A caixa envolvente da raiz é então subdividida em oito octantes de mesmo tamanho que correspondem aos seus nós filhos. Os pontos do nó raiz são distribuídos para os filhos de acordo com o octante ao qual eles pertencem. A subdivisão é realizada para todos os nós filhos até que um certo nível de profundidade seja alcançado ou o nó permaneça vazio. A *octree* resultante consistirá em células vazias (nós internos) e não vazias (folhas) como ilustrado na Figura 3.1-esquerda.

A simplicidade de construção das *octrees* a fazem uma estrutura muito popular quando se deseja trabalhar com processamento de geometria, como por exemplo simplificação e reconstrução de superfície representadas por pontos. Em métodos de simplificação uma *octree* é construída sobre um modelo P gerando um conjunto de clusters C definido pelos nós filhos. Um modelo simplificado P' é obtido substituindo cada cluster C_i por um ponto representativo, tipicamente com coordenadas definidas por:

$$\bar{\mathbf{p}}_i = \frac{1}{|C_i|} \sum \mathbf{p}_j. \tag{3.1}$$

No entanto, quando as células estão próximas à superfície, e esta não está alinhada com os eixos da *octree*, os cluster tendem a ficar desbalanceados. Para aliviar essa deficiência uma versão modificada de *octree* foi proposta em [32] com o nome de *Volume Surface Tree*.

Uma outra aplicação é a determinação de vizinhança de um dado ponto do modelo. Uma *octree* possui 26 direções possíveis de vizinhança: 6 ao longo das faces, 12 ao longo as arestas e 8 nas direções dos vértices. Como o cômputo dos vizinho não se limita apenas a um nó, a determinação da vizinhança pode ser custosa. Uma estrutura mais adequada para este de tipo de busca é a *kd-tree*, apresentada a seguir.

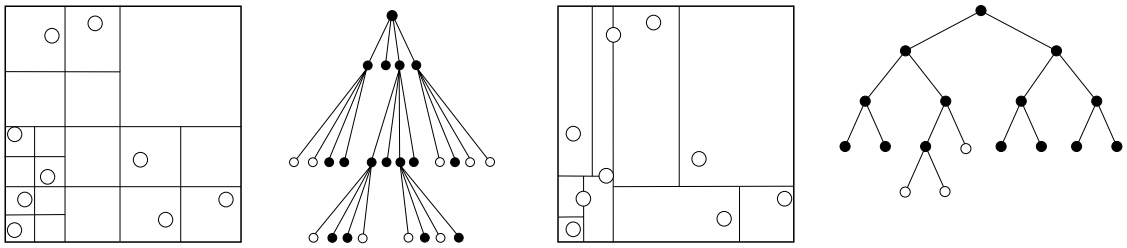


Figura 3.1: Esquerda: *quadtree* equivalente a uma *octree* em $2D$. Direita: *kd-tree* em $2D$. Na *kd-tree* a subdivisão é realizada alternando os dois eixos coordenados de acordo com o ponto médio. Se o conjunto de pontos é par, a linha divisória será a média dos pontos medianos; caso contrário o ponto mediano é alocado para o filho de baixo ou da esquerda. Na figura, a subdivisão foi realizada até que os nós folhas armazenassem apenas 1 ponto. A estrutura em árvore correspondente é ilustrada a direita.

3.1.2 *kd-tree*

Uma *kd-tree* é em geral uma árvore multidimensional de busca em k dimensões; no entanto, para dados em $3D$ a árvore correspondente é geralmente chamada de *kd-tree* tridimensional ao invés de $3d$*-tree*. Elas são um caso especial de árvores binária de particionamento espaciais (*BSP Tree* - *Binary Space Partitioning Tree*). A *kd-tree* usa planos de cortes que são perpendiculares a um dos eixos coordenados (também chamados de hiperplanos), ou seja, uma especialização de uma *BSP Tree* onde planos de cortes arbitrários podem ser usados. Um conjunto de pontos em uma *kd-tree* é subdividido em caixas não interceptantes alinhados aos eixos. Um dos critérios populares para subdivisão da *kd-tree* é a alternância do hiperplano de corte: na raiz, o conjunto de pontos é dividido por um hiperplano perpendicular ao eixo x em dois subconjuntos com o mesmo tamanho, gerando os filhos da raiz (profundidade 1); no próximo nível a partição é baseada no eixo y (nó de profundidade 2), e assim por diante. A recursão pára quando uma determinada quantidade de pontos em um nó é alcançada (Figura 3.2).

Uma *kd-tree* será usada nesta dissertação como uma estrutura de busca dos k vizinhos mais próximos a um ponto x. A busca em uma *kd-tree* requer achar a célula que contem x e computar todas as células que interseptam uma esfera de raio

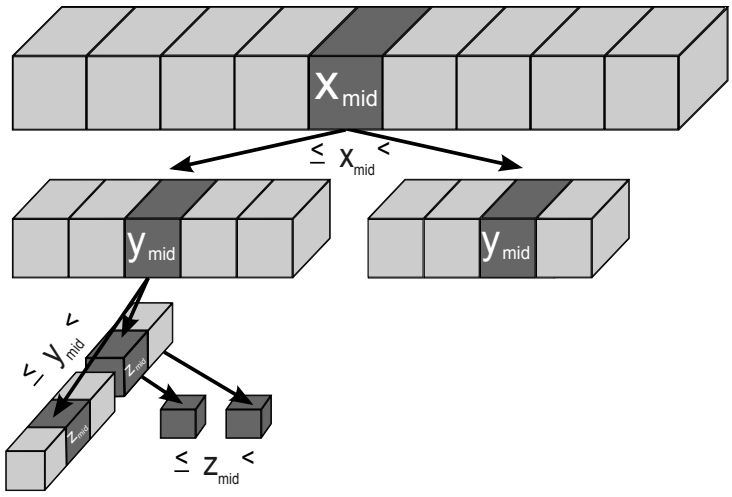


Figura 3.2: Esquema de subdivisão de uma kd-tree alternando os hiperplanos. Para cada nó filho o a mediana dos subconjuntos é encontrada de acordo com o hiperplano de corte.

r centrada no ponto (veja 3.3). As células de interesse são computadas subindo na hierarquia e depois testando todos os filhos destas células contra a esfera. Pontos que estão alocados nestas células são candidatos ao teste de intersecção com a esfera de busca, e os pontos que estão dentro da esfera são retornados como vizinhos de x. Para mais detalhes ver [27].

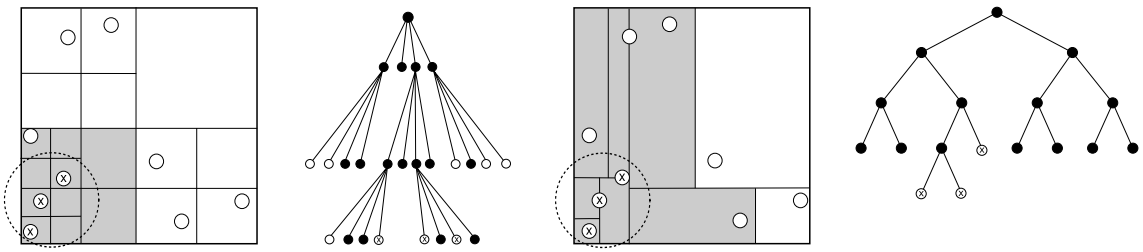


Figura 3.3: Círculo de busca em uma *quadtree* (esquerda) e *kd-tree* (direita) em $2D$. Células em destaque têm seus pontos testados contra o círculo de busca. Note que na *kd-tree* a busca é mais eficiente.

3.1.3 Hierarquia de Volumes Envolventes

Hierarquia de Volumes Envolventes (BVHs) são estruturas muito populares em modelagem e renderização de modelos baseados em pontos. BHV de esferas foi usado,

por exemplo, no *QSplat* [33] para renderização progressiva (este algoritmo será descrito com mais detalhes na seção 3.3). Diferentemente de métodos de partição de dados para indexação no espaço, como as *Octrees* e *kd-trees* descritas anteriormente, as BVHs não são obrigatoriamente uma partição baseada no espaço. Sendo assim, algumas restrições são removidas permitindo a construção de uma hierarquia espacial mais genérica. De fato, ela permite qualquer hierarquia que agrupe os elementos, sem estar necessariamente baseada em seleção espacial. O único requisito em uma BVH é que o volume envolvente (i.e., uma caixa ou esfera) de cada nó englobe todos os elementos da sua subárvore. Geralmente uma estrutura de partição espacial pode ser estendida para uma BVH, gerando o volume envolvente atribuído a cada nó. As BVHs podem ser construídas de maneira *top down*, ou seja, de cima para baixo, similar a uma *octree* ou *kd-tree*.

No exemplo da Figura 3.4, a construção de uma BVH de esferas envolventes usada no *QSplat* é ilustrada. Dado todos os pontos do modelo, uma esfera é calculada usando um algoritmo como o proposto por [34], passando a ser a raíz da hierarquia. A seguir, os pontos são separados em dois subconjuntos de acordo com um dos planos paralelos aos eixos coordenados. Este plano é determinado como o plano que divide a caixa envolvente ao longo da sua maior extensão e para cada subconjunto uma esfera é computada gerando os filhos esquerdo e direito do nó raiz. Esta subdivisão é realizada até que cada esfera contenha um valor mínimo de pontos ou que um dado nível de profundidade seja alcançado. Similar à *kd-tree*, outros critérios de divisão podem ser usados (por exemplo a mediana).

3.2 Multiresolução e Nível de Detalhe

Nesta seção iremos apresentar alguns trabalhos que inspiraram esta dissertação, com ênfase em dois deles em particular. O primeiro é o *QSplat* [33], um sistema para renderização de pontos baseado em uma hierarquia de esferas envolventes. O segundo é o *Sequential Point Tress* (SPT) [6], uma estrutura de dados que permite renderizar em Níveis de Detalhes e diretamente na GPU modelos de pontos.

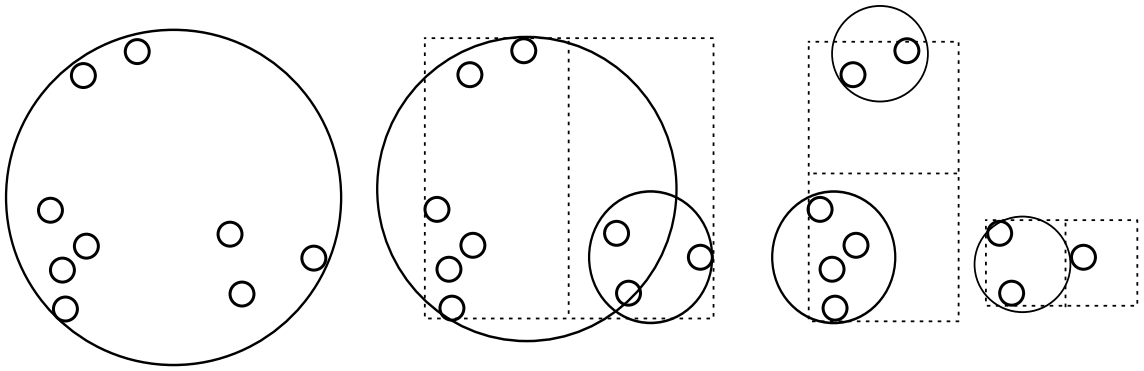


Figura 3.4: **Esquerda**: esfera envolvente do conjunto de pontos iniciais. **Centro**: a divisão é realizada ao longo da maior extensão da caixa envolvente dos pontos. **Direta**: nós no próximo nível.

3.3 *QSplat*

Nesta seção iremos descrever o *QSplat*, um sistema para representação e renderização progressiva de modelos grandes (entre 100 milhões e 1 bilhão de pontos), como por exemplo os produzidos no Projeto Michelangelo Digital [9]. O *QSplat* combina uma hierarquia de esferas envolventes com renderização baseada em pontos. Os nós internos da hierarquia armazenam atributos (posição, normal, cor) que são estimados pelos seus nós filhos. O algoritmo de renderização percorre a hierarquia até que o tamanho da projeção da esfera envolvente seja menor que um valor pré-determinado (geralmente um *pixel*); o nó é então renderizado e seus filhos ignorados.

3.3.1 *QSplat* Estrutura de Dados

O *QSplat* usa uma hierarquia de esferas envolventes que também é usada para controle de nível de detalhe, *view frustum cullling* e *back facing culling* [33]. Cada nó da hierarquia (Figura 3.6) contém o centro e o raio da esfera envolvente, uma normal, o ângulo do cone de normais, opcionalmente uma cor, além de alguns *bits* para representar a estrutura da árvore. A hierarquia é criada em um pré-processamento e guardada em disco. Na Figura 3.5 temos uma ilustração desta hierarquia.

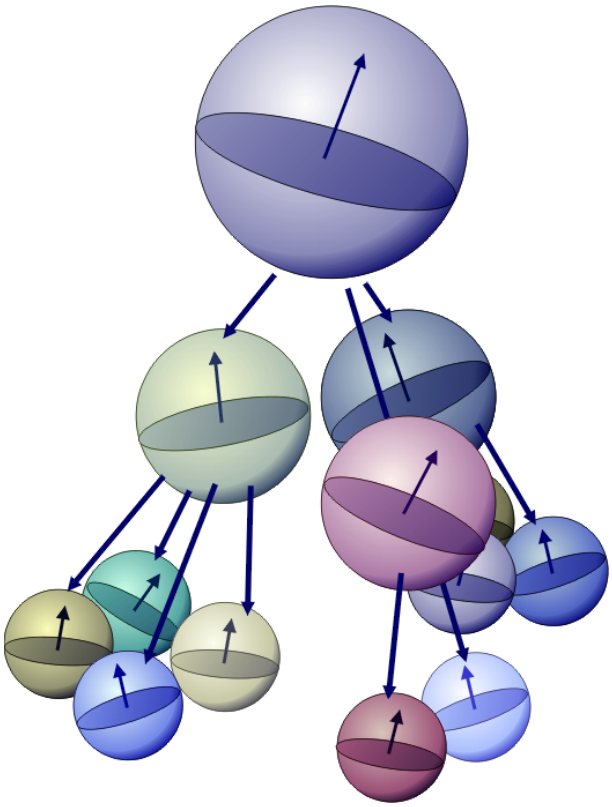


Figura 3.5: Figura esquemática da hierarquia de esferas de *QSplat* (Retirada de [5])

Posição e Raio

A posição e o raio de cada esfera é codificada relativamente aos seus parentes na hierarquia de esferas envolventes e, a fim de economizar memória, são quantizados em 13 valores. O raio de uma esfera varia entre $^{1}/_{13}$ e $^{13}/_{13}$ do raio de seus parentes, enquanto seu centro relativo ao centro de seus parentes (em cada uma das coordenadas **x**, **y** e **z**) é algum múltiplo de $^{1}/_{13}$. Para garantir que o processo de quantização não introduza nenhum buraco durante a renderização, todos os valores são arredondados para o maior valor que englobe seus parentes.

Normal

A normal é codificada em 14 *bits*. Sua quantização usa uma grade de 52×52 em cada uma das 6 faces do cubo gerando $52 \times 52 \times 6 = 16224$ direções possíveis. Uma tabela é usada para decodificar o vetor normal.

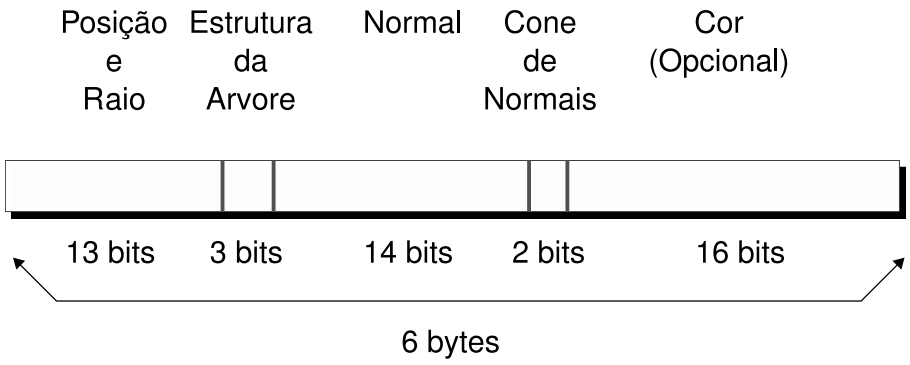


Figura 3.6: Estrutura de um Nó

Cone de Normais

O ângulo do cone de normais é codificado em apenas 2 *bits*. Os quatros valores possíveis que representam a metade do ângulo de abertura do cone são $^{1}/_{16}$, $^{4}/_{16}$, $^{9}/_{16}$ e $^{16}/_{16}$.

Cor

A cor por ponto é codificada em 16*bits*, sendo distribuídos da seguinte forma entre os canais de vermelho, verde e azul: $5 - 6 - 5$ (R-G-B[1]).

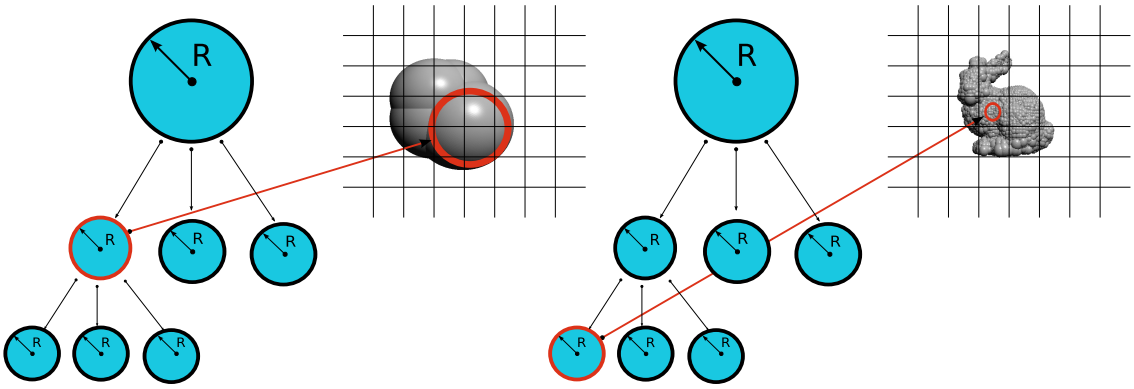
3.3.2 Renderização

O processo simples de renderização é ilustrado na Figura 3.7, enquanto os estágios do algoritmo serão detalhados a seguir.

Visible Culling

Como é usado uma hierarquia de esferas envolventes, nós não visíveis são ignorados durante o percurso. *Frustum Culling* é realizado testando cada esfera contra os planos do tronco da pirâmide que representa o campo de visão. Se a esfera estiver fora, ela e sua subárvore são ignoradas, se ela estiver dentro do campo de visão, ela e seus filhos estão visíveis e não precisam mais passar pelo teste. *Backface Culling* também é realizado durante o processo de renderização utilizando o ângulo do cone de normais.

[1]Sistema de cores RGB



(a) (b)

Figura 3.7: Esquema do algoritmo de renderização. *(a)* Tamanho da imagem projetada do nó é maior que um *pixel*: continua o percurso nas subárvores. *(b)* Tamanho da imagem projetada do nó é menor que um *pixel*: Renderiza o *splat*.

Heurística de Renderização

A heurística usa o tamanho da imagem projetada na tela, ou seja, testa se a área da esfera projetada na tela excede um determinado valor (geralmente um *pixel*).

Renderizando um *Splat*

Quando se atinge um nó desejado de acordo com os critérios mencionados anteriormente, o *splat* é renderizado definido pela esfera corrente. O tamanho do *splat* é baseado no diâmetro da projeção da esfera, e sua cor é obtida usando um cálculo de iluminação baseada na normal e cor da mesma.

3.3.3 Discussão

O *QSplat* possui um processo bem simples, mas infelizmente ele não usa todo o potencial computacional que as GPUs oferecem. Dada a granularidade na sua determinação de nível de detalhes, um modelo pode chegar a ser renderizado ponto por ponto. Este "modo imediato" envia ponto a ponto para renderização e, consequentemente, subutiliza a GPU. No entanto, esta simplicidade torna o *Qsplat* um algoritmo utilizável em outras aplicações: por exemplo, sua hierarquia de esfera é uma boa estrutura para *Ray Tracing* e para aplicações em rede como a do *Stream QSplat*[35], que permite visualizar modelos $3D$ de forma progressiva e remotamente.

3.4 *Sequential Point Trees*

O *QSplat* possui um processamento de dados muito simples e de fácil implementação, mas infelizmente sua estrutura hierárquica recursiva é difícil de ser implementada em GPU. Os pontos renderizados não são armazenados de forma contínua e, portanto, não são processados sequencialmente. A *CPU* (*Central Processor Unit*) percorre a árvore e faz chamadas independentes para renderizar cada nó. Isso causa um "gargalo" entre a CPU e a GPU, onde esta última permanece por vezes ociosa esperando por dados da CPU. *Sequential Point Trees* propõe o uso da estrutura do *QSplat*, porém de uma forma sequencial facilmente tratada em GPU. Sendo assim, mais trabalho é transferido para GPU diminuindo o "gargalo". Nas seções que seguem o SPT será apresentado com mais detalhes.

3.4.1 Hierarquia de Pontos

Inicialmente o SPT utiliza uma hierarquia de pontos representada por uma *octree* (ver Seção 3.1.1), onde cada nó da hierarquia representa parte do objeto. Cada nó armazena um centro $\mathbf{c}$ e uma estimativa da normal $\mathbf{n}$. Um nó armazena também um diâmetro $\mathbf{d}$ da esfera envolvente centrada em $\mathbf{c}$. Um nó interno representa a união de seus nós filhos, ou seja, o diâmetro cresce a medida que se sobe na hierarquia. Os nós folhas possuem pontos que são distribuídos uniformemente no objeto, possuindo diâmetros aproximadamente iguais.

3.4.2 Métricas de Erro

Cada nó na hierarquia pode ser aproximado por um disco de mesmo centro, normal e diâmetro do nó. O erro desta aproximação é avaliado por duas métricas: o erro *perpendicular* $\mathbf{e}_p$ e o erro *tangencial* $\mathbf{e}_t$.

Erro Perpendicular

O erro perpendicular $\mathbf{e}_p$ é a distância mínima entre dois planos paralelos ao disco que engloba todos os filhos. Este erro é uma medida de variância e pode ser calculado como:

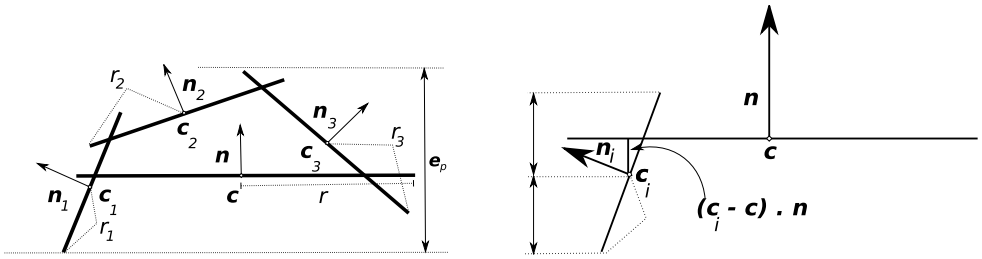


Figura 3.8: Como erro perpendicular, usa-se a distância entre dois plano paralelo ao disco que engloba todos os filhos (Retirada de [6]).

$$\mathbf{e}_p = \max\{((c_i - c)\cdot n) + d_i\} - \min\{((c_i - c)\cdot n) - d_i\} \tag{3.2}$$

$$\text{com} \quad d_i = r_i\sqrt{1-(n_i\cdot n)^2}. \tag{3.3}$$

Durante a renderização, o erro perpendicular é projetado na imagem, resultando em um erro $\tilde{\mathbf{e}}_p$ proporcional ao seno do ângulo entre o vetor de visão $\mathbf{v}$ e a normal do disco $\mathbf{n}$, capturando erros ao longo das silhuetas. Este erro diminui com $^{\mathbf{1}}/_{\mathbf{r}}$, onde $\mathbf{r} = |\mathbf{v}|$:

$$\tilde{\mathbf{e}}_p = \mathbf{e}_p\frac{\sin(\alpha)}{\mathbf{r}} \quad \text{sendo} \quad \alpha = \angle(\mathbf{n},\mathbf{v}). \tag{3.4}$$

Erro Tangencial: O erro tangencial $\mathbf{e}_t$ analisa a projeção dos discos dos filhos no disco do pai, como mostrado na Figura 3.9. O erro é medido usando várias retas ao redor dos filhos projetados de forma a separá-los de uma região não coberta. $\mathbf{e}_t$ é definido como a menor distância entre estas retas e a borda do disco pai, ou seja, mede a cobertura do disco pai; erros negativos são setados em zero. O erro tangencial é projetado no espaço de imagem como:

$$\tilde{\mathbf{e}}_t = \mathbf{e}_t\frac{\cos(\alpha)}{\mathbf{r}} \tag{3.5}$$

Erro Geométrico: Os erros perpendicular e tangencial podem ser combinados em um único erro geométrico $\mathbf{e}_g$, onde o erro no espaço de imagem $\tilde{\mathbf{e}}_g$ depende apenas de r, e não mais do ângulo do visão:

$$\tilde{\mathbf{e}}_g = \frac{\mathbf{e}_g}{\mathbf{r}}, \tag{3.6}$$

onde

$$\mathbf{e}_g = \max\{\mathbf{e}_p\sin\alpha + \mathbf{e}_t\cos\alpha\} = \sqrt{\mathbf{e}_p^2 + \mathbf{e}_t^2}. \tag{3.7}$$

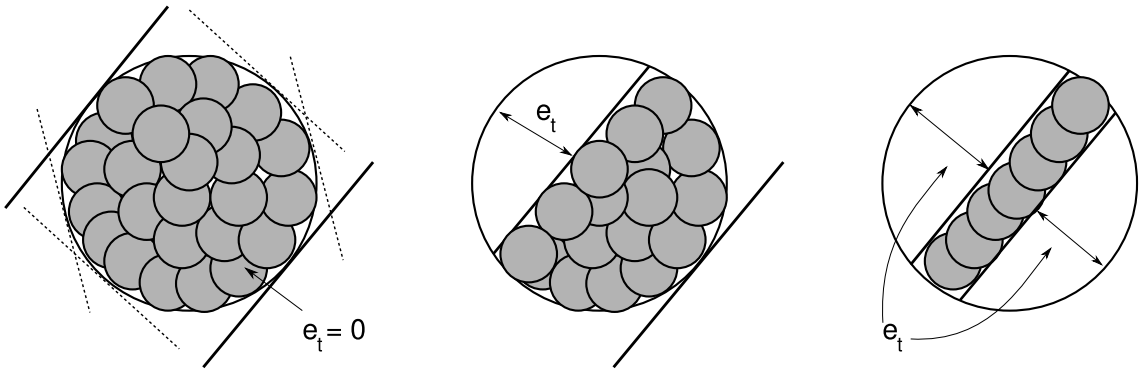


Figura 3.9: Erro tangencial, mede a cobertura do disco pai em relação aos filhos no plano tangente (Retirada de [6]) .

3.4.3 Renderização Recursiva

Um objeto é renderizado na hierarquia de pontos usando um percurso em profundidade, onde para cada nó o erro tangencial projetado $\tilde{\mathbf{e}}_g$ é calculado. Se $\tilde{\mathbf{e}}_g$ está abaixo de um limite de erro ϵ estabelecido e o nó não é um nó folha, seus filhos são percorridos recursivamente; caso contrário, um *splat* de tamanho $\tilde{\mathbf{d}} = {}^{\mathbf{d}}/_{\mathbf{r}}$ é renderizado. Note que esta hierarquia de pontos não se adapta apenas à distância ao observador $\mathbf{r}$, mas também às propriedades da superfície: áreas planas extensas são detectadas com erro geométrico pequeno e podem, consequentemente, ser renderizadas com *splats* grandes.

3.4.4 Arranjamento Sequencial

O procedimento de renderização da seção anterior é recursivo e não se adapta ao processamento não hierárquico da GPU. Assim, há um conversão da estrutura em árvore para uma estrutura em lista e o teste recursivo é substituído por um percurso sequencial sobre a lista de pontos.

Para tanto, o erro simplificado $\tilde{\mathbf{e}}_g$ é substituído por um que seja mais apropriado. Assumindo ϵ constante, ao invés de usar o teste recursivo $\tilde{\mathbf{e}}_g = {}^{e_g}/_{r} < \epsilon$, é armazenado uma distância mínima $\mathbf{r}_{\min} = {}^{e_g}/_{\epsilon}$ no nó simplificando o teste recursivo para $\mathbf{r} > \mathbf{r}_{\min}$. Entretanto, quando os nós da árvore são processados sequencialmente sem informação hierárquica, há necessidade de um teste não recursivo. Para este fim, é adicionado um parâmetro $\mathbf{r}_{\max}$ em cada nó, que é simplesmente o $\mathbf{r}_{\min}$ do seu nó

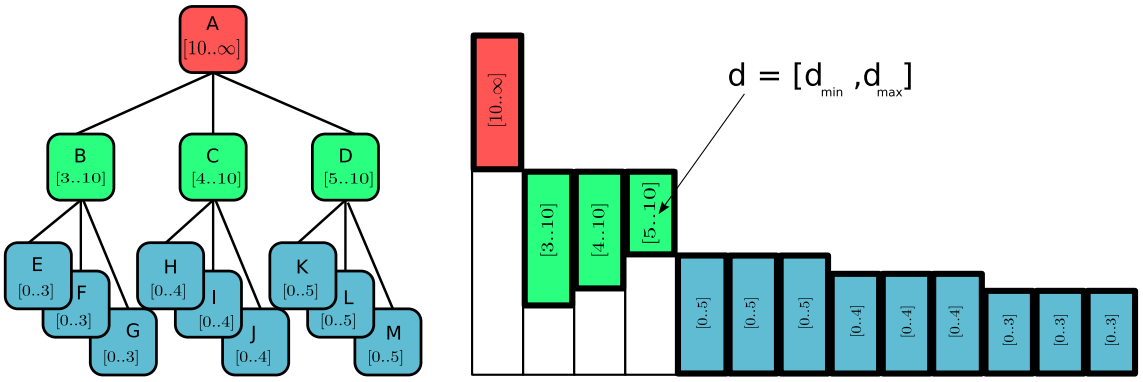


Figura 3.10: Conversão da hierarquia de pontos em um lista. **Direita:** Hierarquia de pontos dos nós $\mathbf{A} - \mathbf{M}$ com seus respectivos $[\mathbf{r}_{\min}, \mathbf{r}_{\max}]$. **Esquerda:** Representação em lista da mesma hierarquia de pontos ordenada por $\mathbf{r}_{\max}$.(Retirada de [6]) .

pai. Usa-se $\mathbf{r} \in [\mathbf{r}_{\min}, \mathbf{r}_{\max}]$ como um teste não recursivo, guiando o algoritmo de renderização com um teste *intervalar* para cada nó.

Após esta substituição de testes, a hierarquia de pontos é transformada em uma lista, para ser processada sequencialmente. Neste estágio, $[\mathbf{r}_{\min}, \mathbf{r}_{\max}]$ é usado para ordenar a lista de forma decrescente a partir de $\mathbf{r}_{\max}$ como ilustrado na Figura 3.10.

Um exemplo de como o algoritmo de renderização funciona é ilustrado na Figura 3.11. Para diferentes valores de $\mathbf{r}$ uma porção da lista é selecionada enquanto outras são descartadas.

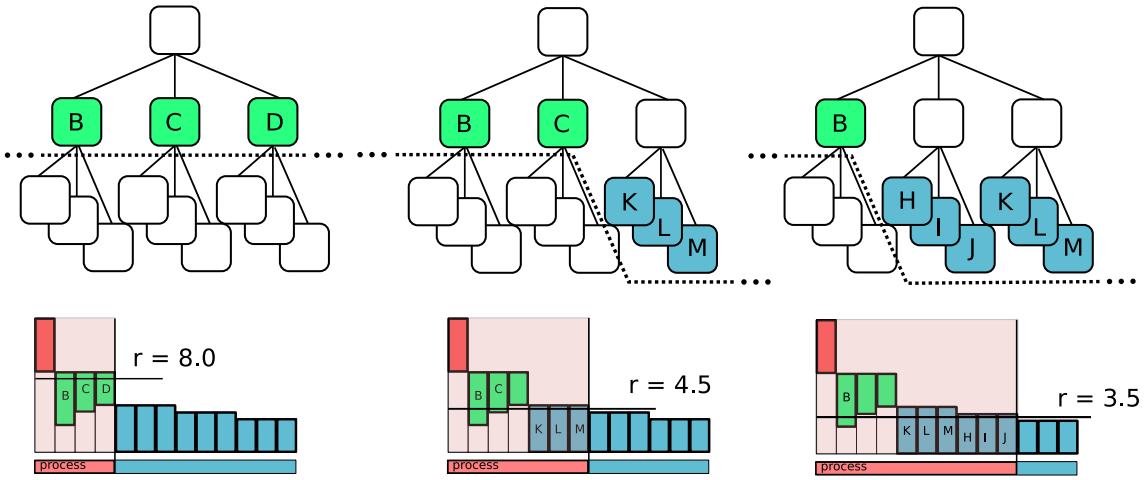


Figura 3.11: **No topo:** Corte na árvore por diferentes pontos de vista. **Em baixo:** O mesmo corte da árvore de cima, mas agora na lista de pontos (Retirada de [6]) .

3.4.5 Discussão

O algoritmo *Sequential Point Trees* é simples, de fácil implementação e provê uma renderização contínua usando níveis de detalhes. O autor enfatiza que grande parte do trabalho é movido para GPU, deixando a CPU livre para outras tarefas. No entanto, o SPT só é eficiente se o modelo estiver inteiramente na memória de vídeo, o que nem sempre é possível.

O algoritmo renderiza pontos em baixa qualidade, já que utiliza pontos renderizados usando funções do *pipeline* fixo do *OpenGL*, mais especificamente *GL_POINTS* com tamanhos variáveis. Outra desvantagem é que por não realizar *frustum culling* o método pode perder em eficiência em alguns casos.

Capítulo 4

Simplificação de Superfícies

Como mencionado anteriormente, modelos baseados em pontos complexos podem chegar a milhões ou mesmo bilhões de amostras. Para que seja possível processar ou visualizar estes modelos em tempo real, muitas vezes é necessário reduzir a complexidade realizando operações de simplificação. Estas técnicas geram aproximações adequadas do modelo reduzindo o número de amostras e procurando manter da melhor forma possível as características do modelo original. Formalmente, a definição de uma simplificação de superfícies baseadas em pontos pode ser formulada da seguinte forma:

Seja uma superfície $\mathcal{S}$ definida por uma nuvem de pontos $\mathcal{P}$. Dado um número amostras $n < |\mathcal{P}|$, compute uma nuvem de pontos $\mathcal{P}'$, onde $|\mathcal{P}'| = \mathbf{n}$, tal que a distância $\varepsilon = d(\mathcal{S}, \mathcal{S}')$ da superfície simplificada $\mathcal{S}'$ da superfície original $\mathcal{S}$ seja mínima.

Diversos algoritmos de simplificação utilizam diferentes heurísticas baseadas em erros locais para encontrar a melhor aproximação. Neste capítulo serão apresentados alguns métodos de simplificação divididos em duas categorias: superfícies baseadas em pontos *Puros* e baseadas em *Splats*.

4.1 Simplificação Baseada em Pontos *Puros*

Uma abordagem simples e muito usada na simplificação de nuvens de pontos é avaliar a importância de cada ponto. Esta importância é quantificada em um valor que indica a quantidade de informação que o ponto possui para formar a superfície,

ou a sua redundância. Linsen [36] defini a informação contida em um ponto como uma soma ponderada de fatores estimados pelos seus vizinhos, como distância e curvatura por exemplo. Em Alexa et al.[37] os pontos são removidos de acordo com a importância de sua contribuição na representação de uma superfície MLS (*Moving Least Squares*). Kalaiah e Varshney [38] medem a importância de cada ponto baseado nas propriedades diferenciais fornecidas pela sua vizinhança. Após a determinação da importância de cada ponto, a simplificação é então realizada removendo os pontos de baixa contribuição ou alta redundância.

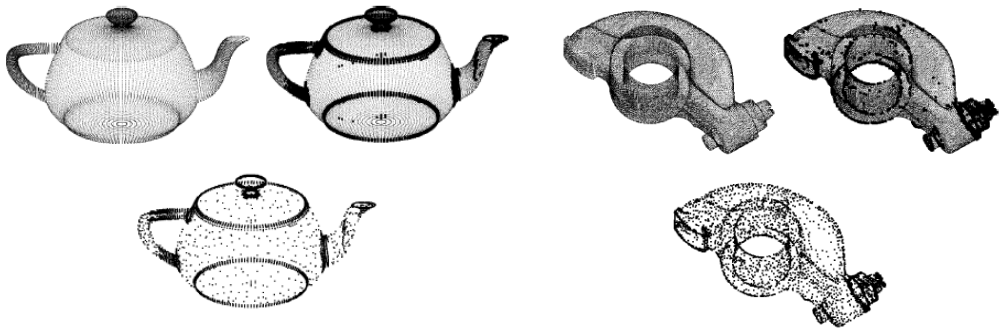


Figura 4.1: Método de simplificação proposto por [7]. Para os dois modelos exemplos são mostrados: o modelo original (**topo-esquerda**), os pontos classificados como de bordas destacados (**topo-direita**), e o modelo simplificado (**baixo**).

Os métodos de simplificação citados acima são baseados na suposição de que a superfície determinada pela nuvem de pontos seja suave. Contudo, modelos de peças mecânicas por exemplo, contem descontinuidades que são geralmente bordas afiadas separando a superfície em duas partes distintas. Song e Feng [7] abordaram este problema criando um método de simplificação baseado na mesma ideia do métodos citados acima, porém com duas diferenças principais. Primeiro, ao invés de operar sobre toda a nuvem de pontos, o algoritmo considera apenas uma região suave distinta por vez. Segundo, o grau de importância dos pontos é medido de forma diferente: enquanto Alexa et al.[37] focam em uma simplificação contínua, e Kalaiah e Varshney [38] na renderização e em uma distribuição homogênea, Song e Feng [7] buscam uma simplificação que preserve as partes característica do modelo, como ilustrado na Figura 4.1.

Os algoritmos citados acima criam uma nuvem de pontos simplificada realizando uma amostragem do modelo, ou seja, definindo um subconjunto da nuvem original ao

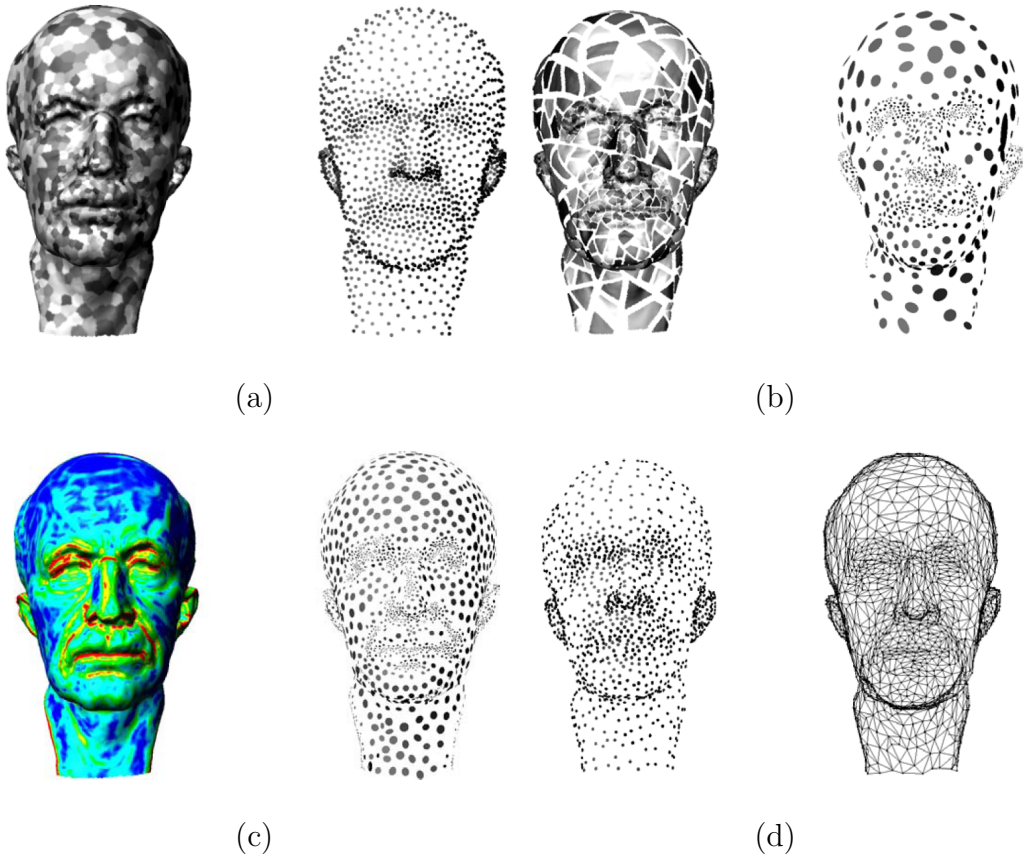


Figura 4.2: Métodos de simplificação propostos por [8]. **(a)** Clusterização Incremental. **(b)** Clusterização Hierárquica. **(c)** Simulação de Partículas. **(d)** Simplificação Interativas.

remover iterativamente pontos de acordo com uma métrica de erro. Enquanto estes métodos são simples e eficientes, suas estratégias de simplificação não garantem uma distribuição global uniforme. Além disso, os erros propostos por estes métodos por vezes não produzem um boa estimativa para todos casos, não sendo trivial formular métricas genéricas mais precisas.

Uma maneira sistemática de simplificação de pontos foi proposta por Pauly et al.[8], onde vários métodos de simplificação de polígonos foram adaptados para pontos. São eles:

- *Métodos de Clusterização:* Onde a nuvem de pontos $\mathcal{P}$ é dividida em subconjuntos. Os pontos deste subconjunto são então substituídos por um ponto representativo (geralmente seu centróide). Existem dois tipos de clusterização: incremental, onde clusters são criados sistematicamente ao crescer regiões através da agregação de pontos vizinhos; e o hierárquico, onde clusters são

criados tomando a nuvem de pontos original e dividindo-a em conjuntos menores.

- *Simplificação Iterativa:* Onde repetidamente seleciona-se pares de pontos juntando-os em um só. Note que para cada junção, há um acréscimo no erro total, no entanto, cria-se uma lista ordenada com todos pares possíveis, colocando no topo a junção que provoca o menor erro. Pares de pontos são então aglomerados de forma a minimizar o erro. Para avaliar o erro de junção dos pares de pontos é usada uma adaptação do Quadric Error Metric apresentado para malhas poligonais em [39].

- *Simulação de Partículas:* Onde há uma distribuição aleatória de um número $\mathbf{n}$ de pontos na superfície, tal que $|\mathcal{P}'| = \mathbf{n}$. Cada ponto é considerado como uma partícula que aplica forças de repulsão às vizinhas, causando um espalhamento até que a distribuição desejada seja alcançada.

Como discutido em [8], o método de clusterização incremental é o que possui o maior erro médio, seguido pelo método hierárquico, simulação de partículas e do método iterativo, que possui menor erro. A distribuição dos pontos produzida pelos métodos de clusterização se aproxima da distribuição original, que é desejável em alguns caso. Entretanto, simulação de partículas é a alternativa mais viável quando se deseja uma densidade uniforme ou um controle da densidade local da distribuição. Métodos de clusterização são os mais eficientes devido a sua simplicidade. Clusterização incremental, simplificação iterativa e simulação de partículas possuem custo de armazenamento linear em relação ao número total de amostras, requerendo muitas vezes uma capacidade grande de memória, enquanto a clusterização hierárquica depende apenas do tamanho do modelo simplificado. O autor discute que as três técnicas operam diretamente na nuvem de pontos ao invés de reconstruir a superfície a priori para aplicar o processo de simplificação. A Figura 5.4 ilustra estes métodos.

4.2 Simplificação Baseada em *Splats*

Como discutido no capítulo 2, para preencher buracos entre os pontos de forma mais eficiente, representações baseadas em pontos são frequentemente estendidas para

representações baseadas em *Splats* [23], onde a superfície é aproximada por discos ou elipses. Esta representação é de especial interesse no contexto de renderização, onde os *splats* facilitam a concepção de algoritmos eficientes [12, 28, 13, 14] que geram imagens de alta qualidade.

Embora a representação baseada em *splats* seja bastante utilizada na prática, muitos esquemas de simplificação tomam apenas o centro do *splat* em consideração (como discutido na seção anterior). A ideia é primeiro gerar o conjunto de pontos na qual a densidade se adapta a curvatura da superfície, e em um processo posterior converter os pontos em *splats* estimando sua normal e tamanho pela análise local de sua vizinhança. Em contraste, técnicas de simplificação baseadas em *splats*, exploram toda a geometria do disco ou elipse para obter o subconjunto que recobre a superfície. No entanto, estas técnicas são computacionalmente mais intensas.

Wu e Kobbelt [25] desenvolveram uma técnica de simplificação otimizada baseada em *splats* que explora a flexibilidade de modelos de pontos, ou seja, sem informação de topologia. Utilizando um esquema de otimização global é computada uma aproximação de toda a superfície com um conjunto mínimo de *splats* que satisfaz um erro global pré-determinado.

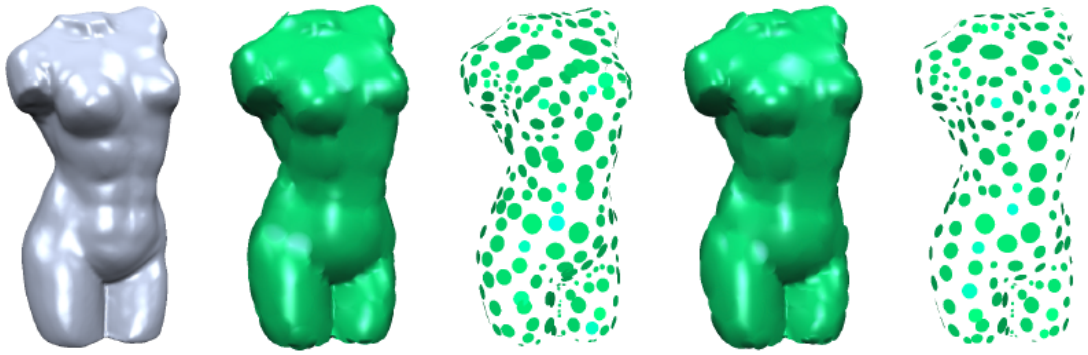


Figura 4.3: **Esquerda:** Modelo de um torso feminino 171.000 pontos. **Meio:** Modelo aproximado por 422 *splats* após o segundo passo (seleção gulosa). **Direita:** 333 *splats* após finalizado o terceiro passo (otimização global usando sistemas de partículas) A segunda e quarta imagem mostram os *splats* renderizados usando filtro EWA. Note a melhor distribuição dos *splats* após passo de otimização global

O objetivo da simplificação otimizada é computar um conjunto mínimo de *splats*

$\mathcal{T} = \{\mathbf{s_j}\}$ que aproxime o conjunto original $\mathcal{P}$ com um erro abaixo de uma tolerância ε. Em uma fase de pré-processamento propriedades locais da superfície, como vetor normal $\mathbf{n}_i$ e densidade ω_i, são calculadas para cada ponto $\mathbf{p}_i$ baseadas em uma vizinhança local. Dependendo da aplicação, pode-se decidir entre *splats* circulares ou elípticos. Um *splat* circular é definido por um centro $\mathbf{c}_i$, normal $\mathbf{n}_i$ e um raio $\mathbf{r}_i$; no caso do elíptico o raio $\mathbf{r}_i$ é substituído por dois vetores $\mathbf{t}_i^1$ e $\mathbf{t}_i^2$ que definem os seus eixos principais. Para assegurar o erro ε máximo, uma nova métrica de distância *splat*-ponto é introduzida: para um ponto $\mathbf{p}_i$ a distância ao *splat* $\mathcal{T}$ é definida como sendo a projeção ortogonal no plano do *splat* $\mathbf{s}_j$:

$$dist(\mathbf{p}_i, \mathcal{T}) = dist(\mathbf{p}_i, \mathbf{s}_j) = |\mathbf{n}_i^T(\mathbf{p}_i - \mathbf{c}_i)| \tag{4.1}$$

se

$$||(\mathbf{p}_i - \mathbf{c}_i) - \mathbf{n}_j^T(\mathbf{p}_i - \mathbf{c}_i)\mathbf{n}_j||^2 \leq \mathbf{r}_j^2 \tag{4.2}$$

no caso de um *splat* circular ou

$$(\mathbf{t}_j^{1^T}(\mathbf{p}_i - \mathbf{c}_i))^2 + (\mathbf{t}_j^{2^T}(\mathbf{p}_i - \mathbf{c}_i))^2 \leq 1 \tag{4.3}$$

se for um *splat* elíptico. Se $\mathbf{p}_i$ projetar no interior de vários *splats* a distância mínima é escolhida. Se nenhum $\mathbf{s}_j$ for encontrado pelas equações 4.2 ou 4.3, sua distância é definida como $dist(\mathbf{p}_i, \mathcal{T}) = \infty$.

A simplificação otimizada é realizada em três passos. No primeiro passo é computado o *splat* máximo para cada ponto $\mathbf{p}_i$ cujo erro é limitado por um ε. Começando por uma semente $\mathbf{p}_i$, o *splat* $\mathbf{s}_i$ cresce ao incorporar os vizinhos de acordo com suas distâncias. Esta distância é calculada usando as equações 4.2 e 4.3, sendo que os *splats* elípticos possuem a vantagem de adaptarem-se melhor à curvatura da superfície. Em um segundo passo, para o conjunto inicial de *splats*, um subconjunto que cobre a superfície sem que haja buracos entre os vizinhos é selecionado usando um procedimento guloso.

Por último, o procedimento guloso anterior é otimizado usando um procedimento global. A ideia é substituir iterativamente subconjuntos de *splats* por um novo subconjunto que possua os mesmo elementos ou que tenha uma distribuição melhor. Para melhorar o procedimento anterior, é utilizado um sistemas de partículas como em [8]. A figura 4.3 mostra o efeito desta simplificação otimizada após a finalização do último passo.

Wu el al. [26] desenvolveram um algoritmo iterativo com um abordagem gulosa para criar uma representação progressiva de *splats*. O seu funcionamento é semelhante ao *Progressive Meshes* de Hoppe [40], no entanto, são criados *splats* para cada conjunto de pontos. Em seguida, todas as possíveis operações de *junção* de splats são organizadas em uma lista de prioridades de acordo com uma *métrica de erro*, que avalia o erro causado pela respectiva operação de junção.

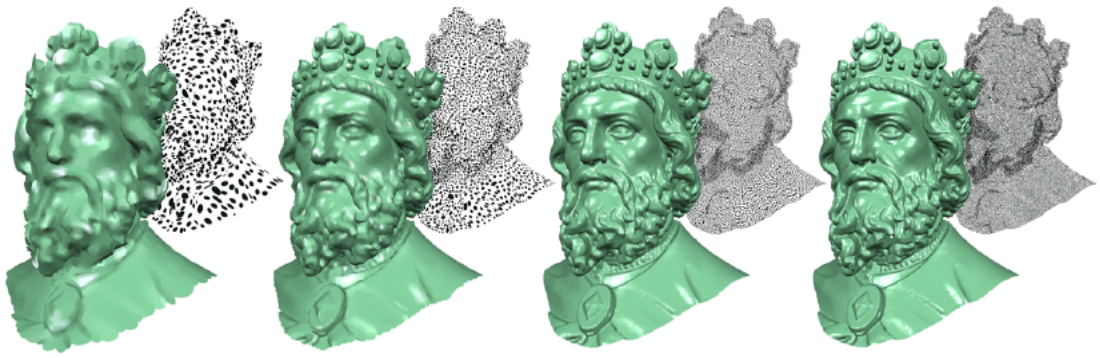


Figura 4.4: Representação progressiva do modelo de Carlos Magno (600.000 de pontos). Da esquerda para direita, modelos com: 2.000, 10.000, 70.000 e 600.000 *splats*

A entrada do algoritmo é um conjunto de pontos $\mathcal{P} = \{\mathbf{1} \ldots \mathbf{p}_i\}$ que são transformados em um conjunto de *splats* $\mathcal{T} = \{\mathbf{1} \ldots \mathbf{s_i}\}$. Cada *splat* representa uma elipse em $3D$ com centro $\mathbf{c}_i$, normal $\mathbf{n}_i$ e dois vetores ortogonais $\mathbf{t}_i^1$ e $\mathbf{t}_i^2$.

Similar a [8] e [25] os k-vizinhos de $\mathbf{p}_i$, $\mathcal{N}_k(\mathbf{p}_i)$, são computados previamente para analisar localmente a superfície (computar normal $\mathbf{n}_i$) e gerar o splat inicial $\mathbf{s}_i$. Para tanto, um plano $\mathcal{H}$ é formado para $\mathbf{p}_i$ e $\mathcal{N}_k(\mathbf{p}_i)$ definindo $\mathbf{s}_i$ com normal $\mathbf{n}_i$ e centro $\mathbf{c}_i = \mathbf{p}_i$. Como os *splats* iniciais são círculos, $\mathbf{t}_i^1$ e $\mathbf{t}_i^2$ serão dois vetores ortogonais paralelos a $\mathcal{H}$ com mesmo tamanho $\mathbf{r}_i$:

$$\mathbf{r}_i = \max_j ||(\mathbf{p}_j - \mathbf{c}_i) - \mathbf{n}_i^T(\mathbf{p}_j - \mathbf{c}_i)\mathbf{n}_i||, \tag{4.4}$$

para todo $\mathbf{p}_j \in \mathcal{N}_k(\mathbf{p}_i)$. Assim como feito por [8], $\mathcal{N}_k(\mathbf{p}_i)$ será usado para criar uma topologia dinâmica para auxiliar o processo de junção de *splats*. Sendo assim, é criado um grafo $\mathcal{G}(\mathcal{P}, E)$ onde uma aresta (i, j) pertence a E se $\mathbf{p}_j \in \mathcal{N}_k(\mathbf{p}_i)$. Então uma operação de junção Φ juntará dois *splats* $\mathbf{s}_a$ e $\mathbf{s}_b$ associados a dois pontos $\mathbf{p}_a$

e $\mathbf{p}_b$ de um aresta $\mathbf{e} \in E$ em um *splat* $\mathbf{s}_m$. A lista de prioridades conterá todas as possíveis operações de junção de todas as arestas pertencentes a E. Para avaliar a operação de junção de dois splats, dois erros foram entendidos. Um para medir o erro relativo à distância (L^2) e outro ao desvio da normal ($L^{2,1}$).

Métrica L^2

A métrica L^2 é baseada na distância Euclidiana. Para computar o erro causado pela operação Φ em relação ao conjunto de pontos originais, uma lista adicional de índices $\{f_i\}$ dos pontos originais é mantida para cada *splat* $\mathbf{s}_i$, onde o índice i refere-se ao ponto $\mathbf{p}_i$.

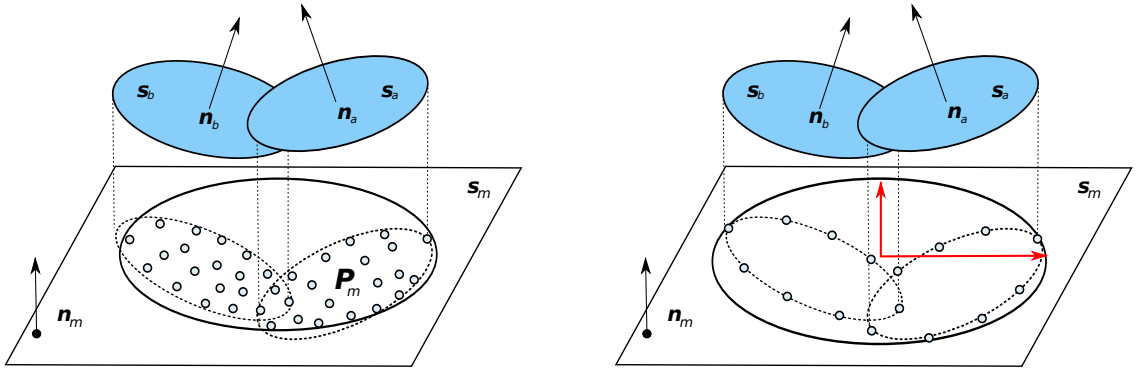


Figura 4.5: Junção dos *splats* de acordo com a métrica L^2 (esquerda) e $L^{2,1}$ (direita)

O erro da junção de dois *splats* $\mathbf{s}_a$ e $\mathbf{s}_b$ em um novo *splats* $\mathbf{s}_m$ é definido como:

$$\varepsilon_\Phi = ||e|| \cdot \sum_{f \in \{f_m\}} |dist(\mathbf{p}_f, \mathbf{s}_m)|^2, \{f_m\} = f_a + f_b. \tag{4.5}$$

Note que o erro é ponderado pelo tamanho da aresta a fim de penalizar a junção de *splats* muito distantes.

Dado a métrica de erro 4.5 e dois *splats* $\mathbf{s}_a$ e $\mathbf{s}_b$ para a junção, $\mathbf{s}_m$ é determinado pela método de análise das componentes principais (PCA) do conjunto de pontos $\mathcal{P}_m = \{\mathbf{p}_f\}$ diretamente em $3D$. Assim, teremos o ponto médio $\bar{\mathbf{p}}$ e três autovalores $\lambda_1 \geq \lambda_2 \geq \lambda_3$ e seus autovetores correspondentes $\mathbf{v}_1$, $\mathbf{v}_2$, $\mathbf{v}_3$.

Desta forma, $\mathbf{s}_m$ será determinado com centro $\mathbf{c}_m = \bar{\mathbf{p}}$, normal $\mathbf{n}_m = \mathbf{v}_3$ e os dois eixos $\mathbf{t}_m^1$ e $\mathbf{t}_m^2$ definidos pelas direções de $\mathbf{v}_1$ e $\mathbf{v}_2$ com tamanho respectivo a $\sqrt{\lambda_1/\lambda_2}$.

Os tamanho dos eixos são ajustados de forma que o *splat* englobe todo os pontos $\mathcal{P}_m$ quando projetados no plano definido por $\mathbf{s}_m$ (Figura 4.5).

Métrica $L^{2,1}$

A métrica de erro $L^{2,1}$ mede o desvio na direção da normal, sendo uma extensão da métrica original proposta em [41]. Neste caso o computo é simples e não há necessidade da lista de índices. Dada a operação de junção Φ e dois *splats* $\mathbf{s}_a$ e $\mathbf{s}_b$, com suas respectivas áreas $|\mathbf{s}_a|$ e $|\mathbf{s}_b|$, de forma similar a Equação 4.5 o erro ponderado pelo tamanho da aresta é definido como:

$$\varepsilon_\Phi = ||e|| \cdot |\mathbf{s}_a| + |\mathbf{s}_b| \cdot ||\mathbf{n}_a - \mathbf{n}_b||^2. \tag{4.6}$$

De acordo com a métrica $L^{2,1}$ o centro do novo *splat* $\mathbf{s}_m$ é definido como:

$$\mathbf{c}_m = \frac{|\mathbf{s}_a| \cdot \mathbf{c}_a + |\mathbf{s}_b| \cdot \mathbf{c}_b}{|\mathbf{s}_a| + |\mathbf{s}_b|} \tag{4.7}$$

e normal

$$\mathbf{n}_m = \frac{|\mathbf{s}_a| \cdot \mathbf{n}_a + |\mathbf{s}_b| \cdot \mathbf{n}_b}{|\mathbf{s}_a| + |\mathbf{s}_b|}. \tag{4.8}$$

Os vetores $\mathbf{t}_m^1$ e $\mathbf{t}_m^2$ são calculados da mesma forma que para métrica L^2, porém, como o conjunto de pontos $\mathcal{P}_m$ não é mais mantido para ser projetado, n pontos (geralmente 8 é o suficiente) são calculados na borda dos *splats* $|\mathbf{s}_a|$ e $|\mathbf{s}_b|$ e projetados no plano definido por $|\mathbf{s}_m|$. A partir destes aplica-se PCA e computa-se os tamanhos de $\mathbf{t}_m^1$ e $\mathbf{t}_m^2$.

Com as duas métricas de erro a sequência de operações de junção de *splats* Φ_i e sua operação inversa, operação de divisão do *splat*, podem ser criadas durante o processo de simplificação. Esse procedimento é similar ao proposto para malhas poligonais [40], porém sem manter informação de topologia.

4.3 Discussão

A aquisição de modelos de pontos (via $3D$ *scanners*) produz um grande número de amostras, tornando difícil um processamento eficiente deste modelos. Por isso técnicas de simplificação são importantes em qualquer processo de modelagem ou visualização de nuvem de pontos. Neste capítulo foram discutidos várias técnicas de

simplificação de modelos, considerando apenas pontos. Enquanto estas técnicas são eficientes, elas possuem uma desvantagem de produzirem uma aproximação muito simples. Uma representação mais elaborada baseada em *splats* é uma extensão popular de superfícies baseada em pontos, principalmente quando se deseja uma visualização eficiente e com qualidade. Poucos algoritmos utilizam toda a geometria dos *splats* para guiar a simplificação, o que, apesar de serem mais elaborados, evitam um processo posterior para recriar os splats. No capítulo a seguir será apresentada uma simplificação baseada em *splats*.

Capítulo 5

Método Proposto

Neste capítulo apresentaremos um algoritmo para simplificação de superfícies representada por *splats*. Utilizamos uma abordagem incremental, onde clusters são criados sistematicamente ao crescer regiões através da agregação de *splats* vizinhos. Este crescimento é guiado por dois erros: o *erro perpendicular*, que mede a variação dos *splats* na direção da normal; e o *erro tangencial*, que nos fornece uma medida de cobertura. Partiremos da suposição de que o modelo já possui uma representação em *splats* e que inicialmente todos são *splats* circulares.

5.1 Motivação

Como mencionado no Capítulo 4, modelos baseados em pontos são complexos devido à grande quantidade de amostras necessárias para representá-lo, tornando operações de simplificação uma ferramenta necessária para que seja possível alcançar taxas interativas de processamento e visualização; splats são muitos usados no contexto de renderização quando se deseja qualidade e eficiência (Seção 2.4). Embora a representação de superfícies baseadas em *splats* seja comum na prática, muitos algoritmos de simplificação utilizam apenas o centro em consideração; a ideia é em um primeiro momento gerar uma superfície com uma densidade de pontos que se ajuste a curvatura da superfície e em um processamento posterior, converter estes pontos em splats circulares ou elípticos. Em contraste a esta abordagem, simplificação de superfícies baseada em *splats* levam em consideração toda a geometria do *splat*.

Muitos algoritmos de simplificação [33, 28, 30] são baseados em estruturas

hierárquicas. Os pontos são organizados em uma estrutura de partição espacial (Capítulo 3) e os *splats* criados pela análise local da superfície. Estas técnicas são simples e rápidas, mas como não existem estratégias de otimização mais elaborada, os resultados geralmente são conservativos e tendem a produzir uma quantidade significativa de *splats* redundantes. A fim de produzir amostras com maior qualidade, usaremos uma abordagem gulosa e incremental para renderização eficiente usando *splats*, onde clusters são criados adicionando vizinhos de acordo com métricas de erro definidas a diante (Seções 5.3 e 5.4).

5.2 Clusterização Incremental

Um cluster C_0 é construído adicionando-se sucessivamente os vizinhos mais próximos de $\mathbf{s}_j$ a partir de uma semente $\mathbf{s}_j \in \mathcal{T}$ escolhida randomicamente. Este crescimento incremental para quando um limitante φ_p (para erro perpendicular) e φ_t (para o erro tangencial) é alcançado. O próximo cluster C_1 é então construído seguindo o mesmo processo, com uma nova semente escolhida na vizinhança de C_0. Inicialmente, cada vizinho que possui erro perpendicular abaixo de φ_p é utilizado como possível candidato a membro do cluster. Com isso, o novo centro $\mathbf{c}_m$ e normal $\mathbf{n}_m$ do *splat* $\mathbf{s}_m$ que representa o cluster são atualizados (ver 5.2.1). Com $\mathbf{c}_m$ e $\mathbf{n}_m$, podemos atualizar as extensões do *splat* $\mathbf{s}_m$ (ver 5.2.2) e, consequentemente, computar o novo erro tangencial $\mathbf{e}_t$. Caso este erro esteja acima de φ_t, a configuração do último vizinho adicionado é estabelecida como o cluster final. O processo continua até que todos os *splats* de $\mathcal{T}$ pertençam a algum cluster C_i.

5.2.1 Centro e Normal

Para calcular o novo centro $\mathbf{c}_m$ e normal $\mathbf{n}_m$, utilizamos uma soma ponderada pela área dos *splat* de acordo com a equação 4.7 e 4.8 (métrica L^2). Assim, dado um conjunto de *splats*, o centro $\mathbf{c}_m$ e a normal $\mathbf{n}_m$ do novo *splat* podem ser definidos respectivamente como :

$$\mathbf{c}_m = \frac{\sum_i |\mathbf{s}_i| \cdot \mathbf{c}_i}{\sum_i |\mathbf{s}_m|}, \tag{5.1}$$

$$\mathbf{n}_m = \frac{\sum_i |\mathbf{s}_i| \cdot \mathbf{n}_i}{\sum_i |\mathbf{s}_m|}. \tag{5.2}$$

5.2.2 Eixos da Elipse

Similar ao método apresentado na Seção 4.2, a fim de obter os eixos da elipse $\mathbf{s}_m$, projetamos 8 pontos da borda das elipses $\mathbf{s}_i$ no plano definido por $\mathbf{c}_m$ e $\mathbf{n}_m$. Aplicamos *PCA* no conjunto de pontos projetados C' para obter os autovalores $\lambda_1 \geq \lambda_2 \geq \lambda_3$ e autovetores unitários v_1, v_2, v_3. A direção dos eixos da elipse $\mathbf{s}_m$ é então definida pelos vetores v_1 e v_2. Assumindo que a excentricidade da elipse é dada por $\sqrt{\frac{\lambda_1}{\lambda_2}}$, pode-se computar as dimensões dos eixos principais como:

$$|t_m^1| = \sqrt{\lambda_1} \cdot \bar{\mathbf{e}}, \tag{5.3}$$

$$|t_m^2| = \sqrt{\lambda_2} \cdot \bar{\mathbf{e}}, \tag{5.4}$$

onde

$$\bar{\mathbf{e}} = \frac{\max_{q \in C'} \{(\langle q, v_1 \rangle^2) \cdot \lambda_1 + (\langle q, v_2 \rangle^2) \cdot \lambda_2\}}{\lambda_1 \cdot \lambda_2}. \tag{5.5}$$

5.3 Erro Perpendicular

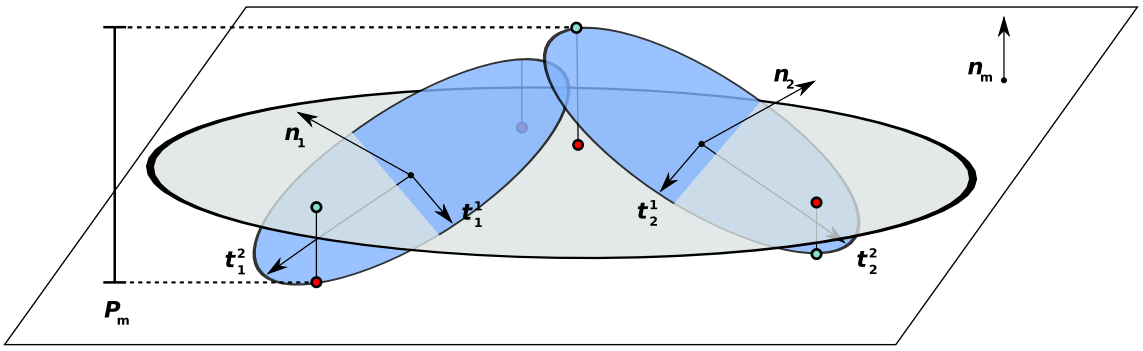


Figura 5.1: O erro perpendicular mede a distância máxima dos *splats* $\mathbf{s}_i$ na direção perpendicular ao plano $\mathbf{P}_m$ definido por pelo *splat* $\mathbf{s}_m$

O problema de determinar a distância máxima de uma elipse $\mathbf{s}_i$, do conjunto que está sendo agregado, ao plano da elipse $\mathbf{s}_m$ (que passa a representar esse conjunto), não é tão imediato como no caso circular. Para calcular a chamada distância

perpendicular entre $\mathbf{s}_i$ e $\mathbf{s}_m$, que notaremos como $\mathcal{D}(i,m)$, precisamos executar o seguinte processo. Seja:

$$v(\alpha) = t_i^1 \cos(\alpha) + t_i^2 \sin(\alpha) + c_i, \tag{5.6}$$

a representação paramétrica do contorno da elipse $\mathbf{s}_i$ e n_m a normal ao plano P_m da elipse $\mathbf{s}_m$. Para maximizar em α a distância entre $v(\alpha)$ e P_m precisamos resolver o seguinte problema de otimização:

$$\max_{\alpha\in[0,2\pi]} \langle v(\alpha) - c_m, n_m\rangle = \langle t_i^1, n_m\rangle \cos(\alpha) + \langle t_i^2, n_m\rangle \sin(\alpha) + \langle c_i - c_m, n_m\rangle. \tag{5.7}$$

Para resolver esse problema começamos igualando a zero a derivada da função objetiva, o que determina que a solução procurada α_{max} deve satisfazer a:

$$\langle t_i^2, n_m\rangle \cos(\alpha_{max}) - \langle t_i^1, n_m\rangle \sin(\alpha_{max}) = 0 \Longrightarrow \frac{\langle t_i^2, n_m\rangle}{\langle t_i^1, n_m\rangle} = \tan(\alpha_{max}), \tag{5.8}$$

onde:

$$\begin{aligned} cos(\alpha_{max}) &= \pm\frac{\langle t_i^1, n_m\rangle}{\sqrt{\langle t_i^1, n_m\rangle^2 + \langle t_i^2, n_m\rangle^2}} \\ sin(\alpha_{max}) &= \pm\frac{\langle t_i^2, n_m\rangle}{\sqrt{\langle t_i^1, n_m\rangle^2 + \langle t_i^2, n_m\rangle^2}}. \end{aligned} \tag{5.9}$$

Para identificar inteiramente α_{max} resta apenas decidir acerca dos sinais que aparecem nas expressões acima, o que significa distinguir os casos de distância máxima e distância mínima a P_m. Isto é feito observando que os resultados acima nos permitem escrever $\mathcal{D}(i,m)$ da seguinte forma:

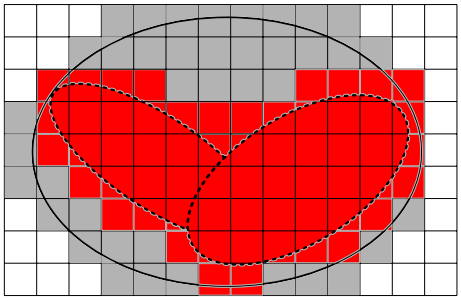
$$\begin{aligned} \mathcal{D}(i,m) &= \max_{sign\in\{-1,1\}} |sign \cdot \frac{\langle t_i^1, n_m\rangle^2 + \langle t_i^2, n_m\rangle^2}{\sqrt{\langle t_i^1, n_m\rangle^2 + \langle t_i^2, n_m\rangle^2}} + \langle c_i - c_m, n_m\rangle| \\ \mathcal{D}(i,m) &= \sqrt{\langle t_i^1, n_m\rangle^2 + \langle t_i^2, n_m\rangle^2} + |\langle c_i - c_m, n_m\rangle|. \end{aligned}$$

Podemos então utilizar $\max_i\{\mathcal{D}(i,m)\}$ como medida do erro perpendicular. Alternativamente, podemos estender a definição de erro perpendicular dada na seção 3.4.2 para o caso circular, meramente substituindo na equação 3.3, d_i por $\sqrt{\langle t_i^1, n_m\rangle^2 + \langle t_i^2, n_m\rangle^2}$, o que nos permitirá obter:

$$\begin{aligned} \mathbf{e}_p &= \max\{\langle c_i - c, n\rangle + \sqrt{\langle t_i^1, n_m\rangle^2 + \langle t_i^2, n_m\rangle^2}\} - \\ &\quad \min\{\langle c_i - c, n\rangle - \sqrt{\langle t_i^1, n_m\rangle^2 + \langle t_i^2, n_m\rangle^2}\}. \end{aligned}$$

5.4 Erro Tangencial

O erro tangencial mede a diferença entre a elipse que representa o cluster ($\mathbf{s}_m$) e o conjunto (Γ) formado pelas projeções das elipses que fazem parte do cluster sobre o plano de $\mathbf{s}_m$. Há duas maneiras naturais de avaliar a diferença entre dois conjuntos planares. A primeira é dada pela área da diferença simétrica entre os conjuntos. No caso, como $\mathbf{s}_m$ contem Γ, essa diferença é simplesmente $\mathbf{s}_m - \Gamma$. A segunda pela maior distância de um ponto de um conjunto a outro. Como $\Gamma \subseteq \mathbf{s}_m$, podemos nos restringir a encontrar o ponto q em $\mathbf{s}_m$ mais afastado de Γ. Para obter computacionalmente a área de $\mathbf{s}_m - \Gamma$ ($\mathcal{A}(\mathbf{s}_m - \Gamma)$), pensamos em duas estratégias :



(a)

Figura 5.2: Segunda proposta de erro tangencial. Imegir as elipses e calcular a diferença de área pela grade regular.

1. A primeira visa obter essa área de forma exata. De fato, identificando os pontos extremos de cada elipse em relação a uma das coordenadas e os ordenando segundo ela, podemos por um algoritmo de varredura de linhas identificar as interseções dos contornos das elipses de Γ e, a partir delas, $\mathcal{A}(\mathbf{s}_m - \Gamma)$. Esse procedimento, entretanto, tem uma complexidade que torna impraticável a sua utilização para nossos propósitos. Para concluir isso, basta considerar que ele é $O(n_x \log n_x)$, onde n_x é o número de interseções entre elipses de Γ, o qual já é quadrático no número dessas elipses. Além disso, o simples cômputo de uma interseção entre duas elipses em posição genérica já envolve a resolução de uma equação do quarto grau. Assim, pretender encontrar o valor exato de $\mathcal{A}(\mathbf{s}_m - \Gamma)$ é uma tarefa impraticável.

2. A segunda forma busca uma solução aproximada ao imergir as elipses envolvidas numa malha regular e determinar a situação de pertinência em relação a

$\mathbf{s}_m$ e Γ do centro de cada célula. Uma célula cujo centro pertence a $\mathbf{s}_m$ e não a Γ, contribui com sua área para o cômputo de $\mathcal{A}(\mathbf{s}_m - \Gamma)$ (Figura 5.2). A determinação da pertinência com relação a todas as elipses de Γ, precisando ser efetuada para um número de pontos quadrático na resolução da malha, prejudica a performance dessa alternativa. Isto obriga que se empregue uma malha muito grosseira ou que se limite o número de elipses sendo agregadas, tornando, portanto, esta opção também não inteiramente satisfatória.

Tendo em vista as dificuldades apontadas acima na obtenção da área de $\mathbf{s}_m - \Gamma$ nos concentramos em obter o erro tangencial via uma aproximação computacionalmente viável da $dist(\mathbf{s}_m, \Gamma)$, a distância a Γ do ponto de s_m mais afastado dele. A determinação de $dist(\mathbf{s}_m, \Gamma)$ fica facilitada se assumirmos que o ponto de $\mathbf{s}_m$ mais distante de Γ estiver na sua borda. Isso certamente é verdade se o número de elipses em Γ for igual a 2 e em casos reais, devido a própria limitação do tamanho do cluster e do fato do próprio processo de expansão que gera os clusters não fazer curvas acentuadas. Notamos que essa suposição não causou distorções acentuadas no cálculo do erro tangencial. Se podemos limitar a busca do ponto mais distante de Γ a borda de $\mathbf{s}_m$, podemos aproximar $dist(\mathbf{s}_m, \Gamma)$ de forma aceitável e com custo computacional adequado, da seguinte forma:

- O contorno da elipse $\mathbf{s}_m$ que representa o cluster é discretizado num conjunto, $\mathcal{C} = \{p_1, \ldots, p_8\}$, contendo 8 de seus pontos. Especificamente, são aqueles que definem com o centro da elipse raias que fazem com o eixo maior um ângulo múltiplo de 45^o.

- Seja $\mathbf{s}_i$ uma elipse do conjunto $\mathcal{T} = \{\mathbf{s}_1, \ldots, \mathbf{s}_n\}$, que está sendo condensado, e $\mathbf{s}'_i$ uma elipse do conjunto $\mathcal{T}' = \{\mathbf{s}'_i, \ldots, \mathbf{s}'_n\}$ de elipses projetadas sobre o plano definido por $\mathbf{s}_m$. A distância de cada um dos pontos ($p_k \in \mathcal{C}$) em que se discretiza $\mathbf{s}_m$ a $\mathbf{s}'_i$ – que notaremos por $dist(p_k, \mathbf{s}'_i)$ – é obtida de forma aproximada da forma indicada abaixo. Os elementos referidos no texto a seguir estão representados na Figura 5.3.

1. No caso óbvio em que $p_k \in \mathbf{s}'_i$, teremos $dist(p_k, \mathbf{s}'_i) = 0$.

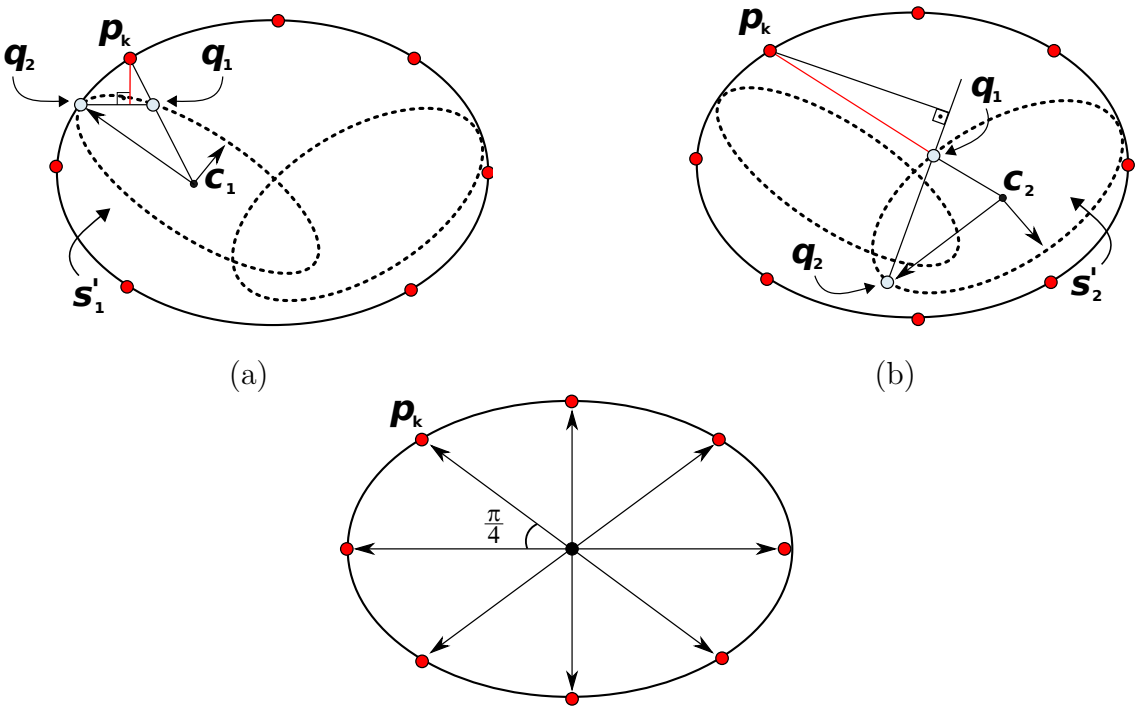


Figura 5.3: Distância aproximada de uma elipse $\mathbf{s}'_i$ do conjunto das elipses projetadas $\mathcal{T}'$ ao contorno de $\mathbf{s}_m$ discretizado em 8 pontos.(b) Quando a projeção n o está no segmento delimitado por $\overline{q_1 q_2}$, o erro tangencial é tomado com sendo a distância de p_k a q_1. O segmento em vermelho representa o erro tangencial. Embaixo, ilustra a discretização de $\mathbf{s}_m$.

2. Caso contrário, considere o segmento de reta delimitado pelo ponto q_1, onde a raia $[c_i, p_k]$ corta a elipse $\mathbf{s}'_i$, e q_2 sendo a extremidade do eixo maior de $\mathbf{s}'_i$ que está mais próxima de p_k.

3. Computa-se a distância mínima de p_k ao segmento $[q_1, q_2]$ e usa-se essa distância como aproximação de $dist(p_k, \mathbf{s}'_i)$).

Utilizando a distância aproximada definida acima, se obtém para cada p_k sua distância ao conjunto de elipses $\mathcal{T}'$ – dado por $\min_{\mathbf{s}'_i \in \mathcal{T}'}(dist_{p_k \in \mathcal{C}}(p_k, \mathbf{s}'_i))$. O erro tangencial será finalmente obtido tomando-se a maior dessas distâncias. Ele pode ser então expresso por:

$$e_t \quad = \quad \max\{\min_{\mathbf{s}'_i \in \mathcal{T}'}(dist_{p_k \in \mathcal{C}}(p_k, \mathbf{s}'_i))\} \tag{5.10}$$

5.5 Resultados

A Tabela 5.1 expõe os resultados das simplificações de 3 modelos. O limitante para erro perpendicular φ_p foi tomado como sendo o dobro da distância do vizinho mais próximo à semente: $2 \cdot \min_{p \in \mathcal{N}_k(\mathbf{s}_{seed})} dist(\mathbf{s}_{seed}, p)$. O limitante tangencial φ_t foi tomado como sendo a dimensão do eixo menor da semente $||\mathbf{t}^1_{seed}||$.

Modelo	N° *Splats* Originais	N° *Splats* Simplificação
Bunny	139990	11258
Armadillo	172974	11964
Dragon	437645	37959

Tabela 5.1: Número de *splats* após simplificação usando nosso algoritmo.

Foi usada uma *K-D Tree* para computar os $k-$vizinhos mais próximos da semente. Foram usados 32 vizinhos mais próximos, mas este número é dependente da densidade do modelo; sendo assim, o tamanho do cluster fica limitado ao máximo de 32.

5.6 Discussão

Este capítulo apresentou um algoritmo de simplificação baseado em *splats* bem como duas extensões de métricas de erro para o caso elíptico, o erro perpendicular e o tangencial. O erro perpendicular fornece uma medida de variância na direção da normal, permitindo distinguir regiões com curvatura acentuada de regiões planares. O erro tangencial nos fornece uma medida de cobertura, evitando que os cluster cresçam de forma descontrolada e, consequentemente, cubram áreas desnecessárias.

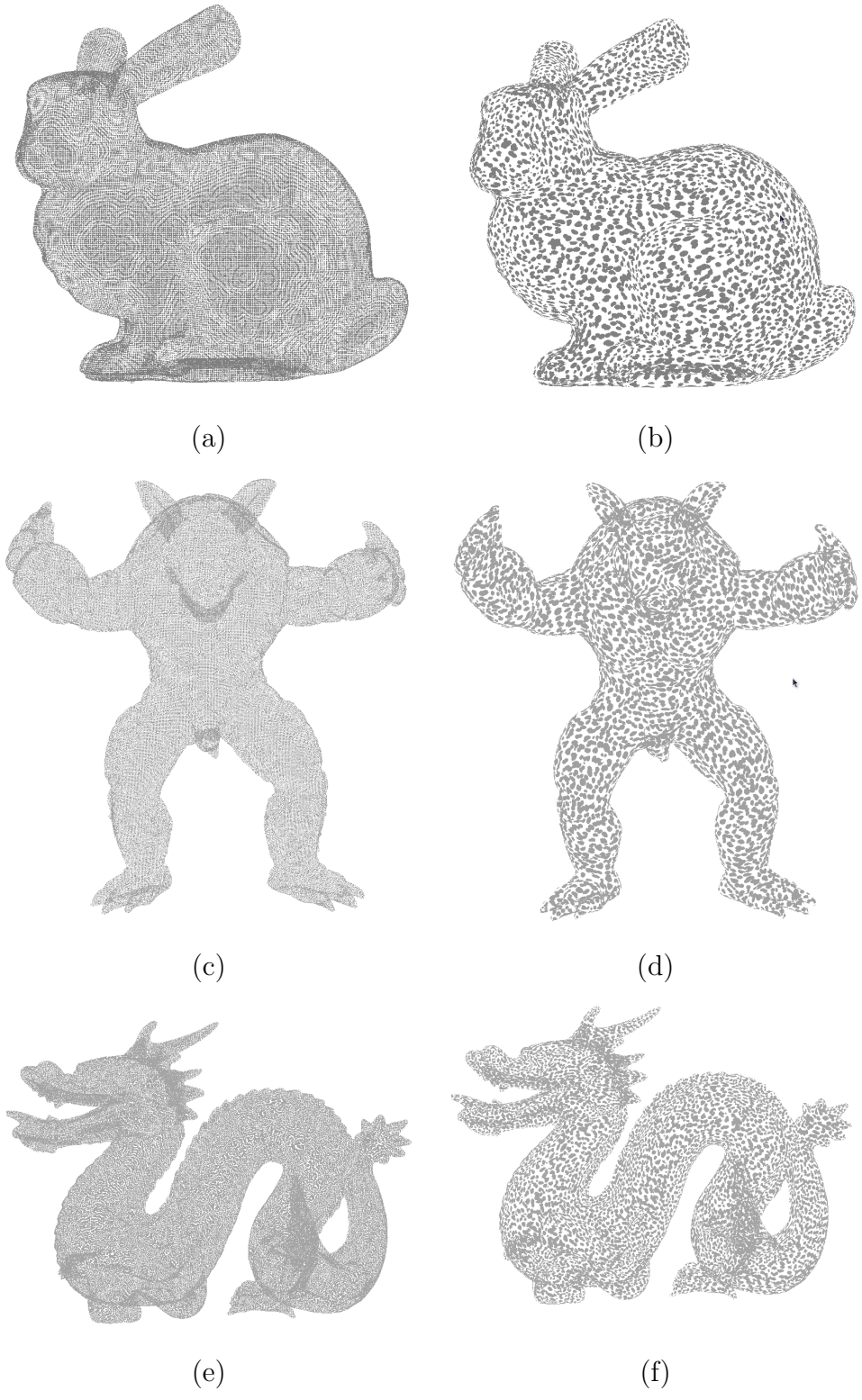


Figura 5.4: Modelos apresentados na tabela 5.1.**(a)** Modelo *Bunny* original: 139990. **(b)** *Bunny* simplificado: 11258. **(c)** Modelo *Armadillo* original: 172974. **(d)** Modelo simplificado: 11964.**(e)** Modelo *Dragon* original: 437645 **(f)** Modelo simplificado 37959.

Capítulo 6

Conclusão e Trabalhos Futuros

Representação de superfícies por *splats* se mostra vantajosa em muitos casos, principalmente quando o objetivo é visualizar interativamente modelos contendo um grande número de amostras, como os produzidos por $3D$ *scanners*. *Splats* permitem a renderização em alta qualidade ao acrescentar informações locais da superfície aos pontos, como por exemplo, valores de densidade e propriedades diferenciais (normal e vetores tangentes).

A contribuição desta dissertação é a elaboração de um algoritmo de simplificação que atua diretamente nos *splats*. Diferentemente dos algoritmos que trabalham com apenas a posição geometrica (pontos "puros"), os *splats* permitem guiar a simplificação de uma forma mais precisa, além de não necessitar de conversões ou de manter diferentes representações do modelo. Ainda mais, a utilização de *splats* elípticos permitem uma aproximação com menos elementos por se adaptarem melhor à superfície. Objetivando uma simplificação de alta qualidade, uma aborgagem incremental de clusterização foi utilizada.

6.1 Trabalhos Futuros

A abordagem incremental utilizada facilita a criação de uma hierarquia de volumes envolventes, e consequentemente gerar estruturas de dados específicas para visualização. Com este intuito, e com base nos trabalhos relacionados, existem dois passos naturais a serem seguidos a partir da abordagem de simplificação apresentada:

- gerar uma hieraquia de volumes envolventes para criar uma estrutura de ren-

derização em nível de detalhes como a aprensentada pelo *QSplat* (Seção 3.3);

- converter esta estrutura, usando os erros estendidos apresentados, para uma estrutura como a apresentada pelo *SPT* (Seção 3.4).

Adicionalmente, algumas otimizações podem ser feitas no algoritmo de simplificação, entre elas:

- uma forma mais eficiente para encontrar os $k-$vizinhos mais próximos;
- obter incrementalmente o próximo candidato a membro do cluster, na forma atual o *splat* é re-computado a cada iteração;
- realizar otimizações globais a fim de obter uma melhor distribuição dos cluster na superfície.

Finalmente, é de especial interesse guiar a simplificação utilizando outros aspectos além da geometria, por exemplo, levando em consideração atributos como a cor das amostras.

Referências Bibliográficas

[1] BOUBEKEUR, T., *Hierarchical Processing, Editing and Rendering of Acquired Geometry*, Ph.D. Thesis, University of Bordeaux, September 2007.

[2] LEVOY, M., WHITTED, T., *The Use of Points as a Display Primitive*, Tech. Rep. 85-022, Department of Computer Science,University of North Carolina at Chapel Hill, 1985.

[3] ADAMS, B., *Point-Based Modeling, Animation and Rendering of Dynamic Objects (Puntgebaseerde modellering, animatie en visualisatie van dynamische objecten)*, Ph.D. Thesis, Department of Computer Science, K.U.Leuven, Leuven, Belgium, May 2006, Dutré, Philip (supervisor), Willems, Yves (cosupervisor).

[4] BOTSCH, M., SPERNAT, M., KOBBELT, L., "Phong Splatting ". In: *SPBG'04: Proceeding of the Symposium on Point-Based Graphics 2004*, pp. 25–32, 2004.

[5] WAND, M., *Point-Based Multi-Resolution Rendering*, Ph.D. Thesis, Department of computer science and cognitive science, University of Tübingen, 2004.

[6] DACHSBACHER, C., VOGELGSANG, C., STAMMINGER, M., "Sequential point trees", *ACM Trans. Graph.*, v. 22, n. 3, pp. 657–662, 2003.

[7] SONG, H., FENG, H.-Y., "A progressive point cloud simplification algorithm with preserved sharp edge data", *The International Journal of Advanced Manufacturing Technology*, v. 45, n. 5-6, pp. 583–592, November 2009.

[8] PAULY, M., GROSS, M., KOBBELT, L. P., "Efficient simplification of point-sampled surfaces". In: *VIS '02: Proceedings of the conference on Visuali-*

zation '02, pp. 163–170, Washington, DC, USA, IEEE Computer Society, 2002.

[9] LEVOY, M., PULLI, K., CURLESS, S., etal., "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, New York, NY, USA, ACM Press/Addison-Wesley Publishing Co., 2000.

[10] CORRÊA, W. T., FLEISHMAN, S., SILVA, C. T., "Towards Point-Based Acquisition and Rendering of Large Real-World Environments". In: *In Proceedings of the 15th Brazilian Symposium on Computer Graphics and Image Processing*, 2002.

[11] 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.

[12] 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.

[13] BOTSCH, M., KOBBELT, 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, Washington, DC, USA, IEEE Computer Society, 2003.

[14] MARROQUIM, R., KRAUS, M., CAVALCANTI, P. R., "Efficient Point-Based Rendering Using Image Reconstruction". In: *Symposium on Point-Based Graphics 2007, Prague-Czech Republic*, September 2007.

[15] SCHEIBLAUER, C., ZIMMERMAN, N., WIMMER, M., "Interactive Domitilla Catacomb Exploration". In: *VAST 2009 ? 10th International Symposium on Virtual Reality, Archaeology, and Cultural Heritage*, pp. 65–72, 2009.

[16] GROSS, M., PFISTER, H., *Point-Based Graphics (The Morgan Kaufmann Series in Computer Graphics)*. San Francisco, CA, USA, Morgan Kaufmann Publishers Inc., 2007.

[17] ALEXA, M., (EDITORS, S. R., WEYRICH, T., etal., "Post-processing of Scanned 3D Surface Data", 2007.

[18] CSURI, C., HACKATHORN, R., PARENT, R., etal., "Towards an interactive high visual complexity animation system". In: *SIGGRAPH '79: Proceedings of the 6th annual conference on Computer graphics and interactive techniques*, pp. 289–299, New York, NY, USA, ACM, 1979.

[19] BLINN, J. F., "Light reflection functions for simulation of clouds and dusty surfaces", *SIGGRAPH Comput. Graph.*, v. 16, n. 3, pp. 21–29, 1982.

[20] REEVES, W. T., "Particle Systems — a Technique for Modeling a Class of Fuzzy Objects", *ACM Trans. Graph.*, v. 2, n. 2, pp. 91–108, 1983.

[21] XU, H., NGUYEN, M. X., YUAN, X., etal., "Interactive Silhouette Rendering for Point-Based Models". In: *Proceeding of the 1st Eurographics Symposium on Point-Based Graphics*, pp. 13–18, color plate 215, 2004.

[22] PFISTER, H., ZWICKER, M., VAN BAAR, J., etal., "Surfels: surface elements as rendering primitives". In: *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pp. 335–342, New York, NY, USA, ACM Press/Addison-Wesley Publishing Co., 2000.

[23] ZWICKER, M., PFISTER, H., VAN BAAR, J., etal., "Surface Splatting". In: *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pp. 371–378, New York, NY, USA, ACM, 2001.

[24] KOBBELT, L., BOTSCH, M., "A survey of point-based techniques in computer graphics", *Comput. Graph.*, v. 28, n. 6, pp. 801–814, 2004.

[25] WU, J., KOBBELT, L., "Optimized Sub-Sampling of Point Sets for Surface Splatting", *Comput. Graph. Forum*, v. 23, n. 3, pp. 643–652, 2004.

[26] WU, J., ZHANG, Z., KOBBELT, L., "Progressive splatting", *Symposium on Point-Based Graphics*, v. 0, pp. 25–142, 2005.

[27] SAMET, H., *Foundations of Multidimensional and Metric Data Structures (The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling)*. San Francisco, CA, USA, Morgan Kaufmann Publishers Inc., 2005.

[28] BOTSCH, M., WIRATANAYA, A., KOBBELT, L., "Efficient high quality rendering of point sampled geometry". In: *EGRW '02: Proceedings of the 13th Eurographics workshop on Rendering*, pp. 53–64, Aire-la-Ville, Switzerland, Switzerland, Eurographics Association, 2002.

[29] SAINZ, M., PAJAROLA, R., "Point-based rendering techniques", *Comput. Graph.*, v. 28, n. 6, pp. 869–879, 2004.

[30] PAJAROLA, R., "Efficient Level-of-Details for Point Based Rendering". In: *Proceedings IASTED Invernational Conference on Computer Graphics and Imaging*, Calgary, AB, Canada,, ACTA Press, 2003.

[31] PAJAROLA, R., SAINZ, M., GUIDOTTI, P., "Confetti: Object-Space Point Blending and Splatting", *IEEE Transactions on Visualization and Computer Graphics*, v. 10, n. 5, pp. 598–608, 2004.

[32] BOUBEKEUR, T., HEIDRICH, W., GRANIER, X., etal., "Volume-Surface Trees", *Computer Graphics Forum (Proceedings of EUROGRAPHICS 2006)*, v. 25, n. 3, pp. 399–406, 2006.

[33] 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, New York, NY, USA, ACM Press/Addison-Wesley Publishing Co., 2000.

[34] GÄRTNER, B., "Fast and Robust Smallest Enclosing Balls". In: *ESA '99: Proceedings of the 7th Annual European Symposium on Algorithms*, pp. 325–338, London, UK, Springer-Verlag, 1999.

[35] RUSINKIEWICZ, S., LEVOY, M., "Streaming QSplat: a viewer for networked visualization of large, dense models". In: *I3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics*, pp. 63–68, New York, NY, USA, ACM, 2001.

[36] LINSEN, L., *Point Cloud Representation*, Tech. Rep. 03, Fakultät für Informatik, Universität Karlsruhe, 2001.

[37] ALEXA, M., BEHR, J., COHEN-OR, D., etal., "Point set surfaces". In: *VIS '01: Proceedings of the conference on Visualization '01*, pp. 21–28, Washington, DC, USA, IEEE Computer Society, 2001.

[38] KALAIAH, 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.

[39] GARLAND, M., HECKBERT, P. S., "Surface simplification using quadric error metrics". In: *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pp. 209–216, New York, NY, USA, ACM Press/Addison-Wesley Publishing Co., 1997.

[40] HOPPE, H., "Progressive meshes". In: *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pp. 99–108, New York, NY, USA, ACM, 1996.

[41] COHEN-STEINER, D., ALLIEZ, P., DESBRUN, M., "Variational shape approximation". In: *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pp. 905–914, New York, NY, USA, ACM, 2004.

[42] MOENNING, C., DODGSON, N. A., "A New Point Cloud Simplification Algorithm". In: *In Proceedings 3rd IASTED Conference on Visualization, Imaging and Image Processing*, pp. 1027–1033, 2003.

[43] PAULY, M., GROSS, M., "Spectral processing of point-sampled geometry". In: *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pp. 379–386, New York, NY, USA, ACM, 2001.

[44] ELDAR, Y., LINDENBAUM, M., PORAT, M., etal., "The farthest point strategy for progressive image sampling", *Image Processing, IEEE Transactions on*, v. 6, n. 9, pp. 1305–1315, Sep 1997.

[45] 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, Aire-la-Ville, Switzerland, Switzerland, Eurographics Association, 2002.

[46] ROSSIGNAC, J., BORREL, "Multi-Resolution 3D Approximations for Rendering Complex Scenes". In: *Modeling in Computer Graphics: Methods and Applications*, pp. 455–465, New York, NY, USA, Springer-Verlag, 1993.

[47] GARLAND, M., *Quadric-based polygonal surface simplification*, Ph.D. Thesis, Department of Computer Science , Carnegie Mellon University., Pittsburgh, PA, USA, 1999, Chair-Heckbert, Paul.