Pontifícia Universidade Católica
do Rio de Janeiro

Heloisa Reis Leal

Operações Booleanas na Modelagem por Pontos

Dissertação de Mestrado

Dissertação apresentada como requisito parcial para obtenção do título de Mestre pelo Programa de Pós-Graduação em Informática da PUC-Rio.

Orientadores: Prof. Waldemar Celes Filho
Prof. Luiz Velho

Rio de Janeiro, agosto de 2004

Pontifícia Universidade Católica
do Rio de Janeiro

Heloisa Reis Leal

Operações Booleanas na Modelagem por Pontos

Dissertação apresentada como requisito parcial para obtenção do título de Mestre pelo Programa de Pós-Graduação em Informática da PUC-Rio. Aprovada pela Comissão Examinadora abaixo assinada.

Prof. Waldemar Celes Filho
Orientador
Departamento de Informática-PUC-Rio

Prof. Luiz Velho
Co-orientador
Instituto de Matemática Pura e Aplicada (IMPA)

Prof. Bruno Feijó
Departamento de Informática-PUC-Rio

Prof. Marcelo Dreux
Departamento de Mecânica-PUC-Rio

Prof. Paulo Cezar Pinto Carvalho
Instituto de Matemática Pura e Aplicada (IMPA)

Prof. José Eugênio Leal
Coordenador Setorial do Centro Técnico Científico - PUC-Rio

Rio de Janeiro, 3 de agosto de 2004

Heloisa Reis Leal

Graduou-se em Ciência da Computação pela Universidade Federal de Minas Gerais (UFMG) em abril de 1992. Trabalhou como Analista de Sistemas na empresa Telecomunicações de Minas Gerais S.A. (TELEMIG) entre 1993 e 1997, onde atuou no desenvolvimento de sistemas de controle da rede telefônica de Minas Gerais. Trabalhou como Analista de Sistemas no Tribunal de Justiça do Estado de Minas Gerais (TJMG) entre 1998 e 2001, onde atuou no desenvolvimento de sistemas controle de processos e controle de equipamentos. Desde 2002, trabalha no Serviço Federal de Processamento de Dados (SERPRO) desenvolvendo sistema de controle da Malha Fiscal do Imposto de Renda - Pessoa Física.

Ficha Catalográfica

Leal, Heloisa Reis

Operações booleanas na modelagem por pontos / Heloisa Reis Leal ; orientadores: Waldemar Celes Filho, Luiz Velho. – Rio de Janeiro : PUC, Departamento de Informática, 2004.

75 f. : il. ; 30 cm

Dissertação (mestrado) – Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática.

Inclui referências bibliográficas.

1. Informática – Teses. 2. Computação gráfica interativa. 3. Modelagem 3D. 4. Modelagem por pontos. 5. Operações booleanas. I. Celes Filho, Waldemar. II. Velho, Luiz. III. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

À minha mãe, Tetê, Dálton e Lucas.

Agradecimentos

Aos meus orientadores Waldemar Celes e Luiz Velho pelo apoio, dedicação e paciência que tornaram este trabalho possível.

À PUC-Rio pelo apoio financeiro.

Aos membros da banca examinadora pelas correções e sugestões.

À Cristina Griner, chefe de projeto do SERPRO, pelo apoio, compreensão e ajuda.

Aos meus professores de mestrado pelos conhecimentos transmitidos.

À Mauren Paola Ruthner pela amizade e colaboração durante todo o mestrado.

Aos colegas do Departamento de Informática da PUC-Rio pelos momentos compartilhados.

À minha mãe pela revisão do texto desta dissertação e pelo total apoio, sem o qual este trabalho não seria possível.

À Dálton e Tetê, meus queridos irmãos, pelo apoio, incentivo e colaboração.

Resumo

Leal, Heloisa Reis;Celes, Waldemar (Orientador). **Operações Booleanas na Modelagem por Pontos.** Rio de Janeiro, 2004. 75p. Dissertação de Mestrado - Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Operações booleanas em modelagem 3D são usadas para criar novos modelos ou para modificá-los. Na maioria dos tipos de representação de objetos 3D, estas operações são bastante complexas. Nos últimos anos tem sido muito explorado um novo tipo de modelagem, a modelagem por pontos, que apresenta muitas vantagens em relação às outras representações como maior simplicidade e eficiência. Dois trabalhos exploram as operações booleanas na modelagem por pontos, o trabalho de Adams e Dutré e o trabalho de Pauly et. al. Dada a grande importância deste novo tipo de modelagem e do uso de operações booleanas, esta dissertação apresenta uma introdução à modelagem por pontos, implementa o algoritmo proposto em Adams e Dutré com algumas melhorias e o compara com o método de Pauly et. al.

Palavras-chave

Modelagem 3D; Modelagem por pontos; Operações Booleanas; Computação Gráfica Interativa

Abstract

Leal, Heloisa Reis; Celes, Waldemar(Advisor). **Boolean Operations on Point-based models.** Rio de Janeiro, 2004. 75p. MSc Dissertation - Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Boolean operations are used to create or modify models. These operations in the majority of 3D object representations are very complex. In the last years a significant trend in computer graphics has been the shift towards point sampled 3D models due to their advantages over other representations, such as simplicity and efficiency. Two recent works present algorithms to perform interactive boolean operations on point-based models: the work by Adams and Dutré and the work by Pauly et. Al.. Due to great importance of this novel representation and of the use of boolean operations, the present work makes an introduction to point-based representation, implements the algorithm proposed by Adams and Dutré with some improvements, and compares this implementation with the work by Pauly et. al..

Palavras-chave

3D Modeling; Point-based Modeling; Boolean Operation; Interactive Computer Graphics

Sumário

Lista de figuras

Lista de tabelas

1
Introdução

O mundo digital está conquistando cada vez mais espaço nos meios de comunicação. Sons, imagens, vídeos digitais são mídias muito usadas em diversas áreas como entretenimento, medicina, desenho industrial, engenharia, arquitetura, educação, arte, ciência, entre outras. Podemos citar alguns exemplos dentro destas áreas como jogos, efeitos especiais, filmes digitais, visualização científica, vídeo-conferência, comércio na internet. Nos últimos anos verificamos que o modelo 3D vem se tornando um novo meio de comunicação digital. Com o desenvolvimento da tecnologia de aquisição e do processamento de modelos 3D, ele vem sendo cada vez mais utilizado. Jogos 3D, animações, aplicações em realidade virtual, visualização científica, aplicações para área médica e odontológica, criação de réplicas para estudo e preservação do patrimônio histórico, sistemas CAD/CAM, catálogos de modelos 3D na Web para comercialização, criação de protótipos para arquitetura, engenharia são alguns exemplos de aplicações que usam modelos 3D.

A área que estuda a representação de um modelo 3D no computador é chamada de modelagem em computação gráfica. Existem diversos tipos de modelagem para as diversas aplicações que usam um modelo 3D, como por exemplo, representações poligonais como malhas triangulares que são geralmente usadas na área de entretenimento. Este trabalho explora um novo tipo de modelagem, a modelagem por pontos.

A modelagem por pontos começou a ser largamente explorada nesta década e trata-se de uma área muito promissora em computação gráfica dadas às várias e grandes vantagens em relação aos outros tipos de representação como: esquema de representação CSG (*Constructive Solid Geometry*) [1], ou malhas poligonais. Dentre as principais vantagens podemos citar: grande simplicidade e flexibilidade, boa performance, robustez. Mas a principal vantagem desta modelagem, que foi o fator que impulsionou o seu grande desenvolvimento nesta década, é que esta modelagem por pontos é a melhor opção para tratar os dados de saída de equipamentos de aquisição de modelos 3D. O uso destes equipamentos tem aumentado muito nesta década. A

tecnologia destes equipamentos tem progredido muito e eles têm se tornado mais baratos. A tendência é que o uso destes equipamentos se torne o mais utilizado meio de obtenção de modelos 3D. Isto porque construir modelos 3D bastante detalhados por meio de um software de modelagem é muito trabalhoso. O processo de aquisição de modelos por meio destes equipamentos automatiza este processo de construção.

No trabalho com modelos 3D, freqüentemente são usadas as operações booleanas, que são operações de união, interseção ou diferença. No esquema de representação CSG, estas operações fazem parte da representação do objeto. Estas operações também podem ser usadas para criar novos modelos ou para alterá-los. Na maioria dos tipos de representação de objetos 3D, estas operações são bastante complexas.

A simplicidade da representação por pontos permite que as operações booleanas sejam realizadas com facilidade se compararmos com a realização destas operações em outros tipos de representação como malhas poligonais. Motivada pelas vantagens desta modelagem e pela utilidade destas operações, esta dissertação apresenta uma introdução geral de modelagem por pontos e trabalha, mais especificamente, com operações booleanas em objetos modelados por pontos. Dois trabalhos que exploram as operações booleanas na modelagem por pontos serão abordados: o trabalho de Adams e Dutré [2] e o trabalho de Pauly et. al [3]. O presente trabalho implementa o método de Adams e Dutré com algumas melhorias e faz uma comparação entre o método de Pauly [3] e o método de Adams e Dutré [2] com nossas melhorias.

Como contribuições dadas por este trabalho, apresentamos:

- um estudo sobre modelagem por pontos.
- um estudo sobre operações booleanas com pontos.
- proposta de melhorias ao algoritmo proposto por Adams e Dutré [2].
- comparação dos dois métodos propostos para realizar estas operações: método de Pauly et. al. [3] e de Adams e Dutré [2].

Este trabalho vem apresentado nesta dissertação organizada em sete capítulos, conforme descrito, resumidamente, a seguir.

O capítulo 2 cita alguns trabalhos sobre modelagem por pontos que estão relacionados a este trabalho.

O capítulo 3 apresenta uma visão geral do modelo por pontos e explica melhor alguns trabalhos citados no capítulo 2.

O capítulo 4 apresenta operações booleanas com pontos e os dois métodos para realizá-las na modelagem por pontos: método de Pauly et. al. [3] e de Adams e Dutré [2].

O capítulo 5 apresenta algumas melhorias ao algoritmo de Adams e Dutré [2] e a nossa implementação deste algoritmo.

O capítulo 6 traz os resultados obtidos.

E, finalmente, o capítulo 7 encerra o trabalho com uma conclusão e com possíveis trabalhos futuros.

2
Trabalhos relacionados

O primeiro pesquisador a propor o uso de pontos como primitiva gráfica foi Levoy em [4]. Ele propõe o uso de pontos como primitiva de renderização e discute a conversão dos outros modelos para o modelo por pontos.

Nos últimos anos, vários trabalhos importantes têm sido realizados na área em diversos segmentos: em visualização podemos destacar os trabalhos de Pfister [5] e de Zwicker [6] que usam pontos como primitiva de renderização e com resultados práticos interativos muito bons.

Em [7] Pauly et. al. propõem algoritmos para modelagem multi-resolução em modelos por pontos.

Os trabalhos de Curless [8] e Rusinkiewicz [9] discutem os métodos de aquisição, que consistem em obter uma amostragem de pontos da superfície do objeto real por meio de equipamentos como, por exemplo, o scanner 3D.

Nas operações booleanas, a interseção das superfícies dos modelos iniciais será parte do modelo resultado e para ser representada corretamente, precisa ser reconstruída. A reconstrução da superfície também é usada na visualização de um modelo. Um trabalho significativo na área de reconstrução de superfícies está em [10] de N. Amenta, et.al. Este trabalho apresenta uma solução prática com garantias teóricas para este problema e uma heurística para o tratamento de "arestas vivas", a questão mais difícil desta área. Um exemplo de uma "aresta viva" é o encontro de dois planos não coincidentes. Freqüentemente, a interseção citada acima (interseção entre os modelos iniciais) gera uma "aresta viva" (seção 4.2.2.3).

O algoritmo aqui implementado é um *plugin* para o software PointShop 3D [11] de Zwicker, et. al., software de edição de modelos com pontos. O PointShop tem seu código aberto, documentação e trabalha com *plugins*. Ele permite a visualização e edição interativa do modelo por pontos. É o único software livre disponível com estas características. Esta implementação na forma de um *plugin* facilitou muito o trabalho, pois não foi necessário implementar a visualização do modelo 3D. Além disso, o Pointshop contém também um *plugin* com a implementação do método de Pauly, et. al. [3] discutido nesta dissertação. A

execução dos dois *plugins* que realizam operações booleanas pôde ser comparada.

Os trabalhos aqui discutidos são o de Bart Adams e Philip Dutré [2] e de Pauly, et. al.[3] que apresentam métodos diferentes para realização de operações booleanas com pontos: Adams e Dutré utilizam *octrees* [1], e Pauly Kd-Tree [16] e MLS (*Moving Least Squares*) [18]. Estes algoritmos são discutidos no capítulo 4.

O trabalho de Sara F. Frisken [12] apresenta um algoritmo para realizar operações em *octree* de forma mais eficiente. Como estas operações estão presentes em todo o nosso algoritmo, este trabalho certamente oferece uma forma de otimização, mas que não foi implementada.

3
Modelagem por Pontos

A computação gráfica não lida diretamente com o objeto ou com o fenômeno físico, lida com modelos gráficos ou geométricos. Modelos gráficos são estruturas computacionais (abstratas) que capturam aspectos de objetos de interesse para uma aplicação, como forma, cor, textura, etc. Se apenas a forma ou a geometria do objeto é relevante, temos modelos geométricos.

Estes modelos são usados em várias aplicações em computação gráfica como robótica, visão computacional, simulações realistas, análises estruturais, prototipagem, projeto e manufatura de produtos eletromecânicos, arquitetura.

Existem vários tipos de modelos gráficos. Esta dissertação trata de um tipo muito em voga atualmente que é o modelo por pontos.

Neste capítulo daremos uma visão geral do modelo por pontos. Introduziremos este modelo e seu processo de criação. Será apresentado o que é um modelo por pontos e o conceito de *surfel* [5]*,* introduziremos os tipos de distribuição dos *surfels,* uma estruturação aplicada a estes *surfels,* e dados que eles podem conter. As quatro próximas subseções (Idéia da representação, Tipos de distribuição dos *surfels,* Estrutura de multi-resolução e Dados que um *surfel* pode conter) tratam destes tópicos. Um modelo por pontos pode ser criado a partir da conversão de outros tipos de modelagem, NURBS por exemplo, para o modelo por pontos ou a partir de um processo de aquisição. Esta última forma de criação pode ser dividida em duas etapas: aquisição do modelo, reconstrução da superfície. As subseções 3.5 e 3.6 tratam destas duas etapas respectivamente. A edição do modelo 3D é tratada brevemente na subseção 3.8 e a sua visualização na subseção 3.7. A apresentação do processo de aquisição (seção 3.5) é baseada no tutorial de W. T. Corrêa [13]. A etapa de edição 3D consiste em operações para mudança da cor, textura ou forma do modelo. Para esta etapa enfocaremos as operações booleanas que modificam a forma do modelo ou criam novos modelos.

3.1.
Idéia da Representação

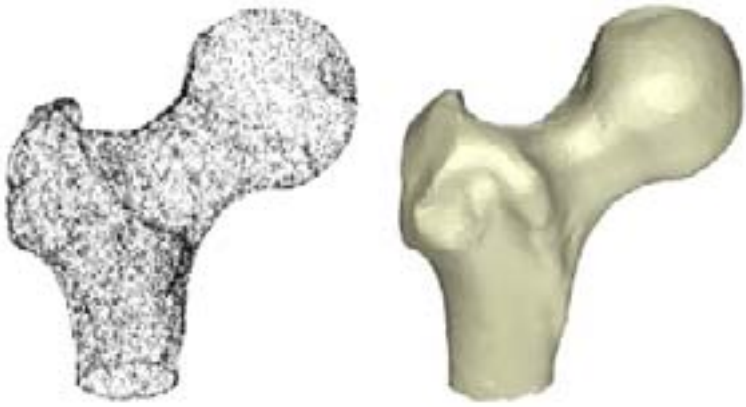


Figura 1: [20] -Um modelo por pontos e sua visualização.

Um modelo por pontos representa um objeto através de seu contorno. É um tipo de representação B-Rep (*Boundary Representation*) [1]. Este contorno consiste em uma superfície 2D posicionada no espaço 3D. Na modelagem por pontos, ela é representada por um conjunto de pontos 3D que são uma amostragem dos (infinitos) pontos que formam a superfície contínua (Figura 1). A amostragem pode ser obtida de diversas maneiras. Uma delas, o processo de aquisição, progrediu muito nos últimos anos e este foi um dos fatores que tornou muito importante o desenvolvimento deste modelo por pontos. Outra forma de gerar amostragem é fazer uma conversão de um outro modelo, NURBS por exemplo, para o modelo por pontos.

Estes pontos são denominados *surfels* e podem conter, além da posição 3D, outras informações como normal à superfície, cor, textura, raio de alcance. A Figura 2 mostra um conjunto de *surfels* representando uma superfície. No *surfel* em destaque podemos ver o raio de alcance r_t e a normal *n* à superfície.

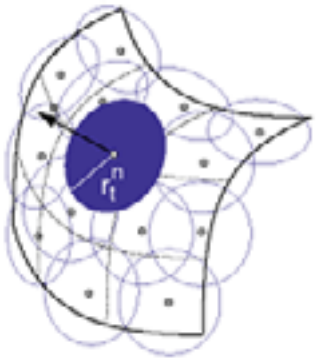


Figura 2 [5]: A superfície modelada por surfels. O surfel em destaque tem raio de alcance r_i e normal *n*.

Como o modelo não armazena qualquer informação sobre a conexão entre os *surfels*, a reconstrução da superfície contínua a partir dele pode exigir algumas restrições na amostragem (seção 3.6).

3.2. Tipos de distribuição dos *surfels*

Existem dois tipos de distribuição dos *surfels* na modelagem por pontos: uniforme e não uniforme. Na distribuição uniforme cada um deles descreve partes de mesmo tamanho da superfície. Na não uniforme cada *surfel* armazena a informação de seu raio de alcance que é o raio do disco que este *surfel* representa.

A distribuição uniforme tem a vantagem de ser mais simples e não ser necessário armazenar o raio de alcance do disco, mas ela apresenta a seguinte desvantagem: se o modelo possui regiões com muito detalhe e regiões com pouco detalhe, ele terá que manter uma alta densidade mesmo nas regiões de poucos detalhes.

3.3. Estrutura de multi-resolução

Um objeto pode ser representado em vários níveis de detalhes. Por exemplo, podemos representar um mesmo objeto com várias resoluções diferentes. Este conceito pode ser aplicado ao modelo por pontos. Desta forma, um modelo é representado por uma estrutura hierárquica (uma árvore, por exemplo), onde em cada nível temos uma representação do objeto em uma

determinada resolução. Em [7] Pauly et. al. propõem algoritmos para modelagem multi-resolução em modelos por pontos.

3.4. Dados que um *surfel* pode conter

Cada *surfel* armazena sua localização e pode também conter informações como a cor ou coordenadas de textura. Se a amostragem não for uniforme, o *surfel* contém a informação do raio de alcance que é o raio de um disco cujo centro é a posição do *surfel*. O *surfel* pode também conter o vetor normal à superfície e assim o surfel pode ser visto como um disco orientado. A superfície também pode ser vista como uma superfície orientada.

3.5. Aquisição do modelo

Esta introdução sobre aquisição do modelo aqui apresentada baseia-se no tutorial de W. T. Corrêa [13].

O processo de aquisição consiste em obter uma amostragem de pontos da superfície do objeto real por meio de equipamentos como, por exemplo, o scanner 3D. É uma área que tem progredido muito nos últimos anos. A resolução e a precisão dos equipamentos de aquisição têm melhorado bastante e o custo deles reduzido de forma significativa. Isso possibilita que a aquisição de modelos 3D seja usada em aplicações de desenho industrial, arte ou em produtos para a área de entretenimento. O conseqüente maior uso destes equipamentos criou a necessidade de avanços no tratamento dos seus dados de saída que são um conjunto de pontos.

Este processo pode ser dividido em três partes [13]: aquisição da geometria, aquisição da textura e correção de falhas da aquisição.

3.5.1. Aquisição da geometria

Existem vários métodos de aquisição da geometria. Curless [8] e Rusinkiewicz [9] classificaram estes métodos em três categorias: métodos por contato, transmissão, e reflexão.

O método por contato toca a superfície do objeto com uma sonda e grava a posição. Este método, do qual o *Coordinate Measuring Machines* (CMMs) é

um exemplo, é bastante preciso, mas é muito lento e não adequado a objetos frágeis.

Métodos por transmissão projetam ondas de energia sobre o objeto e gravam a energia transmitida. Por exemplo, tomografia computadorizada que projeta ondas de *raio-x* sobre o objeto e mede a quantidade de radiação que passa através dele. O resultado é um modelo geométrico de alta resolução. Este método pode captar detalhes do objeto que não são visíveis de fora do objeto. Por outro lado, ele é caro e arriscado.

Métodos por reflexão podem ser óticos ou não-óticos. Não-óticos incluem radares que medem a distância a um objeto enviando microondas ou ondas sonoras para o objeto e gravando o tempo que a onda gasta para retornar. Estes métodos não são muito precisos, nem rápidos.

Métodos por reflexão óticos podem ser passivos ou ativos. Métodos óticos passivos não interagem com o objeto e incluem técnicas de visão computacional como *shape-from-shading*, *stereo triangulation*, *optical flow* e *métodos de decomposição para video streams*. Estes métodos raramente constroem modelos precisos e sua maior aplicação é em reconhecimento de objetos.

Métodos óticos ativos projetam luz sobre um objeto de maneira estruturada e, através da reflexão da luz, determinam a forma do objeto. Exemplos são *depth from defocus*, *photometric stereo*, *projected-light triangulation*, *time of flight*. Métodos por reflexão óticos ativos têm muitas vantagens sobre os passivos: têm um melhor desempenho quando não existe textura, são mais baratos e mais robustos e produzem uma amostragem precisa.

3.5.2. Aquisição da textura

Idealmente, nós gostaríamos de ter uma completa descrição de como um ponto da superfície reflete a luz dependendo do vetor normal a esta superfície, da direção do raio de luz incidente e do comprimento de onda. Tal descrição é conhecida como *Bidirectional Reflectance Distribution Function* (BRDF). Medir BRDFs com precisão é um problema difícil.

Pode-se ter uma aproximação da aquisição de cores do objeto tirando fotos dele. O objeto é fotografado e a cor de cada ponto da superfície é obtida a partir de um mapeamento das fotos 2D para o modelo 3D. Para mapear a fotografia à geometria, precisamos conhecer os parâmetros de projeção da câmera e a sua posição no momento em que a foto foi batida. Para cada ponto

3D do modelo, encontramos sua projeção 2D no plano da câmera e associamos a cor do *pixel* nesta projeção à cor do ponto 3D.

3.5.3. Correção das falhas da aquisição

O processo de aquisição deixa freqüentemente buracos no modelo devido a ruídos no processo ou à impossibilidade de aquisição de determinadas partes de uma superfície. Estes buracos devem ser corrigidos. Pode-se usar, por exemplo, um filtro de passa-baixa para remoção de ruídos [14]. Métodos de difusão geométrica têm sido aplicados para preenchimento dos buracos [15]. A correção das falhas é tida como um pré-processamento para o processo de reconstrução da superfície contínua (seção 3.6).

3.6. Reconstrução da superfície (contínua) a partir do modelo (discreto)

A reconstrução da superfície contínua consiste em, a partir de um conjunto não estruturado de pontos, obter um modelo contínuo desta superfície. Esta reconstrução é freqüentemente exigida para um modelo por pontos. Na sua visualização, podemos desejar focar qualquer parte do modelo. Para fazermos operações booleanas com os modelos, podemos querer determinar exatamente onde eles se intersectam.

No entanto, não precisamos ter necessariamente a representação contínua de todo o objeto ao mesmo tempo. A reconstrução desta superfície a partir dos pontos amostrados pode ser feita no momento que for mais adequado à aplicação. Na visualização, a reconstrução pode ser feita na projeção 2D da parte do modelo em foco (seção 3.7). Para operações booleanas pode-se reconstruir apenas a área da interseção dos modelos.

Este processo de reconstrução de superfície tem sido muito estudado em modelagem geométrica e geometria computacional nos últimos anos. O trabalho mais importante desta área é o trabalho de Amenta et. al. [10] que foi a primeira solução com garantias demonstráveis para este problema de reconstrução de superfície. A seção 4.4 explica estes conceitos e o trabalho [10]. Este trabalho demonstra que, teoricamente, para que a reconstrução da superfície (contínua) a partir das amostras seja correta, a amostragem deve ser adequada. Ele usa o conceito de "Eixo Medial" para definir o que é uma amostragem adequada

(seção 4.4). Em duas dimensões, os vértices de *Voronoi* de uma amostragem densa aproximam o “Eixo Medial” da amostragem.

Por meio deste conceito de "Eixo Medial" [10,16] obtêm-se parâmetros teóricos para a definição de "amostragem adequada" e, baseado neste conceito, o algoritmo apresentado em [10] traz uma solução prática para o problema, obtendo uma malha poligonal a partir do modelo por pontos.

O *PointShop 3D* [11], software de edição de modelos por pontos utilizado nesta pesquisa, faz, quando necessário, a reconstrução da superfície. Para isso, o PointShop 3D explora a vizinhança local (*surfels* vizinhos de acordo com suas posições). Ele usa o conceito de "Eixo Medial" para justificar seu método de reconstrução da superfície que na prática apresenta resultados bastante satisfatórios, apesar de teoricamente não ter garantias [16].

3.7. Visualização do modelo

A primeira abordagem para visualização de um modelo por pontos foi construir uma malha de polígonos a partir dos *surfels* e renderizá-la. Mas a construção de malhas a partir dos *surfels* é um processo muito caro e lento, principalmente para modelos complexos. Na tentativa de obter uma visualização interativa do modelo por pontos, estão sendo estudadas técnicas que utilizam *surfels* como primitiva de visualização. A idéia foi apresentada inicialmente por Levoy [4].

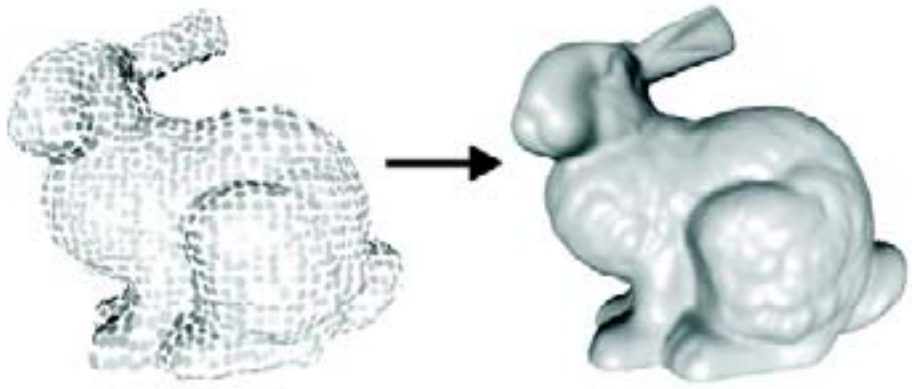


Figura 3 [21] -Modelo por pontos e sua renderização usando *Surface Splatting.*

O objetivo da renderização a partir dos *surfels* é projetá-los na tela e reconstruir as superfícies de forma contínua sem buracos. Para que esta renderização seja amplamente utilizada deve-se ter cuidado com o uso de memória e com a qualidade da imagem gerada. Técnicas recentes como *Surface Splatting* [6] faz a renderização interativamente, com qualidade (Figura 3) comparada a outros métodos de renderização como *Image Based Rendering*. Em geral, a visualização do modelo por pontos pode ser dividida nas seguintes etapas:

- *warping* - projeção em perspectiva de cada ponto do modelo na tela: processo análogo à projeção de vértices dos triângulos na malha triangular. Se um pixel da tela estiver contido em mais de uma projeção de *surfels* (vizinhos), ou seja, mais de um ponto cobre este pixel, os atributos destes pontos serão combinados;

- *shading* por ponto - o processo de *shading* é realizado para cada ponto visível (os pontos que não são vistos do ponto de vista em questão devem ser retirados). Qualquer modelo local de *shading* pode ser aplicado. Uma nova técnica foi proposta em *Surface Splatting* [6];

- reconstrução da imagem – pixels que não “receberam” projeção de nenhum ponto formam buracos na imagem. Esta imagem deve ser reconstruída sem buracos , sem *aliasing* e as superfícies devem ser reconstruídas de forma contínua e devem ser “suavizadas” (*smooth surface*);

Os algoritmos de renderização por pontos podem tratar a questão de buracos na visualização de três formas [17]:

Detecção e preenchimento de buracos no espaço da tela: pontos são projetados na tela e pixels que não receberam pontos são detectados (buracos). Para preencher estes buracos que não receberam pontos é feita uma interpolação entre os vizinhos;

Geração de mais pontos: a superfície é interpolada no espaço do modelo para garantir que todos os pixels recebam pelo menos a projeção de um ponto;

Splatting: um surfel é projetado na tela e sua contribuição é espalhada de forma diferenciada para os pixels vizinhos. À medida em que o vizinho se torna mais distante, a contribuição do *surfel* diminui.

Räsänen faz uma analogia do processo de renderização proposto em [6] com o mapeamento de textura [15]. No mapeamento de textura, para um determinado pixel da tela, são encontrados os pontos da textura que afetam este

pixel. Num sentido inverso, em *splatting*, para um determinado *surfel* (que representa a textura) são encontrados os pixels da tela que este *surfel* afeta.

3.8. Edição do modelo

Edição de um modelo 3D consiste em operações que modificam características deste modelo como cor, textura, forma ou criam novos modelos. O software PointShop 3D [11], software de edição de modelos por pontos 3D, implementa várias destas operações (ver seção 5.2).

Para modificar a forma de um modelo ou criar um novo, são muito usadas as operações booleanas, que é o nosso enfoque nesta dissertação. Na modelagem CSG (*Constructive Solid Geometry*), as operações booleanas são usadas, juntamente com primitivas sólidas, para definir um modelo. No entanto em alguns tipos de modelos, estas operações são difíceis, como, por exemplo, em malhas poligonais. Com pontos, elas podem ser feitas interativamente [3,2].

4
Operações Booleanas com Pontos

Operações booleanas são as operações de união, interseção e diferença definidas na teoria matemática de conjuntos. Estas operações são aplicadas em modelos sólidos para alterá-los ou para criar novos modelos. A Figura 4 abaixo mostra a criação de três novos modelos a partir de quatro modelos iniciais.

Figura 4 [22] : Três modelos novos foram criados a partir dos quatro iniciais (esfera, cilindro, dois cubos). O primeiro é a esfera menos um cubo, em seguida retirou-se outro cubo e por último retirou-se o cilindro.

Estas operações são aplicadas em objetos modelados por pontos em dois trabalhos: no trabalho de Bart Adams e Philip Dutré [2] e no trabalho de Pauly, et. Al. [3]. Neste capítulo, a seção 4.1 apresenta a parte comum aos dois trabalhos e as seções 4.2 e 4.3 apresentam as particularidades de cada um. A seção 4.4 discute uma questão não explicitada em nenhum dos dois trabalhos.

4.1.
Métodos para realizar operações booleanas em objetos modelados por pontos

A Figura 5 mostra o resultado do método de Adams[2]: operações de união, diferença e interseção de objetos sólidos 3D modelados por pontos.

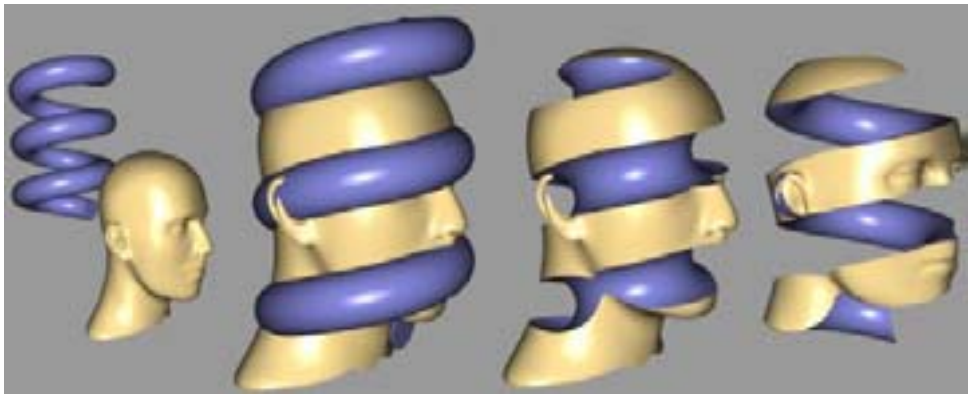


Figura 5 [2]-A partir de dois modelos por pontos, uma hélice e uma cabeça, as três operações booleanas são mostradas: da esquerda para a direita: união, diferença e interseção.

Sejam A e B, objetos 3D sólidos modelados por pontos. Definem-se os seguintes elementos:

A_f – superfície (contorno) de A que está fora de B;
A_d – superfície (contorno) de A que está dentro de B;
B_f – superfície (contorno) de B que está fora de A;
B_d – superfície (contorno) de B que está dentro de A;

O resultado da operação booleana entre A e B, será um modelo C cuja superfície de contorno pode ser determinada pela seguinte regra [2], que aqui chamamos de Regra 1:

Regra 1:
A U B: $A_f + B_f$;
A ∩ B: $A_d + B_d$;
A – B: $A_f + B_d$ (com a orientação invertida);
B – A: A_d (com a orientação invertida) + B_f;

A Figura 6 mostra as operações com dois modelos *A* e *B*.

(a) (b)

(c) (d)

Figura 6 [22]: (a) Dois modelos A e B; (b) Posicionados de forma que A tem uma parte dentro de B, é mostrado o resultado A U B: $A_f + B_f$; (c) Resultado de $A - B$: $A_f + B_d$ (d) Resultado de $A \cap B$: $A_d + B_d$.

Bart [2] e Pauly [3] apresentam dois métodos para determinar os elementos acima. Estes dois métodos são construídos a partir dos seguintes conceitos: os elementos A_f ,A_d ,B_f ,B_d são uma parte da superfície modelada por pontos e, portanto, são conjuntos de *surfels*. Para determiná-los, é necessária, então, a classificação de um *surfel* (que pode se resumir à classificação de um ponto que é o seu centro) de uma superfície como "dentro" ou "fora" de um outro modelo. Os dois métodos utilizam a seguinte regra para esta classificação, aqui denominada Regra 2:

Regra 2:

Dado um ponto *x* qualquer, um volume *V* determinado por uma superfície de contorno *S*, uma amostragem *P* da superfície *S,* um ponto $p \in P$ que é a amostra mais próxima a *x*, um vetor n_p normal à *S* em *p* e orientado para fora de *S,* temos que:

$(x - p) \cdot n_p > 0 \Leftrightarrow x \notin V$.

Ou seja, para classificar um ponto em relação a um modelo, verificamos o produto interno entre a normal em *p* (*surfel* mais próximo a *x*) e o vetor $(x - p)$: se este produto interno for maior que zero, *x* não pertence ao volume *V* e será classificado como "fora"; senão será classificado como "dentro". De maneira ainda mais informal: se a normal em *p* apontar para a direção oposta a *x*, *x* será classificado como "dentro", se apontar para *x, x* será classificado como "fora".

A Figura 7 mostra a Regra 2.

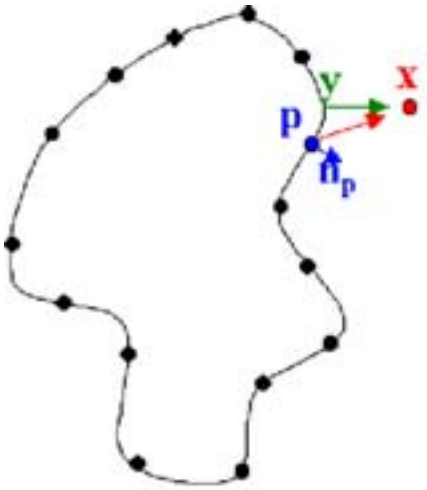


Figura 7: Para classificar o ponto *x* , encontra-se o surfel *p* (com normal n_p) mais próximo a *x*. Se o produto interno entre n_p e *(x – p)* for maior que zero, então *x* é classificado como "fora"; senão será classificado como "dentro".

A validade da Regra 2 é discutida na Seção 4.4. Uma das restrições teóricas para garantir-lhe a validade é que a superfície seja "suave".

Cada um dos dois métodos dá um tratamento diferente para o uso desta regra. Pauly usa a regra diretamente, encontrando o surfel mais próximo a um ponto e classificando este ponto a partir deste *surfel*. Ele utiliza também a representação implícita MLS (Moving Least Squares [18]) para casos em que a Regra 2 poderia falhar e para tratamento de interseções [3]. Seu método vale para modelos sólidos ou não sólidos. O método de Adams e Dutré [2] utiliza *octrees* [1], criando uma classificação dos nós da *octree* a partir desta Regra 2. Os pontos são então classificados de acordo com o nó da *octree* que o contém. Este segundo método aplica-se somente a modelos sólidos. Os dois métodos serão descritos em detalhes nas seções abaixo.

Como um *surfel* é considerado um disco, podemos dizer que *surfels* de dois modelos diferentes podem se intersectar. A interseção destes dois *surfels* corresponde à interseção destes dois discos. Estes *surfels* geram problemas para visualização: se forem acrescentados ao resultado sem um tratamento específico, apresentarão artefatos não desejáveis na renderização do modelo resultado (Figura 8); se não forem acrescentados ao resultado, deixarão buracos no modelo.

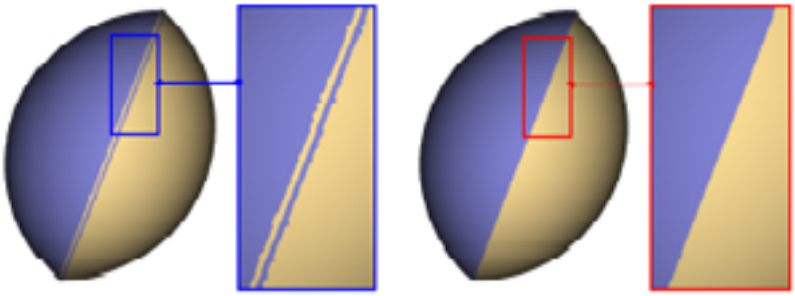


Figura 8 [2] -Interseção de duas esferas: à esquerda: sem nenhum tratamento de interseção; à direita: após o tratamento de interseção.

Além disso, nem sempre esta interseção forma uma superfície "suave". Esta interseção pode formar uma "aresta viva" (seção 4.2.2.3), gerando problemas também para o uso da Regra 2 que é válida apenas para superfícies suaves. É necessário, então, fazer um tratamento desta superfície de interseção (formada por *surfels* originados de diferentes modelos). O tratamento de cada um dos métodos para este caso será também abaixo apresentado.

4.2. Método de Adams e Dutré

O método aplica-se somente a modelos sólidos. Para determinação dos elementos da aqui chamada Regra 1 (seção 4.1), este método utiliza uma *octree* [1] que subdivide o espaço do modelo em três: fora, dentro ou no contorno do modelo. Estes espaços são determinados pela classificação dos nós da *octree* a partir da Regra 2. Também é usado o método TINN - *Triangle Inequality Nearest Neighbour* [19] - para otimizar a pesquisa dentro do nó. A seguir descrevemos o modelo usado neste método, apresentando os dados associados aos pontos, o tipo de distribuição e o algoritmo que realiza as operações.

4.2.1. Modelo por Pontos em Adams e Dutré

Os *surfels*, neste método, contêm as seguintes informações: posição do *surfel* x_s, orientação do *surfel* n_s e raio de alcance do *surfel* r_s. O *surfel* pode ser pensado como um disco ortogonal a n_s, com centro x_s e raio r_s. O raio é escolhido de tal forma que a projeção dos discos no plano da imagem se sobreponham.

Os objetos representados são sólidos. A distribuição dos *surfels* é uniforme, embora o algoritmo não exija esta condição. Para cada sólido é definido um r_{max} = Max r_s, o raio de alcance do maior *surfel* pertencente à superfície do sólido. Como o raio é escolhido de tal forma que a projeção dos discos no plano da imagem se sobreponham, o controle da amostragem é, então, ligado à renderização.

4.2.2. Algoritmo de Operações Booleanas

Utilizando objetos modelados da forma descrita acima, o método realiza operações de união, interseção e diferença com modelos sólidos. Supondo dois modelos sólidos S1 e S2. O algoritmo pode ser dividido nas seguintes etapas: Criação de uma *octree* para cada modelo, Teste Dentro-Fora, Tratamento da interseção. Estas etapas são apresentadas nas próximas seções.

4.2.2.1. Criação de uma *octree* para cada modelo

É criada uma *octree* [1] para cada modelo. A criação de uma *octree* é feita da seguinte forma:

Gera-se a "Caixa envolvente", caixa alinhada com os eixos, que contém todo o modelo. A "Caixa envolvente" corresponde ao nó raiz da *octree*. Este nó contém, inicialmente, todos os *surfels* do modelo. Em seguida, o nó raiz é dividido em oito nós filhos. Os *surfels* são distribuídos entre estes oito nós. Cada um deles, caso contenha *surfels*, é dividido novamente em 8 filhos e seus *surfels* são redistribuídos entre eles. Este processo é repetido recursivamente, enquanto o número de *surfels* do nó for maior que zero ou até uma profundidade máxima pré-definida da *octree* ser alcançada.

Cada nó da *octree* deve ser classificado. O objetivo desta classificação é determinar que parte da *octree* está dentro do modelo e que parte está fora. Os nós da *octree* são, então, classificados como "dentro", "fora" e "contorno". Podemos ilustrar esta classificação em 2D: a Figura 9 mostra uma *quadtree* classificada.

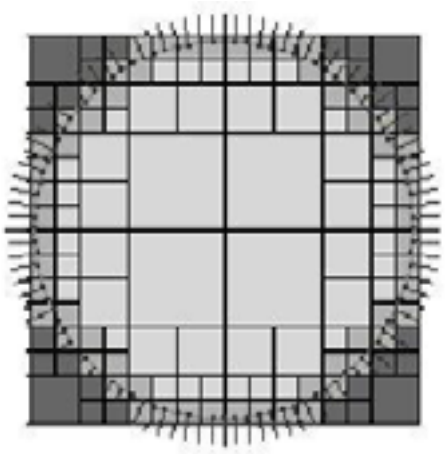


Figura 9-Um objeto representado por pontos com suas normais e sua quadtree correspondente classificada: os nós cinza claro estão dentro do modelo; nós cinza mais escuro estão fora do modelo; e nós com cinza intermediário são nós cheios, nós que contém pontos.

O nós que contém *surfels* são classificados como “contorno”. Para classificar um nó vazio, existem três casos:

. o nó vazio tem apenas um vizinho-irmão não vazio;

. o nó vazio tem mais de um vizinho-irmão não vazio;

. o nó vazio não tem vizinhos-irmãos não vazios;

No primeiro caso, o nó vazio é classificado pelo seu vizinho não vazio. No segundo caso, considera-se qualquer um dos vizinhos não vazios. No terceiro caso, classifica-se primeiro os nós vizinhos e é dada a mesma classificação para o nó vazio. Como cada nó não folha de uma *octree* tem pelo menos um nó filho não vazio, sempre pode ser feita a classificação dos nós vazios. A classificação é feita da seguinte forma: dentre os *surfels* do vizinho-irmão escolhido, encontra-se o *surfel* mais próximo ao nó a ser classificado. Se a normal deste *surfel* aponta para o nó vazio a ser classificado, então ele é classificado como “fora”. Se a normal aponta no sentido contrário, então ele é classificado como “dentro”. De maneira mais formal, sejam c_v e c_c coordenadas dos centros do nó vazio e do nó contorno, respectivamente. Seja s o *surfel*, mais próximo ao nó vazio e n_s a normal em s. Se $(c_v - c_c) \cdot n_s > 0$ (produto interno entre a normal em *s e* o vetor $c_v - c_c$ for maior que zero) , o nó vazio será classificado como “fora”. Senão, será classificado como “dentro”.

Em geral, os nós “contorno” são subdivididos em três partes: uma que está fora do modelo, uma que está dentro do modelo e uma terceira que contém os *surfels*. Esta subdivisão é feita por meio da criação de dois planos paralelos que

dividem o nó. Para determinar estes planos, calcula-se o vetor, n_m, igual à média das normais dos *surfels* do nó: $n_m = \sum n_s / \| \sum n_s \|$, onde n_s é a normal de cada *surfel* do nó.

Em seguida, encontra-se o *surfel* com maior produto interno entre a sua normal e a normal média: $d_1=Max(n_m \cdot n_s)$. Este *surfel* juntamente com a normal média define um dos planos. Encontra-se também o *surfel* com menor produto interno entre a sua normal e a normal média: $d_2=Min(n_m \cdot n_s)$. Este *surfel* juntamente com a normal média define o outro plano. A Figura 10 mostra a divisão de um nó de uma quadtree nas três partes: dentro do modelo, fora do modelo e no "contorno".

Esta subdivisão aplica-se apenas a nós que contém *surfels* cujas direções não variem em mais de 90 graus. Caso isso ocorra, ele não será subdividido. Ou seja, se um nó possuir dois vetores normais (de dois *surfels* diferentes) que formem um ângulo maior que 90 graus, ele não será subdividido.

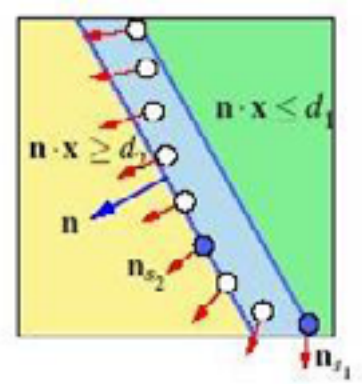


Figura 10 [2] -Divisão de nó de contorno: a parte à esquerda está fora do modelo, a parte à direita está dentro do modelo e a faixa intermediária contém os surfels.

4.2.2.2. Teste Dentro-Fora

O Teste Dentro-Fora faz a classificação dos *surfels* de um modelo como "dentro" ou "fora" de outro modelo. Portanto aplicaremos este mesmo teste duas vezes: uma para o modelo S1 em relação ao modelo S2 e outra para S2 em relação a S1. Vamos mostrar o teste Dentro-Fora para o caso da classificação dos *surfels* do modelo S1 em relação a S2.

No item anterior foram criadas duas *octrees*, uma para o modelo S1 e outra para o modelo S2. Chamemos de A e B as *octrees* de S1 e S2 respectivamente. Para classificar S1 em relação a S2 inicialmente é feita uma avaliação da posição de A em relação a B. A e B podem estar nas seguintes situações:

- A não intersecta B;
- A intersecta apenas nós “fora” de B;
- A intersecta apenas nós “dentro” de B;
- A intersecta nós “fora” e nós “dentro” de B;

Para testar estas situações, os nós são aumentados de r_{max}, o raio do maior *surfel* do modelo sólido. Isto é feito para compensar o fato de que o *surfel* não é apenas um ponto, mas sim considerado como um disco.

No primeiro e no segundo caso, todos os *surfels* de A são classificados como “fora”. No terceiro caso, todos os *surfels* de A são classificados como “dentro”.

No quarto caso, temos que percorrer a *octree* A. Começando pelo nó raiz, percorre-se os filhos deste nó. Para cada filho, os quatro testes acima são feitos. Se o nó filho encontra-se em um dos três primeiros casos, a classificação é dada como foi citada acima. Se um filho está no quarto caso e não é um nó folha, repete-se recursivamente o mesmo procedimento para seus filhos até encontrar-se um nó folha. Para um nó que está no quarto caso e é folha, percorre-se a lista de *surfels*, que este nó contém, classificando cada um destes *surfels*. Seja s_a um *surfel* desta lista. Para este caso percorre-se a *octree* B para classificar s_a. Primeiramente, encontramos o nó de B que contém s_a. Se este nó de B foi dividido, testa-se de acordo com esta divisão do nó: se o surfel s_a está na parte do nó que está dentro do modelo, ele é classificado como “dentro”; se s_a está na parte que está fora do modelo, ele é classificado como “fora”; se s_a está na parte intermediária ou se o nó de B não foi subdividido em planos, encontra-se s_b que é o *surfel* , pertencente a este nó de B, que está mais próximo de s_a. s_a será, então, classificado de acordo com s_b. Se $(s_a - s_b) \cdot n_b > 0$, s_a é classificado como “fora”, senão é classificado como “dentro”. Se os discos de s_b e s_a se intersectam, s_a é marcado como "interseção". Para encontrar s_b , o algoritmo implementa o método TINN – Triangle Inequality Nearest Neighbour [19].

4.2.2.3.
Tratamento da interseção

Os *surfels* de A que intersectam *surfels* de B são classificados como "interseção". Se eles não forem incluídos no modelo resultado, o sólido resultado terá buracos. Se eles forem incluídos sem algum tratamento, eles não representarão adequadamente a região. Em geral, o *surfel* s_a (um *surfel* "interseção" de A) é dividido em dois pelo plano que contém o *surfel* s_b (*surfel* de B que s_a intersecta): uma parte que está fora de B e outra que está dentro de B. Dependendo da operação que está sendo realizada, vamos querer manter uma destas partes do *surfel* s_a no modelo resultado. Seja x_a a posição do *surfel* s_a, e o *surfel* s_a, uma circunferência de centro x_a e raio r_a igual ao raio de alcance de s_a. O plano que contém o surfel s_b define uma corda desta circunferência, como mostra a Figura 11. O *surfel* s_a será substituído por um *surfel* menor que ultrapassa esta corda de uma distância *e*, definida pelo usuário. Para cobrir completamente a interseção serão adicionados *surfels* menores que cruzam a corda de interseção.

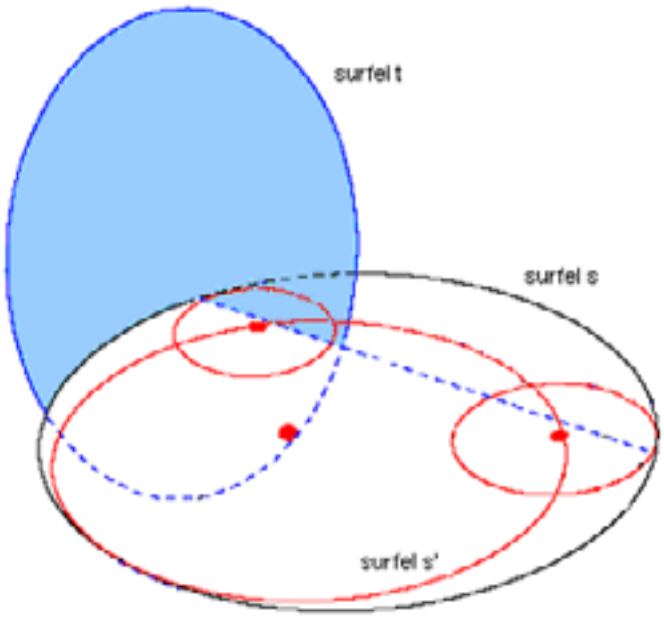


Figura 11 -Interseção do surfel s e do plano que passa pelo surfel t, o surfel mais próximo a s. O surfel s é substituído por três surfels menores.

O tratamento de interseção proposto por Adams e Dutré [2] não só evita os buracos, mas também cria uma forma de renderização de "arestas vivas". A colocação de vários pequenos *surfels* na região da interseção permite esta correta renderização das “arestas vivas”. Desta forma o problema das "arestas vivas" é resolvido na modelagem (Figura 12). A vantagem de se resolver o caso na modelagem é que não é necessário fazer um tratamento especial na renderização para que mostre esta "aresta viva". Diferente do tratamento feito em Pauly (seção 4.3.1) que resolve esta questão em tempo de renderização. Além disso, o tratamento resolve alguns casos onde a Regra 2 poderia falhar (veja em Validade da Regra 2-seção 4.4).

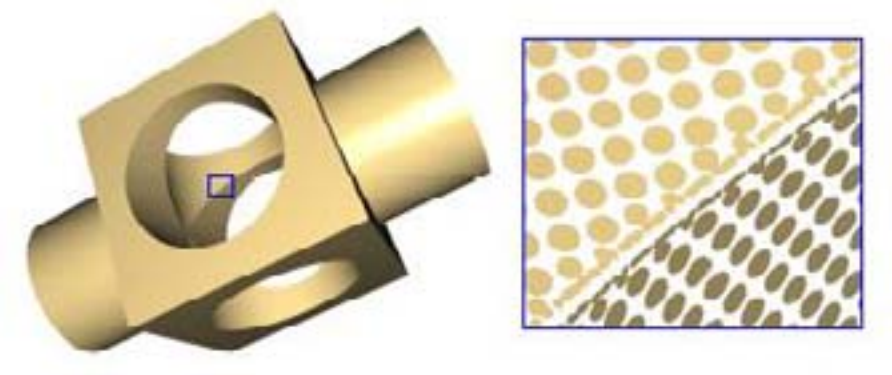


Figura 12 [2] -À direita é mostrado o resultado do tratamento de interseção para o caso de *Arestas Vivas* do objeto à esquerda.

4.3. Método de Pauly

O método aplica-se a objetos sólidos e não sólidos. Os modelos podem ser obtidos a partir de diferentes entradas como nuvem de pontos irregulares ou por meio de scanners 3D. Cria um modelo híbrido destes objetos: usa o modelo por pontos juntamente com a representação implícita Moving Least Squares - MLS [18]. Realiza operações booleanas com pontos e utiliza adicionalmente MLS para alguns casos de classificação de *surfels* e no tratamento de interseção. A seguir apresentamos as características do modelo por pontos deste método e o algoritmo que realiza as operações.

4.3.1.
Modelo em Pauly

O modelo é uma estrutura híbrida de pontos com a representação implícita *Moving Least Squares* - MLS [18]. Esta estrutura é usada para fazer operações booleanas, e também operações de *free-form* (não abordadas nesta dissertação). Para operações booleanas, a representação MLS permite que a curva de interseção entre os modelos seja reconstruída e *surfels* sejam colocados nesta curva e em alguns casos de classificação de *surfels* mostrados na seção 4.3.2. Para este modelo, não existem restrições quanto à forma do objeto.

Os *surfels* contêm informações sobre sua posição e sua normal. Além disso, *surfels* podem conter informações adicionais como textura, cor, propriedades de materiais.

A distribuição dos *surfels* é não-uniforme e a amostragem é aproximadamente "adequada". (Veja seção 4.4).

4.3.2.
Algoritmo de Operações Booleanas

O algoritmo classifica *surfels* de um modelo como “dentro” ou “fora” de outro modelo. Antes de colocarmos como é feita esta classificação, devemos acrescentar que Pauly estende esta classificação para modelos que não são sólidos. Ou seja, mesmo um modelo não sólido divide o espaço da cena (espaço em que se encontra o modelo) em dois sub-espaços: dentro ou fora do modelo. Pauly faz esta subdivisão da seguinte forma:

Como a superfície que determina o modelo é orientada (contém vetores normais à superfície que apontam para “fora” do modelo), Pauly considera que um modelo não sólido divide o espaço em dois sub-espaços: um “fora” do modelo (os vetores normais apontam para este sub-espaço) e um que está “dentro” do modelo (os vetores normais apontam para direção oposta a este sub-espaço). Por exemplo, um plano é representado como um conjunto de *surfels* contidos num determinado espaço geométrico plano *π.* Cada *surfel* contém um vetor normal direcionado. Este modelo divide, então, o espaço em dois sub-espaços: o sub-espaço que é apontado por estes *surfels* (considerado “fora” do modelo) e o sub-espaço que está na direção oposta (considerado “dentro” do modelo). Desta forma ele aplica as classificações de “dentro” e “fora” de um modelo também a modelos não sólidos. A partir desta consideração, as

operações booleanas são realizadas com os modelos não sólidos da mesma forma que com os modelos sólidos.

Para determinar esta classificação (dentro ou fora do modelo), Pauly utiliza a Regra 2. Para pesquisa do *surfel* mais próximo, o método usa a estrutura *Kd-tree* [16] e também o algoritmo ANN [16]. No entanto, o método apresenta uma restrição da Regra 2: ela só deve ser usada para classificações de pontos x , se a distância $\|x-p\|$ for maior que o espaçamento da amostragem local η_p em p. Este espaçamento da amostragem local η_p em p é um cálculo aproximado da distância média entre os surfels vizinhos a p. Se x está muito próximo à superfície, a classificação pode falhar. Neste caso, ele calcula o ponto exato de S mais próximo a x usando a projeção MLS ("*Moving Least Squares*").

Por questões de melhoria de desempenho, utiliza-se o fato de que sendo $\Omega p\ (x)$ a classificação do ponto x em relação à amostragem P, tem-se que $\Omega p\ (x) = \Omega p\ (x')$ para todos os pontos de x' que estão dentro da esfera de centro x e raio igual a $\|x-p\| - \eta_p$ (onde p é o surfel mais próximo e η_p é o espaçamento local da amostragem em p acima citado) . Ou seja, sendo x um ponto qualquer, p o surfel mais próximo a x, η_p é o espaçamento local da amostragem em p, se desenharmos uma esfera com centro x e raio igual a $\|x-p\| - \eta_p$, temos que todos os pontos x' que estão dentro da esfera terão a mesma classificação de x. Desta forma o número de vezes que a classificação é feita (pesquisando-se o surfel mais próximo ao ponto e usando a Regra 2) diminui em até 90%. Assim, em geral a classificação de um *surfel* é obtida a partir do seu *surfel* vizinho.

Dados:

S_1 a superfície MLS de contorno que define o modelo A;

S_2 a superfície MLS de contorno que define o modelo B;

P_1 o conjunto de *surfels* que representa a superfície S_1;

P_2 o conjunto de *surfels* que representa a superfície S_2;

$\Omega p_1\ (x)$ a classificação do ponto p em relação a uma amostragem P_1,;

Q_1 o subconjunto de P_1 que vai compor o resultado;

$\Omega p_2\ (x)$ a classificação do ponto p em relação a uma amostragem P_2,;

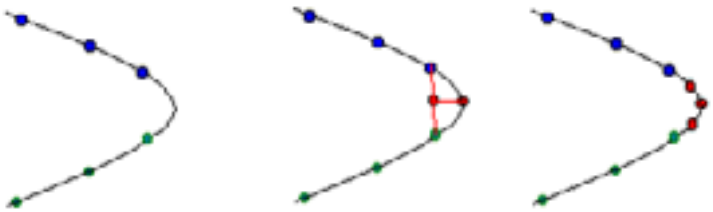
Q_2 o subconjunto de P_2 que vai compor o resultado;

Os conjuntos Q_1 e Q_2 são obtidos por meio da Tabela 1 mostrada abaixo.

	Q_1	Q_2
$S_1 \cup S_2$	$\{\mathbf{p} \in P_1 \vdots \Omega_{P_2}(\mathbf{p}) = 0\}$	$\{\mathbf{p} \in P_2 \vdots \Omega_{P_1}(\mathbf{p}) = 0\}$
$S_1 \cap S_2$	$\{\mathbf{p} \in P_1 \vdots \Omega_{P_2}(\mathbf{p}) = 1\}$	$\{\mathbf{p} \in P_2 \vdots \Omega_{P_1}(\mathbf{p}) = 1\}$
$S_1 - S_2$	$\{\mathbf{p} \subset P_1 \vdots \Omega_{P_2}(\mathbf{p}) = 0\}$	$\{\mathbf{p} \subset P_2 \vdots \Omega_{P_1}(\mathbf{p}) = 1\}$
$S_2 - S_1$	$\{\mathbf{p} \in P_1 \vdots \Omega_{P_2}(\mathbf{p}) = 1\}$	$\{\mathbf{p} \in P_2 \vdots \Omega_{P_1}(\mathbf{p}) = 0\}$

Tabela 1: Classificação para operações booleanas

Para o tratamento de interseção, neste método, são computados pontos que caem exatamente na curva de interseção e estes são acrescentados ao modelo resultado. Estes pontos são determinados de forma que a densidade da amostragem vai crescendo à medida que os surfels se aproximam da curva de interseção e convergem para este ponto de interseção, como mostra a Figura 13.



(a) (b) (c)

Figura 13 - Os novos surfels são colocados exatamente na curva de interseção e convergem para o ponto de interseção.

A Figura 14 mostra o processo de encontrar os *surfels* que caem nesta curva de interseção. Inicialmente, são encontrados todos os pontos de Q_1 e Q_2 que estão próximos à curva de interseção por meio da função de distância do operador de projeção MLS. Em seguida, procura-se todos os pares mais próximos ($q_1 \in Q_1$, $q_2 \in Q_2$), e computa-se o ponto q na curva de interseção da seguinte forma (veja a Figura 14):Seja r o ponto na linha de interseção dos dois planos tangentes em q_1 e q_2 e que está mais próximo a estes dois pontos, isto é, que minimiza a distância $||r-q_1|| + ||r-q_2||$. r é a primeira aproximação de q e agora pode ser projetado sobre S_1 e S_2 para obter dois novos pontos de partida q_1' e q_2' para a iteração. Este procedimento pode ser repetido iterativamente até que os pontos q_1 e q_2 convirjam ao ponto q na curva de interseção. Normalmente este processo requer menos de três iterações.

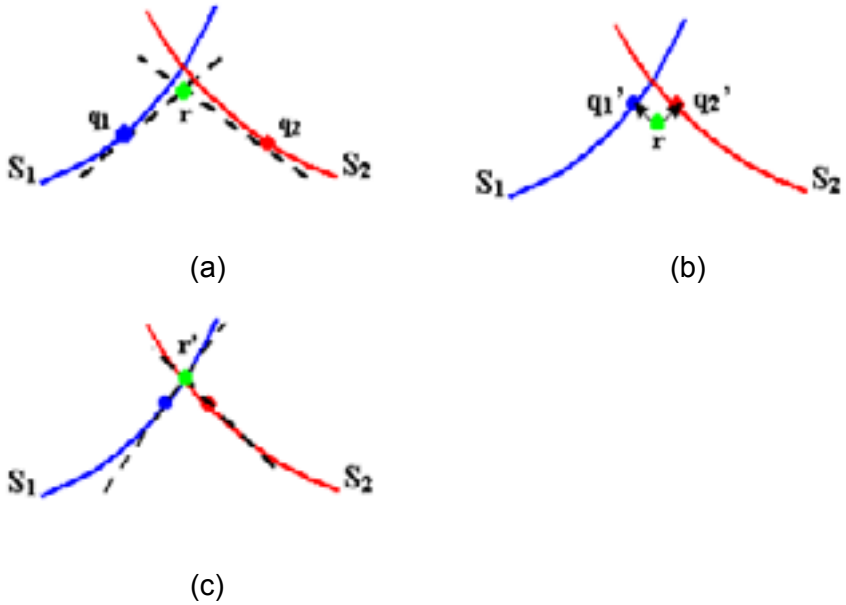


Figura 14: (a) Suponhamos a reta que é a interseção do plano tangente a S_1 em q_1 e o plano tangente a S_2 em q_2. O ponto r é o ponto desta reta que está mais próximo a q_1 e q_2. (b) r é projetado sobre S_1 e S_2, obtendo-se q_1' e q_2', dois novos pontos de partida para nova iteração. (c) A nova iteração gera r' que é uma aproximação melhor de q.

Pauly faz o tratamento de "arestas vivas" em tempo de renderização. Ele faz uma alteração no renderizador. Marca os *surfels* de interseção e, ao renderizá-los, são cortados para gerar uma "aresta viva".

4.4. Validade da Regra 2

Os dois métodos analisados nesta dissertação, utilizam a seguinte regra para a classificação de um *surfel*:

Regra 2: dado um volume V determinado por uma superfície de contorno S, uma amostragem P da superfície S, um ponto x qualquer, um ponto $p \in P$ que é a amostra mais próxima a x, um vetor n_y normal a S em p e orientado para fora de S, temos que: $(x - p) \cdot n_p > 0 \Leftrightarrow x \notin V$.

Em [16] é colocado que esta regra é tirada da seguinte propriedade da geometria diferencial (aqui chamada Propriedade 1):

Propriedade 1: Dado um volume V determinado por uma superfície de contorno S contínua e duplamente diferenciável, um ponto x qualquer, um ponto

$y \in S$ que é o ponto de S mais próximo a x, um vetor n_y normal a S em y e orientado para fora de S, temos que : $(x - y) \cdot n_y > 0 \Leftrightarrow x \notin V$.

A Figura 15 ilustra esta propriedade.

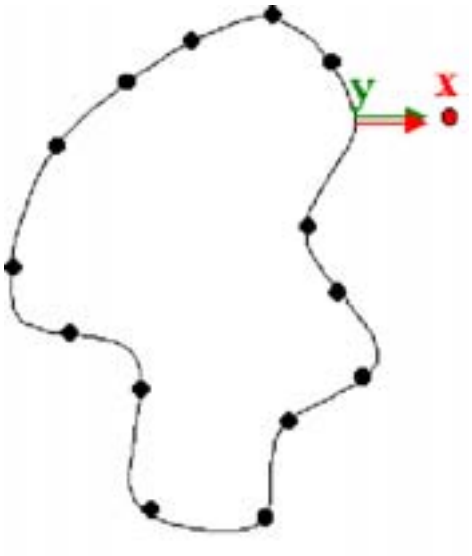


Figura 15: Para verificar se o ponto *x* está fora de um contorno *C*, encontra-se o ponto *y* de *C*. Se o produto interno entre a normal em *y* e o vetor que liga *y* a *x* for maior que zero, então *x* está fora de *C*, senão está dentro de *C*.

Pauly assume que, tendo uma amostragem *P* da superfície *S*, podemos usar a Propriedade 1, substituindo *y* por *p*, o *surfe*l mais próximo ao *ponto x:* $(x - p) \cdot n_p > 0 \Leftrightarrow x \notin V$ (Regra 2). Esta amostragem *P*, em seu trabalho, é (aproximadamente) uma “amostragem adequada”. Também seu tratamento de interseção cria uma aproximação dela (seção 4.3). Empiricamente, a “discretização da propriedade” é válida a partir desta suposição. Mas isso não foi demonstrado teoricamente em seu trabalho.

Pauly coloca em [16] que não existe ainda uma teoria matemática rigorosa de amostragem, análoga a teoria de Fourier, para superfícies discretas.

A melhor aproximação teórica para esta área é o uso do "Eixo Medial" que foi colocado em [10]. Vamos descrever este trabalho brevemente nesta seção. Este trabalho apresenta um algoritmo de reconstrução de superfície que parte de uma "amostragem adequada" (abaixo definida) de pontos e obtém uma malha que é topologicamente equivalente à superfície do objeto e à medida que a densidade da amostragem cresce, a saída converge para esta superfície.

Para definir amostragem adequada, o trabalho parte do conceito de "Eixo Medial".

"Eixo Medial" de uma superfície 2D posicionada no R^3 é o conjunto (fechado) de pontos p_i para os quais, buscando-se o ponto da superfície que está mais próximo a p_i , mais de um ponto pertencentes a esta superfície são encontrados. A Figura 16 abaixo ilustra o "Eixo Medial" em 2D.

Figura 16 [10]: As curvas de cor mais clara são o "Eixo Medial" da curva em preto.

Em duas dimensões, os vértices de *Voronoi* de uma amostragem densa aproximam o "Eixo Medial" da amostragem (mas esta propriedade não se estende a 3D). A Figura 17 mostra o Eixo Medial em 3D.

Figura 17 [10]: Em 3D, geralmente, o "Eixo Medial" é uma superfície 2D (em cor mais clara). Aqui o quadrado é o "Eixo Medial" da superfície transparente que o contorna.

Podemos definir agora a amostragem adequada da seguinte forma:

Supondo uma superfície suave, que quer dizer uma 2-*manifold* duplamente diferenciável posicionada no R^3 , uma "amostragem adequada" desta superfície é aquela na qual a densidade da amostragem é (pelo menos) inversamente proporcional à distância ao "Eixo Medial". A Figura 18 mostra uma "amostragem adequada".

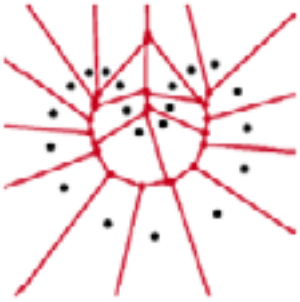


Figura 18 [10]: A figura mostra o diagrama de Voronoi de uma amostragem de pontos *S*. Da mesma forma que o a amostragem aproxima a curva, os vértices de Voronoi aproximam o "Eixo Medial".

Na prática, na reconstrução da superfície, uma aproximação deste “Eixo Medial” é computada (devido à dificuldade de computá-lo exatamente e em alguns casos até impossibilidade).

5
Melhorias Propostas

No capítulo 4, apresentamos dois métodos que realizam operações booleanas em modelos por pontos: o método de Adams e Dutré [2] e o método de Pauly[3]. Para esta dissertação, usamos o método de Adams e Dutré [2] e propomos as seguintes melhorias:

- Alteramos o momento em que é feita a classificação da *octree*. Fazemos esta classificação durante a sua criação. Os nós filhos de um nó pai *P* qualquer são classificados logo após a criação de todos estes filhos de *P*, independentemente do restante da *octree* ter sido criada. Isso pode ser feito porque a *octree* é construída na ordem "raiz-esquerda-direita" ("*preorder traversal*") e, portanto, após todos os filhos de um nó pai serem criados, a parte necessária da *octree* para a classificação destes filhos já está criada.

- A classificação da *octree* é otimizada classificando-se os cantos dos seus nós como dentro ou fora do modelo. Na classificação de um nó n_1 a partir de seu vizinho n_2, se os cantos que são comuns a n_1 *e* n_2 tiverem a mesma classificação, esta será dada ao nó. Assim, diminuímos o número de pesquisas pelo *surfel* mais próximo, que é a parte crítica da classificação dos nós da octree em termos de tempo de processamento.

- Acrescentamos um valor mínimo (maior que zero) para o número de *surfels* contidos em um nó, que é uma das condições de parada na criação da *octree*. Verificamos que desta forma diminui-se a chance de erro na classificação de partes críticas como “arestas vivas”.

Uma comparação entre a nossa implementação e a de Pauly também será feita no capítulo 6.

5.1.
Descrição do algoritmo

O algoritmo implementado para esta dissertação, baseado no método de Adams e Dutré [2], pode ser dividido nas seguintes partes: Criação e classificação de uma *octree* para cada modelo, Teste DentroFora, Criação do modelo resultado. Não foi implementado tratamento especial algum de interseção para o nosso *plugin*. As próximas seções apresentam estas partes.

5.1.1.
Criação e classificação de uma octree para cada modelo

As idéias do algoritmo são aqui ilustradas em 2D, mas a extensão delas para 3D é feita facilmente.

Para cada modelo, a *octree* é criada e classificada da seguinte forma:

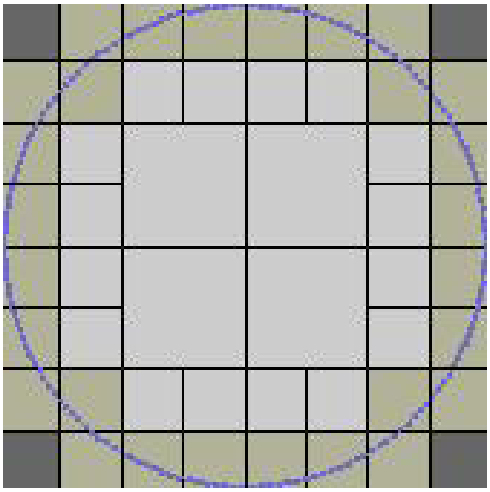


Figura 19-Quadtree classificada. Os nós mais escuros estão fora do modelo, os mais claros dentro e os em tom intermediário são nós de contorno.

Gera-se uma “caixa envolvente” do modelo contendo todos os *surfels* e cria-se o nó raiz da *octree* que corresponde a esta “caixa envolvente”. Este nó raiz é subdividido em oito filhos e seus *surfels* são distribuídos entre eles. Ao invés de testarmos os valores de x,y e z para cada *surfel* a fim de determinar para qual filho ele irá, utilizamos listas auxiliares. Primeiro os *surfels* são subdivididos em dois grupos pelos valores de x e armazenados em uma lista auxiliar. Em seguida, cada uma das listas é subdividida pelo valor de y e as quatro listas obtidas serão subdivididas pelo valor de z. Ao final teremos 8 listas de *surfels*, cada uma delas pertencente a um filho.

Cada nó será, então, novamente subdividido enquanto contiver um número de surfels > n_{min} (valor pré-definido) ou enquanto a profundidade da árvore < p_{max} (valor pré-definido). Na figura 19, $p_{max} = 3$ e $n_{min} = 3$.

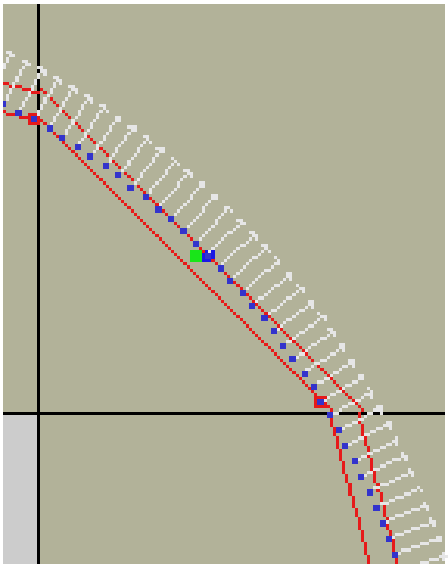


Figura 20-O nó cheio é dividido em três partes: à esquerda, a região está dentro do modelo, à direita está fora e entre as duas linhas que dividem o nó está a região que contém os surfels.

Os nós cheios recebem um tratamento especial. Eles são subdivididos em três partes da seguinte forma: uma que está fora do modelo, outra que está dentro e uma que contém os *surfels*. Esta subdivisão é feita por meio da criação de dois planos paralelos que dividem o nó. Para determinar estes planos, calcula-se o vetor, n_m, igual à média das normais dos surfels do nó: $n_m = \sum n_s / \| \sum n_s \|$, onde n_s é a normal de cada *surfel* do nó. Em seguida, encontra-se o *surfel* com maior produto interno entre a sua normal e a normal média: $d_1=Max(n_m \cdot n_s)$. Este *surfel* juntamente com a normal média define um dos planos. Encontra-se também o *surfel* com menor produto interno entre a sua normal e a normal média: $d_2=Min\ (n_m \cdot n_s)$. Este *surfel* juntamente com a normal média define o outro plano (Figura 20). Se o nó contém surfels de superfícies de orientações diferentes, ele não será subdividido, ou seja, os planos coincidirão com os planos que delimitam o próprio nó. Este teste foi implementado da seguinte forma: escolhe-se um *surfel* qualquer do nó. Calcula-se o produto interno entre a normal deste surfel e as outras normais dos outros surfels deste nó. Se algum produto interno for menor que zero, este nó não será subdividido. Após a etapa de subdivisão, classificam-se os oito cantos deste nó como "dentro" ou "fora" do modelo por meio da classificação de um ponto que descreveremos na sub-seção 5.1.1.1 .

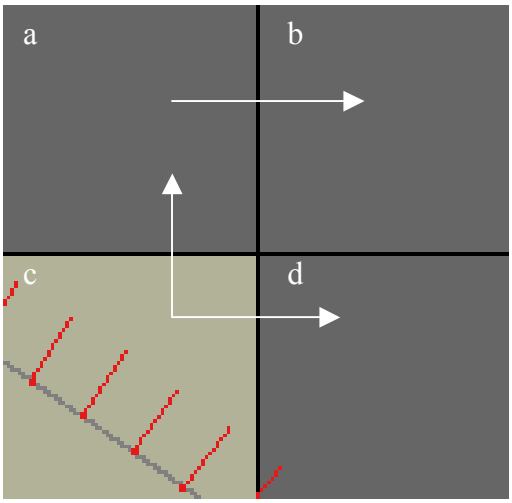


Figura 21-O nó *a* é classificado pelo nó *c;* o nó *d* também é classificado pelo nó *c*; a classificação do nó *a* é propagada para o nó *b*.

Já os nós vazios da *octree* serão classificados como “dentro” ou “fora” do modelo da seguinte forma: cada nó folha vazio é classificado a partir de um de seus irmãos vizinhos adjacentes. Se ele possuir um vizinho adjacente cheio ele será classificado por este vizinho, se não possuir, os vizinhos adjacentes serão classificados primeiro e ele terá esta mesma classificação (Figura 21). Como tem que existir sempre pelo menos um nó cheio entre os filhos de um nó pai, então sempre esta classificação poderá ser feita. Com a classificação de todos os filhos, tem-se também a classificação dos cantos do nó pai.

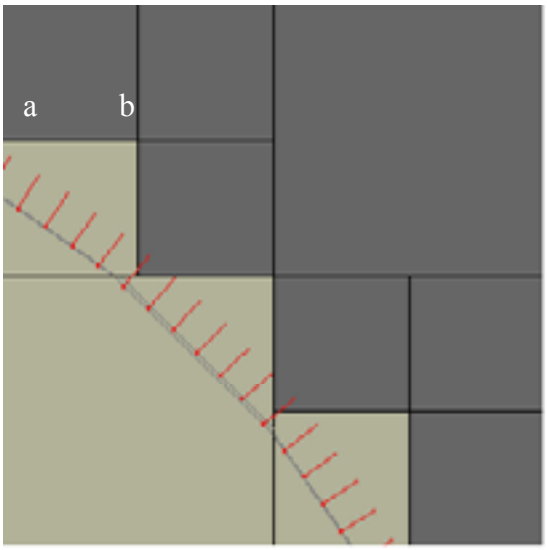


Figura 22-Os cantos *a* e *b* do nó cheio possuem a mesma classificação: fora do modelo. Esta classificação foi dada ao nó vazio adjacente.

Para classificar um nó vazio por meio de um nó cheio, verificam-se os cantos comuns destes dois nós: se todos os cantos tiverem a mesma classificação, esta será dada ao nó vazio (Figura 22). Se houver classificações diferentes escolhe-se o surfel do nó cheio mais próximo ao nó vazio; faz-se o produto interno entre a normal deste *surfel* e o vetor que liga o centro do nó cheio ao centro do nó vazio (ou o vetor que liga o surfel ao centro do nó vazio). Se o produto interno for maior que zero, o nó é classificado como “fora”, senão é classificado como “dentro”. Para encontrar o *surfel* (pertencente a um nó cheio) mais próximo ao nó vazio, existem dois casos:

- O nó cheio é folha: percorre-se toda a lista de surfels e escolhe o *surfel* mais próximo da fronteira entre o nó cheio e o vazio.

- O nó cheio não é um nó folha: percorrem-se os filhos do nó cheio, ou seja esta pesquisa é feita descendo na *octree* a partir do nó cheio, procurando o *surfel* mais próximo à fronteira do pai com o nó vazio a ser classificado.

Classificando o centro do nó vazio, todo o nó deve ter a mesma classificação pois este nó não contém *surfels*. Para encontrar o *surfel* do nó vazio mais próximo da fronteira entre o nó cheio e o vazio basta testar o valor da coordenada: se a fronteira é o lado direito do nó cheio escolhe-se o surfel com maior coordenada ‘x’; se é o lado esquerdo, escolhe-se o surfel com menor ‘x’ ; se lado superior, surfel com maior ‘y’, etc. A seguir apresentamos a classificação de um ponto qualquer pertencente a um nó cheio.

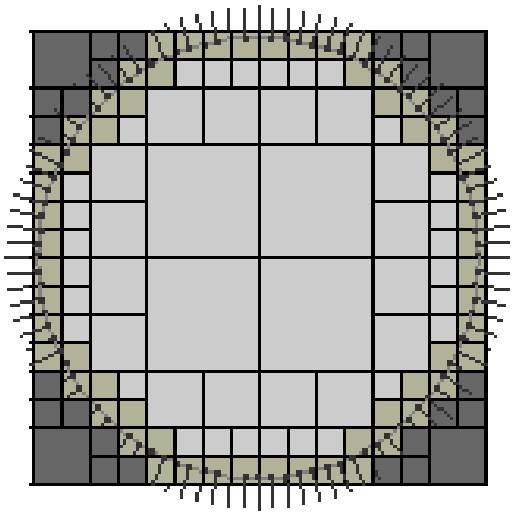


Figura 23- Quadtree classificada.

5.1.1.1. Classificação de um ponto de um nó cheio

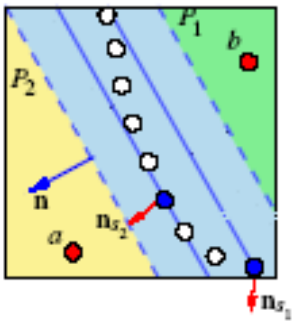


Figura 24 :[2] A normal **n** é a normal média e os surfels $\mathbf{s_1}$ e $\mathbf{s_2}$ juntamente com a normal **n** definem as retas que dividem o nó. O ponto **a** está fora do modelo e o ponto **b** está dentro do modelo.

Para um ponto localizado dentro do nó, faz-se o seguinte teste: se o nó foi dividido, testa-se de acordo com a divisão do nó: se o ponto está na parte do nó que está dentro do modelo, ele é classificado como “dentro”; se o ponto está na parte que está fora do modelo, ele é classificado como “fora” (Figura 24); se o ponto está na parte que contém os *surfels*, encontra-se o *surfel s*, pertencente ao nó, mais próximo do ponto e o classifica de acordo com *s*: seja n_s a normal em *s*, c_s a coordenada de *s*, c_p a coordenada do ponto . Se $(c_p - c_s) \cdot n_s > 0$, o ponto será classificado como “fora”. Senão, será classificado como “dentro”.

Se o nó não foi dividido em planos, esta classificação, a partir do surfel mais próximo, será sempre feita.

5.1.2. Teste DentroFora

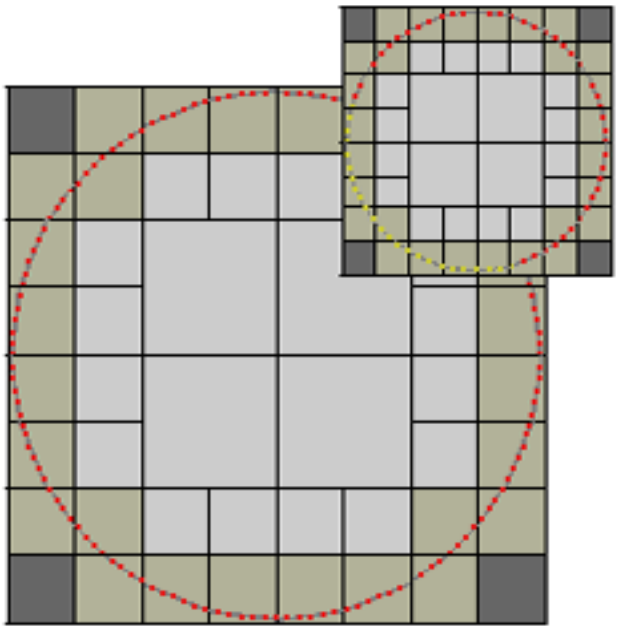


Figura 25-Duas circunferências e suas quadtrees correspondentes.

O Teste DentroFora classifica os *surfels* de um modelo como dentro ou fora de outro modelo, ou na sua interseção. Este teste foi aqui implementado exatamente como foi proposto em Adams e Dutré [2] , exceto que não foi implementado o método TINN - Triangle Inequality Nearest Neighbour [19]. Cada *octree* é percorrida, verificando-se se os seus nós ou *surfels* estão dentro, fora ou no contorno do outro modelo (Figura 25). Sejam *A* e *B* duas octrees, e queremos classificar a octree *A*. As duas octrees (*A,B*) são percorridas, partindo da raiz.

Para cada nó de *A* existem quatro casos (Figura 26). Para testar estas situações, os boxes dos nós são aumentados de r_{max}, o raio do maior *surfel* do modelo. Isto é feito para compensar o fato de que o surfel não é apenas um ponto, mas sim considerado como um disco.

Os quatro casos são:

. Se o nó de *A* não intersecta *B*, então todos os surfels deste nó são classificados como "fora".

. Se o nó de *A* intersecta apenas nós de *B* que estão "fora" do modelo, todos os surfels deste nó são "fora".

. Se o nó de *A* intersecta apenas nós de *B* que estão "dentro" do modelo, todos os surfels deste nó são "dentro".

. Se o nó de *A* intersecta nós de *B* que estão "dentro" e também nós de *B* que estão "fora", testa-se se este nó de *A* é ou não folha. Se não for folha, percorre-se os filhos deste nó de *A* até encontrar um nó folha cheio que intersecta *B*. Para cada *surfel* s_a deste nó folha de A , encontra-se o *surfel* de B mais próximo e classifica-se a coordenada de s_a da forma apresentada na seção 5.1.1.1. Seja s_b o surfel usado na classificação de s_a. Se os discos de s_b e s_a se intersectam, s_a é classificado como “interseção”.

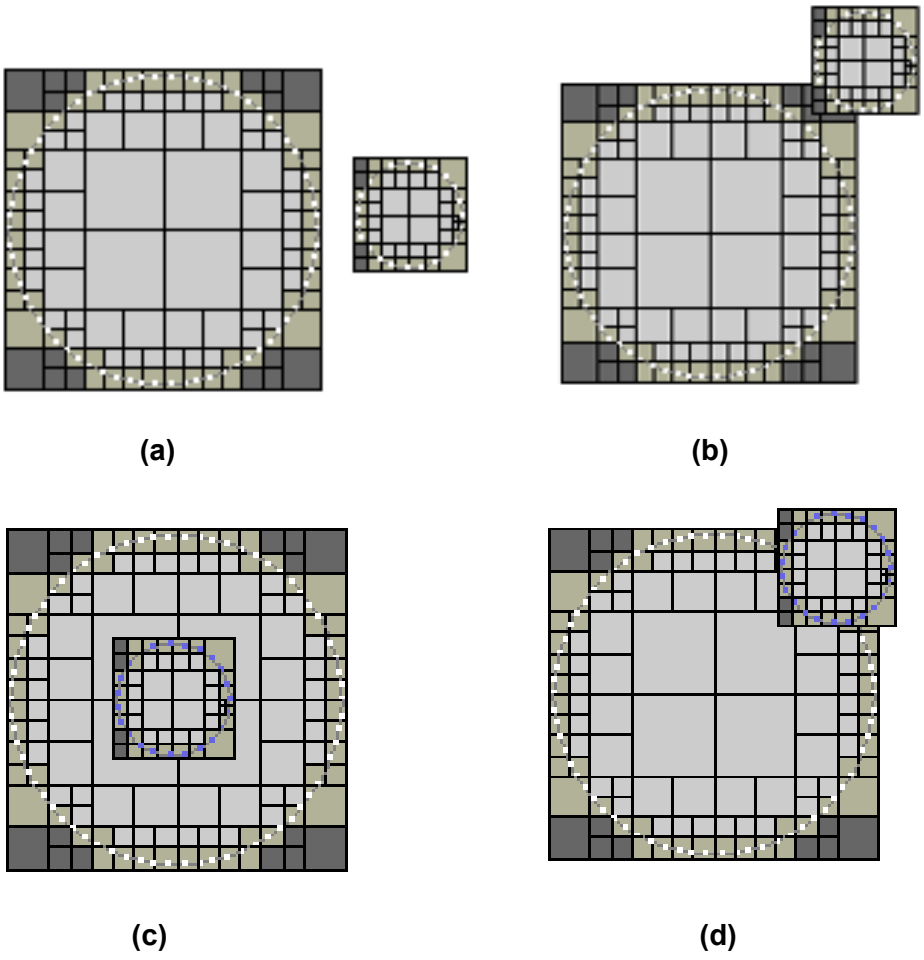


Figura 26- a. modelos não se intersectam. b. modelo intersecta nós fora. c. modelo intersecta apenas nós dentro. d. modelo intersecta nós dentro e nós fora.

5.1.3. Criação do modelo resultado

Um novo modelo é criado com os seguintes *surfels* (de acordo com a operação booleana escolhida pelo usuário - veja Regra 1 da seção 4.1):

União - surfels de A que estão fora de B + surfels de B que estão fora de A;

Interseção - surfels de A que estão dentro de B + surfels de B que estão dentro de A;

Diferença A-B - surfels de A que estão fora de B + surfels de B que estão dentro de A (com orientação invertida);

Como não foi implementado tratamento especial algum de interseção para o nosso *plugin, o*s surfels que se intersectam são mantidos no modelo resultado final sem modificações.

Descrito o algoritmo, vejamos a seguir a sua implementação.

5.2. Implementação

Este algoritmo foi implementado na forma de um *plugin* para o software PointShop 3D [11]. Este *plugin* foi desenvolvido na linguagem C e no sistema operacional Windows.

A escolha deste tipo de implementação permitiu-nos usar o software PointShop para os testes do algoritmo. Estes testes foram feitos com os modelos disponibilizados para o PointShop [11]. Nesta seção apresentaremos o software PointShop e as principais estruturas de dados usadas em nosso *plugin*.

O Pointshop é um software aberto, com seu código disponível, e preparado para trabalhar com *plugins*. É um software de edição de modelos 3D por pontos . Ele permite além de visualizar o modelo, alterar as suas características como cor, brilho, e fazer mapeamento de textura, iluminação, filtragem (Figura 27).

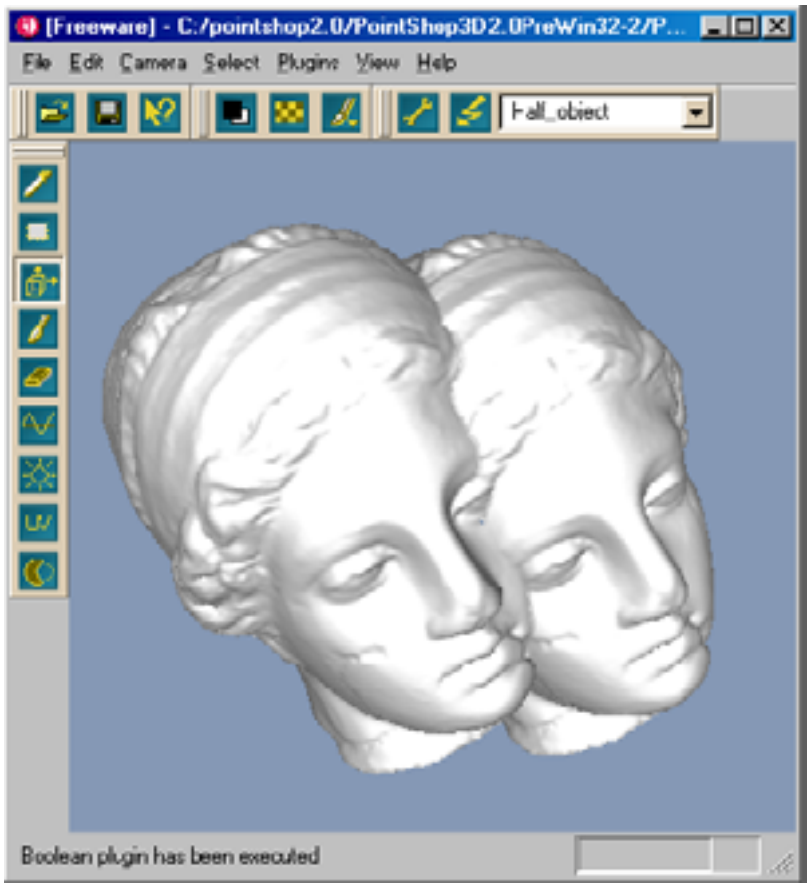


Figura 27-Software PointShop 3D com dois modelos carregados na sua cena.

Além disso, pode-se criar uma cena com vários modelos e visualizá-los de qualquer ponto de vista. Implementa o *virtual trackball* para rotacioná-los e transladá-los. Usa uma iluminação simples direcional. A sua arquitetura é extensível, permite que o usuário adicione seus próprios componentes ao sistema. Possui três tipos de componentes:

"Tools" fornece funcionalidades para edição interativa de superfície. São disponibilizados na "*toolbar"*.

"*Plugins*" fornece funcionalidades para processar, em um só passo, modelos ou partes deles selecionadas. São disponibilizados no menu "*Plugins*".

"Renderes" fornece funcionalidades de "display" para os modelos.

Para esta dissertação, foi escolhida a construção do componente "*Plugin*" por adequar-se à nossa proposta e ser a mais simples de ser construída. O PointShop [11] já contém uma *tool* que executa operações booleanas e deformação *free-form* sobre os modelos carregados na cena usando o método de Pauly [3] apresentado no capítulo 4. O nosso *plugin* "Operações Booleanas", implementa o algoritmo descrito na seção anterior, baseado na proposta de

Adams e Dutré [2]. Uma comparação entre o nosso *plugin* e a *tool* de Pauly [3] é feita no capítulo 6.

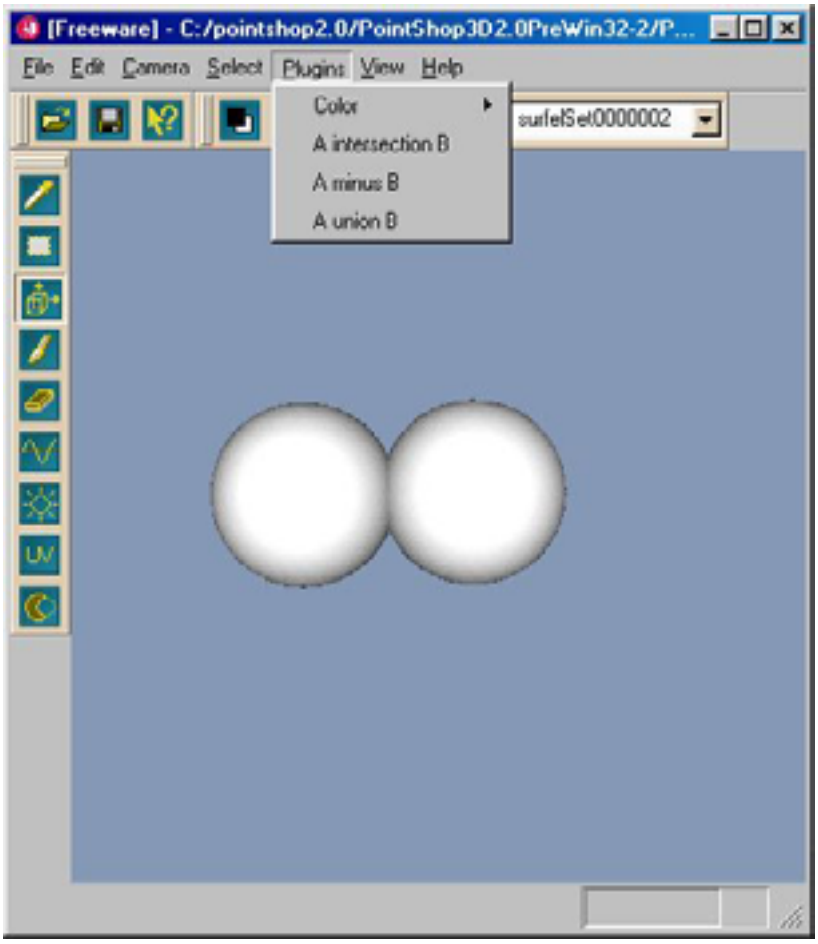


Figura 28-Menu do PointShop com o *plugin* e as opções para operações booleanas.

Para se executar o *plugin*, o usuário precisa carregar dois modelos na cena do PointShop, acionar o *plugin* informando a operação que será executada. O *plugin* executará o algoritmo acima proposto e carregará o resultado na cena do Pointshop, retirando os modelos anteriores.

5.2.1. Estruturas de Dados

Como vimos, o software PointShop [11] trabalha com modelos por pontos. Um modelo deste software, descrito na seção 4.2, contém todas as informações que o algoritmo proposto em Bart [2] necessita. Abaixo relacionamos as

principais estruturas de dados do PointShop usadas no *plugin* e as principais estruturas de dados do nosso *plugin*.

As principais classes do PointShop usadas no *plugin* são:

. Classe *Scene*

. Classe *Objects*

. Classe *SurfelColections*

. Classe *Surfel*

. Classe *StatusBar*

. Classe *Transformer*

A hierarquia de dados é a seguinte: a *Scene* contém vários *Objects*, os quais contém *SurfelColection*s. Cada *SurfelColections* contém *Surfels* que suportam uma ou mais propriedades, tais como cor difusa, cor especular.

. Classe *Scene*: Classe que representa uma cena que contém modelos.

. Classe *Object*: Um modelo é representado pela classe *Object*, que contém uma *SurfelCollection* e as matrizes de transformação para rotação, escala e translação. Cada *Object* é identificado pelo seu nome, que é único na *Scene*. O método *getUniqueName* retorna um nome único. Cada *Object* tem sua própria matriz de transformação que pode ser modificada com *setRotation*, rotate, *setPosition*, *translate*, *setScal*e, e *scale*. *Object*s podem ser copiados com *copy*. *Objects* consiste em um conjunto de surfels os quais podem ser recuperados pela *getSurfelCollectio*n. O *Objec*t pode ser ativado com *setActiveObject* e recuperado com *getActiveObject*.

Objects podem ser adicionados ou removidos com *addObjec*t e *removeObject*. Os *Objects* podem ser obtidos pela *getFirstObject* e *getNextObject*.

.*Classe SurfelCollection***: r**epresenta o conjunto de surfels de um modelo. Cada *surfelcollection* consiste em um conjunto de surfels (*SurfelInterfac*e) de tamanho obtido por *getNofSurfels.* Os surfels podem ser recuperados com *getFirstSurfel*, *getCurrentSurfel e getNextSurfel.*

A SurfelCollection contém um PropertyDescriptor, e uma SurfelList.

A SurfelList é uma lista duplamente encadeada de surfels. Possui os atributos: *firstSurfel*, *lastSurfel*, *currentSurfel* que são ponteiros para uma *SurfelInterface; nSurfels* que é o número de *Surfels*.

A *SurfelInterface* define a API para os *surfels* que são os menores elementos que compõem um *Object.* Podem ser considerados como pontos com uma normal que define a orientação do círculo (ou elipse) que será desenhado

tão logo o surfel seja renderizado. Tem como atributos dois ponteiros (*next* e *prev*) para *SurfelInterface*.

PropertyDescriptor consiste em até 32 valores descrevendo quais as propriedades a *SurfelInterface* suporta.

. Classe Surfel: **c**lasse que representa o surfel.

. Classe *StatusBa*r: **c**lasse que imprime mensagens na tela. Os *plugins* podem mandar mensagens para o usuário pela *StatusBar.*

Mensagens texto podem ser impressas com *showMessage*. O progresso de uma operação pode ser indicado com *setProgress* ou *resetProgress.*

. Classe *Transformer:* **r**epresenta a transformação entre sistemas de coordenadas. Esta classe fornece métodos para transformar pontos e vetores entre os diferentes sistemas de coordenadas do Pointshop.

A seguir apresentamos as principais estruturas de dados do *plugin* Operações Booleanas.

As principais estruturas de dados do *plugin* são: *boundingbox, tree*, no, lista_surfel, surfel.

Boundingbox: contém dois pontos 3D que definem uma caixa que contém o modelo: um ponto com menores coordenadas x,y e z e outro ponto com maiores x,y,z;

Tree: representa a octree do modelo. Cada registro contém um ponteiro para um nó e oito ponteiros para as 8 octrees filhas.

No**:** representa um nó da octree. Contém os seguintes dados:

- localização do nó que é o canto inferior esquerdo do nó;
- lista_surfel (preenchida apenas para nós folha cheios)
- quantidade de surfels do nó;
- classificação do nó como dentro, fora ou contorno;
- classificação do nó em relação a outro modelo;
- indica se o nó é folha ou não;
- vetor que é a média das normais dos surfels do nó;
- ponto que é a média das localizações dos surfels do nó;
- surfel que tem (entre todos os surfels do nó) o maior produto interno entre a sua normal e a normal média;
- surfel que tem (entre todos os surfels do nó) o menor produto interno entre a sua normal e a normal média;
- produto interno máximo e mínimo entre as normais dos surfels do nó e a normal média deste nó;

- indica se o nó contém surfels com normais em sentido oposto (produto interno entre elas é menor que zero);

- classificação dos oito cantos do nó;

lista_surfels: lista em que cada registro contém um apontador para um surfel e um apontador para o próximo registro.

surfel: representa o surfel da octree. Contém a classificação em relação a outro modelo e um ponteiro para o surfel do Poinshop.

6
Resultados

Neste capítulo apresentamos os testes realizados com o *plugin* para o PointShop implementado para esta dissertação, a comparação deste *plugin* com o *plugin* de Pauly [3] e a comparação entre o método de Adams e Dutré [2] e o método de Pauly [3].

6.1.
Testes

Os testes foram feitos em um PC x86 *Family* 6, 261600 KB de RAM, sem placa gráfica; sistema operacional Windows 2000. O *plugin* foi implementado na linguagem C.

Os modelos usados para os testes são modelos disponibilizados no PointShop [11]. Vale lembrar que o método implementado em nosso *plugin* é restrito a modelos sólidos e, portanto, foram usados apenas estes modelos sólidos para os testes.

O modelo na Figura 29, que é um dos modelos disponibilizados no PointShop [11], foi usado para mostrar o resultado das operações realizadas pelo *plugin* implementado nesta dissertação. O modelo *Igea* é um modelo sólido, que contém 134.345 surfels.

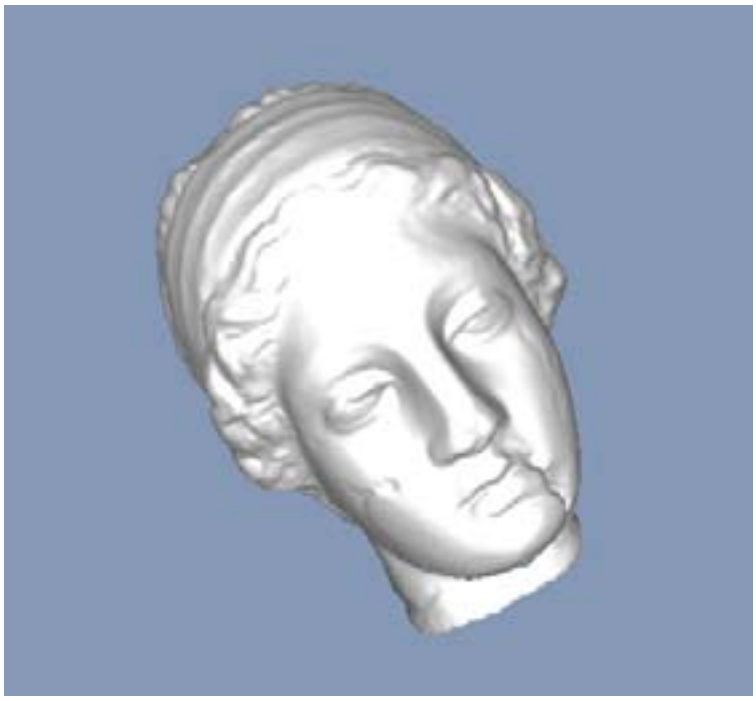


Figura 29-Modelo do PointShop Igea.

Para realizar as operações, este modelo foi carregado na cena do PointShop duas vezes e um deles colorido de vermelho. As Figuras 30, 31, 32 mostram o resultado da operação de união, interseção e diferença, respectivamente. Os números de surfels dos modelos resultantes são mostrados na Tabela 2.

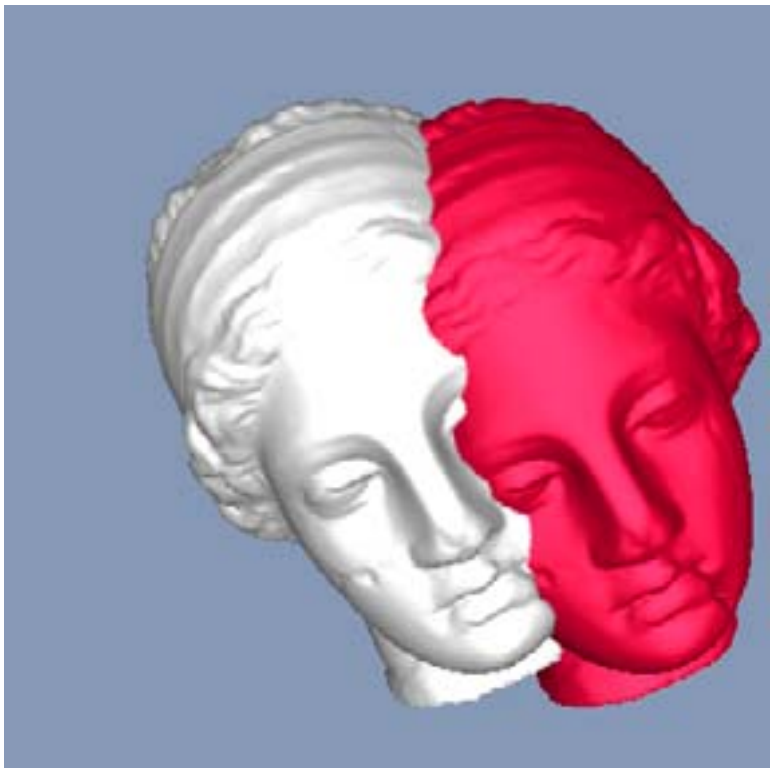


Figura 30-Resultado da união de duas Igeas.

Figura 31-Resultado da interseção de duas Igeas.

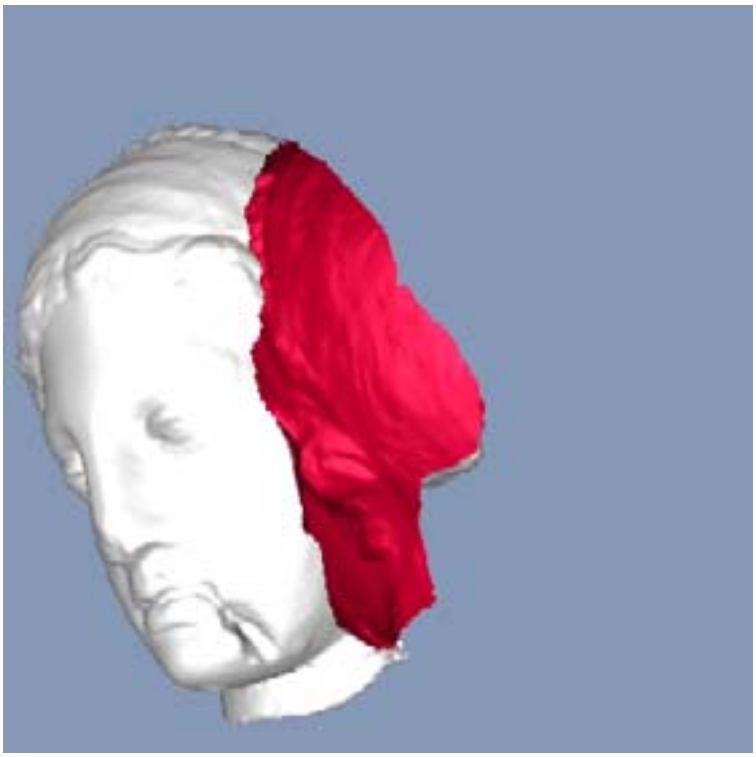


Figura 32-Resultado da diferença entre duas Igeas.

Operação	Número de surfels do modelo resultado
União	189.715
Interseção	72.225
Diferença	134.552

Tabela 2-Número de surfels dos modelos resultantes

Observamos que o número de surfels dos modelos resultantes está coerente com as figuras dos seus modelos.

Para avaliar o tempo de processamento das operações, foram realizados vários testes com o objetivo de obter uma média dos tempos de realização de cada operação. Também foram armazenados os tempos da criação das duas octrees e da classificação de todos os *surfels* dos dois objetos (como dentro ou fora do outro modelo). Estes testes foram feitos usando o modelo *Igea.* A média dos tempos de processamento foi calculada e os resultados são mostrados na Tabela 3. Verificamos que o maior tempo gasto é para criação das *octrees*. Verificamos também, que este resultado apresentado é similar ao resultado verificado empiricamente com o *plugin* de Pauly [3]. Os tempos gastos para realizar as operações são similares e o maior tempo gasto no *plugin* de Pauly também é para inicialização de suas estruturas de dados, que incluem a *kd-tree*.

Operação	Criação das Octrees (seg)	Classificação (seg)	Criação do resultado (seg)	Total (seg)
União	3,9375	1,324	1,8925	7,154
Interseção	3,8585	1,585	0,9547	6,3982
Diferença	3,99875	1,504	1,729	7,29475

Tabela 3- Média dos tempos de processamento para as operações com duas Igeas.

Com o objetivo de expor melhor os resultados deste trabalho, apresentamos abaixo mais um teste das operações booleanas usando o nosso *plugin*. Os modelos usados para este segundo teste são o *Max Planck* e o *Male*, ambos modelos sólidos disponibilizados no PointShop. Estes modelos são mostrados nas figura 33 e 34.

O primeiro, *Max Planck*, contém 52.809 surfels.

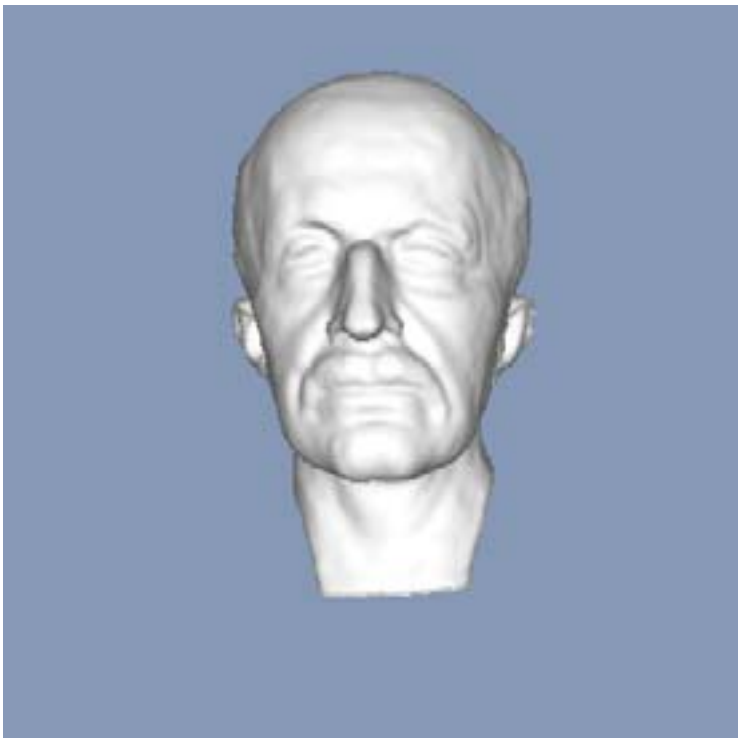


Figura 33-Modelo do PointShop MaxPlanck.

O segundo modelo abaixo, *Male*, contém 148.138 surfels.

Figura 34-Modelo do PointShop Male.

As Figuras 35,36,37 mostram o resultado da operação de união, interseção e diferença, respectivamente, usando os dois modelos acima. Podemos observar pela figura 35, que o resultado da operação de diferença, neste caso, gerou um modelo resultado com três partes desconexas. Os números de surfels de cada modelo resultado são mostrados na Tabela 5.

Figura 35- Resultado da União do modelo Male com MaxPlanck.

Figura 36-Resultado da interseção do modelo Male com o MaxPlanck.

Figura 37-Resultado da diferença do modelo Male do modelo MaxPlanck.

As médias de tempos de processamento foram calculadas da mesma forma que o primeiro teste com o modelo *Igea*, acima apresentado. Os resultados são mostrados na Tabela 4. Podemos observar que este teste

apresenta tempos um pouco menores que o teste com o modelo *Igea.* Isto se deve ao fato de que o modelo *MaxPlanck* ser significativamente menor que o modelo *Igea: MaxPlanck* possui 52.809 *surfels e Igea* contém 134.345 *surfels.*

Operação	Criação das Octrees (seg)	Classificação (seg)	Criação do resultado (seg)	Total (seg)
União	3,012	1,254	1,02	5,285
Interseção	2,985	0,901	0,861	4,747
Diferença	2,965	0,911	0,791	4,667

Tabela 4-Tempo de processamento das operações com os modelos MaxPlanck e Male.

Operação	Número de surfels do resultado
União	126.532
Interseção	88.414
Diferença	77.635

Tabela 5-Número de surfels dos modelos resultados das operações com os modelos MaxPlanck e Male.

Em seguida apresentaremos uma comparação entre o nosso *plugin* e o *plugin* de Pauly[3] e também uma comparação entre o método de Adams e Dutré [2] e o método de Pauly [3].

6.2. Comparação entre os dois métodos que realizam operações booleanas

A comparação entre os métodos de Pauly[3] e de Adams e Dutré foi feita em termos de tempo de processamento e em termos conceituais. Em termos de tempo de processamento comparamos o *plugin* de Pauly[3] com o nosso *plugin*.

Em termos de tempo de processamento, verifica-se empiricamente que os dois métodos apresentam tempos bastante similares. Entretanto, o tempo de processamento do algoritmo aqui implementado ainda pode ser melhorado com

o uso do método TINN [19] que foi proposto em Adams[2] - para otimizar a pesquisa do *surfel* mais próximo a um ponto que está localizado na região de um nó que contém *surfels* - veja seção 4.2. O uso deste método não foi implementado para este trabalho e a sua implementação melhoraria o tempo de resposta do algoritmo. Além disso, o *trabalho* de Sara F. Frisken [12], citado no capítulo 2, apresenta uma otimização no uso de octrees, que pode ser incorporada no algoritmo implementado neste trabalho.

Em termos conceituais, vamos fazer a comparação do uso do que chamamos aqui de Regra 2, do tratamento de interseção, e do tratamento de "arestas vivas".

Como vimos na seção 4.1, tanto o algoritmo implementado nesta dissertação, baseado no trabalho de Adams e Dutré, como o método de Pauly [3], parte do que chamamos aqui de Regra 2 para realizar operações booleanas com pontos. Mas cada um destes métodos dá um tratamento diferente para o uso desta regra.

O método de Pauly usa a Regra 2 para classificação de cada ponto ou de cada *surfel*. Para classificar um *surfel* de B, s_b, como dentro ou fora de A, é encontrado o *surfel* de A, s_a , mais próximo a s_b e s_b é classificado a partir de s_a. Para encontrar s_a Pauly utiliza uma kd-tree. Como esta pesquisa para encontrar s_a é cara (mesmo usando uma kd-tree), Pauly utiliza uma esfera que contém pontos com a mesma classificação (seção 4.3.2) e diminui em 90% este número de pesquisas. Pauly utiliza também a representação implícita MLS (Moving Least Squares) para casos em que a Regra 2 poderia falhar e para tratamento de interseções. Entretanto, o número destes casos em que a Regra 2 poderia falhar é muito pequeno. E a representação MLS não é uma estrutura simples de ser implementada. Portanto, não encontramos vantagens em usar MLS para estes casos. O método de Adams e Dutré não trata estes casos e, na prática apresenta resultados bastante satisfatórios, como foi mostrado. Portanto, consideramos que valeria a pena fazer algum tratamento especial para estes casos, se fosse um procedimento simples. Para a execução do algoritmo de Pauly, o uso de MLS não interfere significativamente porque a construção desta representação não é feita somente para realizar operações booleanas. Os testes empíricos que fizemos com o algoritmo de Pauly não incluem a construção desta representação.

Aparentemente os dois algoritmos são bastante parecidos, um usa *octrees* e o outro usa *kd-tree*. As duas estruturas de dados são simples e otimizam a classificação dos *surfels* de um modelo como dentro ou fora do outro modelo.

Mas existe uma diferença conceitual entre dois que merece ser destacada. O algoritmo aqui implementado usa a Regra 2 para classificação dos nós de uma *octree* e armazena esta *octree* classificada (ver capítulo 5). Assim na maioria dos casos não será necessário encontrar o *surfel* mais próximo para classificar um ponto (ou um *surfel*) como fora ou dentro de um modelo. Caso o ponto esteja em uma área da *octree* classificada, ele terá a classificação dada àquela área. A pesquisa ao *surfel* mais próximo será feita apenas nos casos em que o ponto a ser classificado está localizado entre os planos que subdividem um nó cheio.

O uso desta *octree* classificada levanta uma questão que não foi citada no trabalho de Adams e Dutré que é a proximidade desta estrutura (armazenada) de *octree* classificada com uma representação volumétrica do objeto. Sendo assim, esta estrutura pode ser aproveitada para outras funções além da classificação dos *surfels*. Por exemplo, podemos ter uma aproximação do volume de uma cena, somando os volumes aproximados dos modelos que a compõem. Como o enfoque deste trabalho é operação booleana, não vamos nos aprofundar nesta questão.

Outra importante diferença entre os dois trabalhos, que já foi explicada na seção 4.3.2. é que o método de Pauly aplica-se a modelos sólidos ou não sólidos, enquanto que o método de Adams e Dutré aplica-se somente a modelos sólidos.

Quanto ao tratamento de interseção, em termos de visualização todos os dois resolvem bem o problema. O tratamento de interseção nos dois trabalhos também resolve o problema da validade da Regra 2 (ver seção 4.4). Entretanto, o tratamento de Adams e Dutré não depende da renderização sendo, portanto, mais portável. Além disso, uma desvantagem do método de Pauly é o uso da representação implícita MLS (“Moving Least Squares”) do objeto para obter os pontos exatamente na curva de interseção. Esta representação torna o tratamento de interseção mais complexo e a vantagem apresentada não é tão significativa para este tratamento de interseção.

7
Conclusão

Concluímos, então, que as escolhas de Adams e Dutré por usar uma *octree* classificada e pelo tratamento dado à interseção são mais abrangentes que a escolha de Pauly. E consideramos que quanto mais abrangente for uma escolha conceitual, mais resultados poderemos tirar dela. No caso de Adams e Dutré, eles estendem o problema de classificar um ponto para classificar todo o modelo. Como foi citado, ele quase fornece uma representação volumétrica do modelo. Esta aproximada representação volumétrica pode nos fornecer outras informações como, por exemplo, o volume aproximado do modelo.

Além disso, o uso da representação implícita MLS no método de Pauly cria uma complexidade a mais para o método sem apresentar vantagens para a realização de operações booleanas, que é o nosso enfoque.

Logo, podemos dizer que o método de Adams e Dutré apresenta as seguintes vantagens em relação ao de Pauly:

- a estrutura de *octree* classificada, além de ser usada para classificação, fornece informações adicionais como volume aproximado do objeto.

- o tempo de processamento no método de Adams e Dutré tende a ser melhor que no método de Pauly, se implementadas as melhorias acima citadas e se colocássemos em consideração o uso da representação implícita MLS (*Moving Least Squares*).

- o tratamento de interseção de Adams e Dutré apresenta a vantagem da portabilidade, ou seja, irá funcionar independente do algoritmo de renderização utilizado.

E as vantagens do método de Pauly em relação ao de Adams e Dutré:

- a colocação de surfels exatamente na curva de interseção.
- a possibilidade de trabalhar com modelos sólidos ou não sólidos.

E, finalizando, concluímos que o estudo relatado nesta dissertação foi importante por:

- fornecer um breve estudo sobre modelagem por pontos, uma nova e promissora área dentro da computação gráfica.

- por apresentar uma avaliação dos dois trabalhos existentes sobre operações booleanas com pontos, encontrando as vantagens de cada um e levantando questões que não tinham sido citadas nos trabalhos originais.
- por propor e implementar melhorias ao trabalho de Adams e Dutré.

7.1. Trabalhos Futuros

Nossa implementação pode ser ainda evoluída com as propostas apresentadas na seção 6.2. Como observamos, o tempo de processamento do algoritmo aqui implementado ainda pode ser melhorado com o uso do método TINN [19] - que foi proposto em Adams[2] - para otimizar a pesquisa do surfel mais próximo a um ponto que está localizado na região de um nó que contém surfels. Além disso, o trabalho "Simple and Efficient Trasversal Methods for Quadtrees and Octrees" de Sara F. Frisken [12], citado no capítulo 2, apresenta uma otimização no uso de octrees. Outra proposta para melhorar o tempo de processamento do algoritmo aqui implementado é usar a forma de trabalhar com *octrees* apresentada neste trabalho.

O trabalho de Adams e Dutré [2] é aplicado apenas a objetos sólidos. O nosso *plugin*, portanto, aplica-se somente a este tipo de objeto. Já o trabalho de Pauly[3] é aplicado também a modelos não sólidos. Apresentamos como proposta a um trabalho futuro a extensão do algoritmo aqui implementado para modelos não sólidos. O surfel pode conter uma informação adicional que indica se ele está delimitando um volume ou não. Assim pode-se ter um modelo com “partes sólidas” e “partes não sólidas”. Por exemplo, a união de um plano com uma esfera poderia resultar numa esfera com “abas” (seriam as partes do plano que estariam fora da esfera). O modelo resultado teria uma parte correspondente a um espaço 3D (esfera) e outra parte que seria uma superfície 2D posicionada no espaço 3D sem delimitar algum volume ("abas").

Muitos trabalhos podem ser desenvolvidos aplicando-se o modelo por pontos em animação, jogos, visualização de terrenos, criação de protótipos, etc, considerando-se a eficiência e simplicidade do modelo. E mais especificamente as operações booleanas com pontos podem ser aplicadas para criação de cenários, testes de colisões e nestes casos a representação usando *octrees* pode ser mais explorada, considerando-se o seu potencial de representação volumétrica de um objeto.

8
Referências Bibliográficas

[1] MÄNTYLÄ, M. An Introduction to Solid Modeling. **Computer Science Press**, 1988.

[2] ADAMS, B.; DUTRÉ, P. Interactive Boolean Operations on Surfel-Bounded Solids. **Proceedings of SIGGRAPH 2003**, San Diego, USA, 2003.

[3] PAULY, M.; KEISER, R.; KOBBELT, L.; GROSS, M. Shape Modeling with Point-Sampled Geometry. **SIGGRAPH 2003**, Computer Graphics Proceedings, Annual Conference Series. ACM Press / ACM SIGGRAPH, 2003.

[4] LEVOY, M.; WHITTED, T. The Use of Points as Display Primitives. **Technical Report** TR 85-022. The University of North Carolina at Chapel Hill, Department of Computer Science, 1985.

[5] PFISTER, H.; ZWICKER, M.; VAN BAAR, J.; GROSS, M. 2000. Surfels: Surface Elements as Rendering Primitives. **SIGGRAPH 2000**, pp 335–342. New Orleans, LA, July 23-28, 2000.

[6] ZWICKER, M.; PFISTER, H.; VAN BAAR, J.; GROSS, M. Surface Splatting. **SIGGRAPH 2001**.

[7] PAULY, M.; KOBBELT, L.; GROSS, M. Multiresolution Modeling of Point-Sampled Geometry. **CS Techinical Report** #378 September, 16 2002.

[8] CURLESS, B. **New Methods for Surface Reconstruction from Range Images.** PhD thesis, Stanford University, 1997.

[9] RUSINKIEWICZ, S. **Real-time Acquisition and Rendering of Large 3D Models.** PhD thesis, Stanford University, 2001.

[10] AMENTA, N.; BERN, M.; KAMVYSSELIS, M. A new Voronoi-Based Surface Reconstruction Algorithm. **SIGGRAPH 98 Conference proceedings**. pp 415-421. New York, 1998.

[11] ZWICKER, M.; PAULY, M.; KNOLL, O.; GROSS, M. Pointshop 3D: an Interactive System for Point-Based Surface Editing. **Proceedings of SIGGRAPH 2002**, San Antonio, TX. July 2002.

[12] FRISKEN, S.; PERRY, R. Simple and Efficient Traversal Methods for Quadtrees and Octrees. **Journal of Graphics Tools**, Vol. 7, Issue 3. May 2003.

[13] CORRÊA, W. T.; OLIVEIRA, M. M.; Silva, C. T.; Wang, J. Modeling and Rendering of Real Environments. **Revista de Informática Teórica e Aplicada (Brazilian Journal of Theoretic and Applied Computing)**, 9(2):127-156. 2002.

[14] DAVIS, J.; MARSCHNER, S.; GARR, M.; LEVOY, M. Filling Holes in Complex Surfaces Using Volumetric Diffusion. **First International Symposium on 3D Data Processing, Visualization, Transmission.** June, 2002.

[15] RÄSÄNEN, J. **Surface Splatting: Theory, Extensions and Implementation.** Master's Thesis, Dept. of Computer Science, Helsinki University of Technology, May 28, 2002.

[16] PAULY, M. **Point Primitives for Interactive Modeling and Processing of 3D Geometry.** PhD thesis, Federal Institute of Technology (ETH) of Zurich, 2003.

[17] HOPPE, H.; DEROSE, T.; DUCHAMP, T.; MCDONALD, J.; STUELZLE, W. Surface Reconstruction from Unorganized Points. **Computer Graphics**, 26(2):71–78. July 1992.

[18] ALEXA, M.; BEHR, J.; COHEN-OR, D.; FLEISHMAN, S.; LEVIN, D.; SILVA, C. T. Point Set Surfaces. **IEEE Visualization 2001**, pp. 21–28. October 2001.

[19] GREENSPAN, M.; GODIN, G.; TALBOT, J. Acceleration of Binning Nearest Neighbor Methods. **Proceedings of Vision Interface 2000**, pp. 337-344.

[20] MENCL, R. **Reconstruction of Surfaces from Unorganized Three-Dimensional Point Clouds.** Dissertation, Dortmund Univ., 2001.

[21] KRIVANEK, J. Representing and Rendering Surfaces With Points. **http://citeseer.nj.nec.com/krivanek03representing.html**. Acesso em 30 jul. 2003.

[22] KRISTJANSSON ET. AL. Approximate Boolean Operations on Free-form Solids. **Proceedings of SIGGRAPH 01.** 2001.