
UNIVERSIDADE FEDERAL FLUMINENSE
INSTITUTO DE COMPUTAÇÃO
CIÊNCIA DA COMPUTAÇÃO

CARLOS EDUARDO VAISMAN MUNIZ

# EXTRAÇÃO DE MODELOS POLIGONAIS A PARTIR DE MODELOS VOLUMÉTRICOS

NITERÓI - RJ

2010

CARLOS EDUARDO VAISMAN MUNIZ

# EXTRAÇÃO DE MODELOS POLIGONAIS A PARTIR DE MODELOS VOLUMÉTRICOS

Projeto final apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal Fluminense, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação. Área de Concentração: Computação Gráfica.

ORIENTADOR: PROF. DSC. ANSELMO ANTUNES MONTENEGRO

Niterói - RJ
2010

CARLOS EDUARDO VAISMAN MUNIZ

# EXTRAÇÃO DE MODELOS POLIGONAIS A PARTIR DE MODELOS VOLUMÉTRICOS

Projeto final apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal Fluminense, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação. Área de Concentração: Computação Gráfica

Aprovado em abril de 2010

BANCA EXAMINADORA

---

Prof. DSc. Anselmo Antunes Montenegro - Orientador
UFF

---

Prof. DSc. Esteban Walter Gonzalez Clua
UFF

---

Prof. DSc. Aura Conci
UFF

Niterói - RJ
2010

Ao meu pai e minha irmã que tanto me apoiaram, me incentivaram durante todo meu percurso na faculdade, além de todo carinho que recebi deles.

À minha querida mãe, que apesar de não poder estar comigo hoje, me deu uma grande lição de vida, muito carinho, sempre me apoiou mesmo nos momentos mais difíceis e me ensinou a correr atrás dos meus sonhos.

À minha querida namorada Adriana, por todo seu carinho e apoio moral.

AGRADECIMENTOS

À Universidade Federal Fluminense,

Ao Prof. DSc Anselmo Antunes Montenegro por me orientar neste trabalho.

A todos os professores com os quais tive o privilégio de ser aluno.

Ao Daniel Cordeiro Monteiro, Arnaldo Cotrim Barbosa e a todos os amigos e colegas que contribuíram para a realização deste trabalho.

Ao Will Sutton, Kamil “Plasmadroid”, Koen Van de Sande, Stuart Carey e todos que programaram e contribuíram com o Voxel Section Editor III.

A todos os visitantes do Project Perfect Mod e usuários do Voxel Section Editor III que me apoiaram e motivaram a fazer este trabalho.

À Electronic Arts Los Angeles que permitiu o uso de formatos proprietários da série de jogos Command & Conquer e à NVIDIA pela doação da GeForce 7800 GTX usada para testar o trabalho.

## RESUMO

Este trabalho apresenta um estudo sobre a reconstrução de superfícies poligonais a partir de modelos volumétricos, propondo uma nova forma de construção de objetos gráficos, em que se misturam as técnicas de manipulação de voxels com a modelagem baseada em vértices, difundida nos aplicativos de edição de imagens 3D disponíveis no mercado. Enquanto que o uso de representações baseadas em bordos é recomendado para a criação de objetos com malhas complexas ou de grandes extensões, em certos contextos, como, por exemplo, jogos eletrônicos, a manipulação de voxels é mais intuitiva para o detalhamento dos modelos, dando maior controle ao usuário do que texturas baseadas em funções. O método desenvolvido neste trabalho conduz o processo de exportação de voxels para um conjunto de faces, arestas e vértices, detecta a orientação das superfícies, suaviza as cores e a geometria, além de reduzir o número de polígonos da malha gerada. Esta técnica foi testada com o auxílio do programa Voxel Section Editor III, escrito na linguagem de programação Delphi, para o qual existe uma diversidade de modelos gráficos gratuitos, sendo criados usando o formato de arquivo suportado por ele.



Palavras-chave: Poligonização de dados volumétricos, suavização de malhas, simplificação de malhas, modelos volumétricos.

**ABSTRACT**

This work presents a study on the reconstruction of polygonal surfaces from volumetric models, striving to devise new way of building graphical objects mixing voxel manipulation techniques with vertex based modeling, available in 3D image editing applications. While the use of boundary representation is recommended for constructing complex and large meshes, for some applications as, for example, videogames, the manipulation of voxels is more intuitive in the process of adding details to models, providing a more accurate control to the user than function based textures. The method developed in this work conducts the process of exporting volumetric pixels into sets of faces, edges and vertices, detects the surface normals, smooths the colors and the geometry of the surfaces and also reduces the polygon count of the generated mesh. This technique was tested with the help of the program Voxel Section Editor III, written in Delphi programming language, which has a vast amount of free graphical models, being constantly created using the file format supported by it.

Key words: Polygonization of volumetric models, mesh fairing, mesh simplification, polygon reconstruction, volumetric models.

**SUMÁRIO**

## LISTA DE FIGURAS

## LISTA DE TABELAS

## LISTA DE ABREVIATURAS, SIGLAS E SÍMBOLOS

| | |
|---|---|
| 2D | Duas Dimensões |
| 3D | Três Dimensões |
| 3DS MAX | Autodesk 3ds Max – Programa de manipulação de imagens em 3D baseado em bordo |
| API | Application Programming Interface - Interface de Programação de Aplicativos |
| B-REP | Boundary Representation – Representação baseada em bordo |
| CPU | Central Processing Unit – Unidade Central de Processamento |
| EMC | Extended Marching Cubes – técnica de reconstrução de superfícies poligonais |
| GIMP | GNU Image Manipulation Program – Programa de manipulação de imagens em 2D baseadas em pixels. |
| GPU | Graphics Processing Unit – Unidade Gráfica de Processamento |
| GNU | GNU's Not Unix – Sistema operacional em código livre |
| MC | Marching Cubes – técnica de reconstrução de superfícies poligonais |
| MS | Microsoft – Multinacional desenvolvedora de software e hardware |
| PIXEL | Picture Element – Elemento mínimo da imagem |
| PBRT1 | Problema de baixa resolução de dados do tipo 1, explicado na seção 2.2.2.2. |
| PBRT2 | Problema de baixa resolução de dados do tipo 2, explicado na seção 2.2.2.2. |
| QEF | Quadric error function – Função quádrica de erro. |
| RAM | Random Access Memory – Memória de acesso aleatório |
| SIGGRAPH | Special Interest Group on GRAPHics and Interactive Techniques – A principal conferência anual internacional de computação gráfica |
| SINC | Sinus cardinalis – Cardinal seno (sen(x)/x) |
| UFF | Universidade Federal Fluminense |
| VOXEL | *Vo*lume Pi*xel* – O menor elemento renderizável do volume |
| VXLSE III | Voxel Section Editor III – Editor de imagens volumétricas baseadas em pixels |

# Capítulo 1 – Introdução

No final da década de 90, com a disponibilidade de placas de vídeo com aceleração 3D no mercado e bibliotecas gráficas como OpenGL (2010) e DirectX (2010), houve uma difusão do uso de objetos tridimensionais em vídeos, aplicações, jogos, filmes, produções artísticas e de propaganda, etc. Estes são gerados a partir de modeladores, que lidam quase que exclusivamente com objetos gráficos baseados em vértices, arestas e faces.

A manipulação de malhas que representam tais objetos é intuitiva para muitos usuários, pois seu grau de expressividade é coerente com o resultado final da visualização, além de restringir todas as operações a uma única interface. No entanto, o armazenamento das informações das cores nestes modelos não pode ser feito de forma vetorial, já que o número de vértices, faces e arestas pode aumentar substancialmente, especialmente se houver uma riqueza de detalhes. Para simplificar o armazenamento destes detalhes utilizam-se mapas de textura.

A edição dos mapas de textura não é uma atividade trivial, já que o resultado final é diferente daquilo que é manipulado pelo usuário. Além disso, o uso de múltiplas interfaces ou até mesmo de outros programas pode ser necessário para esta finalidade. A superfície do objeto deve ser parametrizada em um espaço bidimensional, onde suas faces são projetadas. O posicionamento e a vizinhança das faces do modelo neste espaço diferem-se da malha original. Os mapas de textura contêm uma estrutura de dados diferente do resto do modelo, pois armazenam matrizes de pixels com dados de cores, iluminação, profundidade, entre outros.

Outra abordagem para a geração de objetos tridimensionais é a que se baseia em editores de modelos volumétricos (LORENSEN e CLINE, 1987), onde é possível visualizar e manipular dados de três dimensões através de projeções 2D (LEVOY, 1988). Modelos volumétricos representam objetos gráficos em um ou mais reticulados tridimensionais, formados por conjuntos de *voxels*, termo em inglês para denominar elementos de imagens volumétricas.

O desinteresse por editores de *voxels* para geração de modelos complexos pode ser explicado pelo excesso de esforço necessário para colori-los elemento a elemento. Em contrapartida, esta estrutura de dados possibilita ao usuário a edição manual de um conjunto discreto de elementos correspondentes a uma amostragem dos pontos da superfície desejada, dando maior controle ao usuário sobre aquilo que está sendo editado.

Para alguns usuários, principalmente aqueles que não são especialistas em design, esta abordagem é mais intuitiva do que o uso de um programa externo de edição de imagens 2D, ou um componente de interface específico em um sistema 3D, para criar texturas e, em seguida, mapeá-las na malha através de uma função em duas ou mais variáveis. O principal obstáculo que inviabiliza a utilização de voxels para texturização de modelos geométricos é o desconhecimento de um método que permita exportar conjuntos de *voxels* para malhas sucintas e com aparência correta.

A pesquisa correspondente a este trabalho teve início durante o desenvolvimento do Voxel Section Editor III (VXLSE3, 2010), um software de criação e edição de modelos volumétricos de código aberto (VXLSE3SVN, 2010), no início de 2009, com a re-escrita do código de renderização de voxels e uma tentativa de adaptação da técnica *Marching Cubes* (LORENSEN e CLINE, 1987) para a estrutura de dados do programa, com o intuito de gerar representações baseadas em polígonos.

Diferentemente de outros trabalhos, a técnica de poligonização desejada deveria considerar o máximo de detalhes artísticos inseridos pelo usuário na construção do modelo final, mesmo que estes fossem ruídos ou levassem a introdução de descontinuidades topológicas. Além disso, tinha-se como interesse levar em consideração os recursos artísticos e estilísticos utilizados pelo modelador para expressar as formas que ele pretendesse produzir, dentro do contexto de um espaço descrito por elementos volumétricos.

Os resultados obtidos através da técnica de *Marching Cubes* não satisfizeram os requisitos desejados, o que levou a uma busca por uma metodologia alternativa.

O objetivo deste projeto é investigar, desenvolver e testar técnicas eficientes para transformar objetos volumétricos em representações poligonais, usando a posição, cor e a direção do vetor normal de cada elemento para produzir malhas suaves e com um número baixo de polígonos, porém com boa expressividade considerando as características do processo original de modelagem em um espaço volumétrico, conforme especificadas anteriormente.

A metodologia proposta para alcançar os objetivos enumerados se baseia em um processo composto por quatro etapas: reconstrução de cada voxel como um cubo facetado, extração da malha correspondente à superfície externa (bordo) do modelo; suavização da malha resultante através de filtros baseados na composição de funções de transferência com o operador laplaciano discreto; e, finalmente, aplicação de um método de simplificação de malhas.

A transformação do modelo volumétrico em uma coleção de cubos descritos por seis faces, a partir da qual se extrai o bordo do modelo, é uma das etapas críticas, visto que nem sempre é possível obter um *manifold* 2D (variedade bidimensional que informalmente pode ser vista como uma generalização da noção de superfície). Isto ocorre porque, durante a modelagem, o usuário não necessariamente constrói um objeto cuja representação discreta corresponde a um *manifold* 2D, como será visto posteriormente no trabalho.

Inicialmente, um grande esforço foi empreendido para solucionar tais problemas topológicos, entretanto, tais aspectos acabaram por não ser considerados na versão final deste texto, ficando como um trabalho a ser investigado futuramente.

Os testes para a validação do método foram feitos com o Voxel Section Editor III, que tem um modelo de dados compatível com o que é utilizado neste trabalho, requerendo poucas adaptações. O código fonte desta pesquisa encontra-se disponível no SVN do programa e, espera-se um lançamento de uma nova versão do programa (1.4), com a finalização desta técnica e com a importação de dados geométricos em modelos volumétricos em meados de 2010.

Com isso, pretende-se que no futuro seja possível conciliar as duas metodologias de modelagem em um único editor de modelos tridimensionais com foco em jogos.

Esta dissertação está organizada de seguinte forma: o capítulo 2 apresenta alguns fundamentos sobre representação e reconstrução de objetos gráficos 3D, além de introduzir o conceito de frequência; o capítulo 3 descreve o problema da reconstrução de superfícies poligonais a partir de objetos volumétricos e os principais desafios a serem considerados na resolução do problema; o capítulo 4 descreve pesquisas anteriores relacionadas ao assunto, abordando algoritmos baseados no *Marching Cubes* e métodos baseados em processamento de imagens para suavização de malhas; o capítulo 5 apresenta e detalha o método proposto neste trabalho; o capítulo 6 descreve a metodologia e o ambiente utilizado para testes; o capitulo 7 descreve os resultados obtidos pela aplicação do algoritmo e comparações com técnicas alternativas e, finalmente, o capítulo 8 conclui o trabalho e apresenta sugestões para continuação desta linha de pesquisa.

# Capítulo 2 – Fundamentos

Este capítulo tem por objetivo, rever os principais conceitos sobre representação e reconstrução de objetos gráficos. Adicionalmente, são apresentadas noções sobre operações de filtragem no domínio da freqüência, utilizadas freqüentemente em processamento de imagens. A compreensão de tais técnicas é útil para um melhor entendimento das técnicas de suavização de malhas, que compõem uma parte essencial deste trabalho.

## 2.1 – Modelos para representação de objetos gráficos

A computação gráfica é um conjunto de técnicas e ferramentas para a manipulação de dados gráficos em um computador (AZEVEDO e CONCI, 2003). Uma de suas utilidades é servir de instrumento de concepção de arte, sintetizando objetos gráficos que podem ser animados. Eles podem ser utilizados em jogos, filmes ou qualquer software que trabalhe com imagens. A geração de imagens e animações, além de seu uso na interação com usuários são formas de mostrar e implementar ideias e conceitos, podendo ser usadas para qualquer finalidade e por qualquer segmento do mercado de trabalho.

Num sentido mais amplo, pode-se afirmar que a computação gráfica lida com os problemas relacionados à síntese, manipulação e processamento de objetos gráficos. Um objeto gráfico corresponde a uma descrição matemática da forma e aparência de um objeto do mundo real. Cada objeto gráfico é representado matematicamente através de um par $O=(U,f)$, onde $U$ é um subconjunto de pontos de um subespaço euclidiano que descreve sua geometria e f uma função de atributos que descreve a aparência do objeto em cada um dos pontos que o compõem.

Objetos gráficos podem ser classificados em objetos gráficos planares ou espaciais segundo a dimensão do espaço em que estão imersos. Além disso, podem ser classificados conforme a dimensão do conjunto que descreve seu suporte geométrico. Curvas são exemplos de objetos unidimensionais que podem ser planares ou espaciais, enquanto que regiões são objetos gráficos bidimensionais planares. Superfícies são objetos bidimensionais espaciais (possuem área mas não volume) e, por sua vez, sólidos são exemplos de objetos gráficos espaciais tridimensionais.

É muito comum a descrição de objetos gráficos através de modelos funcionais que podem ser paramétricos ou implícitos. Modelos paramétricos descrevem os pontos que caracterizam um objeto através de funções definidas em um espaço de parâmetros. Já os modelos implícitos descrevem o suporte geométrico de um objeto através das raízes de uma equação. Curvas implícitas são descritas através das raízes de uma equação da forma $f(x,y)=0$; superfícies são determinadas pelas raízes de uma equação $f(x,y,z)=0$. Os objetos gráficos de nosso interesse são os sólidos já que consistem de abstrações dos objetos tridimensionais que encontramos no mundo real. Existem basicamente duas formas de se representar sólidos: através da representação por bordo ou através de uma descrição dos elementos que o constituem numa decomposição do espaço ambiente.

### 2.1.1 – Representação baseada em decomposição do espaço

Neste tipo de representação, o espaço no qual o objeto está imerso é subdivido em um conjunto discreto de elementos. A forma do objeto em questão é descrita através de uma função característica que determina quais elementos do espaço fazem parte do objeto ou então por meio de uma enumeração.

A representação matricial de um objeto é baseada na decomposição do plano. Uma imagem digital como mostrado na figura 2.1, por exemplo, tem como suporte geométrico uma decomposição do plano em um conjunto de elementos discretos formando um reticulado. Cada

elemento tem uma cor e é chamado de *pixel*, palavra que tem origem na contração das palavras *picture element*.

Um processo análogo ocorre na descrição de imagens tridimensionais e objetos volumétricos. A diferença fundamental é a de que o espaço ambiente é tridimensional e cada elemento da decomposição é um *pixel* volumétrico denominado *voxel*.

Voxels possuem cor e posição bem definidas, sendo esta última caracterizada por coordenadas inteiras. Transformações de rotação e redimensionamento podem ter resultados inconsistentes, com perda ou distorção de informações uma vez que as coordenadas não são definidas no conjunto dos números reais e sim no conjunto dos inteiros. Por isso, esta representação não é ideal para gerar animações.

Neste trabalho, tem-se por objetivo gerar superfícies através da poligonização de objetos volumétricos, sendo então importante definir a noção de vetor normal a um voxel que faz parte da parte externa ou casca do modelo.

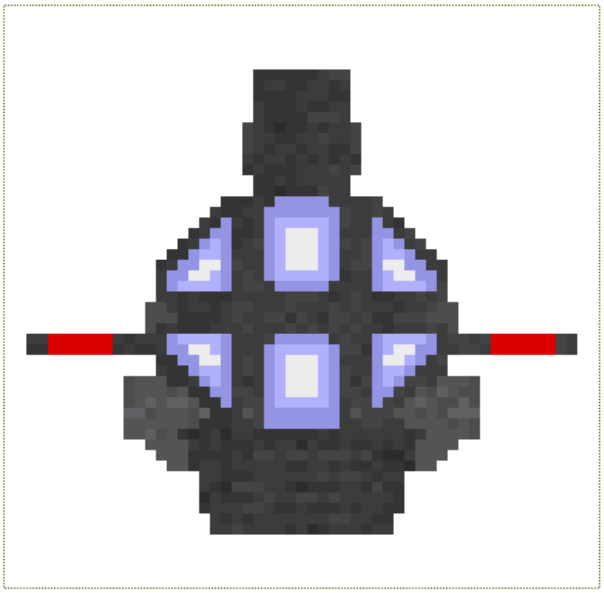

Figura 2.1 – Exemplo de imagem criada a partir pixels.

2.1.2 – Representação baseada em bordo (b-rep)

A outra abordagem para representar sólidos chama-se *boundary representation*, ou simplesmente *b-rep*.

A geometria e a topologia do objeto representado são descritas pela posição dos vértices caracterizada através de coordenadas reais, pelas arestas que fazem a ligação entre dois vértices e pelas faces, que contém três ou mais arestas.

O bordo do objeto pode ser visto como a discretização de uma superfície determinada por uma ou mais funções de múltiplas variáveis cujo domínio é o corpo dos números reais ou, em alguns casos, dos números complexos. Transformações de escala, rotações e translações são consistentes e reversíveis, sem perdas de informação. Vértices ou faces contêm cores e normais, representadas de acordo com a estrutura de dados utilizada na sua codificação. A figura 2.2 mostra um exemplo de um objeto com representação baseada em bordo.

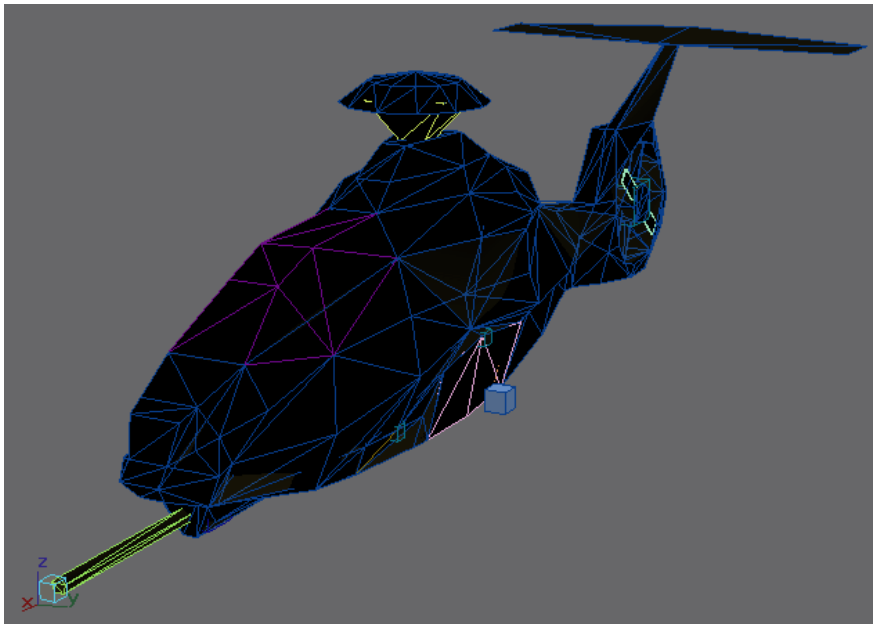

Figura 2.2 – Exemplo de imagem sintetizada a partir de representação baseada em bordo.

### 2.1.3 – Vantagens e desvantagens da representação por borda e decomposição espacial

As duas representações podem ser comparadas a partir de diversos critérios:

- Espaço em disco: o espaço utilizado pela representação volumétrica (decomposição espacial) é proporcional ao tamanho do volume, enquanto que na representação b-rep é proporcional à quantidade de detalhes ou ruídos na malha.

- Manipulação pelo usuário: a representação por bordo é mais intuitiva para a geração de objetos grandes, já que a modelagem é feita com linhas e faces baseadas em conceitos geométricos usados por profissionais de várias áreas do mercado usando programas como AutoCAD (2010), 3ds Max (3DSMAX, 2010), Maya (2010), Blender (2010), etc. A representação em pixels ou voxels é mais intuitiva para pintar detalhes de cores e ruídos no objeto e é utilizada, no caso bidimensional, para geração de texturas, por programas como Adobe Photoshop (2010), GIMP (2010), MS Paint, etc.

- Processamento e uso de memória na renderização: a representação de bordo sempre precisa ser convertida para uma matriz de *pixels*, portanto ela requer muito mais processamento e memória do sistema do que a representação por decomposição espacial para enviar a imagem para o dispositivo de saída. No entanto, o custo de memória para manipular a imagem segue a regra descrita para o atributo 'espaço em disco', ou seja, na representação *b-rep* é proporcional à quantidade de detalhes na malha enquanto que na representação volumétrica é proporcional à quantidade de voxels.

- Transformações lineares: a representação baseada em bordo consegue manter sua integridade após sofrer inúmeras rotações ou redimensionamentos, enquanto a representação baseada em decomposição espacial tem seus resultados distorcidos se a rotação for feita em um ângulo não múltiplo de 90º.

Para compensar as desvantagens da representação em bordo, é comum muitos programas projetarem imagens sobre as faces de uma malha que descreve a geometria de um objeto 3D. Essa projeção é chamada de mapeamento de textura (MÖLLER et al, 2008) (HECKBERT, 1989) e pode conter outras informações além de cores, como iluminação, distorções de geometria, etc. Os mapas de textura são bastante utilizados em programas de geração de efeitos (chamados de *shaders*) executados pela placa de vídeo durante o processo de renderização. Um exemplo de mapas de texturas utilizados em um objeto com representação b-rep (soldado) pode ser vista na figura 2.3.

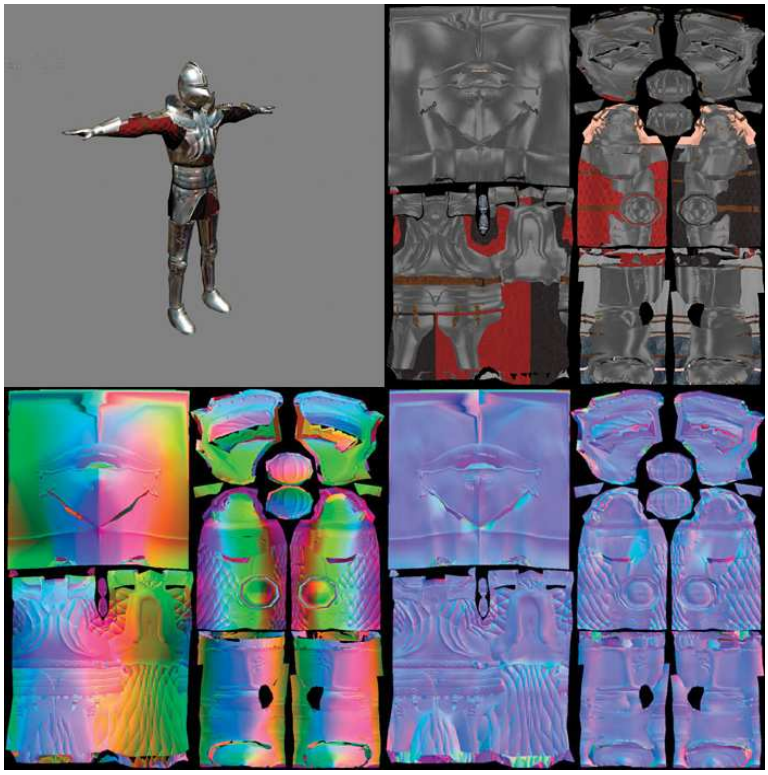

Figura 2.3 – Exemplo de imagem em bordo com texturas com mapa de difusão (no topo à direita), mapa de tangente (na parte de baixo à esquerda) e mapa de normais (abaixo à direita).

Apesar do uso de malhas texturizadas ajudar a reduzir o uso de espaço em memória e disco dos objetos gráficos, aumentar o nível de detalhamento no objeto, agilizar o tempo de renderização com o uso de shaders e permitir transformações lineares consistentes, este modelo não é intuitivo para modelagem de objetos tridimensionais, já que o usuário não consegue editar o resultado final texturizado diretamente.

As texturas são editadas separadamente, o que faz com que o usuário não tem uma noção completa sobre a que face cada pixel da imagem pertence, além de ter que resolver como as coordenadas de texturas serão geradas. A edição de um modelo com representação volumétrica não possui esse problema, pois todos os voxels do modelo estão unificados em uma só imagem localizados em suas posições originais.

## 2.2 – Frequências Espaciais

A noção de frequência pode ser compreendida de modo intuitivo como a quantidade de ocorrências de um determinado evento em um período de tempo. A frequência espacial é uma adaptação do conceito de frequência para espaço, onde se mede a quantidade de ciclos ou oscilações em um sinal por uma unidade de espaço, bastante utilizada na área de processamento de sinais e imagens. Visto que é possível representar uma imagem como uma função, é possível definir o conceito de frequência para tais objetos matemáticos. A figura 2.4 mostra alguns conceitos básicos utilizados na análise de oscilações de sinais:

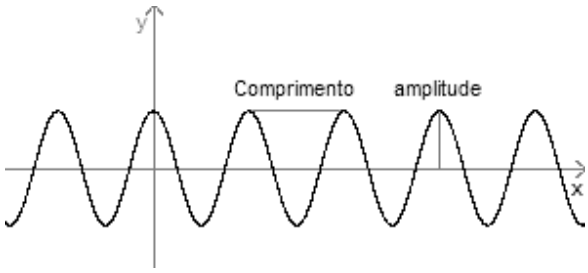


Figura 2.4 – Exemplo de ciclos em uma função periódica e conceitos básicos de oscilações como comprimento e amplitude.

Em uma imagem com representação baseada em decomposição do espaço, os dados da função são os tons de cinza ou valores de um canal de cor. Cada canal de cor é tratado separadamente. A frequência em uma imagem captura a variação do sinal em uma unidade de tempo e é um conceito fundamental na definição de filtros utilizados em procedimentos de realce

de imagens para eliminar ruídos (suavizar ou borrar a imagem), encontrar bordas, e outras formas de processamento.

### 2.2.1 – Realce de imagens para pixels a partir de filtragem de frequência

Técnicas de realce de imagens têm como objetivo modificar uma imagem original para torná-la mais apropriada para um objetivo específico determinado pelo problema a ser resolvido. Elas são feitas a partir da transformação dos dados de uma imagem em frequência, o processamento desta, e sua transformada inversa para uma nova imagem como mostrado na figura 2.5.


Figura 2.5 – Fluxograma do procedimento de realce de imagens.

A Transformada de Fourier (GONZALEZ e WOODS, 2000) é um mecanismo que é usado para transformar uma função no domínio de tempo/espaço em uma função no domínio de frequência, servindo como um instrumento de detecção e comparação de frequências. Em 2D:

$$F(u,v) = \iint f(x,y) \cdot e^{-2\pi\sqrt{-1}\cdot(ux+vy)} dxdy$$

As variáveis *u*, *v* são de frequência, enquanto *x, y* são de tempo/espaço. Ambas as funções pertencem aos números reais.

A Transformada Inversa de Fourier converte uma função de domínio de frequência em tempo/espaço. Em 2D, ela tem a seguinte fórmula:

$$f(x,y) = \iint F(u,v) \cdot e^{2\pi\sqrt{-1}\cdot(ux+vy)} dudv$$

Para o uso de dados discretos, no caso de pixels, existe a Transformada Discreta de Fourier e sua inversa, que respectivamente são:

$$F(u,v) = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) e^{-2\pi\sqrt{-1}(\frac{ux}{M}+\frac{vy}{N})}$$

$$f(x,y) = \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u,v) e^{2\pi\sqrt{-1}(\frac{ux}{M}+\frac{vy}{N})}$$

Onde *M*, *N* são usados para rastrear os pixels vizinhos nos eixos x e y respectivamente.

Devido sua alta complexidade computacional, é comum utilizar o algoritmo de Transformada Rápida de Fourier e sua inversa, que não são objeto de estudo deste trabalho.

A Transformada de Fourier e suas derivações têm várias propriedades como separabilidade, rotação, translação, periodicidade e simetria conjugada, mudança de escala, entre outros, que não serão explorados neste trabalho. A única propriedade que é fundamental para o realce de imagens é a convolução.

Seja *f*(*x*,*y*) a imagem original, *g*(*x*,*y*) a imagem processada e *h*(*x*,*y*) uma função de transferência que determina qual tipo de filtragem e processamento será feita com a imagem, gerada a partir de uma transformada inversa da função *H*(*u*,*v*), então, pelo teorema da convolução, temos:

$$f(x,y) * h(x,y) \Leftrightarrow F(u,v)H(u,v)$$

Além disso, a inversa também é válida:

$$F(u,v) * H(u,v) \Leftrightarrow f(x,y)h(x,y)$$

*F*, *G* são os resultados da aplicação da Transformada de Fourier nas funções *f* e *g* respectivamente. Além disso, * é o símbolo da operação de convolução. Esta pode ser descrita da seguinte maneira, com dados discretos:

$$g(x,y) = \frac{1}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f(x-m, y-n)h(m,n)$$

Portanto, utiliza-se a convolução acima para gerar filtros (função *h*) para tratar imagens (função *f*) no domínio de frequência. A função *h*, também chamada de núcleo (kernel) da convolução, pode ser representada como uma matriz de *M* por *N* elementos, onde cada elemento é o peso de cada pixel da vizinhança do pixel (*x*, *y*) atual na transformação. O valor resultante da operação preencherá os canais de cores vermelho, verde e azul separadamente. É normal que *M* e *N* tenham o mesmo valor e que também sejam números ímpares maiores que 1, para que o efeito seja distribuído para vizinhança de forma proporcional a distância do elemento da matriz ao seu centro.

Existem vários tipos de filtros distintos e quase todos eles não serão objeto de estudo deste trabalho, com exceção dos filtros de passa baixa. Estes servem para eliminar ou atenuar altas frequências da imagem, gerando um efeito de suavização ou borramento da mesma. Eles são usados para reconstruir imagens ou para interpolar dados. Neste trabalho, analisaremos três filtros de forma sucinta: Média, Gaussiana e Lanczos.

O filtro de Média é gerado a partir do somatório de todos os pixels na vizinhança de um pixel considerado, dividido pelo número total de pixels na máscara. Neste caso, todos os elementos da matriz de convolução têm o mesmo valor, sendo 1 dividido pelo número de elementos. O exemplo a seguir mostra um filtro de média com uma matriz 3x3:

| | | |
|---|---|---|
| $\frac{1}{9}$ | $\frac{1}{9}$ | $\frac{1}{9}$ |
| $\frac{1}{9}$ | $\frac{1}{9}$ | $\frac{1}{9}$ |
| $\frac{1}{9}$ | $\frac{1}{9}$ | $\frac{1}{9}$ |

O filtro Gaussiano é modelado a partir da seguinte função gaussiana (SHAPIRO e STOCKMAN, 2001):

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

O $\sigma$ é o desvio padrão, que pode ser usado como uma constante ou calculado a partir da distância dos pontos vizinhos com o atual usando a fórmula:

$$\sigma = \sqrt{\frac{1}{N}\sum_{1}^{N}((x_i - x)^2 + (y_i - y)^2)}$$

Neste caso, *N* é o número de vizinhos do ponto a ser avaliado.

Este filtro tem como propriedade o fato de que sua Transformada de Fourier e a inversa serem iguais no domínio contínuo. Portanto ele suaviza ruídos de pixels de forma coerente com o nível de sua frequência. Exemplo de filtro Gaussiano composto de valores inteiros:

$$\frac{1}{273}$$

| | | | | |
|---|---|---|---|---|
| 1 | 4 | 7 | 4 | 1 |
| 4 | 16 | 26 | 16 | 4 |
| 7 | 26 | 41 | 26 | 7 |
| 4 | 16 | 26 | 16 | 4 |
| 1 | 4 | 7 | 4 | 1 |

O filtro de Lanczos foi modelado a partir da criação de uma janela da função *sinc(x)*. Esta é considerada um filtro ideal de passa baixa por ser resultado da aplicação da transformada de Fourier em uma função retângulo que retorna valor 1 para números entre -0,5 e 0,5 e zero para os outros valores.

$$Sinc(x) = \frac{sen(x)}{x}$$

Portanto, esta função pode ser usada como base para filtrar apenas as frequências abaixo de uma determinada constante. Para retornar resultados entre 0 e 1, esta função é geralmente normalizada se transformando em:

$$Sinc(x) = \frac{sen(\pi x)}{\pi x}$$

O filtro de Lanczos (DUCHON, 1979) é uma janela da função *sinc(x)*, sendo ideal para aplicação da função *sinc(x)* em uma região de suporte limitado. Ela é descrita através da seguinte fórmula:

$$L(x,y) = Sinc(\sqrt{x^2+y^2})Sinc(\frac{\sqrt{x^2+y^2}}{\alpha}) = \alpha \frac{sen(\pi\sqrt{x^2+y^2})sen(\frac{\pi\sqrt{x^2+y^2}}{\alpha})}{\pi^2(x^2+y^2)}$$

O $\alpha$ é um parâmetro que regula o efeito de suavização da função e a rejeição de frequências altas. Caso $x$ e $y$ sejam 0, retorna-se o valor 1. Usando $\alpha = 3$ em uma matriz 5x5, temos o seguinte exemplo de filtro de Lanczos:

| | | | | |
|---|---|---|---|---|
| 0,003 | 0,029 | 0 | 0,029 | 0,003 |
| 0,029 | -0,146 | 0 | -0,146 | 0,029 |
| 0 | 0 | 1 | 0 | 0 |
| 0,029 | -0,146 | 0 | -0,146 | 0,029 |
| 0,003 | 0,029 | 0 | 0,029 | 0,003 |

2.2.2 – Filtragem de frequência para realce de malhas.

Neste trabalho, o conceito de frequência espacial será usado para suavização de malhas. O sinal a ser processado é a própria malha do objeto gráfico. Em cada vértice, os detalhes podem ser percebidos como oscilações locais no sinal, a “crista de uma onda”, que estão associados a frequência do sinal. Como o objetivo da suavização de malhas é perturbar a posição de cada vértice, o sinal deve ser analisado localmente em cada vértice.

Apesar de ser um dos instrumentos mais utilizado para detectar e comparar frequências espaciais, o uso da *Transformada Rápida de Fourier* com uma janela foi descartado devido a seu alto custo computacional, com uma complexidade computacional na ordem de n log n.

Optou-se por uma metodologia alternativa onde a análise do sinal em cada vértice pode ser feita a partir da detecção do ângulo formado pela crista da onda gerada pelo vértice. Esta onda pode ser vista como um volume formado por um plano descrito por todos os vértices vizinhos ao vértice avaliado e as respectivas ligações de todos estes vértices a ele. Para facilitar a visualização deste conceito, mostrado na figura 2.6, pode-se considerar o plano formado pelos vizinhos como a base da onda e a distância do vértice avaliado em relação a esta base como sua altura em um determinado tempo *t*.

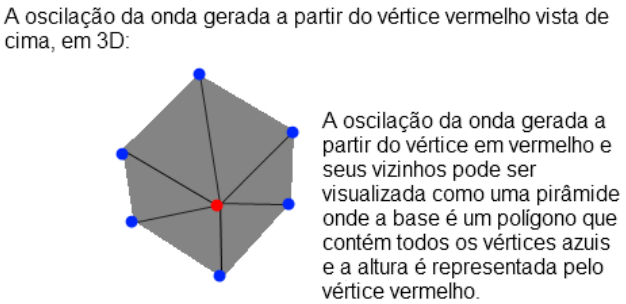


Figura 2.6 – Visualização gráfica da oscilação de uma onda em um vértice.

Para fazer qualquer operação de realce de malha, é necessário que se faça a comparação entre frequências diferentes. Esta comparação só é possível se tivermos ondas com amplitudes

iguais. Para isso, devemos considerar que em um determinado momento $t + t_i$, sendo $t_i$ um valor positivo, a onda atingirá a amplitude comum a todas as outras ondas, se fizermos uma operação de escala na onda ao longo do tempo.

O momento $t + t_i$ é equivalente a metade do período da onda, já que neste tempo, temos apenas a crista positiva da onda sem a parte negativa, e, portanto, não obtemos um ciclo completo. A frequência é estimada a partir do número de oscilações por unidade de tempo. A partir disso, podemos concluir que a frequência máxima que pode ser obtida a partir de um vértice é de meia oscilação por unidade de tempo e ocorre quando $t_i$ for igual a 0. Neste caso, a amplitude é atingida no tempo $t$. A frequência mínima será nula e ocorrerá quando a altura da onda é nula no tempo $t$. Pode-se concluir que nesta situação $t_i$ é infinito e a altura da onda jamais atingirá a amplitude das outras ondas.

Frequências baixas representam ângulos suaves enquanto as altas representam mudanças abruptas na geometria podendo ser ruídos ou ter algum significado artístico relevante.

Neste trabalho, a comparação de frequências é feita a partir da derivada da posição do vértice em relação aos seus vizinhos através da seguinte fórmula:

$$\nu(x) = K \cdot \frac{\sum_{i=0}^{N-1}(x_i - x)}{N}$$

Onde $K$ é uma constante de escala feita para compensar a disparidade do valor original da frequência com o valor obtido pela derivada, $N$ é o número de vizinhos do vértice atual, $x$ é a posição do vértice atual e $x_i$ é a posição dos vizinhos.

O resultado obtido a partir desta derivada é uma estimativa proporcional a frequência da onda no vértice. Este método é coerente, pois ondas suaves têm derivadas baixas e, portanto, frequências baixas. Ângulos agudos têm resultados altos, assim como suas frequências. O uso da constante $K$ é necessário para regular o resultado, aproximando-o de sua frequência real.

Esta abordagem é viável em modelos gerados a partir de dados discretos, subdivididos espacialmente com tamanho uniforme, onde não existem discrepâncias nas distâncias entre os vértices. Caso existam discrepâncias significativas, a detecção da frequência ficará comprometida, retornando valores baixos para a maioria dos vértices.

# Capítulo 3 – Reconstrução de superfícies poligonais a partir de volumes

## 3.1 – Visão geral do problema

Uma abordagem intuitiva para futuros programas de modelagem de objetos tridimensionais seria o uso de representação baseado em decomposição espacial de modo combinado com a representação baseado em bordo, permitindo o usuário alternar voluntariamente entre elas. Esta combinação permitiria o usuário aproveitar as vantagens de gerar malhas de forma vetorial e detalhar as cores da superfície a partir de voxels, de forma que seja possível visualizar o resultado final em tempo real. Para que esta abordagem seja viável, é necessário converter uma representação para outra e vice-versa da forma mais otimizada possível. Existem vários métodos consistentes de voxelização (CHEN e FANG, 1998) (CHEN e FANG, 2000) (EISEMANN e D'ECORET, 2006) (SILVA et al, 2009), que é o processo de converter uma malha geométrica em um conjunto de voxels. No entanto, não existe uma solução definitiva para o problema de poligonização e os resultados das técnicas dependem de características dos dados de entrada. Em alguns casos, tais métodos não podem ser aplicados diretamente sobre os dados gerados por editores gráficos 3D. Estes métodos serão discutidos com mais detalhes no capítulo 3 deste trabalho.

Neste trabalho, o objetivo da reconstrução de superfícies poligonais é obter vértices, arestas e faces a partir da casca externa de um modelo volumétrico, como mostrado na figura 3.1. Na representação volumétrica, um *voxel* é um elemento pontual, o que pode gerar dificuldades para satisfazer de forma apropriada as operações de visualização, tais como *zoom* e aproximação da câmera quando aplicados ao objeto.

Uma abordagem minimalista para visualização de modelos representados por voxels consiste em sua conversão para cubos representados por modelos facetados. Apesar de muito simples, esta estratégia pode envolver a necessidade de redimensionamento de todas as regiões,

havendo a necessidade de uma reconstrução de detalhes que, até então, não existem ou são desconhecidos.

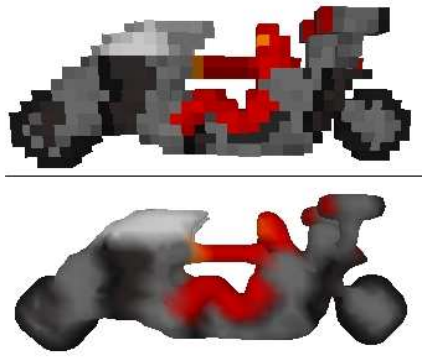

Figura 3.1 – Exemplo de modelo volumétrico de baixa resolução (em cima), cujos voxels são representados com cubos, convertido para uma malha geométrica (em baixo).

## 3.2 – Desafios da reconstrução de superfícies a partir de *voxels*

A resolução deste problema envolve o tratamento de vários subproblemas, alguns dos quais consideravelmente complexos, em particular os que lidam com questões topológicas.

### 3.2.1 – Triangulação

Triangulação é uma das técnicas mais utilizadas para reconstrução de objetos gráficos contínuos a partir de representações discretas. Resolver o problema de triangular um conjunto de vértices representando dados *d*-dimensionais, onde *d* é tipicamente dois ou três, envolve em determinar como conectar um conjunto de vértices formando arestas que pertencem a um conjunto de triângulos *T*, tal que se $T_i$ e $T_j$ são dois triângulos disjuntos pertencentes a *T*, vale a seguinte propriedade:

$T_i \cap T_j = \varnothing$,
$T_i \cap T_j$ é um vértice ou aresta pertencente a $T$

A complexidade computacional dos algoritmos de triangulação é O($n$ log $n$), sendo $n$ o número de arestas. Normalmente, procedimentos mais gerais são evitados por muitos métodos de poligonização de dados volumétricos através da utilização de padrões pré-definidos para geração de faces, como, por exemplo, o Marching Cubes (LORENSEN e CLINE, 1987). O capítulo 3 mostrará como todos os algoritmos conhecidos de reconstrução de superfícies poligonais evitam este subproblema.

### 3.2.2 – Baixa resolução de dados

A representação do objeto através de pixels ou voxels é feita a partir de uma amostragem finita do espaço contínuo que contém o objeto, gerando um conjunto de dados discretos a partir da subdivisão do espaço em partições de mesmo tamanho, com coordenadas definidas a partir de números inteiros positivos. A representação por bordo descreve dados em coordenadas reais, enquanto a paramétrica pode ser gerada a partir de números complexos também. Estas duas últimas não são estruturadas em espaços topológicos discretos. Durante uma conversão de dados para um espaço topológico discreto, ocorrem perdas tanto com respeito à geometria quanto a natureza do sistema de vizinhanças em cada ponto, isto é, sua topologia.

Em um modelo de dados volumétrico, é possível identificar duas situações distintas em que uma amostragem finita, normalmente em baixa resolução, leva a representação de partes do objeto com uma topologia distinta da original. Neste trabalho utilizaremos as seguintes denominações para os problemas de baixa resolução de dados conforme apresentado a seguir:

- *Problema de Baixa Resolução do Tipo 1* (PBRT1) ocorre quando um *voxel* não tiver vizinhos em lados opostos de um mesmo eixo. Este problema pode ser resolvido se o *voxel* for subdividido pela metade em todos os eixos. Um exemplo de PBRT1 pode ser visto na figura 3.2.

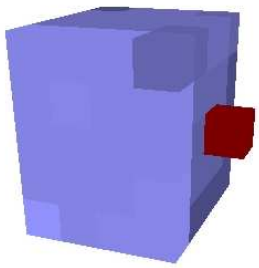

Figura 3.2 – Exemplo de PBRT1 no voxel em vermelho.

- *Problema de Baixa Resolução do Tipo 2* (PBRT2) ocorre quando, considerando um *voxel* como uma região cúbica, dois *voxels* estão ligados apenas por arestas ou vértices, o que se configura como uma descontinuidade de uma topologia de dados discreta, onde a região da aresta onde ocorre a interseção deveria estar dentro do volume do objeto, mas está na superfície. Em 2D, esta descontinuidade ocorre quando os dois *pixels* estão ligados apenas por vértices, considerando o *pixel* como um quadrado. Este problema de baixa resolução de dados também é conhecido como *aliasing* (AZEVEDO e CONCI, 2003) e a figura 3.3 ilustra um exemplo disto.

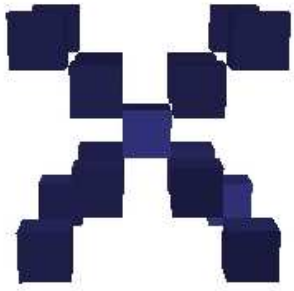

Figura 3.3 – Exemplo de PBRT2 em todos os voxels em azul.

O modelo não terá problemas de baixa resolução de dados se cada um de seus *voxels* tiver uma única orientação (vetor normal) para qual a luz reflete.

Os usuários que usam programas de edição de imagens a partir de voxels não necessariamente levam em consideração que ao pintar detalhes do volume  podem gerar modelos volumétricos cuja discretização torna difícil a extração de uma variedade bidimensional. Uma variedade bidimensional é uma generalização do conceito de uma superfície e pode ser entendida intuitivamente como uma família de conjuntos localmente homeomorfos a um subconjunto do plano. . É muito comum os casos em que, para fins artísticos, os modeladores representam cones e cilindros através diagonais formadas por voxels cujo bordo não corresponde a noção de variedade 2D uma vez que contém elementos de dimensão unitária como as arestas na interseção entre dois voxels consecutivos.

# Capítulo 4 – Pesquisas Relacionadas

No capítulo anterior, descrevemos o problema da extração de superfícies poligonais como um problema de reconstrução de um modelo contínuo a partir de um conjunto de voxels. Como os voxels possuem coordenadas, podem ser tratados como vértices, porém, em certas situações, sua simples ligação não conseguirá formar uma malha consistente com o modelo original, se houver qualquer tipo de descontinuidade mencionado na seção 3.2.2. Um exemplo é o ilustrado na figura 4.1 onde o modelo é formado por três elementos isolados. Sua conexão formaria uma face triangular, o que não corresponde a ideia original.

Figura 4.1 – Exemplo de imagem com descontinuidades de pixels/vértices, que requer vértices adicionais para ser consistente em processos de zoom de câmera.

A maneira mais simples de gerar um modelo geométrico que resolve, em parte, este problema é transformar um pixel em uma região quadrática para imagens em 2D ou um *voxel* em uma região volumétrica para imagens em 3D, duplicando os vértices existentes para todos os eixos e suas combinações. Neste trabalho, chamaremos este procedimento de *regionalização de pixels*.

Qualquer dispositivo de saída de imagens já faz isso por padrão devido à própria forma que a imagem aparece neles, apesar de que o software que for reconstruir uma imagem vai precisar fazer esse processo de forma autônoma para gerar uma malha, armazenando esta informação separadamente.

Modelos gerados a partir de cubos são muito rudimentares e só produzem resultados satisfatórios se cada cubo também ocupar um pixel da tela.

Este capítulo tem por objetivo, descrever, de forma sucinta, as diversas metodologias pesquisadas anteriormente para resolver o problema da reconstrução. Serão descritos os métodos desenvolvidos para poligonização de superfícies descritas na forma implícita e por campos escalares, além a adaptação do uso de técnicas de tratamento de imagem na malha com o uso de filtragem espacial.

## 4.1 – Métodos para extração de superfícies poligonais a partir de superfícies implícitas

A pesquisa em reconstrução de modelos geométricos a partir de volumes foi motivada, em grande parte, pela área de visualização de dados médicos, com o objetivo de gerar malhas tridimensionais a partir de dados obtidos de tomografias e outros exames. Estes dispositivos normalmente utilizam um sistema de coordenadas esféricas, capturando a distância da superfície e, em alguns casos, a cor em relação a cada pixel da imagem, que é vinculado a dois ângulos de rotação em relação à câmera do equipamento. A partir disso, foram criados métodos como o Marching Cubes (LORENSEN e CLINE, 1987), Dual Contouring (JU et al, 2002) e derivados que fazem uso dessas informações.

### 4.1.1 – Marching Cubes

*Marching Cubes* é um algoritmo apresentado no SIGGRAPH 1987 por Lorensen e Cline que extrai uma superfície poligonal a partir modelos volumétricos. A presença de um vértice é gravada em um *bit*, gerando um número de 8 bits. O algoritmo utiliza esse número para consultar

uma tabela que mostrará a disposição dos vértices e faces geradas para cada configuração. A figura 4.2 ilustra qual bit identifica cada vértice e aresta.

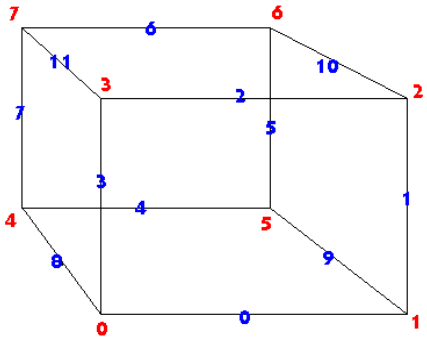


Figura 4.2 – Região cúbica com 8 vértices em vermelho e 12 arestas em azul, cada uma com um identificador.

Para saber se uma superfície passa por uma aresta, basta que exista um vértice da aresta que esteja dentro do volume e outro que esteja fora. As 256 possíveis combinações geradas pelos vértices geram uma tabela de configurações de faces. Estas se resumem a 15 variações mostradas na figura 4.3 (LORENSEN e CLINE, 1987), porém com rotações diferentes:

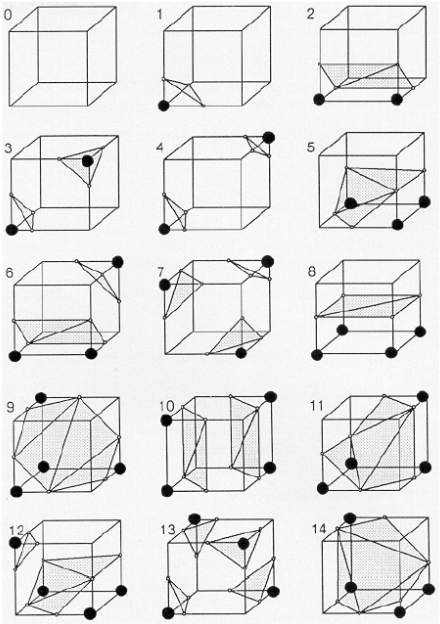


Figura 4.3 – Configurações de faces geradas a partir do Marching Cubes.

O mecanismo para detecção se um vértice da região está ou não dentro do volume depende da estrutura de dados utilizada.

É possível adaptar este método para usá-lo com dados volumétricos sem ter a noção da distância para superfície, considerando que um vértice estará dentro da região interna delimitada pela superfície se e somente se todos os *voxels* aos quais ele pertence não estiverem fora do volume. Essa abordagem falha se ocorrer qualquer um dos tipos de problemas de baixa resolução de dados descritos na seção 3.2.2, gerando buracos na malha final. No caso do tipo 1, o método detectará que todos os vértices estarão fora do modelo. Com o tipo 2, a ligação entre os dois *voxels* que causam esse problema também será ignorada na geometria resultante, por ser considerada fora do volume.

Portanto, uma correta adaptação do método pode necessitar de mais referências e gerar mais configurações, aumentando muito sua complexidade final. Durante esta pesquisa, foram tentadas diversas formas de adaptar o *Marching Cubes* para resolver esses problemas, mas não houve sucesso. A figura 4.4 ilustra o modelo volumétrico original e a 4.5 mostra o resultado de uma das diversas tentativas de implementação desta técnica.

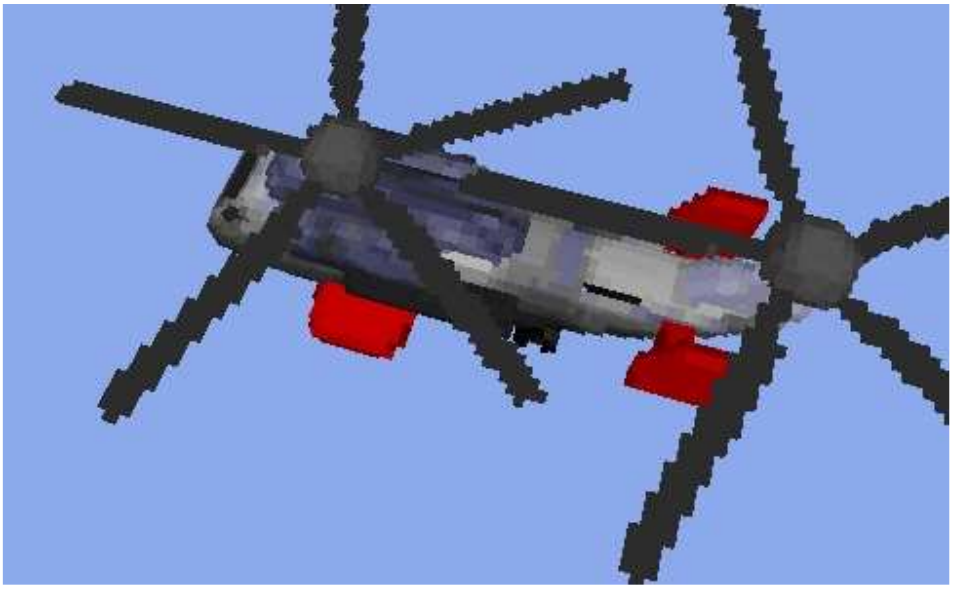

Figura 4.4 – Modelo volumétrico original com voxels representados como cubos.

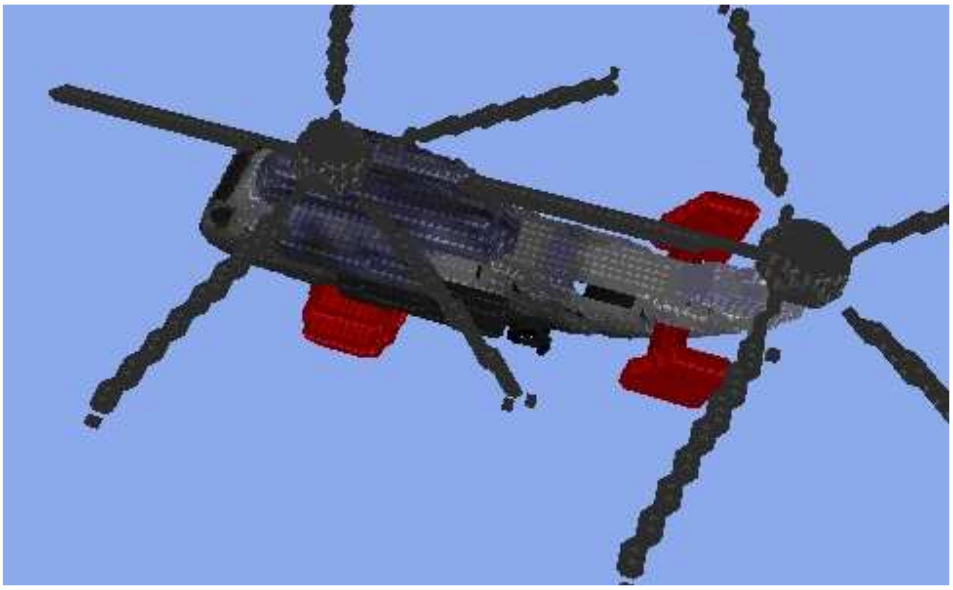

Figura 4.5 – Aplicação do Marching Cubes usando as informações de vértices no modelo da figura anterior. Alguns buracos, como os das hélices, ocorreram devido às PBRT2.

É interessante observar que o fato de as configurações de faces serem pré-determinadas em uma tabela diminui consideravelmente a complexidade do algoritmo, simplificando o processo de triangulação.

Os resultados obtidos tendem a usar menos vértices que o uso de redimensionamento com reconstrução linear. A maioria dos triângulos de uma região tem ângulos múltiplos de 45º entre si. Isso ocorre porque todos os vértices são gerados exatamente no centro das arestas da região cúbica. O principal problema das malhas geradas por este método são a falta de superfícies pontiagudas, fazendo com que a geometria fique mais suave do que deveria.

### 4.1.2 – *Extended Marching Cubes*

*Extended Marching Cubes* (EMC) (KOBBELT et al, 2001) é um algoritmo que estende o *Marching Cubes* apresentado no SIGGRAPH 2001 por Kobbelt et al conseguindo superfícies que preservam mais detalhes que no método original. Ele analisa cada *voxel*, detectando se este faz parte da borda do objeto. Caso fizer, ele detecta o tipo de borda (de vértices ou aresta), as arestas

da região cúbica que receberão os vértices da malha final e a posição destes a partir de interpolação linear. Ele usa uma função quádrica de erro (conhecida como quadric error function ou QEF) a partir destes vértices e suas respectivas normais para encontrar um novo vértice interno à região. As faces são geradas a partir de leques de triângulos (triangle fan) usando este novo vértice como centro do leque. Com isso, consegue-se o efeito pontiagudo. O figura 4.6 mostra como o vértice central é encontrado em duas dimensões e, portanto, usando linha ao invés de triângulo:

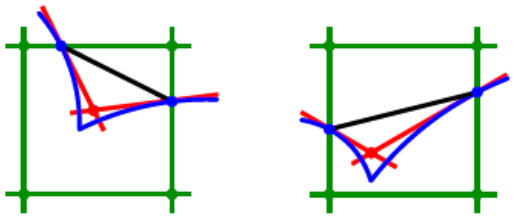

Figura 4.6 – Usando os pontos de interseção com a região e as normais nestes pontos, estima-se um vértice interno, mostrado a partir das linhas vermelhas.

Nos casos em que o *voxel* não fizer parte das bordas de um objeto, utiliza-se o *Marching Cubes* original para gerar a malha de sua região. A figura 4.7 mostra uma comparação de um modelo gerado a partir do *Marching Cubes* e outro usando o *Extended Marching Cubes*:

Como visto na figura 4.6, este método requer as informações das normais de pontos e sua distância à superfície original. Por este motivo, este método não foi pesquisado neste trabalho. Os resultados podem ser vistos na figura 4.7:

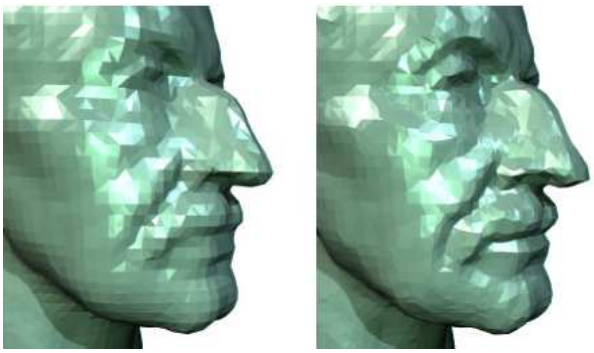

Figura 4.7 – Resultado de uma reconstrução da malha a partir de dados de um scanner 3D. Resultado com Marching Cubes à esquerda e EMC à direita, obtendo melhor detalhamento na boca, nos olhos e mais superfícies pontiagudas.

4.1.3 – *Dual Contouring*.

*Dual Contouring* (JU et al, 2002) é um algoritmo apresentado no SIGGRAPH 2002 por Ju et al que extrai as superfícies poligonais a partir de *octrees* e adapta o cálculo da QEF visto no EMC para calcular os vértices e faces independentemente se houver detalhes pontiagudos ou não. *Octree* é uma subdivisão espacial de um *voxel* em 8 mini regiões cúbicas. Estas por sua vez, podem ser subdivididas em 8 novamente se a resolução de detalhes estiver baixa.

A figura 4.8, a seguir, ilustra todas as etapas do algoritmo do *Dual Contouring*.

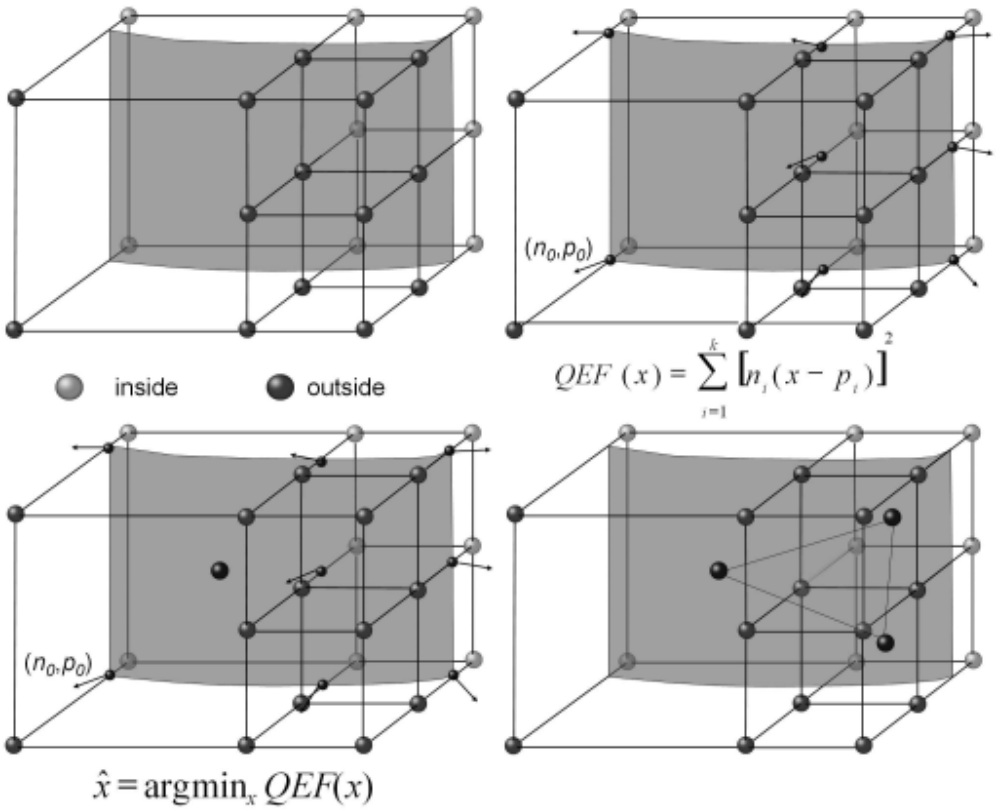


Figura 4.8 – Processo do Dual Contouring. Divide-se o volume em octrees, detectando quem está dentro e fora. Encontra-se a posição a partir do argumento mínimo da QEF. Depois, faz-se a triangulação.

Os resultados obtidos com *Dual Contouring* usam menos vértices e faces do que o que se obtém com EMC, devido a estrutura de *octree*. Além disso, essa estrutura consegue resolver os problemas de baixa resolução do tipo 1 descritos na seção 3.2.2. Apesar de o cálculo da QEF ser mais simples e a estrutura de dados mais leve do que nos métodos baseados no *Marching Cubes*, o cálculo da triangulação não é evitado, fazendo com que esse método seja bastante pesado computacionalmente.

## 4.2 – Reconstrução a partir de filtragem no domínio de frequência e espaço

Os métodos de reconstrução de superfícies poligonais a partir de dados volumétricos definidos de forma implícita são caros computacionalmente e suas adaptações para uso com modelos baseados em voxels, como os considerados neste trabalho, são muito complicadas ou até mesmo impossíveis. Por isso, existe uma linha de pesquisa alternativa em que se adaptam técnicas de realce de imagens (GONZALEZ e WOODS, 2000), originalmente usadas em imagens bidimensionais representadas em pixels, para suavizar uma malha gerada a partir de uma subdivisão discreta e uniforme.

Apesar de ser inicialmente concebida para ser aplicada em imagens digitais, existem pesquisas que visam usar a filtragem espacial como uma abordagem alternativa e mais simples de reconstruir uma malha a partir de voxels, se esta tiver uma subdivisão espacial uniforme, por exemplo, se for gerada a partir da regionalização dos *voxels* em cubos de tamanho fixo. Esta técnica é conhecida como reconstrução e, ao invés alterar cores de pixels, seu objetivo é modificar atributos dos vértices como posição, cores e normais

Enquanto o uso desta técnica para alterar cores se assemelha à filtragem espacial de *pixels*, mudando apenas a forma da detecção da vizinhança e a não necessidade do uso de uma matriz para aplicar a função de transferência, sua utilização para a finalidade de obter a posição dos vértices e direção das normais requer adaptações conceituais mais abruptas. No caso de uma suavização de malha, o conceito de frequência é diferente e, em um filtro passa baixa, o

comportamento da distância é invertido, sendo diretamente proporcional ao peso que esta representa no resultado final. Além disso, cada eixo é tratado separadamente.

O filtro de média para suavização de malhas (FORSTMANN, 2004) consiste em realizar o somatório deste mesmo atributo de seus vizinhos, ligados por arestas, e encontrar um novo valor dividindo-o pelo número de valores somados. A fórmula disto pode ser vista a seguir:

$$x' = x + \frac{\sum_{1}^{N}(x_i - x)}{N}$$

Nesta fórmula, *N* é o número de vizinhos do vértice atual, *x* é a posição antiga e *x'* é a nova posição do vértice no eixo que está sendo calculado.

Esta técnica é utilizada em algumas engine de voxels, como a HVox (2010) e, devido a sua complexidade linear, é possível usá-la em tempo real.

Taubin et al (TAUBIN, 1995) expandiu o estudo de suavização de malhas descobrindo que podem-se usar funções no somatório, que agem como filtros, para manipular o resultado final de acordo com os interesses do programador do software, da mesma forma que é feito com tratamento de imagens, obtendo uma fórmula que pode ser simplificada da seguinte forma:

$$x' = x + \lambda \sum_{1}^{N} h(\frac{x_i - x}{N})$$

Onde $\lambda$ é uma constante de escala e *h* é uma função de transferência. Taubin aplicou uma função polinomial representando um filtro passa banda, enquanto o HVox usa uma função linear onde $f(x) = x$ e $\lambda = 1$ gerando um filtro de média.

Maiores detalhes sobre a análise em frequência do método proposto por Taubin podem ser vistas no artigo original (TAUBIN, 1995).

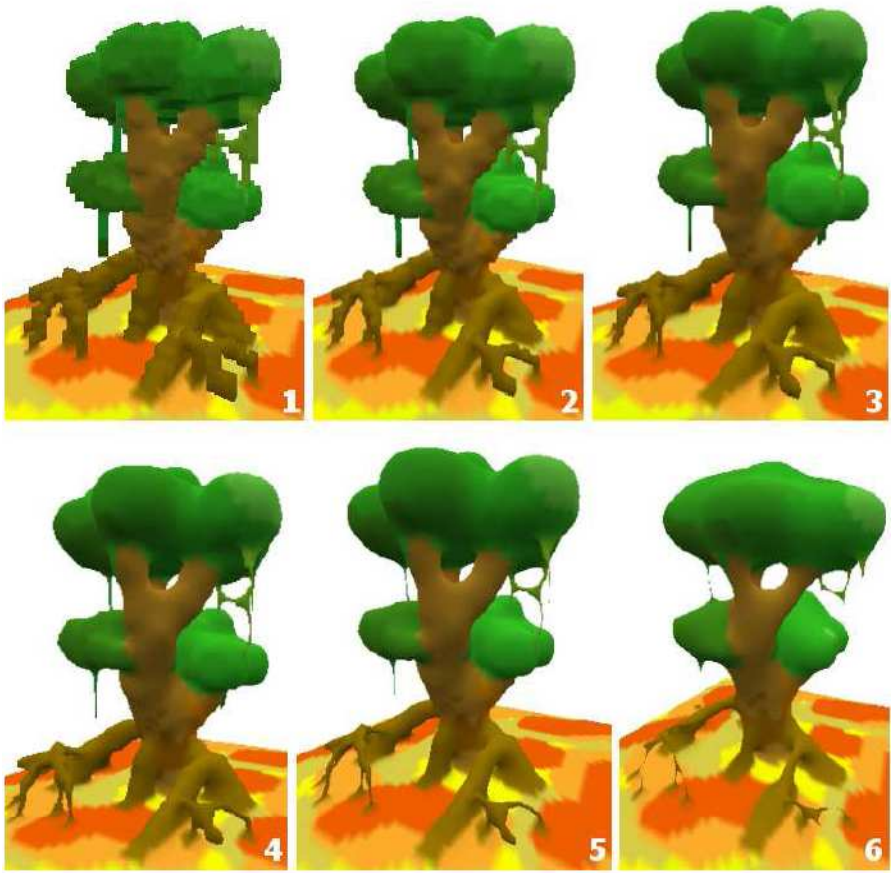


Figura 4.9 – Sucessivas aplicações de suavização de malha linear aplicada em um modelo volumétrico, de 1 a 6 vezes. A quantidade de superfícies pontiagudas reduz com cada aplicação.

Os resultados desta técnica, usando uma função linear para filtrar os dados (FORSTMANN, 2004) podem ser vistos na figura 4.9. Em ambos os casos, a função de suavização teve como objetivo eliminar altas frequências, pois essas foram consideradas simplesmente como ruídos. Por isso, a imagem perde superfícies pontiagudas a cada aplicação da técnica. Além disso, essa técnica gera vários vértices e faces desnecessários.

# Capítulo 5 – O método proposto

No capítulo 4, descrevemos os métodos de extração de superfícies poligonais mais conhecidos. O *Extended Marching Cubes* e o *Dual Contouring* apresentam resultados avançados tratando ruídos de alta e baixa frequências, usando estruturas de dados avançadas específicas e apresentando um alto nível de processamento com cálculos de QEF, sem resolver todos os problemas de baixa resolução de dados. É possível resolver o PBRT2 para o *Dual Contouring* adaptando a malha segundo a metodologia de Bischoff et al (BISCHOFF e KOBBELT, 2006), porém isto aumenta ainda mais o custo computacional do processo. Este custo é levado em consideração porque modelos volumétricos pequenos chegam facilmente aos milhões de *voxels* e, com isso, cada estrutura de dados pode usar grandes volumes de memória.

O desenvolvimento deste trabalho começou com uma adaptação do *Marching Cubes* que usava mais de 4 *gigabytes* de RAM em modelos com um milhão de *voxels* (100x100x100) e mesmo com otimizações, este uso de memória continuou na casa dos gigabytes e a complexidade do algoritmo e seu tempo de execução só aumentou. A técnica original foi descartada após a implementação de técnicas de redimensionamento, com funções de reconstrução mais avançadas do que a linear, com o uso de memória RAM e processamento muito menor, execução do algoritmo quase em tempo real mesmo e sem o uso de GPU, além de se obter resultados melhores do que com o método anterior.

O método proposto neste trabalho, como será explicado adiante, é uma técnica de reconstrução, onde a suavização de malha faz uma filtragem espacial na malha usando uma adaptação do filtro de Lanczos. Diferentemente de outros trabalhos, as altas frequências serão vistas como detalhes artísticos que devem ser capturados, ao invés de serem apenas ruídos.

## 5.1 – Etapas do método

Este método começa com a extração das faces da superfície do modelo original, ignorando as que estão dentro do volume. Depois, faz-se a reconstrução da malha, detectam-se e suavizam-se as normais das faces e as cores. No final, simplifica-se a superfície. A figura 5.1 mostra o fluxograma do método:

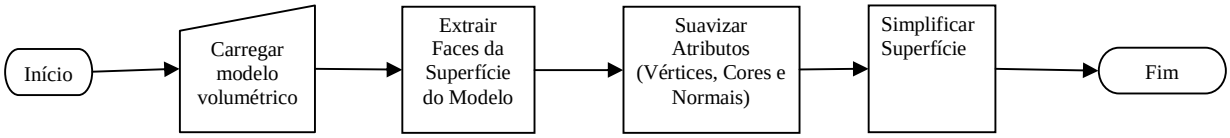


Figura 5.1 – Fluxograma do método.

## 5.2 – Extração das faces da superfície

O objetivo desta parte do método é a obtenção da superfície externa em faces de cubos. Para conseguir isso, temos que detectar a superfície do modelo volumétrico classificando cada *voxel* em 'fora do volume', 'superfície' e 'dentro do volume'. Depois, detectaremos os vértices, faces e finalmente efetuaremos a limpeza de vértices inúteis. A figura 5.2 mostra o fluxograma desta etapa.

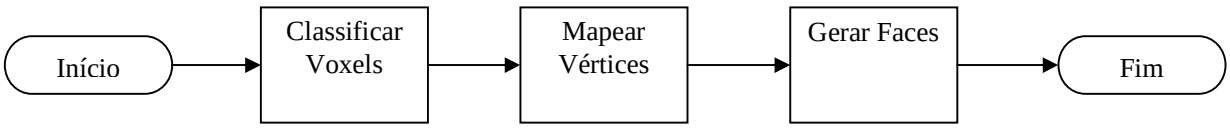


Figura 5.2 – Fluxograma da extração de faces volumétricas.

5.2.1 – Classificação dos voxels

O método de classificação de voxels inicia-se com a detecção dos *voxels* que estão dentro e fora do volume. Depois encontra-se quem está na superfície. O reconhecimento da superfície é fundamental em programas de modelagem de objetos gráficos com modelos volumétricos, porque apesar de o usuário não ser obrigado a pintar a parte interna do volume, a reconstrução de superfícies poligonais requer o conhecimento destes dados.

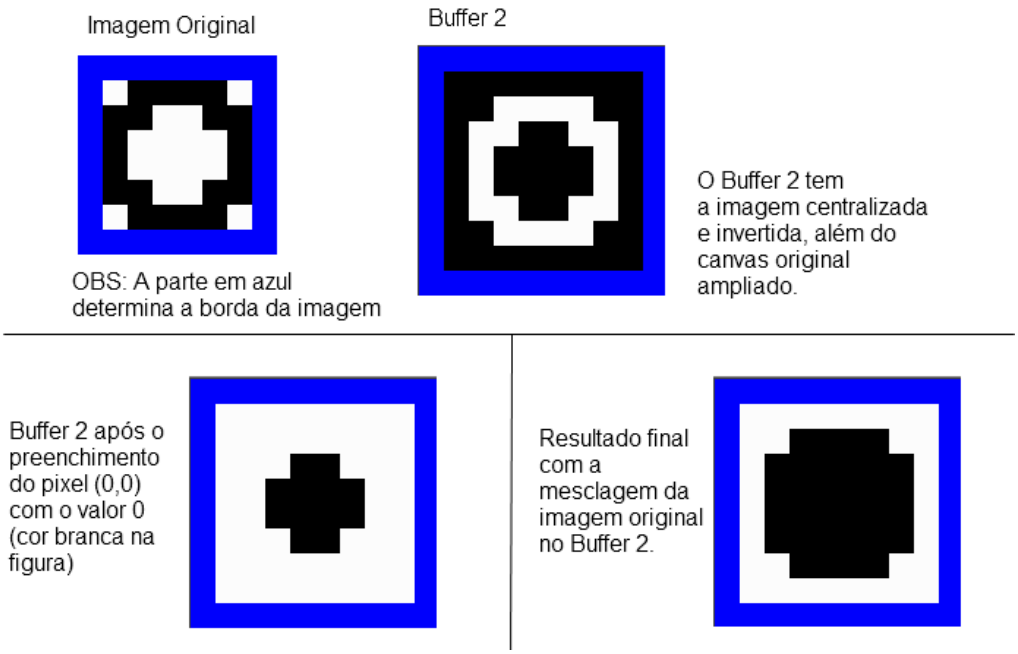


Figura 5.3 – Explicação sucinta do método de classificação de voxel interno e externo.

A detecção de quem está dentro e fora do volume é feita usando dois *buffers* volumétricos. Um deles é a imagem original e o outro tem as dimensões do modelo mais 2 para cada eixo. Por exemplo, se o modelo tiver dimensões 50x40x120, o tamanho do segundo *buffer* será de 52x42x122. Este *buffer* armazenará o modelo de forma que ele fique centralizado no seu espaço. Ele armazenará o valor 0 para as regiões do volume que foram pintadas pelo usuário e 1 para não pintadas. A figura 5.3 ilustra o processo em 2D, com a figura original e o segundo *buffer*.

Aplica-se o algoritmo de preenchimento no segundo *buffer* no ponto (0,0,0) com o valor 0. O algoritmo de preenchimento irá conferir todos os seis vizinhos de face do voxel que está sendo avaliado. Se o valor do vizinho for diferente do valor a ser preenchido, o vizinho recebe o valor 0 e adicionado em uma pilha. O algoritmo para quando a pilha ficar vazia.

Todos os voxels com o valor 1 da imagem original são copiados para o segundo *buffer*. Com isso, neste *buffer*, temos o valor 1 para os voxels internos e 0 para os externos.

Em seguida, detectam-se os voxels que estão na superfície. Para isso, basta conferir todos os voxels internos e, caso um vizinho do *voxel* que está sendo avaliado for externo, marcá-lo como superfície.

### 5.2.2 – Mapeamento de vértices

O mapeamento de vértices tem como objetivo gerar uma lista de vértices que poderá ser usada pelo modelo cúbico. Para isso, cria-se um buffer volumétrico para armazenar a identidade dos vértices e seu tamanho são as dimensões do modelo mais 1, para todos os eixos. Todos seus elementos são inicializados com -1. Além disso, inicializa-se um contador de vértices preenchido com o valor 0. Em seguida, percorre-se cada *voxel* do volume original. Se esse *voxel* estiver pintado, tentaremos preencher os vértices equivalentes no buffer de identidade dos vértices. O preenchimento do vértice ocorre se a ID dele a ser avaliada for -1. Neste caso, o contador de vértices é incrementado. Quando a varredura acabar, a lista de vértices é criada com a quantidade deles especificada no contador. Finalmente, percorre-se o *buffer* volumétrico de vértices e preencher a nova lista com os atributos x, y e z do elemento do *buffer*, se este for maior do que -1.

Este método vai gerar diversos vértices que não serão usados mais tarde, mas ainda assim, o desempenho de um método desses com uma limpeza é mais rápido do que uma avaliação mais complexa dos vértices que realmente serão usados.

### 5.2.3 – Geração de faces

O objetivo da geração de faces é obter uma lista de faces, ligando vértices em polígonos como triângulos ou quadriláteros. Para isso, cria-se um *buffer* de faces volumétrico com o mesmo tamanho do *buffer* criado para os vértices, apesar de que desta vez, cada elemento tem três valores: um para cada dimensão. Além disso, também é necessário criar um contador de faces inicializando-o com o valor 0. Em seguida, percorre-se cada face conferindo se apenas um dos dois voxels que essa face pertence faz parte do volume. Se a condição for verdadeira, preenche-se a face com o valor do contador de faces, e incrementa-se o contador.

Quando a varredura acaba, cria-se um vetor de faces com o tamanho do valor do contador multiplicado ao número de vértices por face. Além disso, cria-se um buffer de vértices usados e inicializam-se todos os elementos dele como inutilizados. O *buffer* de faces é percorrido e para cada elemento encontrado, preenchem-se faces formando um plano, onde os vértices seguem a ordem anti-horária. Em seguida, preenchem-se os valores das cores e armazena-se no *buffer* de vértices usados os vértices deste plano que estão sendo usados.

### 5.2.4 – Limpeza de vértices

O objetivo da limpeza de vértices é eliminar os vértices que o modelo não está utilizando. Criamos um novo *buffer* de vértices, onde cada elemento receberá sua nova identidade. Criamos um contador auxiliar e o inicializamos com valor 0. Percorre-se o *buffer* de vértices usados preenchendo o valor do elemento no novo *buffer* de identidade de vértice com o valor do contador. Caso o elemento seja utilizado, o contador deverá ser incrementado. Depois, aplicam-se as novas identidades nas faces e gera-se uma nova lista de vértices a partir das novas identidades.

5.3 – Processo de suavização de atributos

Este método converte a superfície do objeto original descrito por um conjunto de faces de cubos, reconstruindo os vértices resultantes com uma filtragem espacial usando um filtro de Lanczos. Depois, detectam-se as normais das faces, e usa-se essa informação para encontrar a normal de cada vértice. Finalmente, detectam-se as cores de cada vértice a partir das cores das faces conhecidas previamente na seção 5.2.3. A figura 5.4 ilustra o fluxograma desta etapa.

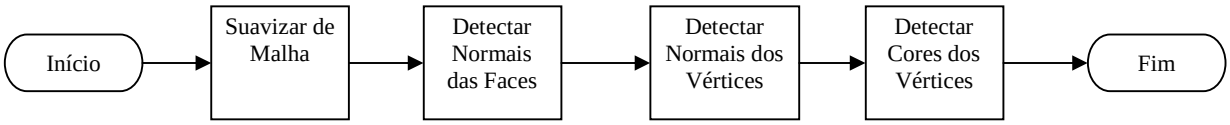


Figura 5.4 – Fluxograma da etapa de suavização de atributos.

A filtragem espacial utilizada neste trabalho é uma adaptação da técnica filtro de média do HVox. A malha obtida no procedimento anterior é formada por um conjunto de regiões planares, onde cada tem um têm 4 vértices formando um quadrado cujo tamanho é 1 em dois eixos e 0 na dimensão restante. Por este motivo, conclui-se que a distância entre dois vértices em um eixo qualquer será sempre um 1 ou 0 na primeira vez que ocorrer uma suavização, limitando a variedade de resultados que seriam obtidos se o filtro fosse aplicado nesta diferença. Por isso, esse trabalho se diferencia dos outros tratando a derivada normalizada da posição do vértice como uma medida de frequência, servindo de parâmetro para uma única aplicação da função de transferência. A fórmula para derivar a nova posição do vértice, mostrada anteriormente na seção 2.2.2, pode ser expressa da seguinte forma:

$$\frac{df}{dx} = K \frac{\sum_{i=0}^{N-1} (x_i - x)}{N}$$

Nesta fórmula, $N$ é o número de vizinhos, $x$ é a posição do vértice e $x_i$ é a posição de um vizinho e $K$ é uma constante de escala determinada pelo programador do software.

A função de transferência foi uma adaptação da função de Lanczos. A escolha deve-se ao fato de a função de Lanczos ser uma janela da função *sinc(x)* e que, apesar de ser um filtro de passa baixa, consegue manter um contraste significativo entre altas a baixas frequências.

A função *sinc(x)* utilizada como base para construção do filtro foi a original, não normalizada.

$$Sinc(u) = \frac{sen(u)}{u}$$

Onde u é a frequência. Com isso, a base da função de Lanczos é a fórmula abaixo:

$$Lanczos(u) = \alpha \frac{sen(u)sen(\frac{u}{\alpha})}{u^2}$$

Como foi visto no capítulo 3, α é um parâmetro que regula o efeito de suavização da função e a atenuação de frequências altas.

Para esta pesquisa, foram feitas algumas adaptações à fórmula original, que era aplicada em suavização de imagens baseadas em pixels gerando os pesos do filtro aplicado no pixel a ser alterado e seus vizinhos:

- Inverteu-se o sentido do aumento do resultado da função, pois seu propósito é reduzir o deslocamento da posição final do pixel à medida que a frequência aumenta. A direção original era útil ao processamento de pixels, onde se reduz o peso de pixels mais afastados do que está sendo processado.

- Aumentou-se a escala dos resultados gerados a partir da multiplicação por 2π. Com isso, e o uso do valor de α para 15, consegue-se gerar valores entre 0 e 1 quando a entrada está entre -1 e 1. A geração de valores com módulo acima de 1 geram ruídos na malha.

Com essas alterações, utilizou-se a seguinte fórmula de transferência (*h*):

$$h(u) = 2\pi \cdot (1 - \frac{15 \cdot sen(u) \cdot sen(\frac{u}{15})}{u^2})$$

No caso de *u* ser negativo, o resultado é multiplicado por -1.

O valor de *u* também pode ser manipulado, em especial, em escala. Durante a pesquisa, não foi detectado uma forma precisa de cálculo das frequências que são manipuladas durante a reconstrução. A partir de experimentos, conclui-se que um aumento de escala de 4/3 nas frequências a serem calculadas pela fórmula de Lanczos melhora os resultados, suavizando-os, sem que haja ruídos. Além disso, reaplicações da suavização mantiveram o comportamento de contração da malha previsto na seção 3.3.2.

A posição final do vértice é determinada pela seguinte fórmula:

$$x' = x + h(\frac{df}{dx})$$

5.3.1 – Suavização da malha

O processo de suavização da malha é feito a partir da reconstrução da posição dos vértices com o uso da função modificada de Lanczos mostrada na seção 5.3. O resultado deste processo pode ser melhorado se as faces forem quadriláteros ao invés de triângulos, porque a detecção de vizinhança funcionará corretamente. A redução do tamanho das faces pode excluir ligações entre arestas. A figura 5.5 ilustra o resultado deste procedimento.


Figura 5.5 – Resultado da suavização de malha de um modelo de tanque de guerra.

5.3.1.1 – Detecção de vizinhos.

A reconstrução requer o conhecimento dos vizinhos de cada vértice. Para descobri-los, usamos uma classe separada, que contém uma lista de ponteiros. O tamanho dela é a quantidade de vértices. Cada ponteiro contém um valor, que é a identidade do vizinho, e aponta para outro

ponteiro, gerando uma lista encadeada de ponteiros para cada vértice, que não pode repetir valores. Inicialmente, todos os elementos desta lista de ponteiros começam apontando para nulo.

O processo de detecção inicia-se com a varredura de todas as faces, onde cada vértice que pertence à face tentará adicionar os outros vértices da face, ou seja, seus vizinhos, na lista de ponteiros. A adição de um mesmo vizinho só ocorrerá uma única vez para um mesmo vértice.

5.3.1.2 – Reconstrução de vértices

Além da lista de vizinhos, este processo usa três *buffers* cujo tamanho de cada um é a quantidade de vértices da malha. Um deles é composto pelas posições originais dos vértices. O outro terá as novas posições e seus elementos são inicializados com o valor (0,0,0). O terceiro terá a quantidade de vizinhos que forem somados durante a operação.

Utiliza-se a lista de vizinhos para fazer o somatório da reconstrução em cada vértice. Este somatório é feito separadamente para cada eixo. Soma-se o valor do vértice no vetor de novas posições com a diferença da posição original do vértice com a posição original do vizinho. Com isso, incrementa-se a quantidade de vizinhos do vértice a cada vizinho somado.

A posição final dos vértices será a posição inicial deles somada com a aplicação da função de transferência na divisão do resultado do somatório pelo número de vizinhos.

$$x' = x + h\left(\frac{\sum_{i=0}^{N-1}(x_i - x)}{N}\right)$$

Neste caso, *x'* é a nova posição, *x* é a antiga, $x_i$ é a posição do vizinho, *N* é o número de vizinhos e a função *h*(u) é a função adaptada de Lanczos abaixo:

$$h(u) = 2\pi - \frac{30\pi \cdot sen(u) \cdot sen(\frac{u}{15})}{u^2}$$

5.3.2 – Detecção e suavização das normais.

Normal é um atributo presente em superfícies tridimensionais e indica a direção para qual a reflexão da luz atinge um valor máximo. Em modelos volumétricos, é comum designar uma normal para cada *voxel*. Porém, modelos baseados em bordos não têm um padrão estabelecido que indique se a normal é um atributo da face ou do vértice. A figura 5.6 mostra a diferença entre a normal por vértice e a normal por face (triângulo).

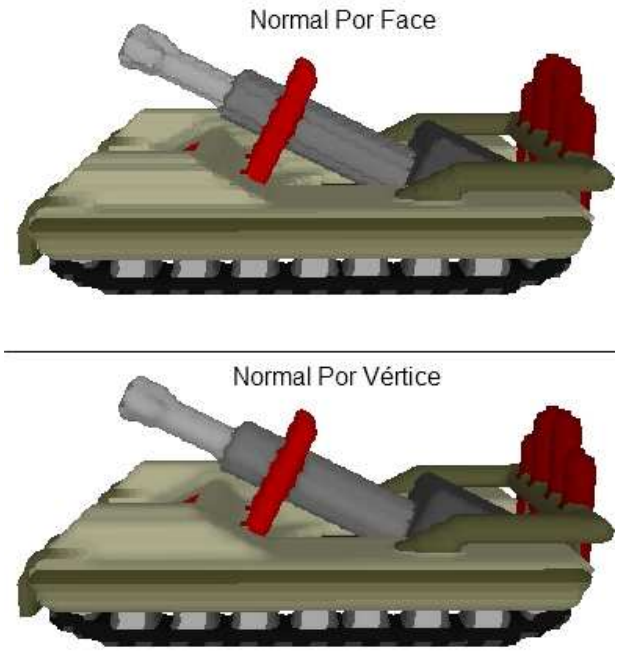


Figura 5.6 – Comparação entre normais por faces e por vértices.

Pode-se perceber que a normal por face gera um contraste maior, com superfícies mais ríspidas, enquanto a normal por vértice é mais suavizada. Esta suavização ocorre porque as normais por vértice são uma média das normais por face. Para chegar a este resultado, temos que encontrar as normais das faces e fazer a média delas para cada vértice.

5.3.2.1 – Detecção das normais das faces

A detecção das normais das faces (AZEVEDO e CONCI, 2003) é feita a partir do produto vetorial entre três vértices das faces no sentido anti-horário, seguindo a fórmula abaixo:

$$N_x = (P3_y - P2_y)\,(P1_z - P2_z) - (P1_y - P2_y)\,(P3_z - P2_z)$$
$$N_y = (P3_z - P2_z)\,(P1_x - P2_x) - (P1_z - P2_z)\,(P3_x - P2_x)$$
$$N_z = (P3_x - P2_x)\,(P1_y - P2_y) - (P1_x - P2_x)\,(P3_y - P2_y)$$

5.3.2.2 – Detecção das normais dos vértices

As normais dos vértices são encontradas a partir das normais das faces. Faz-se uma varredura nas faces, e soma-se a normal de cada uma em seus vértices. No final, divide-se o resultado de cada vértice pelo número de vezes em que ele foi somado.

5.3.3 – Suavização das cores

Assim como a normal, existem diferenças de atribuir uma cor a uma face e a um vértice. Por padrão, a placa de vídeo gera transições de cores ao rasterizar triângulos cujos vértices têm cores diferentes. Por isso, modelos que tenham cores por vértices são mais suaves do que aqueles que têm cores por face. O resultado pode ser visto na figura 5.7.

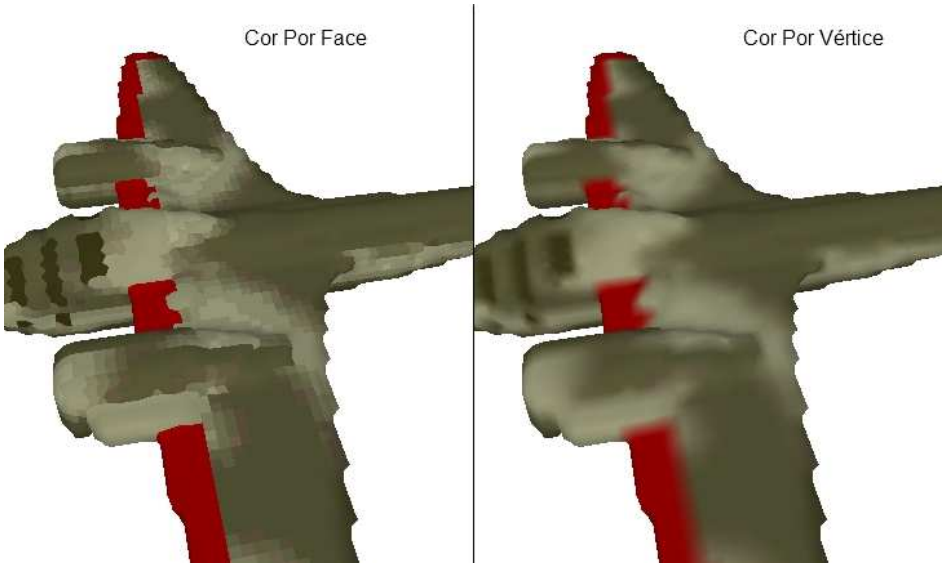


Figura 5.7 – Comparação entre cores por faces e por vértices em um modelo de bombardeiro.

Para gerar cores por vértices, é necessário usar as cores das faces. Faz-se uma varredura nas faces, e somam-se os valores nos componentes vermelho, verde e azul de cada uma em seus vértices. No final, divide-se o resultado de cada vértice pelo número de vezes em que ele foi somado para cada atributo.

## 5.4 – Simplificação da superfície

O método proposto neste trabalho realiza cálculos locais, apenas com base na vizinhança para gerar a malha. A falta de uma visão global tem como consequência a geração de muitos vértices e faces desnecessárias. Além disso, a base inicial do modelo é formada por planos oriundos de objetos cúbicos, que muitas vezes podem ter mais vértices do que a malha precisa em algum ponto. Por isso, há necessidade de simplificar a malha.

Neste trabalho consideraremos um vértice qualquer como inútil, se ele estiver em alguma das seguintes condições:

1) Sua coordenada é idêntica a de um outro vértice.
2) Não tem ligações por arestas.

3) O ângulo entre todas normais das faces vizinhas e o vértice é próximo de zero.

Devemos levar em consideração que o vértice não deve ser eliminado se as cores de todos os vértices vizinhos não forem equivalentes a sua, caso contrário, haverá perda de detalhes. Esta restrição de cor seria desnecessária se as cores dos vértices fossem armazenadas em um mapa de textura, o que aumentaria a eficácia do processo de simplificação de malhas, mas geração de coordenadas de mapas de textura não é objeto de pesquisa deste trabalho.

Devido à limpeza de vértices realizada na geração de superfícies cúbicas explicada na seção 5.2.4, a malha não tem vértices sem arestas, mas os outros dois casos podem ocorrer.

O método de simplificação de superfície aplicado neste trabalho tem como foco a condição três. O ângulo entre os vetores normais das faces vizinhas do vértice e a normal do próprio vértice deverá ser menor do que um ângulo pré-determinado pelo programador. Quanto menor for este ângulo, menor será a perda de qualidade e detalhes do modelo final, em contrapartida, serão eliminadas menos faces. Usando o valor 0 para o ângulo, não haverá perda de qualidade.

O procedimento adotado neste método é de classificar os vértices inúteis, mesclar os que estiverem ligados entre si, removê-los das faces, refazer a lista de vértices e recriar as faces. Esta técnica tem uma postura conservadora, devido à mesclagem de dois ou mais vértices inúteis que estejam ligados entre si, não removendo vértices que não estejam ligados com outros vértices desnecessários, mesmo que estes sejam inúteis. Desta forma, ainda que haja uma perda de eficácia, evita-se o processo de triangulação complexo, agilizando o método. A figura 5.8 mostra o fluxograma do método.

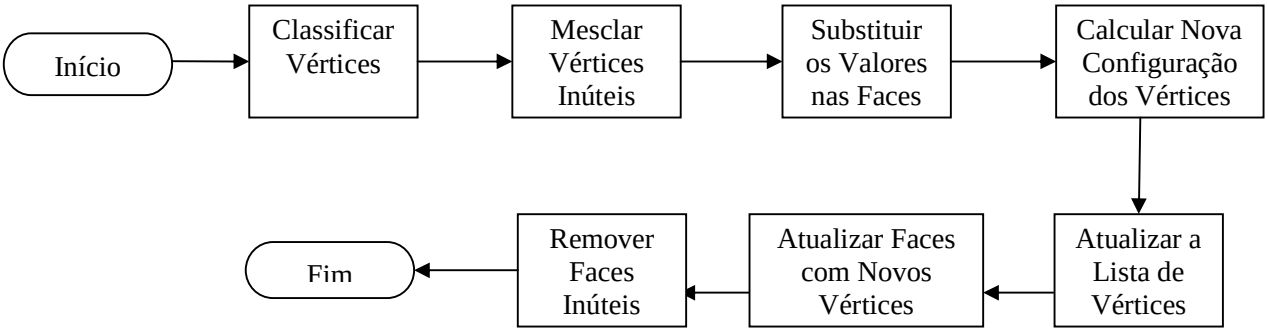


Figura 5.8 – Fluxograma da etapa de simplificação de superfície.

5.4.1 – Classificação de vértices inúteis

Vértices inúteis são aqueles cujos vizinhos têm cores idênticas e a diferença entre sua normal e as de todas as suas faces vizinhas está abaixo do valor de uma constante determinada pelo programador ou usuário.

O método de classificação de vértices inúteis requer a detecção de vértices vizinhos vista na seção 5.3.1.1 deste trabalho. Além disso, é necessário mapear as faces em que cada vértice pertence fazendo uma varredura em cada face e usando a mesma estrutura de dados da detecção de vértices vizinhos, porém a lista encadeada de ponteiros de cada vértice apontará para as faces em que eles pertencem.

Com os vértices vizinhos, podemos fazer a verificação da restrição de cor em cada vértice, conferindo se a cor de cada vizinho é igual a cor dele. Se for, usamos as faces vizinhas e suas normais, encontradas com o método explicado em 5.3.2.1, para conferir se o ângulo entre as normais é menor do que o ângulo gerado a partir da constante ε. Para isto, calcula-se o produto interno (STEINBRUCH e WINTERLE, 1990) entre os vetores normais descrito pela seguinte fórmula:

$$\varepsilon = \cos(\theta) >= (x_i \cdot x) + (y_i \cdot y) + (z_i \cdot z)$$

Caso o vértice for aprovado nas duas condições, deve-se marcá-lo em um *buffer* separado, que denominaremos de “vetor de transformação de vértices”, como um vértice com o potencial de ser descartado. Na implementação, utilizamos o valor -1 para indicá-lo como candidato a ser descartado e a identidade original do vértice para informar que ele não deve ser descartado. O uso de números inteiros neste vetor é necessário, pois eles serão utilizados como referência para remoção de faces com vértices descartados.

### 5.4.2 – Mesclagem de vértices inúteis

Vértices inúteis que estejam ligados a outros inúteis serão mesclados em um único vértice. A coordenada do novo vértice será a soma das coordenadas dos vértices antigos dividida pelo número de vértices mesclados. No vetor de transformação de vértices, os vértices mesclados terão como valor a identidade deste novo vértice.

O rastreamento de vértices vizinhos é feito com o uso de uma estrutura de dados de lista de vértices que inicializa com um vértice marcado como inútil. A lista varre seus vizinhos e tenta adicionar os que forem inúteis. Sua execução para quando a lista estiver vazia.

### 5.4.3 – Manipulação de faces e vértices

Esta etapa engloba a substituição dos valores nas faces até o final do fluxograma da figura 5.8. Os valores das faces são substituídos pelos seus valores dos vértices registrados no vetor de transformação de vértices.

No cálculo da nova configuração dos vértices, este vetor de transformação de vértices será modificado para que se possa adquirir a nova posição dos vértices excluindo os vértices que deverão ser removidos. Para a realização deste procedimento, inicializa-se um contador com

valor 0 e varre-se todos os elementos do vetor. Quando o elemento for igual ao seu índice, ele receberá o valor do contador que será incrementado, caso contrário, ele é marcado para exclusão.

Para a realização da atualização da lista de vértices, faz-se uma cópia de segurança das posições, cores e normais dos vértices atuais. O procedimento prossegue com uma cópia dos valores do backup nas posições novas indicadas pelo vetor de transformação de vértices.

Tendo os dados dos vértices atualizados, precisamos atualizar as faces e remover faces com vértices repetitivos. Esta parte inicia-se com uma nova varredura das faces onde elas receberão os novos valores dos vértices indicados pelo vetor de transformação de vértices. Em seguida, faz-se uma cópia das faces. Analisa-se cada face conferindo se há vértices repetidos. Caso haja, marcamos esta face para ser descartada com o valor -1 na cópia. No final, utilizamos a cópia das faces para transmitir os valores das faces não eliminadas para o vetor de faces da malha.

# Capítulo 6 – Metodologia de Testes

As pesquisas para este trabalho iniciaram-se com diversas tentativas de implementar o *Marching Cubes* em uma representação volumétrica de objetos gráficos sem a noção da superfície original, resolvendo os problemas de **baixa resolução de dados do tipo 1 e 2**.

A resolução desses problemas requer a obtenção de mais dados de referências, como a detecção das arestas e faces de cada região cúbica formada por um *voxel* que estão dentro ou fora do volume. Além de muito lento, o método precisava de várias estruturas de dados muito pesadas, usando *gigabytes* de memória *RAM*, mesmo para modelos de dimensões pequenas e devido às referências adicionais, não era possível evitar o processo complexo de triangulação dos vértices da malha. Os resultados tinham consistência limitada, facilmente perceptível a partir de uma rápida análise visual.

Os testes com as diversas variações de *Marching Cubes* e técnicas alternativas foram essenciais para mudança de direção do projeto ao longo do tempo e sua evolução. Após o abandono da ideia de utilizar o MC, testou-se a reconstrução com diversas fórmulas distintas e, após observações, conclui-se que uma adaptação do filtro de *Lanczos* apresentava resultados mais consistentes sem o excesso de suavidade de outros filtros. Com isso, a pesquisa expandiu-se com técnicas para suavizar cores, normais e simplificação da malha.

O que foi testado durante a pesquisa será explicado neste capítulo com seus respectivos resultados, mostrando alguns dos métodos descartados além do motivo de serem abandonados. No entanto, antes disso, esta seção descreve o software usado para de testes, métricas de testes e o hardware utilizado.

## 6.1 – Software de Testes: Voxel Section Editor III

Durante a pesquisa, utilizou-se o software Voxel Section Editor III (VXLSE3, 2010), também conhecido como VXLSE III. Inicialmente escrito por Will Sutton em meados de 2000, este editor de modelos em *voxels* gratuito foi criado para a edição unidades dos jogos Command & Conquer Tiberian Sun (CLASSICS, 2010) (figura 6.1) e Command & Conquer Red Alert 2 (figura 6.2), que usam um formato volumétrico proprietário criado pela Westwood Studios (2010) e atualmente pertence à Electronic Arts Los Angeles (2010). Sua versão inicial, chamada de Will's VXL Editor, recebeu algumas atualizações até o autor perder o código fonte. O programa foi reescrito como Voxel Section Editor II, e desta vez, seu código fonte foi distribuído, sendo modificado por diversos programadores ao longo dos anos. Em 2004, Stuart Carey refez a interface do programa, que passou a se chamar Voxel Section Editor III (figura 6.4).

Figura 6.1 – Command & Conquer Tiberian Sun.

Figura 6.2 – No Red Alert 2, as unidades usam mais vetores normais do que no Tiberian Sun.

No final de 2005, o autor desta pesquisa assumiu o desenvolvimento deste programa aperfeiçoando as funcionalidades existentes, desenvolvendo métodos para a detecção automatizada das normais dos *voxels* e reescrevendo a *engine* de renderização do programa. Um dos métodos de detecção das normais foi publicado como *short paper* no SIBGRAPI 2007, com a orientação do Prof. Esteban Clua, chamado "Finding Surface Normals From Voxels" (MUNIZ e CLUA, 2007).

O conhecimento prévio do programa e de sua estrutura e a existência de uma base sólida de usuários foram fatores determinantes para a escolha do Voxel Section Editor III nesta pesquisa. Este programa foi escrito na linguagem de programação Delphi. O ambiente de desenvolvimento somente compila programas em 32 bits, o que limita o uso de memória RAM dedicada ao programa para o máximo de 4gb. Acima disso, o programa retorna uma mensagem de falta de memória, mesmo que a máquina utilizada tenha mais de 4gb de RAM livres, rodando sistemas operacionais Windows de 64 bits. Essa limitação afetou gravemente o desenvolvimento

da pesquisa, em especial na técnica de *Marching Cubes*. No final da pesquisa descobriu-se um meio de reduzir bruscamente o uso de memória *RAM* da técnica de reconstrução, permitindo o uso de modelos bem maiores.

A arquitetura do Voxel Section Editor III é baseeada no padrão conhecido como Model View Controller (MVC, 2010), visto na figura 6.3. A interface é composta pelos formulários. Ela acessa os dados volumétricos e poligonais a partir das classes contidas no Modelo. A manipulação desses dados a partir do Controle, onde se encontra toda a implementação das etapas do método proposto neste trabalho.

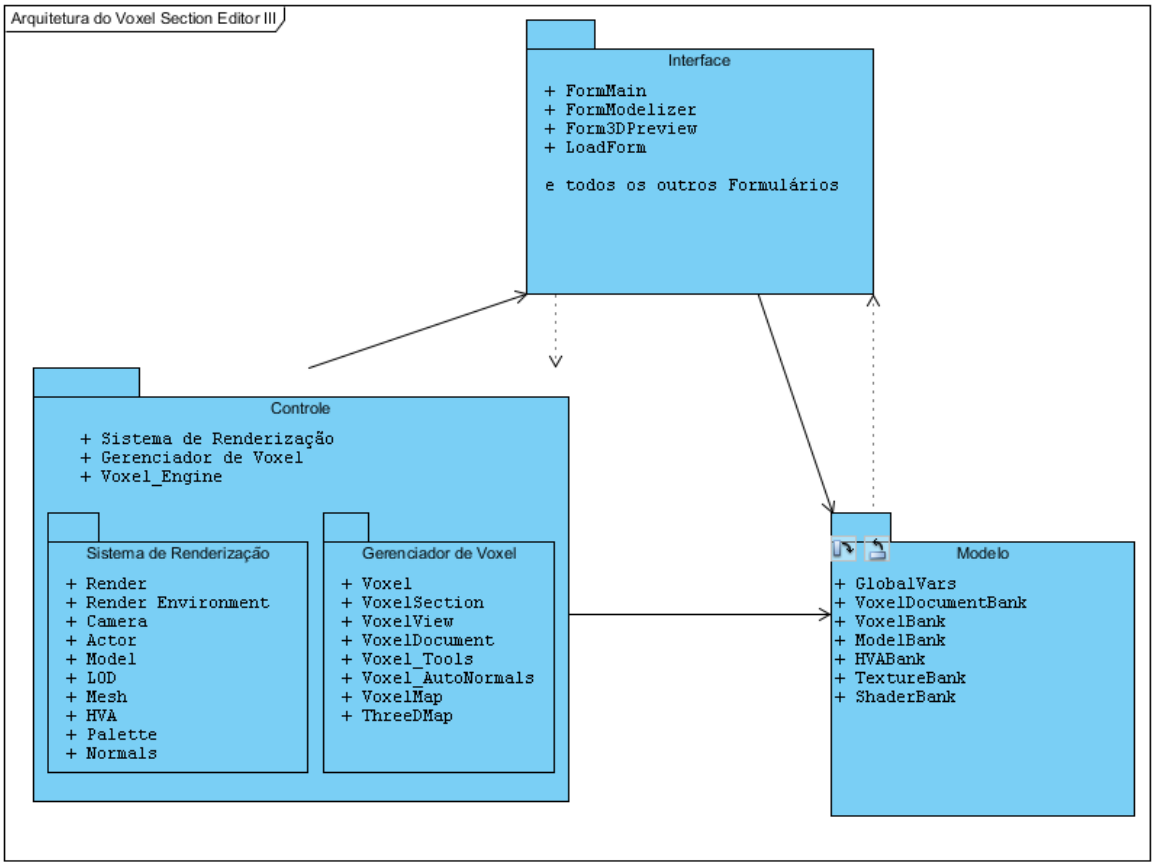


Figura 6.3 – Visão geral da arquitetura do Voxel Section Editor III.

A estrutura de dados dos volumes utilizados no programa tem limitações acentuadas de cor e normal, devido ao formato de volume da Westwood Studios que prioriza a redução de

espaço em disco em relação à qualidade do modelo. O jogo Tiberian Sun restringia seus modelos volumétricos ao uso de uma paleta de 256 cores e outra paleta de 36 direções de vetores normais. O jogo Red Alert 2 usa uma paleta de 244 direções de normais, mantendo a paleta de 256 cores. Cada *voxel* tem uma única cor e vetor normal e o programa mantém todos, inclusive os que não são utilizados, em um buffer tridimensional na memória. Esse *buffer* faz parte de uma seção que também contém as suas dimensões, um nome identificador, uma *bounding box* para cálculo de colisão além de uma matriz de transformação para cada quadro da animação. Um modelo volumétrico pode ter várias seções.

O Sistema de Renderização poder ter um ou mais ambientes de renderização, onde cada um renderiza uma cena distinta em regiões diferentes do programa. Cada ambiente contém um conjunto de câmeras e atores, além de configurações de luzes globais, além de outras configurações. Os atores contêm um ou mais modelos. Estes, por sua vez, contêm dados de hierarquia, animações e diferentes níveis de detalhes (level of detail). Cada um destes níveis contém um conjunto de malhas.

A estrutura de dados para malhas baseadas em bordo do programa foi criada durante a pesquisa. Cada modelo contém malhas, além de um nome identificador. Cada malha consiste de um conjunto de vértices, faces, cores, normais, entre outras propriedades. Algumas variáveis da malha determinam a quantidade de vértices por faces e se cores e normais estão atribuídas às faces ou vértices, além de determinar o método utilizado para renderização e o nível de transparência do conjunto. Propriedades de texturas e materiais não foram completadas durante o trabalho.

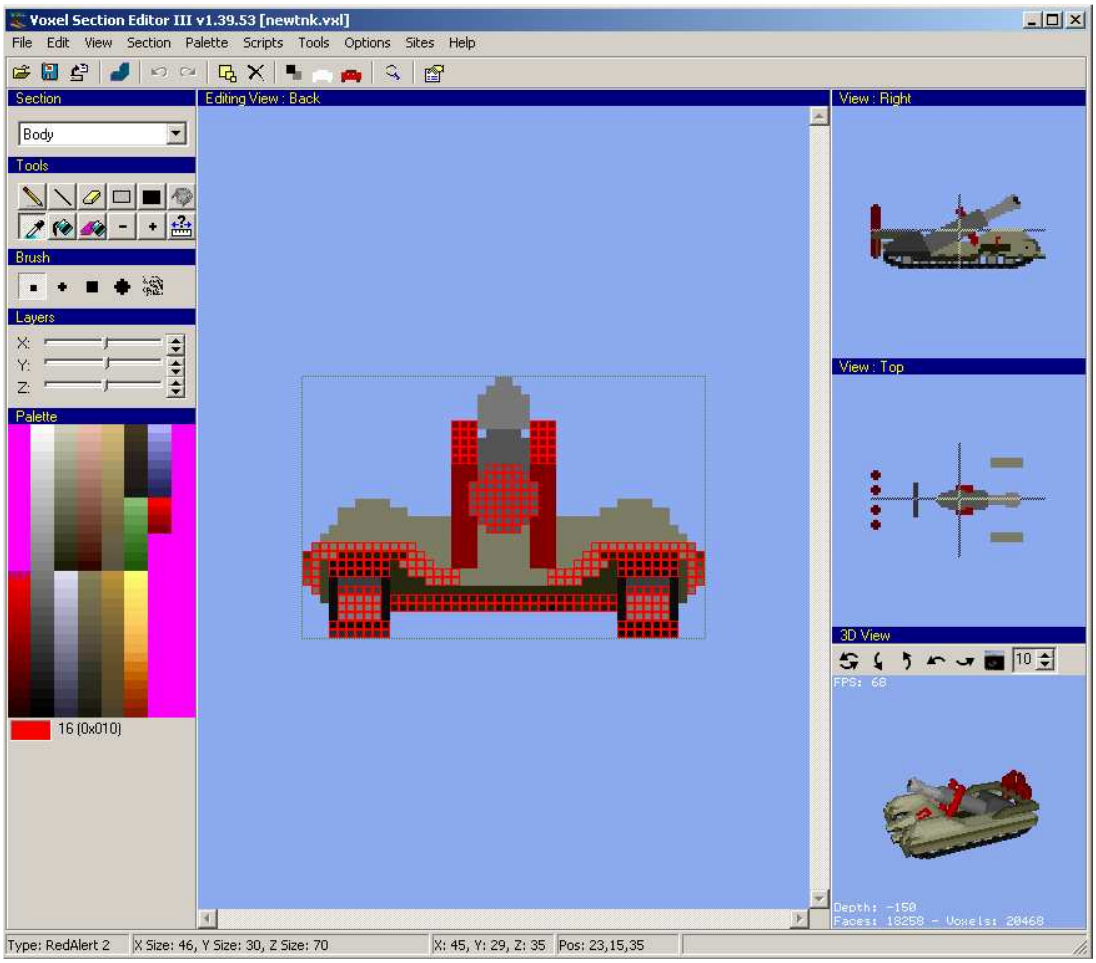


Figura 6.4 – Editor de modelos volumétricos Voxel Section Editor III.

Existem vários objetos volumétricos prontos, arquivos com a extensão *vxl*, que podem ser utilizados pelo Voxel Section Editor III. Além das unidades dos dois jogos que podem ser facilmente extraídas a partir de programas de terceiros gratuitos, como o XCC Utilities (XCC, 2010), há uma enorme base de usuários do programa que produzem novos modelos constantemente e os disponibilizam em sites como Project Perfect Mod (PPMVOXELS, 2010), YR Argentina (YRARG, 2010), Revora (REVORAVOXELS, 2010), ou são utilizados na criação de modificações para estes jogos, como ilustrado na figura 6.5. Os veículos dos jogos contêm em média entre 2 mil e 60 mil voxels usados e a parte interna do volume é parcialmente pintada, enquanto os que são produzidos pelos usuários podem passar de 200 mil voxels usados somente em suas superfícies. Além disso, é possível usar programas de modelagem em 3D como 3ds max

(3DSMAX, 2010), Maya (2010), Blender (2010), Cinema 4D (CINEMA4D, 2010) para gerar malhas no formato *.3ds* e convertê-las para arquivo vxl com uma ferramenta conhecida como 3ds2vxl (2010), que gera vetores normais corretos para cada *voxel*

Figura 6.5 – D-day (DDAY, 2010): modificação para Red Alert 2: Yuri's Revenge com gráficos totalmente feitos pelos fans da série de jogos Command & Conquer.

Apesar de o programa oferecer condições ideais para os testes realizados durante esta pesquisa, o Voxel Section Editor III não é o objetivo final deste trabalho. O principal motivo para isso é o fato de este software ter como foco a geração de modelos volumétricos para jogos antigos, sendo otimizado para eles. Além das limitações de cores e normais, o programa tem falhas conceituais que o inviabilizam como uma solução ideal de edição de imagem. VXLSE III não induz o usuário a gerar o objeto em seções intuitivamente, apesar de a palavra 'Section' fazer parte de seu nome. Além disso, o programa não disponibiliza meios simples para juntar as seções e animá-las. Outro obstáculo é a falta de materiais e meios de associá-los a objetos. Portanto, os conhecimentos adquiridos durante o desenvolvimento deste projeto final deverão ser usados em um futuro editor de imagens em que ambas as representações em *voxels* e bordos deverão ser usadas.

## 6.2 – Métricas e Hardware utilizados

Para validar o método proposto neste trabalho, deve-se medir sua eficácia e eficiência. Para um método ser eficaz, ele deve cumprir com aquilo que ele se propõe a fazer. Neste caso, analisa-se a qualidade do resultado. Um método eficiente deve ser eficaz, usando a menor quantidade de recursos possível, portanto, avalia-se tempo, memória e desempenho.

### 6.2.1 – Métricas de desempenho

Para análise do tempo de execução, incluímos o uso da função QueryPerformanceCounter da API do Windows de tempo de sistema em cada método no modo debug do programa, obtendo uma precisão inicial de bilionésimo de segundos (nanossegundos). Observou-se uma diferença de até 20% no tempo de execução no mesmo tipo de teste. Além disso, todas as operações levaram mais de 4 milissegundos na melhor máquina e com o menor modelo. Por estes dois motivos, usaremos uma precisão de milissegundos, além de calcular a média de pelo menos quatro execuções de uma mesma operação para publicar o desempenho dela neste trabalho.

O uso de memória RAM é determinado a partir do maior valor de memória usada pelo processo vxlse_iii.exe mostrado no gerenciador de tarefas (figura 6.6), durante a execução do método com a precisão de *kilobytes*. Durante os testes, o resultado variava em torno de 15%, devido a vazamentos de memória do programa.

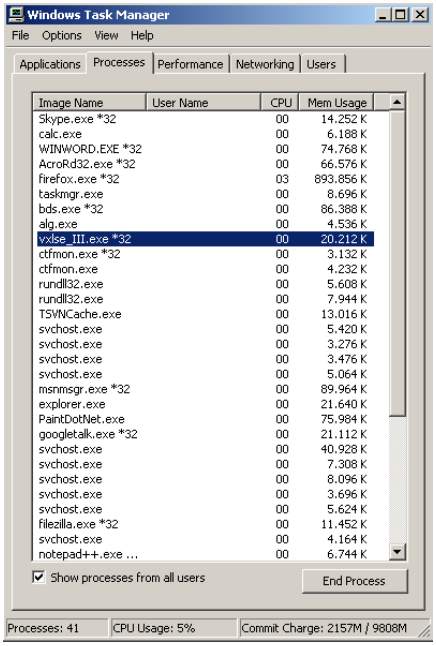


Figura 6.6 – Gerenciador de tarefas, usado para medir o uso de memória RAM nos testes.

Foram utilizados quatro computadores diferentes para avaliar o desempenho do algoritmo:

A máquina 1 é um desktop com um processador *quad core* (AMD Phenom x4) com 2.4ghz por core, 8 gb de memória RAM DDR2 800Mhz, placa de vídeo GeForce 7800 GTX com 512mb e sistema operacional Windows XP Professional SP2 64 bits. Neste sistema, temos os melhores desempenhos e uma maior disponibilidade de memória devido ao sistema operacional.

A máquina 2 é um desktop com um processador com apenas 1 core (AMD Athlon XP) de 2.0 ghz, 1gb de memória RAM DDR1 400Mhz, placa de vídeo GeForce 6600 GT com 256mb e sistema operacional Windows XP Professional SP2 32 bits. Este sistema tem a segunda melhor placa de vídeo, porém uma séria limitação de memória RAM que será seu principal objeto de teste.

A máquina 3 é um desktop com um processador de apenas 1 core (AMD Sempron 64) de 2.1 ghz, 2gb de memória RAM DDR2 667 Mhz, placa de vídeo onboard 8200M cuja memória RAM é compartilhada e sistema operacional Windows XP Professional SP2 de 32 bits. A placa

de vídeo deste sistema é fraca e o processador tem pouca memória cache, apesar de ter uma frequência maior que a máquina 2 e 4.

A máquina 4 é um laptop com processador *dual core* (AMD Turion 64 X2) de 1.8ghz por core, 4gb de memória RAM DDR2 667Mhz, placa de vídeo onboard GeForce Go6150 cuja memória é compartilhada e sistema operacional Windows Vista Home Premium Tablet Edition de 32 bits. Este *tablet* contém um sistema operacional muito pesado, diversos programas rodando ao fundo e um processador de frequência fraca em relação às outras máquinas. A grande quantidade de memória RAM compensa o fraco desempenho permitindo a manipulação de objetos gráficos maiores.

### 6.2.2 – Métricas de qualidade

A avaliação da qualidade será feita a partir de análise visual. Durante a pesquisa não foi possível concluir meios autônomos de determinar a qualidade da conversão de um modelo gráfico. Cogitou-se a possibilidade de medir os ruídos de um modelo, mas tal métrica seria inconsistente com a etapa de simplificação de malha, que tem por objetivo, reduzir a resolução de dados aumentando o nível de ruído, como explicado na seção 2.2.2.3. Além disso, ruídos podem ter valor artístico e não existe nenhum padrão que determine tal valor.

A etapa de simplificação da malha tem como métrica o número de faces da geometria antes e após o procedimento, além da análise visual para avaliar a consistência da figura resultante.

Os maioria dos modelos utilizados nos testes a seguir foram extraídos dos jogos antigos da série Command & Conquer (C&C, 2010) produzida pela Westwood Studios. Alguns outros foram gerados por usuários do site Project Perfect Mod (PPM, 2010). Apesar de muitos outros exemplos terem sido usados durante a pesquisa, os escolhidos a seguir têm características especiais que serão úteis para os testes de qualidade das etapas do método e técnicas alternativas:

* Nod Attack Buggy (figura 6.7): unidade criada pela Westwood Studios para o jogo Tiberian Sun. É um modelo de baixa quantidade de voxels pintados (2812) com superfícies ríspidas na traseira e arredondadas nos pneus e no canhão. A antena é uma PBRT1. É interessante para análise das etapas de suavização de atributos.

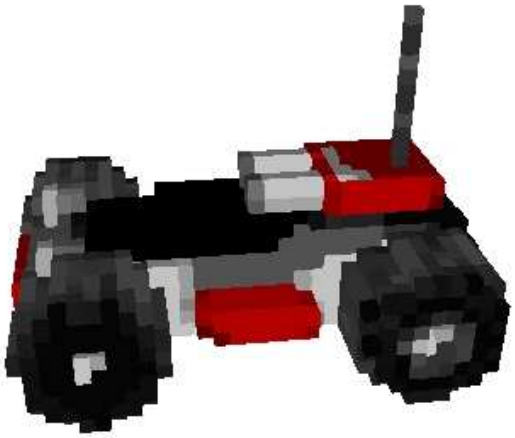

Figura 6.7 – Nod Attack Buggy da Westwood Studios representado com cubos.

* Nod Attack Buggy, Return of the Dawn (figura 6.8): unidade criada pelo usuário "Reaperrr", do fórum do Project Perfect Mod, para a modificação Return of the Dawn (ROTD, 2010) com 3087 voxels pintados. Modelo repleto de problemas de baixa resolução dos tipos 1 e 2. Interessante para análise de suavização de atributos, porém quase inútil para simplificação da malha.

Figura 6.8 – Nod Attack Buggy da modificação Return of the Dawn representado com cubos.

* Yuri Boomer (figura 6.9): submarino criado pela Westwood Studios para o jogo Red Alert 2: Yuri's Revenge com 9488 voxels pintados. Apresenta uma superfície muito curvilínea, sendo muito útil para suavização de malhas e quase inútil para simplificação. Apresenta voxels internos ao volume sendo útil para o teste de extração de cubos da superfície. Este modelo não tem problemas de baixa resolução de dados tão explícitas como os modelos anteriormente citados.

Figura 6.9 – Yuri Boomer representado com cubos.

* Demo Truck (figura 6.10): caminhão criado pela Westwood Studios para o jogo Red Alert 2 com 7893 voxels pintados. Modelo com superfície bastante ríspida, com curvas limitadas. Útil para testes de suavização com altas frequências e o melhor modelo para simplificar a malha.

Figura 6.10 – Demo Truck representado com cubos.

* Nighthawk (figura 6.11): helicóptero criado pela Westwood Studios para o jogo Red Alert 2 com 7303 voxels pintados em 3 seções. Modelo com muitas curvas, PBRT2 nas hélices e nas partes laterais do veículo. A geometria apresenta vários buracos na parte de baixo, perto da hélice menor e nas laterais, para transporte de passageiros e invasão de prédios no jogo. Útil para testes de suavização e de detecção de superfície.

Figura 6.11 – Nighthawk representado em cubos.

* Kirov Airship (figura 6.12): zepelin criado pela Westwood Studios com 35841 voxels pintados de um total de 554880. Modelo com superfícies muito curvilíneas no corpo central e ríspidas nos detalhes laterais. Além de ser útil para o teste de suavização e de extração da superfície, Kirov também é referência de desempenho em todos os métodos medindo o uso de recursos computacionais.

Figura 6.12 – Kirov Airship representado em cubos.

* Warhammer (6.13): super tanque criado pelo usuário "SaneDisruption" do fórum do Project Perfect Mod com 63150 voxels pintados de um total de mais de 270 mil. Contém várias superfícies retas com algumas curvas. Apesar de ser interessante para testar a suavização, é mais útil para avaliar simplificação de malha e o uso dos recursos computacionais do sistema.

Figura 6.13 – Warhammer representado em cubos.

# Capítulo 7 – Resultados

Neste trabalho mostraremos os resultados de cada etapa do método separadamente, com testes de qualidade e desempenho, seguido pela aplicação de técnicas alternativas e suas explicações. O desempenho de técnicas alternativas não será exibido de forma tão detalhada quanto aos dos algoritmos que fazem parte do método proposto.

## 7.1 – Extração de cubos da superfície

O objetivo desta etapa é gerar uma malha a partir de planos com o volume interno oco. As imagens da seção 6.2.2 mostram o resultado da extração de cubos na superfície externa para todos os modelos. A figura 7.1 mostra o Kirov Airship com e sem seu volume interno:

Figura 7.1 – A parte de cima é o Kirov com volume interno e, abaixo, sem o volume interno.

Modelos como o Nighthawk, que contém buracos em sua geometria, não comprometem o resultado deste método, como pode ser visto na figura 7.2:

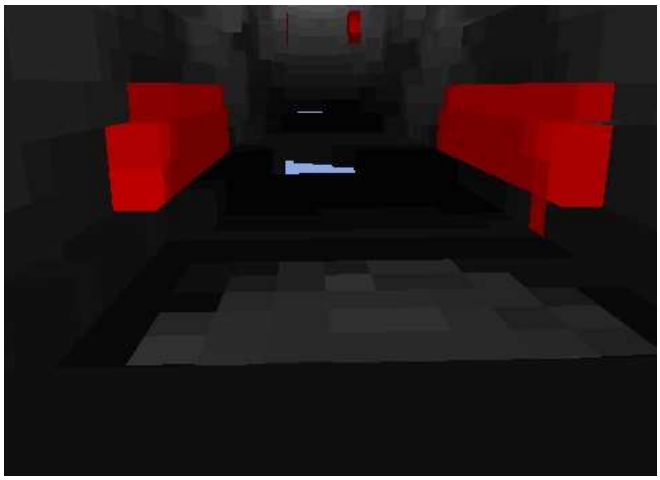

Figura 7.2 – A parte de interna do Nighthawk não foi comprometida pelos buracos no chão.

Durante a pesquisa, a remoção do volume interno do modelo foi fundamental para obter resultados mais confiáveis nas etapas seguintes, pois sem ela, a reconstrução realiza médias e cálculos com vértices que não deveriam existir, o que afeta o resultado final. Este problema pode ser visto com o Boomer na figura 7.3:

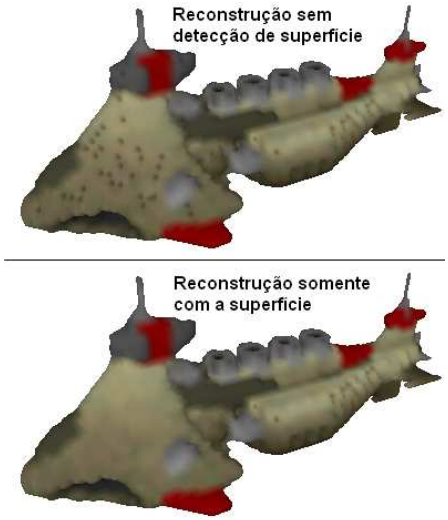


Figura 7.3 – Malha e normais do boomer afetados por vértices internos desnecessários.

No teste de desempenho, mediu-se o uso de memória entre a extração de cubos com detecção de superfície. A versão do Voxel Section Editor III utilizada nos testes usa 14120 kb de RAM ao iniciar, sem nenhum modelo aberto. A tabela a seguir mostra o uso de memória RAM em todos os modelos testados na máquina 1 antes de iniciar a etapa de extração de faces e antes de acabar a etapa (faltando apenas liberar a memória usada no processo).

Tabela 7.1 – Uso de memória RAM na etapa de extração da superfície em cubos.

| **Modelo** | **Antes da Operação** (em kb) | **Durante a Operação** (em kb) |
|---|---|---|
| Attack Buggy (Westwood) | 15032 | 16272 |
| Attack Buggy (Reaperrr) | 15032 | 16748 |
| Yuri Boomer | 18544 | 22344 |
| Demo Truck | 15396 | 17552 |
| Nighthawk | 17136 | 19628 |
| Kirov Airship | 21928 | 43964 |
| Warhammer Tank | 27148 | 41164 |

Considerando *n* como o número de voxels total no modelo, a complexidade dos dados deste método é O(*n*) e os resultados refletem isso. Os modelos Nighthawk e Warhammer relataram um menor uso de memória por terem mais seções e, portanto, usam menos memória por seção para gerar a superfície.

O teste de velocidade usou apenas o modelo do Warhammer já que os requisitos de memória RAM desta etapa são muito baixos e atendidos por todas as máquinas.

Tabela 7.2 – Tempo de execução da etapa de extração da superfície em cubos.

| **Máquina** | **Tempo de Execução** (em ms) |
|---|---|
| 1 | 393 |
| 2 | 521 |
| 3 | 499 |
| 4 | 575 |

Neste teste, o que contou foi a frequência do *core* do processador. A implementação não foi *multi-threaded* devido à dependência dos resultados recém-adquiridos durante a detecção de superfícies e vértices. Considerando *n* como o número de voxels do modelo, a complexidade de execução do algoritmo é de O(*n*). Na máquina 1, a etapa de geração do modelo cúbico sem a detecção de superfície foi executada em 164 ms. Essa proporção se repetiu nas outras máquinas e, portanto, é possível concluir que o processo de detecção de superfícies é mais pesado do que a geração da malha em si.

## 7.2 – Suavização da malha

O objetivo da suavização de malha é alterar as posições dos vértices para obter superfícies próximas daquilo que o usuário idealizou, obtendo um contraste entre regiões curvas e pontiagudas. A análise desta etapa será feita visualmente, já que não foi possível encontrar uma forma ideal de detectar quais regiões deverão ser curvilíneas ou pontiagudas.

No método proposto, a suavização da malha é feita com a aplicação de um filtro de Lanczos que modula o resultado do Filtro de Média, obtido pela distância original dos vizinhos em relação ao vértice. A figura 7.4 mostra a aplicação do filtro de Lanczos, com a detecção de normais por face para facilitar a visualização do resultado:

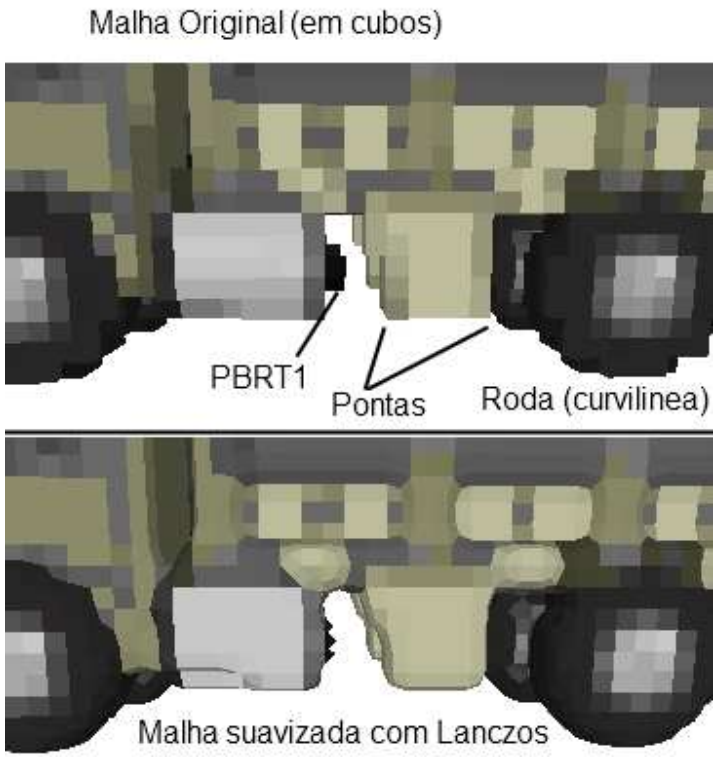


Figura 7.4 – Suavização de malha no Demo Truck.

O método proposto obtém resultados consistentes devido à coerência do Filtro de Média. A figura 7.5 mostra a aplicação do Filtro de Média sobre o mesmo modelo.

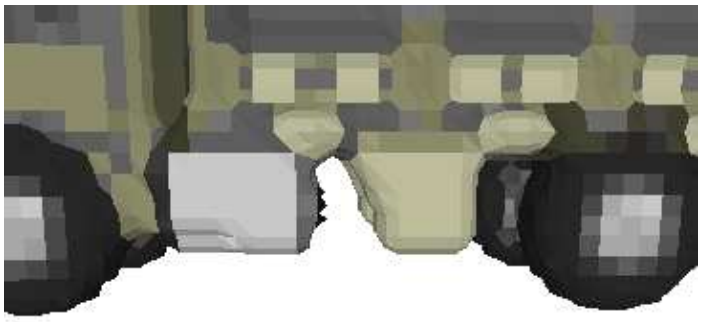

Figura 7.5 – Filtro de média na mesma região do Demo Truck.

A escolha do filtro de Lanczos foi feita após vários testes comparativos com outros filtros. Para cada filtro testado, encontrou-se, de forma experimental, uma constante de escala *K* para multiplicar o valor obtido inicialmente conforme descrito na fórmula do cálculo da derivada da posição do vértice descrita na seção 5.3. Os filtros foram aplicados em cada eixo

separadamente, por isso mostraremos apenas a fórmula para o eixo *x*, mas ela se aplica ao *y* e *z* também. Utilizamos as seguintes funções e suas respectivas constantes de escala:

- **Distância Linear (Filtro de Média)**:

$Distância(x) = x$

Constante de Escala: 1

- **Distância Quadrática:**

$Distância(x) = x^2$

Constante de Escala: 1,3

- **Distância Cúbica:**

$Distância(x) = x^3$

Constante de escala: 1,5

- **Distância Gaussiana:**

$$Distância\ (x) = \frac{x}{\sqrt{2\pi\sigma}} e^{-\frac{x^2}{2\sigma^2}}$$

Constante de Escala: 1,3

A fórmula do desvio padrão ($\sigma$) pode ser vista na seção 2.2.1.

- **Distância Sinc:**

$$Distância(x) = 2\pi(1 - \frac{sen(x)}{x})$$

Constante de Escala: 1,3

- **Distância Lanczos:**

$$Distância(x) = 2\pi(1 - \frac{15sen(x)sen(\frac{x}{15})}{x^2})$$

Constante de Escala: $\frac{4}{3}$

As funções de distância Sinc e Lanczos foram adaptadas para retornar valores entre 0 e 1, sendo próximo de 0 quando *x* tendesse a 0. Na implementação, evitou-se o cálculo dessas funções com *x* = 0 para que não ocorressem divisões por 0. A figura 7.6 mostra o resultado da aplicação destes filtros na unidade Demo Truck

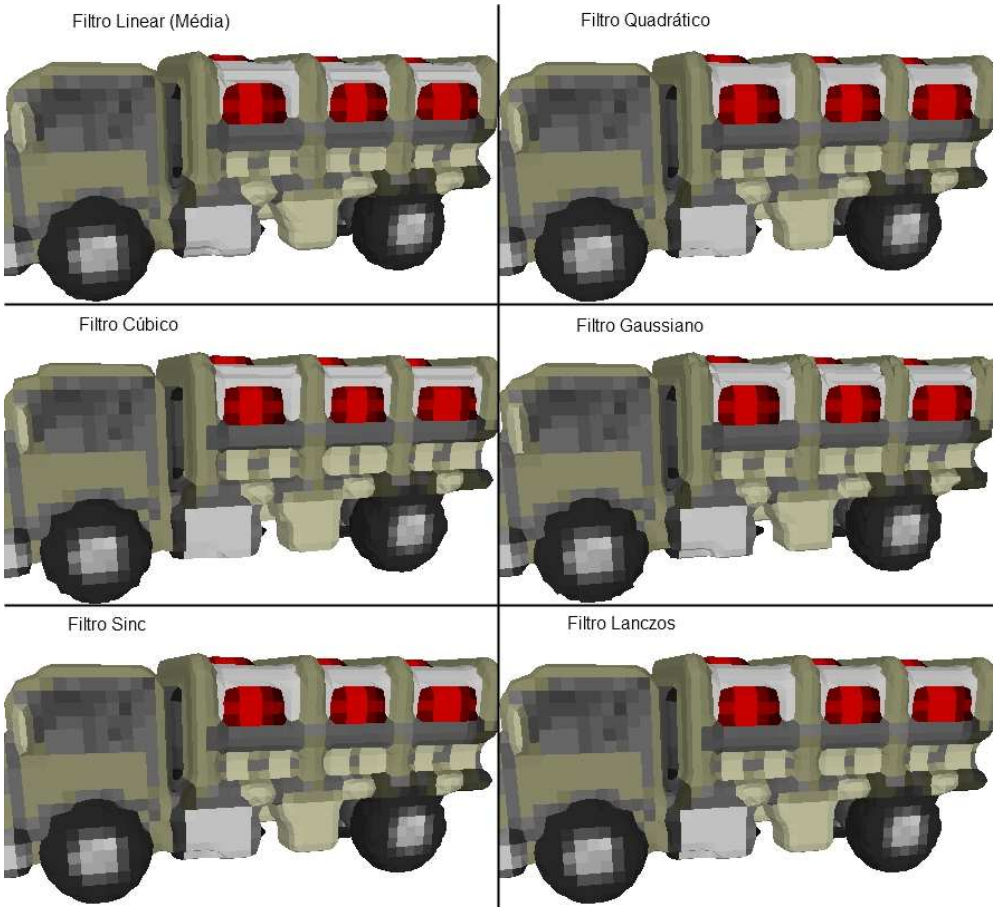


Figura 7.6 – Teste comparativo entre seis filtros distintos com o Demo Truck.

O filtro de média apresenta os resultados mais suavizados, removendo quase todas as regiões pontiagudas. O aumento do expoente de *x* reduz a suavidade do resultado. O filtro de Lanczos obteve resultados semelhantes ao Quadrático, porém é um pouco melhor em algumas curvas.

A constante de escala *K* de $\frac{4}{3}$ foi determinada experimentalmente. Concluiu-se que constantes baixas demais reduzem o efeito do filtro e valores altos criam ruídos, como pode ser visto na figura 7.7.


Figura 7.7 – Exemplos de variações de constante de escala K.

Os problemas de baixa resolução do tipo 2 não causam buracos no modelo, mas seu resultado foi insatisfatório. O resultado esperado para figura 7.8 eram paralelepípedos inclinados se cruzando, porém houve um encolhimento da malha nas ligações entre os voxels.

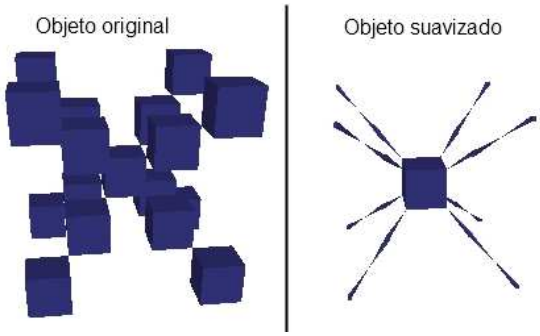


Figura 7.8 – Exemplo de suavização de malha com PBRT2 .

Apesar de insatisfatório, o resultado era esperado, já que não houve tratamento que aumentasse suficientemente a resolução do objeto para que se obtivessem os paralelepípedos esperados. Portanto, faltaram vértices e uma vizinhança ideal para criar o efeito desejável.

Em relação ao teste de desempenho, o uso de RAM foi semelhante à etapa anterior, tendo O(n) como complexidade dos dados de entrada. O teste de velocidade, usando o modelo do Warhammer, teve os seguintes valores:

Tabela 7.3 – Tempo de execução da etapa de suavização de malhas.

| **Máquina** | **Tempo de Execução** (em ms) |
|---|---|
| 1 | 276 |
| 2 | 345 |
| 3 | 338 |
| 4 | 401 |

Neste teste, o que contou novamente foi a frequência do *core* do processador. Considerando *n* como o número de *voxels* do modelo, a complexidade de execução do algoritmo é de O(*n*).

## 7.3 – Suavização das normais

O objetivo da suavização de normais é gerar a direção da orientação dos vértices para obter superfícies com aparência suavizada. O resultado desta etapa pode ser visto na figura 5.6.

Para certos modelos, o procedimento desta etapa não foi suficiente, o que levou ao desenvolvimento de um método de suavização adicional de normais semelhante ao de suavização de malha. No entanto, como o objetivo é eliminar as altas "frequências" das normais, utilizou-se simplesmente o Filtro de Média, em vez de Lanczos. A figura 7.9 mostra o resultado deste procedimento no Boomer:


Figura 7.9 – Operações com normais no Boomer.

O resultado deste procedimento adicional de suavização no lado direito da figura 7.9 apresenta pequenas melhorias em relação às normais obtidas anteriormente, porém esta diferença é pouco perceptível, sendo apenas notada em modelos com muitas curvas. Por este motivo, este procedimento foi considerado como opcional.

No teste de desempenho, o uso de RAM foi semelhante às etapas anteriores, em que a complexidade dos dados de entrada é O($n$). O teste de velocidade, usando o modelo do Warhammer, teve os seguintes valores:

Tabela 7.4 – Tempo de execução da etapa de suavização de normais.

| **Máquina** | **Tempo de Execução** (em ms) |
|---|---|
| 1 | 62 |
| 2 | 94 |
| 3 | 91 |
| 4 | 106 |

Neste teste, a frequência do *core* do processador foi o principal fator responsável pelo tempo de execução. Considerando *n* como o número de voxels do modelo, a complexidade de execução do algoritmo é de O(*n*).

## 7.4 – Suavização das cores

O objetivo da suavização de cores é detectar a cor de cada vértice para obter superfícies com aparência suavizada. O resultado desta etapa pode ser visto na figura 5.7 e na 7.10.

O uso de cores por vértice já tem um resultado suave demais, dispensando a necessidade de suavizações adicionais.


Figura 7.10 – Operações com cores no Boomer.

No teste de desempenho, o uso de RAM foi semelhante às etapas anteriores, em que a complexidade dos dados de entrada é O(*n*). O teste de velocidade, usando o modelo do Warhammer, teve os seguintes valores:

Tabela 7.5 – Tempo de execução da etapa de suavização de cores.

| **Máquina** | **Tempo de Execução** (em ms) |
|---|---|
| 1 | 129 |
| 2 | 169 |
| 3 | 161 |
| 4 | 199 |

Neste teste, a frequência do *core* do processador foi o principal fator responsável pelo tempo de execução. Considerando *n* como o número de voxels do modelo, a complexidade de execução do algoritmo é de O(*n*).

## 7.5 – Simplificação da malha

O objetivo da simplificação da malha é remover vértices e faces inúteis da malha sem que isso afete o formato do objeto. Para avaliar a eficácia desta etapa, serão verificadas a quantidade de faces removidas, além de eventuais mudanças na forma original do objeto.

A técnica de simplificação de malha proposta neste trabalho usa um ângulo máximo $\theta$ para o ângulo da normal do vértice que está sendo avaliado em relação às normais das faces vizinhas para classificar vértices inúteis. A tabela a seguir mostra o efeito da simplificação da malha através da contagem de triângulos usando diversos valores de $\theta$ e modelos diferentes:

Tabela 7.6 – Total de faces sem simplificação e com diferentes valores de θ para cada modelo.

| Modelos/ θ | Total | 0 | 10 | 20 | 30 | 40 | 50 | 60 | 70 |
|---|---|---|---|---|---|---|---|---|---|
| Attack Buggy (W) | 5600 | 5402 | 5386 | 5204 | 5118 | 5008 | **4932** | **4860** | **4852** |
| Attack Buggy (R) | 13184 | 13106 | 13100 | 12968 | 12806 | 12602 | 12436 | 12246 | 12120 |
| Yuri Boomer | 22172 | 22112 | 22030 | 21754 | 21628 | 21506 | 21410 | 21372 | 21356 |
| Demo Truck | 13624 | 12624 | 12430 | 11554 | 11154 | 10726 | **10588** | **10464** | **10430** |
| Nighthawk | 26948 | 26006 | 25582 | 23622 | 22958 | **22332** | **21484** | - | - |
| Kirov Airship | 52620 | 50528 | 50178 | 48378 | 46858 | 45916 | 45122 | **44728** | **44608** |
| Warhammer Tank | 185552 | 175190 | 174928 | 172924 | 172426 | 172088 | 171776 | 171456 | 171336 |

Os valores em laranja indicam distorção das cores no modelo. Esta distorção poderia ser evitada se a malha fosse simplificada após receber coordenadas de textura. Os valores em vermelho indicam a distorção crítica do formato geométrico do modelo, comprometendo sua qualidade. Os valores não preenchidos indicam que o objeto ficou irreconhecível ou distorcido de forma que não reflita mais seu conceito original.

Um dos motivos da baixa eficácia da remoção de faces inúteis é a restrição de cor, pois a técnica não mesclará dois vértices inúteis de cores distintas. Se removermos esta restrição, temos a seguinte tabela:

Tabela 7.7 – Total de faces sem simplificação e com diferentes valores de θ para cada modelo sem a restrição de cor.

| Modelos/ ε | Total | 0 | 5 | 10 | 20 | 30 | 40 | 50 | 60 |
|---|---|---|---|---|---|---|---|---|---|
| Attack Buggy (W) | 5600 | 5108 | 4932 | 4808 | **3260** | **2122** | - | - | - |
| Attack Buggy (R) | 13184 | 12054 | 11948 | 11786 | 10462 | 8626 | **7018** | - | - |
| Yuri Boomer | 22172 | 20556 | 20130 | 19862 | **15866** | **10964** | - | - | - |
| Demo Truck | 13624 | 11430 | 11132 | 10958 | 7602 | **4958** | **2318** | - | - |
| Nighthawk | 26948 | 24106 | 23338 | 22830 | 16472 | **13124** | **10210** | - | - |
| Kirov Airship | 52620 | 48906 | 47364 | 46320 | **31872** | - | - | - | - |
| Warhammer Tank | 185552 | 149342 | 146462 | 143162 | 107948 | **88046** | **68708** | - | - |

Além dos valores em laranja e vermelho, esta tabela tem elementos sem valores preenchidos nos quais a simplificação de malha fez com que o objeto ficasse irreconhecível.

Conclui-se que só será possível aproveitar todo o potencial da simplificação de malha se as coordenadas de textura de cada vértice forem detectadas, o que está fora do escopo deste trabalho.

O uso de RAM foi semelhante às etapas anteriores no teste de desempenho. A complexidade dos dados de entrada é O(*n*). O teste de velocidade, usando o modelo do Warhammer, teve os seguintes valores:

Tabela 7.8 – Tempo de execução da etapa de simplificação da malha com θ = 0.

| **Máquina** | **Tempo de Execução** (em ms) |
|---|---|
| 1 | 187 |
| 2 | 251 |
| 3 | 255 |
| 4 | 277 |

Neste teste, além da frequência do core do processador, outro fator importante para o desempenho foi a velocidade da memória RAM/cache, devido as várias cópias de *buffer* que foram feitas durante o procedimento. Considerando *n* como o número de *voxels* do modelo, a complexidade de execução do algoritmo é de O(*n*).

# Capítulo 8 – Conclusão

Neste trabalho evidenciaram-se as limitações dos processos de modelagem de objetos gráficos, a partir de *voxels* e superfícies em 3D, sendo sugerido que se unifique as duas formas de modelagem para que a geração de malhas e o preenchimento de seus detalhes sejam intuitivos para o usuário. A proposta deste trabalho constituiu-se de resolver o problema de extração de superfícies poligonais a partir de modelos volumétricos, já que as soluções confiáveis do processo inverso já são bastante difundidas. Esta resolução deve levar em conta desafios como evitar processos de triangulações complexos, lidar com problemas de baixa resolução de dados e com o contraste de frequências locais na malha.

Ao longo da pesquisa para este projeto, foram encontradas referências sobre algoritmos de extração de superfície poligonais a partir de isosuperfícies e do uso de técnicas de filtragem espacial em malhas. Estas foram consideradas mais apropriadas por seu benefício aliado ao baixo custo computacional. Elaborou-se assim a metodologia e as etapas a serem cumpridas que se constituíram de extrair uma malha com polígonos de tamanho uniforme, aplicar suavizações na malha, normais e cores e, finalmente, reduzir o número de vértices e faces inúteis. Assim, buscou-se mostrar, analisar e implementar os vários métodos pesquisados com algumas variações para encontrar o melhor resultado. Primeiramente, buscou-se uma forma de renderizar o modelo volumétrico em cubos sem incluir o volume interno e de forma que cada face fosse construída com seus vértices ordenados de forma anti-horária. A segunda parte foi a mais importante da pesquisa, onde se suaviza a malha. Foram testados diversos filtros e o de Lanczos se sobressaiu, obtendo um melhor contraste entre curvas e superfícies com muitos detalhes. Em seguida, detectou-se e suavizaram-se as normais e as cores enfatizando o uso de médias para reduzir o contraste destes atributos entre os vértices. Em quarto lugar, foi pesquisada uma forma de reduzir a malha gerada, removendo faces e vértices redundantes tomando o cuidado de manter a qualidade da malha gerada. Por fim, mostraram-se os resultados do trabalho, por meio de testes de qualidade e desempenho de cada etapa usando o Voxel Section Editor III com análises, além

de explicar a razão de suas técnicas alternativas que não terem sido incluídas no método proposto.

O objeto deste trabalho limitou-se a obter um método de transformar modelos baseados em voxels em geometrias baseada em bordo, em que cada vértice tem uma posição, normal e cor. Consegue-se conservar o máximo de detalhes possíveis e resolver os problemas de baixa resolução do tipo 1. O tempo de execução de todas as etapas é de ordem linear. Esta técnica não resolve os problemas de baixa resolução do tipo 2, porém os resultados obtidos para estes casos não são criticamente comprometidos por esta limitação, sendo visualmente aceitáveis. As técnicas de suavização de atributos sugeridos nesta pesquisa são feitas localmente, sem avaliar o contexto de cada região da superfície. Cada aplicação em uma superfície suavizada anteriormente consegue transmitir o contexto de uma vizinhança mais distante para os vértices da malha, porém a obtenção do contexto não foi objeto de pesquisa deste trabalho.

## 8.1 – Trabalhos Futuros

Outros trabalhos relacionados a processamento de imagens e seu uso com malhas poderão preencher as lacunas a respeito da explicação de frequências, explicações técnicas para filtros de passa baixa e de reconstrução e de formas de capturar o contexto de frequências de vizinhanças. Além disso, outras possíveis repercussões futuras são os estudos para resolver o problema de baixa resolução do tipo 2 e para obter texturas de forma autônoma para diversos tipos de *shaders*. Espera-se ter contribuído para o desenvolvimento de uma forma mais intuitiva de modelagem de objetos gráficos tridimensionais, revolucionando os futuros programas de síntese de imagens e vídeos.

# BIBLIOGRAFIA

(OPENGL, 2010) OpenGL – The Industry Standard for High Performance Graphics. Site oficial do OpenGL: uma API para desenvolvimento de aplicações gráficas e ambientes 3D. Disponível em: <http://www.opengl.org/>. Acesso em 27 de março de 2010.

(DIRECTX, 2010) Microsoft DirectX. Site oficial do DirectX: uma API para desenvolvimento de aplicações gráficas e ambientes 3D. Disponível em: <http://www.microsoft.com/games/en-US/aboutGFW/pages/directx.aspx>. Acesso em 27 de março de 2010.

(LORENSEN e CLINE, 1987) LORENSEN, W. E., CLINE, H. E. "Marching Cubes: A High Resolution 3D Surface Construction Algorithm"; Computer Graphics; Siggraph '87 Conference Proceedings; Jul. 27-31, 1987; vol. 21, No. 4; ACM Siggraph; pp. 163-169.

(LEVOY, 1988) LEVOY, M. "Display of Surfaces from Volume Data"; IEEE Computer Graphics and Applications; Mai. 1988; vol. 8, No. 3; pp. 29-37.

(VXLSE3, 2010) VOXEL SECTION EDITOR III. Ferramenta gratuita de modelagem e edição de modelos volumétricos subdivididos em pixels. Disponível em <http://www.ppmsite.com/index.php?go=vxlseinfo>. Acessso em 27 de março de 2010.

(VXLSE3SVN, 2010) VOXEL SECTION EDITOR III SVN, Código fonte do Voxel Section Editor III. Disponível em < http://svn.ppmsite.com/listing.php?repname=OS%20Voxel%20Tools&path=%2F&sc=0>. Acesso em 27 de março de 2010.

(AZEVEDO e CONCI, 2003) AZEVEDO, E., CONCI, A. “Computação Gráfica – Teoria e Prática”, 1ª. Ed., Elsevier Ltda, Rio de Janeiro, 2003.

(AUTOCAD, 2010) AUTOCAD, Ferramenta comercial de modelagem e design para desenho técnico e projetos arquitetônicos, desenvolvido pela Autodesk. Informações e versão trial disponível em <http://usa.autodesk.com/adsk/servlet/pc/index?id=13779270&siteID=123112>. Acesso em 27 de março de 2010.

(3DSMAX, 2010) 3ds Max, Ferramenta comercial de modelagem e edição de modelos com geometria tridimensional desenvolvido pela Autodesk. Informações e versão trial disponível em <http://usa.autodesk.com/adsk/servlet/index?id=5659302&siteID=123112>. Acesso em 27 de março de 2010.

(MAYA, 2010) Maya, Ferramenta comercial de modelagem e edição de modelos com geometria tridimensional desenvolvido pela Autodesk. Informações e versão trial disponível em <http://usa.autodesk.com/adsk/servlet/pc/index?id=13577897&siteID=123112>. Acesso em 27 de março de 2010.

(BLENDER, 2010) Blender, Ferramenta gratuita de código aberto de modelagem e edição de modelos com geometria tridimensional desenvolvido pelo Blender Foundation. Disponível em <http://www.blender.org/>. Acesso em 27 de março de 2010.

(PHOTOSHOP, 2010) Adobe Photoshop, Ferramenta comercial de edição de imagens bidimensionais desenvolvido pela Adobe. Informações e versão trial disponível em <http://www.adobe.com/products/photoshop/family/?promoid=BPDEK>. Acesso em 27 de março de 2010.

(GIMP, 2010) GNU Image Manipulation Program, Ferramenta gratuita de código aberto de modelagem e edição de imagens bidimensionais desenvolvido pelo The GIMP Team. Disponível em <http://www.gimp.org/>. Acesso em 27 de março de 2010.

(MÖLLER et al, 2008) MÖLLER, T. A., HAINES, E., HOFFMAN, N. "Real Time Rendering"; 3ª Ed; A. K. Peters Ltd, ISBN 978-1-56881-424-7, 2008, 2008.

(HECKBERT, 1989) HECKBERT, P. “Fundamentals of Texture Mapping and Image Warping”, Paul Heckbert, Master's Thesis, UCB/CSD 89/516, CS Division, U.C. Berkeley, Jun 1989

(GONZALES e WOODS, 2000) GONZALEZ, R. C., WOODS, R. E. "Processamento de Imagens Digitais", 1ª. Ed., Edgard Blücher Ltda, Sâo Paulo, 2000

(SHAPIRO e STOCKMAN, 2001) SHAPIRO, L. G., STOCKMAN, G. C: "Computer Vision", pp. 137, 150. Prentice Hall, 2001

(DUCHON, 1979) DUCHON, C. E. "Lanczos Filtering in One and Two Dimensions "; Journal of Applied Meteorology; Jan. 12-Mai 07, 1979; vol. 18, American Meteorological Society; pp. 1016-1022.

(CHEN e FANG, 1998) CHEN, H., FANG, S. “Fast voxelization of three-dimensional synthetic objects,” J. Graph. Tools, vol. 3(4), pp. 33–45, 1998.

(CHEN e FANG, 2000) FANG, S., CHEN, H., “Hardware accelerated voxelization,” Computer Graphics, vol. 24, pp. 200–0, 2000.

(EISEMANN e D’ECORET, 2006) EISEMANN, E., D´ECORET, X., “Fast scene voxelization and applications”, ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games. ACM, 2006, pp. 71–78. Disponível online em: <http://artis.imag.fr/Publications/2006/ED06>. Acesso em 27 de março de 2010.

(SILVA et al, 2009) SILVA, L. F.M.S., PAMPLONA, V. F., COMBA, J. L. D., "Legolizer: A Real-Time System for Modeling and Rendering LEGO Representations of Boundary Models," SIBGRAPI, pp.17-23, 2009 XXII Brazilian Symposium on Computer Graphics and Image Processing, 2009.

(KOBBELT et al, 2001) KOBBELT, L. P., BOTSCH, M., SCHWANECKE, U., SEIDEL, H. P., “Feature sensitive surface extraction from volume data” ; Computer Graphics; Siggraph 2001 Conference Proceedings; Ago. 2001; ACM Siggraph; pp. 57-66.

(JU et al, 2002) JU, T., LOSASSO, F., SCHAEFER, S., AND WARREN, J. "Dual contouring of hermite data. ACM Transactions on Graphics", 2002, vol. 21, No. 3, pp. 339-346.

(FORSTMANN, 2004) FORSTMANN, S., “3D-Reconstruction in Real Time”, Sven Forstmann Master’s Thesis, Institute for Operating and Dialog systems, Faculty for Computer Science, University of Karlsruhe, Mai 2004. Disponível online em <http://www.gpstraces.com/sven/main/publications.htm>. Acesso em 27 de março de 2010.

(HVOX, 2010) HVox, Engine para desenvolvimento de jogos que usa modelos volumétricos criada por Sven Forstmann. Disponível em <http://www.gpstraces.com/sven/HVox/hvox.news.html>. Acesso em 27 de março de 2010.

(TAUBIN, 1995) TAUBIN, G. “A Signal Processing Approach To Fair Surface Design”, Computer Graphics; Siggraph '95 Conference Proceedings; Set. 1995; ACM Siggraph; pp. 351-358.

(BISCHOFF e KOBBELT, 2006) BISCHOFF, S., KOBBELT, L., “Extracting Consistent and Manifold Interfaces from Multi-valued Volume Data Sets”, Bildverarbeitung für die Medizin 2006, Springer Berlin Heidelberg, pp. 281–285.

(STEINBRUCH e WINTERLE, 1990) STEINBRUCH, A.; WINTERLE, P.; "Introdução à álgebra linear.", São Paulo: Makron Books, 1990.

(CLASSICS, 2010) Command & Conquer Classic. Os jogos Command & Conquer Tiberian Sun, Red Alert 1 e Command & Conquer 95 estão disponíveis para download pelo site oficial da franquia Command & Conquer, da Electronic Arts. Disponível em: <http://www.commandandconquer.com/classic>. Acesso em 27 de março de 2010.

(WESTWOOD, 2010) FTP da Westwood.com. Tudo o que restou da Westwood Studios, comprada pela Electronic Arts em 2003 . Disponível em: <ftp://ftp.westwood.com/>. Acesso em 27 de março de 2010.

(EALA, 2010) EA Games. Descrição do estúdio Electronic Arts Los Angeles no site oficial da Electronic Arts. Disponível em: <http://aboutus.ea.com/loc_us.action#1>. Acesso em 27 de março de 2010.

(MUNIZ e CLUA, 2007) MUNIZ, C. E. V., CLUA, E. W. G. "Finding Surface Normals From Voxels"; SIBGRAPI 2007 - Workshop of Undergraduate Works, PI, pp. 49-52.

(MVC, 2010) Trygve /MVC. Página que explica a história da criação do padrão Model View Controller no Smalltalk. Disponível em: <http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html>. Acesso em 16 de abril de 2010.

(XCC, 2010) XCC Homepage. Página que desenvolve e hospeda utilitários usados para modificar jogos da série Command & Conquer. Disponível em: <http://xhp.xwis.net/>. Acesso em 27 de março de 2010.

(PPMVOXELS, 2010) Project Perfect Mod Voxels. Fórum onde membros da comunidade de modificação dos jogos Command & Conquer Tiberian Sun e Red Alert 2 publicam seus modelos volumétricos. Disponível em: <http://www.ppmsite.com/forum/index.php?f=111>. Acesso em 27 de março de 2010.

(YRARG, 2010) Yuri's Revenge: Argentina. Site onde membros da comunidade de modificação dos jogos Command & Conquer Tiberian Sun e Red Alert 2 publicam seus modelos volumétricos. Disponível em: <http://yrarg.cncguild.net/>. Acesso em 27 de março de 2010.

(REVORAVOXELS, 2010) Revora Voxel/SHP Downloads. Fórum onde membros da comunidade de modificação dos jogos Command & Conquer Tiberian Sun e Red Alert 2 publicam seus modelos volumétricos. Disponível em: <http://forums.revora.net/index.php?showforum=328>. Acesso em 27 de março de 2010.

(CINEMA4D, 2010) Cinema 4D, Ferramenta comercial de modelagem e edição de modelos com geometria tridimensional desenvolvido pela Maxon. Informações e versão demo disponíveis em <http://www.maxon.net/products/cinema-4d.html>. Acesso em 27 de março de 2010.

(3DS2VXL, 2010) 3ds2vxl. Ferramenta para converter objetos gráficos no formato .3ds para o modelos volumétricos no formato .vxl usado nos jogos Command & Conquer Tiberian Sun e Red Alert 2 . Disponível em: <http://get3ds2vxl.ppmsite.com/>. Acesso em 27 de março de 2010.

(DDAY, 2010) D-Day. Modificação que transforma o jogo Command & Conquer Red Alert 2 na segunda guerra mundial. Disponível em: <http://dday.migeater.net/>. Acesso em 27 de março de 2010.

(PPM, 2010) Project Perfect Mod. Comunidade que promove modificações dos jogos da franquia Command & Conquer. Disponível em: <http://www.ppmsite.com/>. Acesso em 27 de março de 2010.

(ROTD, 2010) Return of the Dawn. Modificação que traz o jogo Command & Conquer 95 para engine do Command & Conquer Tiberian Sun. Disponível em: <http://www.ppmsite.com/forum/index.php?f=116>. Acesso em 27 de março de 2010.