UMA ESTRUTURA DE DADOS PARA MALHAS PROGRESSIVAS

Margareth Catoia Varela

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

Aprovada por:

Prof. Cláudio Esperança, Ph.D.

Prof. Paulo Roma Cavalcanti, D.Sc.

Prof. Luiz Henrique de Figueiredo, D.Sc.

RIO DE JANEIRO, RJ - BRASIL

MARÇO DE 2004

VARELA, MARGARETH CATOIA

Uma estrutura de dados para malhas progressivas [Rio de Janeiro] 2004

VII, 83 p., 29,7 cm, (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 2004)

Tese – Universidade Federal do Rio de Janeiro, COPPE

1 – Simplificação

2 – Nível de detalhe

3 – Multi-resolução

4 – Transmissão progressiva

5 – Refinamento seletivo

I. COPPE/UFRJ II. Título (série)

Agradecimentos:

Em primeiro lugar, a Deus por tudo que nos provê e aos meus pais Arlindo e Diromar por estarem sempre presentes e tornarem possível a realização deste trabalho, através de incentivos e colaborações durante toda a minha formação acadêmica e profissional.

A todas as pessoas que colaboraram direta ou indiretamente para a elaboração desta tese: meus irmãos, por compreenderem e aceitarem os momentos que bloqueei o uso do nosso computador por longos períodos de trabalho, Rodrigo Zilli, por estar sempre ao meu lado tanto nos momentos felizes quanto nos momentos difíceis, e aos amigos conquistados no Laboratório de Computação Gráfica onde desenvolvi este trabalho. Em especial, agradeço aos amigos Antônio Lopes (o "Alopes") que, com sua boa vontade, contribuiu imensamente nos momentos de dúvida e foi o meu suporte e consultor para assuntos específicos e aleatórios, Mara Franklin, Bruno Tâmara e Walter Herrera que me acolheram com carinho e me ajudaram muito quando cheguei neste laboratório e ainda não tinha conhecimentos suficientes. E aos colegas Álvaro e Ricardo que colaboraram de alguma forma na fase final deste trabalho.

Ao professor e meu orientador Cláudio Esperança e a CAPES, instituição que contribuiu com a parte do apoio financeiro.

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

UMA ESTRUTURA DE DADOS PARA MALHAS PROGRESSIVAS

Margareth Catoia Varela

Março/2004

Orientador: Cláudio Esperança

Programa: Engenharia de Sistemas e Computação

Este trabalho aborda as principais técnicas para simplificação de superfícies e sua exibição em níveis de detalhe. Em particular, é proposto um esquema para armazenamento e reprodução progressiva de superfícies em níveis de detalhe capaz de se adaptar à posição do observador. O esquema foi implementado e testado sobre algumas superfícies com graus variados de complexidade tendo apresentado um desempenho bastante competitivo.

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

A DATA STRUCTURE FOR PROGRESSIVE MESHES

Margareth Catoia Varela

March/2004

Advisor: Cláudio Esperança

Department: Computing and Systems Engineering

This work outlines the main techniques for surface simplification and its visualization in levels of detail. In particular, it is considered a scheme for storage and progressive reconstruction of surfaces in view dependent levels of detail. This strategy was implemented and tested on surfaces with varied degrees of complexity and obtained a competitive performance.

Sumário

Capítulo 1

Introdução

As mais diversas áreas que pretendem representar um objeto por meio da computação gráfica se utilizam de descrições poligonais deste objeto, as quais vamos nos referir pelo termo *malhas*.

Uma malha é constituída de um conjunto de vértices e um conjunto de faces. Cada vértice representa as coordenadas (x, y, z) de um ponto no espaço e cada face define um polígono através da conexão de um subconjunto ordenado dos vértices. Embora os polígonos possam, em geral, ter um número qualquer de vértices, concentramo-nos, neste trabalho, ao caso especial de malhas triangulares, ou seja, aquelas em que seus polígonos têm exatamente 3 vértices. Pode-se facilmente obter malhas triangulares, a partir de outras não triangulares, uma vez que qualquer polígono convexo pode ser subdividido em triângulos [19].

Em geral, as malhas geradas pelas diversas aplicações que existem podem ser bastante complexas, chegando a ordem de um milhão de faces. Como exemplo, podemos citar aquelas geradas para representação de modelos pela Cartografia, sistemas de informação geográfica, simuladores de realidade virtual e aplicações militares, e ainda os modelos de objetos para visualização científica (ver Figura 1.1). A visualização de elementos do corpo humano, por exemplo, precisa ser feita com um alto grau de detalhe, implicando que a aquisição desses dados seja feita em alta resolução, gerando assim modelos grandes.

Malhas complexas são difíceis de renderizar, armazenar e transmitir. Uma proposta para agilizar a renderização é a substituição de uma malha complexa por um conjunto de aproximações em níveis de detalhe, conhecido como LOD (*Level-Of-Detail*).

Assim, uma malha detalhada é usada quando o objeto está próximo ao observador e uma mais grosseira quando o objeto estiver longe ou fora do volume de visão do observador.

Mais ainda, é muito importante que se possa ver um mesmo modelo, ou seja, a mesma malha em diferentes níveis de detalhe. Isto porque a visão em perspectiva faz com que elementos

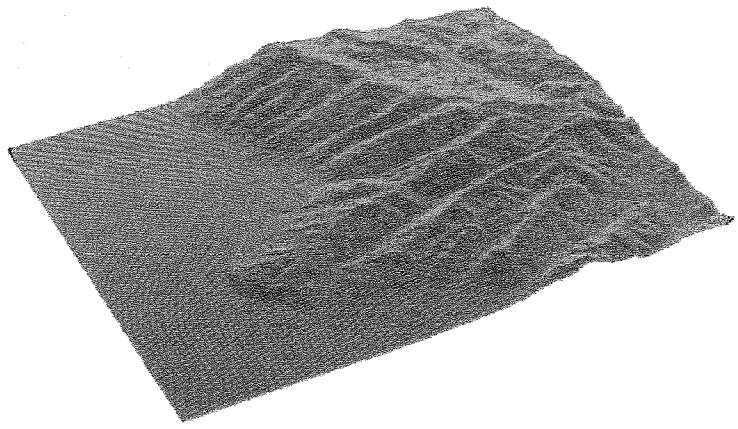


Figura 1.1: Modelo de terreno contendo 45.870 faces e 23.240 vértices.

mais distantes tenham projeções menores do que elementos mais próximos do observador. Ou seja, a parte visível do modelo que está mais próxima do observador deve ser visualizada em um nível de detalhe alto e a parte que está mais longe, ou não visível, deve estar em um nível de detalhe mais baixo.

O surgimento de aplicações 3D na internet e a conseqüente distribuição de informações tridimensionais pela rede tornaram necessária a rapidez na transmissão de elementos geométricos.

A representação LOD de um objeto 3D nos apresenta uma solução incompleta, uma vez que seu objetivo principal é agilizar o processo de visualização.

Enviar uma seqüencia de modelos em níveis de detalhe crescente não é uma boa solução uma vez que há muita redundância nas informações transmitidas. Além disso, para que se possa ter uma visualização de um desses modelos, deve-se esperar que todo ele seja transmitido.

Então, se o problema em questão é a transferência de dados geométricos de forma eficiente, uma metodologia a se analisar é a de transmissão progressiva. Se enviamos uma malha 3D através de uma linha de comunicação de baixa velocidade, a idéia é que na outra ponta possamos receber e desenhar, progressivamente, aproximações do modelo cada vez melhores.

Hoppe propôs uma solução real de transmissão progressiva para dados 3D no seu trabalho *Progressive Meshes* [26]. Inicialmente, é enviada uma aproximação grosseira e seqüencialmente vão sendo transmitidas informações de operações que devem ser aplicadas à malha inicial para que ela possa ser refinada progressivamente. Desta forma, a qualquer momento o receptor está apto a visualizar a malha atual que é localmente reconstruída a partir da execução dessas operações.

O objetivo principal deste trabalho é a simplificação de modelos detalhados de superfícies poligonais, que cabem em memória, em aproximações contendo uma quantidade menor de polígonos. O processo de simplificação pode ser feito de forma automática ou controlada pelo usuário. Além disso, desejamos efetuar o processo inverso, ou seja, o refinamento da aproximação grosseira, de forma que se possa recuperar o modelo original e/ou reconstruir aproximações do mesmo em diferentes níveis de detalhe. A estrutura definida pelo processo de simplificação nos permite essa representação em multi-resolução. Vamos abordar também a construção de uma hierarquia que permita esse refinamento de forma seletiva, ou seja, dependente do volume de visão do observador.

Capítulo 2

Preliminares e Trabalhos Relacionados

Neste capítulo vamos abordar conceitos básicos sobre representação de objetos por malhas e dar uma visão geral dos trabalhos que contribuíram para o desenvolvimento deste, além de alguns outros que fazem parte do estado da arte em simplificação de superfícies. Em seguida faremos algumas considerações sobre modelos em multi-resolução e visualização em níveis de detalhe.

2.1 Representação de sólidos

Muitas vezes a observação clara ou o estudo fácil de certas características de objetos do mundo real é inviável. Seja pelo tamanho do objeto analisado, que pode ser muito pequeno como no caso de moléculas ou muito grande como no caso de espaçonaves, ou pela falta de acesso direto a esse objeto, como partes internas do corpo humano, ou até mesmo por risco envolvido, quando deseja-se estudar a topografia de um vulcão, por exemplo. Assim, em geral, para facilitar a observação de um objeto real, utilizamos um modelo.

Um modelo é um objeto construído artificialmente de forma que a aparência geral, as dimensões relativas e outras características desejadas do objeto original possam estar ali representadas.

O primeiro modelo formal para representação de objetos sólidos foi proposto por Requicha [43] e define um sólido como um subconjunto de pontos do espaço Euclideano de dimensão 3 (E^3). Entretanto, algumas propriedades devem ser adicionadas a esta definição para que ela possa "capturar" a nossa noção real de um sólido. Essas propriedades foram descritas por Mäntylä [38]:

- Um sólido é um subconjunto fechado e limitado do E^3;

- Um sólido deve se manter invariante quando movido de uma posição do espaço para outra através de transformações rígidas, como, por exemplo, translações e rotações;
- Um sólido deve ser regular, ou seja, deve ser todo "preenchido" e não deve conter elementos isolados. Assim, um sólido regular deve conter todos os pontos da sua fronteira e do seu interior.

Uma vez construído um modelo formal de um objeto sólido, precisamos definir um esquema de representação para o mesmo.

2.1.1 Esquemas de representação

Antes de analisarmos os esquemas de representação, vamos fazer algumas definições.

Definição de Simplexos

Os simplexos caracterizam as formas poliédricas mais simples para uma dada dimensão k.

Seja um conjunto $V = \{v_0, v_1, v_2, ..., v_k\}$ de pontos tais que $v_i \in E^3$ onde $0 \leq i \leq k$. Define-se a célula gerada por V como a combinação convexa afim dos pontos pertencentes ao conjunto. Seja, então, a célula $\sigma =< v_0, v_1, v_2, ..., v_k >$. Se os k vetores $u_i = v_i - v_0$ $(0 < i \leq k)$ forem linearmente independentes, então a célula k é dita um k-simplexo (ou simplexo de dimensão k) em $\Re^n$.

Assim,

Simplexos de dimensão 0 ou 0-simplexos: são vértices (os pontos $v_0, v_1, ..., v_k$ do conjunto).

Simplexos de dimensão 1 ou 1-simplexos: são segmentos de reta.

Simplexos de dimensão 2 ou 2-simplexos: são triângulos.

Simplexos de dimensão 3 ou 3-simplexos: são tetraedros.

Um subsimplexo de um k-simplexo σ é definido por um subconjunto dos vértices de σ. No caso particular do $\Re^3$, os subsimplexos de dimensão 1 são chamados de arestas e os subsimplexos de dimensão 2 são chamados de faces.

Representações

Dependendo da dimensão topológica usada para modelar um objeto, podemos distinguir quatro tipos de representação de objetos tridimensionais (ver Figura 2.1):

1. Nuvem de pontos: o objeto é descrito somente pelos seus vértices.

2. Aramado (*wireframe*): o objeto é descrito por vértices e arestas.

3. Superfície ou borda: o objeto é descrito por vértices, arestas e faces.

4. Volumétrica: o objeto é descrito por vértices, arestas, faces e elementos de volume (*voxels*).

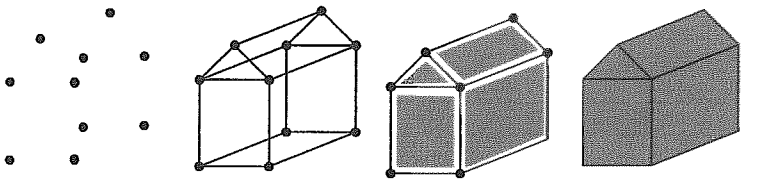


Figura 2.1: Exemplos de representação de um objeto: da esquerda para a direita, nuvem de pontos, aramado, borda, volumétrica.

Classificação dos esquemas de representação

De acordo com Mäntylä [38], os esquemas de representação de objetos são divididos em três categorias:

Modelos de Decomposição, que descrevem um sólido por uma coleção de objetos simples combinados por uma operação de "colagem". Um exemplo clássico de representação de modelos de decomposição é dado pela representação *Octree*[30].

Modelos Construtivos, que descrevem um objeto sólido através de operações de interseção, união e diferença entre elementos de um conjunto de objetos simples. A representação mais comum de um modelo construtivo é a representação CSG (*Construtive Solid Geometry*).

Modelos de Fronteira, que representam um dado objeto sólido através de elementos simples que, juntos, definem a fronteira ou borda desse objeto.

Neste contexto, uma malha é uma representação poligonal que recobre toda a borda ou contorno de um objeto, ou seja, é um modelo de fronteira.

2.1.2 Modelos de borda e os sólidos representados

Na representação por borda, um objeto sólido é inteiramente representado pelas faces que o contornam, ou seja, pelas suas faces de fronteira. Para que essa representação possa ser feita de forma correta, os modelos de representação por borda, ou modelos B-rep (*Boundary Representation*), consistem também de arestas e vértices e, ainda, das relações topológicas entre seus elementos.

Com relação às relações topológicas de um objeto sólido[38]:

- O bordo consiste de um conjunto de faces.
- Cada face tem a sua fronteira definida por um conjunto de arestas. Estas arestas de borda devem ser ordenadas para que possam formar uma curva fechada (um *loop* ou ciclo). A fim de se definir a região interna e a região externa de um objeto sólido, cada ciclo, ou lista de arestas vizinhas, tem que estar orientado. No nosso caso, utilizaremos a orientação ou sentido anti-horário.
- As arestas têm faces vizinhas que se interceptam na própria aresta.
- As arestas são limitadas por vértices vizinhos.
- A cada vértice corresponde um conjunto de arestas vizinhas incidentes.

A representação B-rep não faz restrição alguma quanto ao número de arestas que definem uma face. Contudo, vamos nos restringir a representar as faces de um objeto por triângulos, que são polígonos planos, determinando assim uma superfície poliédrica. Isto nos garante que as arestas, definidas pela interseção de faces, são linhas retas. Ainda com relação aos dados geométricos, os vértices são definidos pelas suas coordenadas (x, y, z). A restrição de utilizar somente faces triangulares não é significativa uma vez que qualquer polígono convexo pode ser triangulado. Além disso, atualmente a maioria dos dispositivos de *hardware* têm suporte direto à renderização desse tipo de primitiva.

Em nosso trabalho, os objetos a serem representados deverão ser objetos sólidos poliédricos válidos e sem buracos.

Um objeto sólido válido tem que ser uma 2-variedade orientável e fechada [38]. Uma variedade (com borda) é uma superfície tal que todos os seus pontos possuem uma vizinhança que é topologicamente equivalente a um disco (ou um semi-disco) aberto.

Mais formalmente, uma superfície é um subconjunto S do espaço euclideano E^3 que é localmente homeomorfo ao plano euclideano E^2. Ou seja, para cada ponto $p \in S$ existe uma

vizinhança esférica $B_\varepsilon^3(p) \subset E^3$ com centro em p, tal que o conjunto $B_\varepsilon^3(p) \subset E^3 \cap S$ é homeomorfo ao disco aberto unitário

$$B_1^2(0) = \{(x, y) \in E^2; x^2 + y^2 < 1\}. \tag{2.1}$$

Para um superfície S com borda, a definição acima é válida para os pontos interiores da superfície e para cada ponto $p \in S$ da borda existe uma vizinhança esférica $B_\varepsilon^3(p) \subset E^3$ com centro em p, tal que o conjunto $B_\varepsilon^3(p) \subset E^3 \cap S$ é homeomorfo ao semi-disco aberto unitário

$$B_1^2(0) = \{(x, y) \in E^2; x^2 + y^2 < 1 \text{ e } y > 0\}. \tag{2.2}$$

Isto significa que uma malha representando o contorno de um sólido poliédrico válido deve satisfazer as seguintes condições:

1. Cada aresta conecta exatamente dois vértices.
2. Cada aresta tem exatamente duas faces vizinhas (ou faces incidentes), exceto as arestas de borda que devem ter exatamente uma.
3. Cada vértice consiste de um ciclo fechado de faces, ou seja, é rodeado por um único ciclo de arestas e faces.

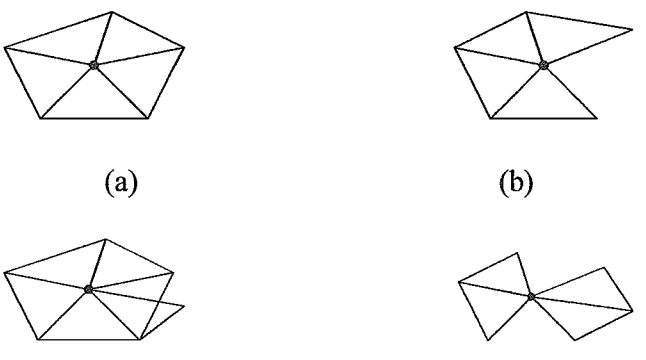


Figura 2.2: Possíveis vizinhanças de um vértice num modelo poligonal. (a) e (b) são variedades, ao contrário de (c) e (d).

A Figura 2.2 ilustra as vizinhanças de um vértice num modelo poligonal: (a) e (b) são representações válidas, enquanto (c) e (d) não.

Considere agora um sólido poliédrico, 2-variedade, orientável e fechado. Seja V o número de vértices, E o número de arestas, F o número de faces, L o número de ciclos de arestas,

H o número de buracos e S o número de cascas, onde uma casca é o número de superfícies conectadas. Então, a fórmula de Euler-Poincaré relaciona esses elementos da seguinte forma [38]:

$$V - E + F - (L - F) = 2 * (S - H), \tag{2.3}$$

ou

$$V - E + F - R = 2 * (S - H), \tag{2.4}$$

onde $R = L - F$ é denominado anel.

2.2 O problema de simplificação

O objetivo de simplificar uma superfície poligonal é produzir automaticamente modelos que aproximem o original tendo uma quantidade menor de polígonos.

Uma aplicação comum de simplificação é a redução da complexidade de modelos de densa amostragem, como os gerados a partir de equipamentos de aquisição de dados volumétricos e iso-superfícies extraídas por algoritmos como Marching Cubes [34].

Como o processo de simplificação se aplica diretamente a esses modelos grandes, é praticamente inviável que esse processo se dê sob supervisão do usuário [17], uma vez que essa manipulação manual se tornaria uma tarefa incômoda. Por isso, o processo de simplificação deve ser automático.

Como podemos, então, a partir de uma malha original, encontrar ou gerar uma nova malha que represente bem essa malha original, mas que seja mais simples, de mais fácil manipulação?

Para resolver esse problema, diversos trabalhos surgiram nos últimos anos, principalmente na década de 1990, quando as técnicas de aquisição de dados tridimensionais já tinham avançado bastante, gerando um volume de dados muito grande, porém a capacidade de manipulação dessas informações no computador ainda não tinha evoluído a tal ponto.

Esses trabalhos podem ser classificados segundo diversos aspectos:

- Caracterização da malha de entrada (malha original): se é uma variedade ou não.
- Caracterização da malha de saída: Sua topologia é preservada em relação à malha original? Seus vértices são um subconjunto dos originais ou têm nova geometria? Pode ser representada em multi-resolução?

- Objetivo da simplificação: Minimizar o tamanho da malha original dentro de um limite de erro ou minimizar o erro dado um tamanho esperado para a malha final?
- Metodologia usada na simplificação.
- Critério para mensuração do erro com relação à malha original.
- Abordagem local ou global: A malha original é modificada através de atualizações locais ou o processo de simplificação é aplicado à malha como um todo?
- Número de passos do algoritmo.
- Qualidade dos resultados.

2.3 Técnicas de simplificação

Nesta seção vamos descrever algumas técnicas de simplificação de superfícies com mais relevância no contexto do presente trabalho. Essas técnicas estão agrupadas por similaridade na sua abordagem.

2.3.1 Métodos incrementais baseados em atualizações locais

Os métodos incrementais baseados em atualizações locais são aqueles em que o processo de simplificação funciona através da aplicação de uma seqüência de operações locais que reduzem o tamanho da malha e, conseqüentemente, diminuem a precisão da aproximação.

A redução do tamanho da malha, implica na remoção de elementos da mesma, ou seja, vértices, arestas ou faces. Assim, temos as seguintes operações de atualização local:

- **Remoção de vértice:** Remove um vértice v da malha e conseqüentemente todas as arestas que incidem sobre ele. Para manter a integridade com relação ao modelo original, é necessária a execução de um procedimento que poligonize o "buraco" resultante. A Figura 2.3 exibe um caso de remoção de vértice de uma malha triangular.

- **Contração de aresta:** Substitui uma aresta e por um novo vértice v. Para tanto, remove as faces incidentes em e e atualiza todas as outras faces que eram incidentes aos vértices delimitadores de e para agora incidirem sobre v (ver Figura 2.4).

- **Contração de triângulo:** Substitui um triângulo t por um novo vértice v. Ou seja, efetua a remoção de t e atualiza a malha de forma que todas faces que incidiam em t passem a incidir sobre v, de acordo com a Figura 2.5.

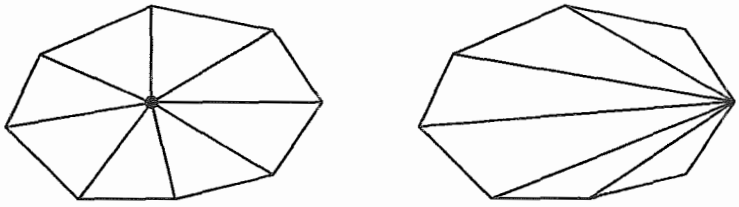


Figura 2.3: Operação de atualização local - Remoção de vértice.

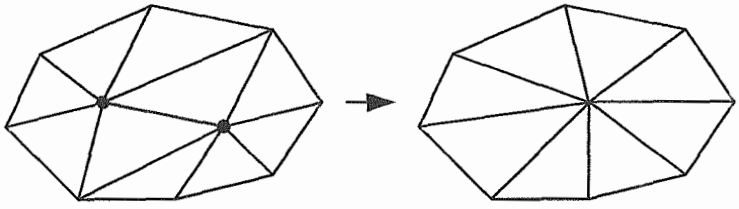


Figura 2.4: Operação de atualização local - Contração de aresta.

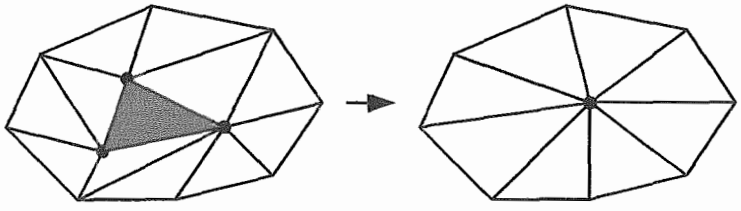


Figura 2.5: Operação de atualização local - Contração de triângulo.

Pode-se, então, definir um pseudo-código comum para esses métodos da seguinte forma:
Enquanto a complexidade ou precisão da malha não for satisfatória, fazer:

1. Escolher elemento a ser removido segundo algum critério de seleção;
2. Remover o elemento escolhido, atualizando os elementos vizinhos de maneira que a integridade da malha seja mantida.

Vamos agora detalhar alguns dos métodos que funcionam segundo este esquema.

Decimação de malhas

O método de decimação de malhas, inicialmente proposto por Schröeder [46], objetiva reduzir o número de triângulos de uma malha através da remoção de algum tipo de elemento (geralmente, vértices), de forma que a topologia seja preservada e a geometria possa ser bem aproximada, com relação ao modelo original (ver Figura 2.6).

O algoritmo executa múltiplos passos sobre uma malha triangular, usando informações de geometria e topologia locais para eleger um vértice para remoção. O vértice é efetivamente removido apenas se satisfizer um critério de ângulo ou distância, conhecido também como *critério de decimação*.

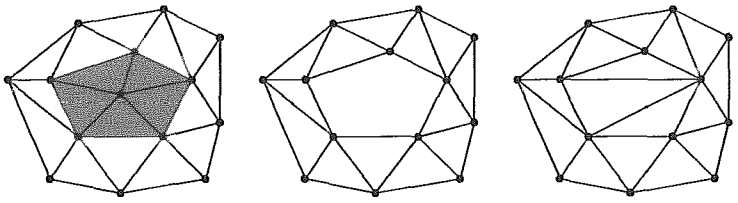


Figura 2.6: Decimação de malhas.

Os vértices que têm mais de um ciclo de faces incidentes são classificados como *complexos*. Os vértices complexos não são candidatos à remoção. Os vértices que estão sobre o bordo da malha e não possuem as características de um vértice complexo, são os *vértices de borda*. Por fim, temos os vértices que possuem exatamente um ciclo de faces incidentes. Neste caso, quando um vértice v tem uma vizinhança aproximadamente plana, ele é classificado como *vértice simples*. Para avaliar a vizinhança de um vértice quanto a sua planaridade, os autores empregam o conceito de *arestas características*. Portanto, um vértice simples é aquele sobre o qual não incide nenhuma aresta característica. Quando possui duas arestas características incidentes, ele é classificado como *vértice de aresta interior*. No caso de haver uma, três ou mais arestas características incidentes, então o vértice é classificado como *vértice de esquina*, que geralmente também não é removido. A Figura 2.7 exibe os casos de classificações deste método.

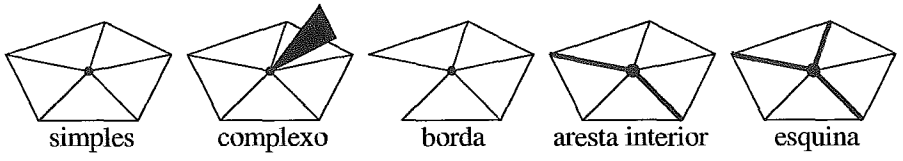


Figura 2.7: Classificação de vértices pelo método de decimação de malhas.

Assim, os vértices simples, de borda ou de aresta interior serão avaliados pelo critério de decimação para serem ou não removidos.

No caso de vértices simples, avalia-se a distância d do vértice ao plano médio das faces incidentes a ele. Nos outros casos, calcula-se a distância d do vértice v até o segmento de reta definido pelos outros dois vértices que compõem a borda ao redor de v ou que delimitam as

arestas características que incidem em v. Assim, se esta distância d for menor do que um valor estimado, então o vértice é removido da superfície. Este critério de decimação está ilustrado na Figura 2.8.

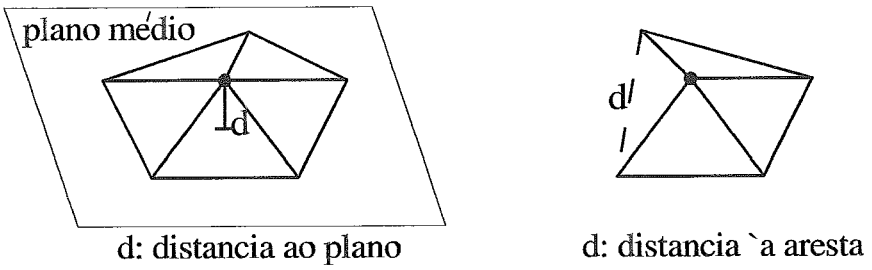


Figura 2.8: Critério de decimação: distância ao plano médio ou distância a aresta.

Esse processo de remoção de vértices é repetido, com possível ajuste do critério de decimação, até que a malha alcance determinado percentual de redução em relação à malha original, ou até que um valor máximo para a decimação seja atingido.

Este método se mostra eficiente no que diz respeito à velocidade e taxa de redução. É simples de implementar e usar e as aproximações obtidas são de boa qualidade. Preserva topologia - característica que pode ser especificada pelo usuário - e se propõe a trabalhar sobre malhas que representam tanto variedades quanto não-variedades. Um problema é que como o critério de avaliação é local, o erro é acumulado e, portanto, o método não pode garantir uma aproximação com erro limitado. Na avaliação feita por Garland e Heckbert [24] a desvantagem deste método está no fato de que a escolha do valor do critério de decimação é feita por parte do usuário e que o gasto de memória é grande.

Algumas extensões deste método surgiram na intenção de melhorar a precisão da aproximação. Seja garantindo erro limitado [10] [22], ou fazendo avaliação global do erro [47] [2] [33] [7], ou ainda alterando o processo de re-triangulação do "buraco" resultante pela remoção de um vértice [2] [7].

Soucy [47] desenvolveu um método de decimação de vértices que preserva cores e atributos. Ciampalini [7] utiliza uma representação que suporta multi-resolução. Renze [42] estendeu a idéia de decimação para malhas tridimensionais (malhas tetraédricas).

Embora os métodos de decimação, em geral, façam a remoção de vértices, podem também fazer a remoção de arestas [21] [22] [44] [1] ou faces [23].

Otimização de Malhas

Dado um conjunto de pontos V_0 e uma malha triangular inicial M_0 construída sobre os mesmos, o método de Otimização de Malhas [29] produz iterativamente uma malha M_j de mesma topologia que se encaixa sobre M_0 e tem um número de vértices menor do que M_0. Isto é feito utilizando-se operações de contração de aresta, partição de aresta ou troca de aresta (*flipping*), conforme ilustrado na figura 2.9.

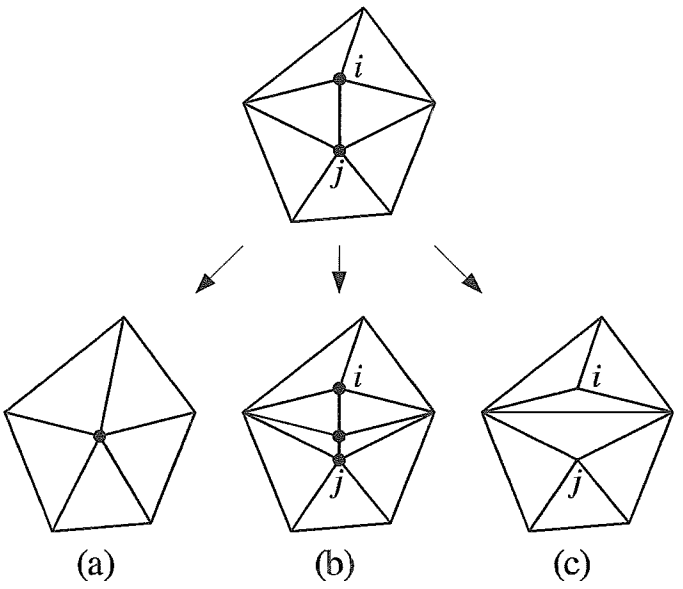


Figura 2.9: Operações de atualização local sobre aresta. No topo, temos a malha na qual será efetuada uma operação sobre a aresta ij. Da esquerda para a direita temos: contração, partição e troca.

Para definir o critério de seleção de uma aresta para contração, os autores desenvolveram e minimizaram uma função de energia

$$E(M) = E_{dist}(M) + E_{rep}(M) + E_{spring}(M), \tag{2.5}$$

onde E_{dist} é a soma das distâncias dos pontos originais a M, E_{rep} é um fator proporcional ao número de vértices em M e E_{spring} é a soma dos comprimentos das arestas.

A minimização da função de energia é feita em dois ciclos. O *ciclo interno* otimiza a geometria dos vértices para que se encaixem sobre os vértices originais. O *ciclo externo* seleciona uma operação legal que reduza a função de energia, executa-a e atualiza a malha.

Uma vantagem dessa técnica é que os resultados das malhas aproximadas são de alta qualidade, porém o tempo de processamento requerido é longo. Não é um método fácil de imple-

mentar nem de usar, principalmente devido à necessidade de especificar parâmetros de ajuste [15]. A topologia do modelo original é preservada e, geometricamente, novos vértices são gerados (no ciclo interno).

Para reduzir a complexidade, a seleção da operação legal é feita randomicamente e o ciclo de minimização interno é executado para um número fixo de iterações.

Embora este método adote uma avaliação de erro global, a aproximação resultante não tem erro limitado [40] [15].

Malhas Progressivas

Trata-se de uma metodologia proposta por Hoppe [26] que surgiu como uma extensão do método de Otimização de Malhas. Estrutura-se numa única operação de atualização da malha: a contração de aresta. Além disso, a função de energia é otimizada recebendo dois novos componentes $E_{escalar}$ e E_{disc}, de forma a preservar outros atributos de aparência da malha, que não geometria. Finalmente, o método introduz uma nova representação em multi-resolução.

Conforme mostra a Figura 2.10, verificou-se que a operação de contração de arestas pode ser facilmente invertida através de uma operação de partição de vértice.

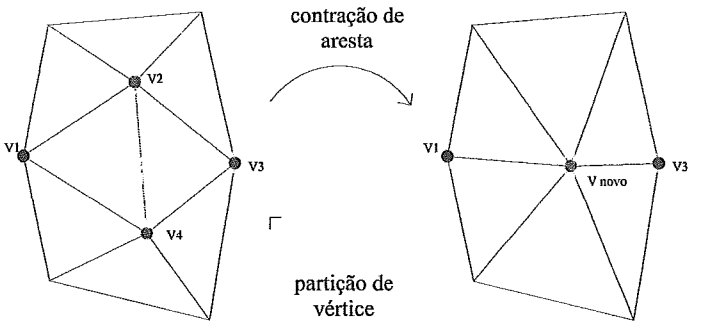


Figura 2.10: Definição das operações de contração de aresta e partição de vértice para Malhas Progressivas.

A utilização dessa operação inversa, também conhecida como *vsplit*, permite armazenar todo o processo de simplificação numa estrutura de dados de multi-resolução, denominada *Malha Progressiva*. Essa representação é composta por uma malha grosseira, de baixa resolução, mais uma seqüência de operações de refinamento (*vsplit*) obtidas pela inversão dos passos de simplificação aplicados à malha original. Ou seja, dado uma malha original $\hat{M}$ simplificada numa malha grosseira M^0 através do processo incremental

$$(\hat{M} = M^n) \overset{ecol_{n-1}}{\rightarrow} \ldots \overset{ecol_1}{\rightarrow} M^1 \overset{ecol_0}{\rightarrow} M^0, \tag{2.6}$$

onde *ecol* representa uma operação de contração de aresta, a representação por Malha Progressiva é formada pela t-upla

$$(M^0, \{vsplit_0, ..., vsplit_{n-1}\}), \tag{2.7}$$

onde $vsplit_i$ é a operação inversa de $ecol_i$.

A representação por malhas progressivas permite efetuar o processo de simplificação numa fase de pré-processamento, dando também a possibilidade de reconstrução de qualquer nível de detalhe interativamente:

$$M^0 \overset{vsplit_0}{\rightarrow} \ldots \overset{vsplit_1}{\rightarrow} M^1 \overset{vsplit_{n-1}}{\rightarrow} (M^n = \hat{M}). \tag{2.8}$$

Durante o processo de simplificação, a escolha das arestas a serem contraídas é feita com o auxílio de uma fila de prioridade, ordenada pelo custo estimado da energia (função de energia) para contrair a aresta.

A vantagem deste método está ligada ao fato de que suporta saída em multi-resolução, transmissão progressiva e refinamento seletivo [1]. Os resultados obtidos são de alta qualidade, uma vez que a etapa de simplificação é feita usando uma metodologia semelhante a de Otimização de Malhas [29], embora seja um pouco mais rápida. As desvantagens também são herdadas daquele trabalho: alto tempo de processamento e dificuldade de implementação e utilização. Herda também a avaliação de erro global, mas gerando aproximações com erro não limitado.

Simplificação de superfície usando métrica de erro quádrico

A métrica de erro quádrico [16] [18] provê uma caracterização competente da forma local de uma superfície, com custo computacional e custo de armazenamento moderados. A combinação da métrica de erro quádrico com o processo iterativo de contração de pares de vértices, resultam num algoritmo rápido que produz aproximações de superfícies poligonais de alta qualidade.

O algoritmo proposto por Garland e Heckbert permite a contração de pares de vértices quaisquer, ou seja, os vértices de cada par a ser contraído não precisam ter, necessariamente, uma aresta que os conecte (ver Figura 2.11). Isto torna o algoritmo capaz de juntar regiões

[1] Estes itens serão detalhados mais adiante.

desconexas do modelo, simplificando, portanto, a topologia da superfície. Em conseqüência, o algoritmo suporta superfícies que não sejam variedades.

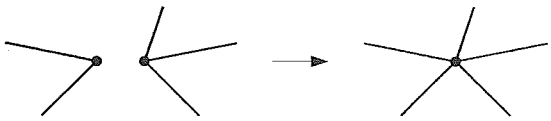


Figura 2.11: Contraindo par de vértices (aresta virtual).

Um par de vértices (v_1, v_2) é válido para uma contração, ou seja, pode ser contraído quando:

- (v_1, v_2) é uma aresta; ou,
- $\|v_1 - v_2\| < t$, onde t é uma parâmetro pré-determinado.

Vemos aqui que a escolha do valor de t deve ser feita com um certo cuidado.

A definição do conjunto de pares de vértices válidos é feita numa fase inicial e durante o curso do algoritmo, somente esses pares são considerados.

A cada vértice é associado o conjunto de pares ao qual ele faz parte. Quando uma operação de contração é efetuada, as arestas incidentes em v_1 e v_2 e os conjuntos de pares de v_1 e v_2 são atualizados para que apontem para o novo vértice gerado; arestas e pares duplicados são removidos.

Durante o percurso do algoritmo, a seleção do par de vértices que deve ser contraído é feita seguindo uma ordem de custo da contração. O custo da contração é definido através do erro associado ao vértice que será gerado, baseado num critério de avaliação de erro por uma função quadrática (ver seção 3.3.3).

O algoritmo pode ser resumido da seguinte forma:

1. Calcular para cada um dos vértices iniciais a função quadrática que avalia o erro;
2. Percorrer todos os pares de vértices válidos para contração, computando a posição ótima do vértice a ser gerado para cada par e calculando o custo da contração, que é dado pelo erro associado à posição escolhida para o novo vértice;
3. Inserir todos os pares numa fila de prioridade ordenada de forma crescente pelo custo de contração de cada par;
4. Repetir até que a aproximação desejada seja alcançada:
    - Retirar do topo da fila o par de vértices a ser contraído;

- Efetuar contração desse par;
- Atualizar o custo de todos os pares válidos associados ao novo vértice.

A Figura 2.12 [16] ilustra alguns resultados do algoritmo.

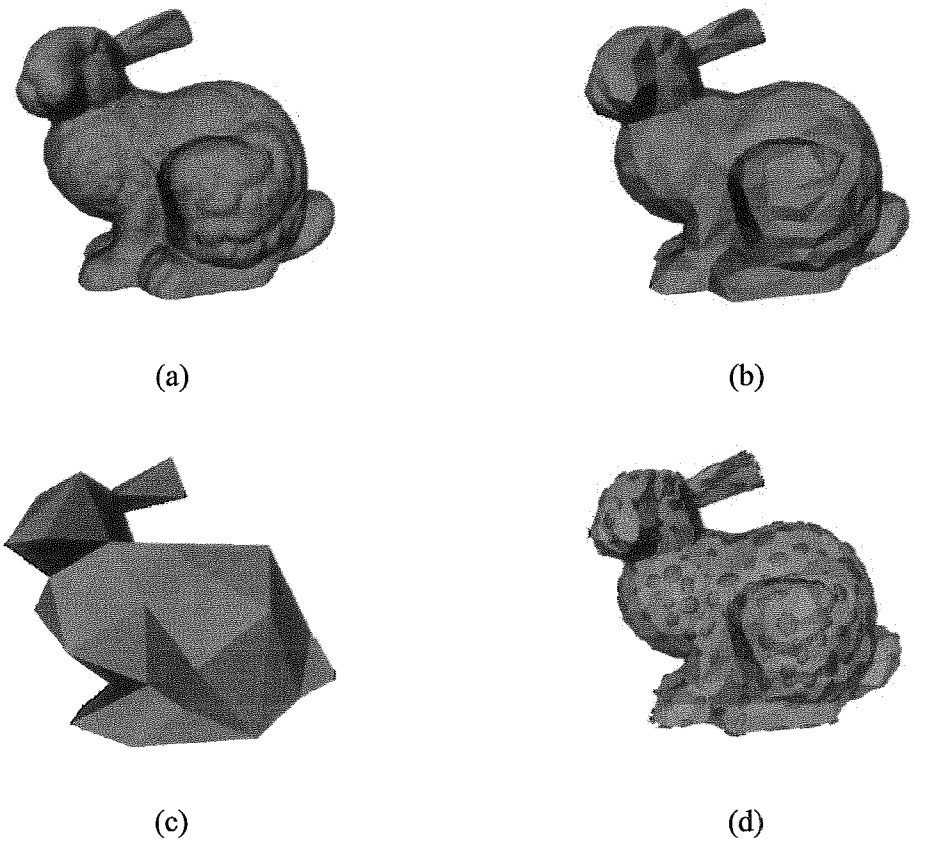


Figura 2.12: Modelos do coelho ("*bunny*") de Stanford. Aproximações obtidas usando métrica de erro quádrico. (a) modelo original com 69.451 faces (b) aproximação com 1000 faces (c) aproximação com 100 faces (d) aproximação com 1000 faces ilustrando as quádricas de erro em cada vértice.

2.3.2 Junção de faces coplanares (*Merging*)

Otimização geométrica

Hincker [25] mostra como polígonos coplanares ou quase coplanares podem ser "juntados" em polígonos complexos e maiores e, então, re-triangulados em polígonos simples e menores do que os originais. As contribuições mais importantes desse trabalho são:

1. Agrupamento rápido de polígonos por coplanaridade.
2. Abordagem rápida para junção de conjuntos de polígonos coplanares.

3. Triangulação simples e robusta para os polígonos criados por 1 e 2.

O algoritmo reduz o número de polígonos começando por um agrupamento de todas as faces em conjuntos de faces quase coplanares. Esse agrupamento é feito pela ordenação das faces, com relação às suas respectivas normais. Uma lista de arestas é criada para cada um desses conjuntos e efetua-se a remoção das arestas duplicadas. Isto resulta numa lista de arestas que contém somente a borda do polígono original. Esses polígonos são construídos e então re-triangulados para formar o objeto reduzido.

Segue abaixo uma breve ilustração do funcionamento do algoritmo 2.13:

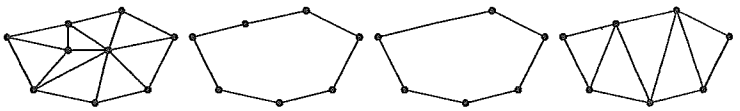


Figura 2.13: Otimização geométrica: Redução de um conjunto de polígonos quase coplanares.

1. Construção dos conjuntos de faces quase coplanares. Isto é feito em tempo $O(n \log n)$, onde n é o número de polígonos da superfície original).

2. Criação da lista de arestas em tempo $O(n \log n)$ e remoção das arestas duplicadas em $O(n)$.

3. Reconstrução do polígono em $O(n \log n)$, onde n é o número de elementos não removidos no item anterior, com remoção dos vértices colineares em tempo $O(n)$, onde n é o número de vértices do polígono reconstruído.

4. Triangulação dos polígonos resultantes.

Este algoritmo pode reduzir significativamente a quantidade de informações de primitivas geométricas requeridas para reproduzir um objeto. Reduzindo um conjunto de polígonos quase coplanares aos segmentos de sua borda, o algoritmo pode remover segmentos redundantes em um único passo. Os resultados do trabalho indicam que o algoritmo descrito funciona melhor quando aplicado sobre áreas com mudanças gradativas na orientação e é ineficiente quando opera sobre superfícies de alta curvatura. A sua complexidade é $O(n \log n)$, onde n é o número de primitivas originais no conjunto de quase coplanares.

Os vértices finais são um subconjunto dos originais e descontinuidades geométricas e topologia são preservadas. A avaliação do erro é altamente imprecisa e as aproximações geradas não são limitadas.

Superfaces

O método *Superfaces* [31] também faz junção de faces por coplanaridade, mas suporta aproximação limitada e propõe uma retriangulação mais robusta.

Para simplificar uma superfície S, as faces de S são agrupadas em conjuntos de regiões da superfície, chamados de *superfaces*, e cada um desses conjuntos é retriangulado satisfazendo um dada tolerância: para todos os vértices da malha original, a distância entre o vértice e o conjunto retriangulado têm que ser menor do que um valor especificado pelo usuário.

As superfaces, ou conjuntos de agrupamento das faces da malha, são criadas da seguinte forma:

- Iterativamente, escolhe-se uma face f_i como a superface corrente Sf_j;
- Encontra-se, por propagação, todas as faces adjacentes a fi e cada uma dessas faces será inserida nesta superface Sf_j se:
  - seus vértices estão a uma distância $\varepsilon/2$ do plano que aproxima Sf_j; e,
  - sua orientação é similar a das outras já inseridas em Sf_j.

Uma vez criada uma superface, sua borda é simplificada baseada na remoção de vértices colineares.

Ao final, o processo de re-triangulação é efetuado dividindo-se as superfaces em polígonos em forma de estrela que serão, então, decompostos.

Quanto ao algoritmo, a heurística usada para detectar faces quase coplanares é mais complexa e mais precisa do que a de Hinker [25], os vértices finais são um subconjunto dos originais e ele preserva as descontinuidades geométricas e a topologia da malha original. A avaliação do erro é mais precisa e o erro é limitado.

2.3.3 *Re-telhamento*

O método automático apresentado por Turk em [49] para criar modelos de superfícies em vários níveis de detalhe a partir de uma descrição poligonal de um dado objeto recebeu um nome curioso: re-telhamento. Isto porque este trabalho mostra como um novo conjunto de vértices pode ser distribuído sobre a superfície de um modelo e conectado a outros de forma a criar uma nova "cobertura" (telha) para a superfície que seja fiel tanto à geometria quanto à topologia da superfície original.

As principais contribuições deste trabalho são:

1. Um método robusto de conectar novos vértices sobre a superfície;

2. Uma forma de usar uma estimativa da curvatura da superfície para distribuir maior quantidade de novos vértices em regiões de alta curvatura; e,

3. Um método de interpolação suave entre modelos que representam o mesmo objeto em diferentes níveis de detalhe.

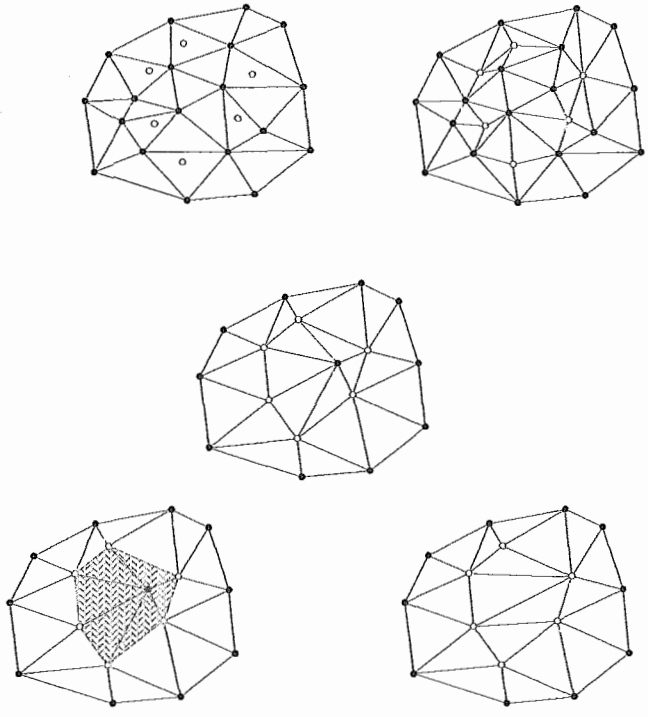


Figura 2.14: Ilustração do processo de re-telhamento.

O processo de re-telhamento começa espalhando aleatoriamente sobre a superfície uma quantidade de pontos definida pelo usuário. Em seguida, um procedimento de relaxamento é aplicado para mover cada ponto numa direção contrária a dos seus pontos vizinhos. Isto é feito seguindo uma força de repulsão que decai linearmente com a distância. Uma vez que todos os pontos tenham sido posicionados, o próximo passo é definir como esses pontos devem ser conectados entre eles mesmos e com os vértices originais da superfície, para formar uma malha triangular que reflita a topologia da superfície original. Este passo foi divido em duas etapas: primeiro cria-se uma superfície poligonal intermediária, chamada tesselação mútua (*mutual tesselation*), que contém tanto os vértices do modelo original quanto os novos pontos que se tornarão vértices da superfície re-telhada. Em seguida, cada vértices original é removido, desde que satisfaça duas condições de consistência topológica. Essa condições garantem que

a topologia da superfície original não seja modificada. Feito isso, a superfície é re-triangulada localmente de forma que as conectividades locais combinem com a superfície original. (Veja a Figura 2.14.)

Este método produz melhores resultados sobre modelos de superfícies curvas. Como exemplo, esta técnica foi aplicada com sucesso em modelos derivados de dados volumétricos e modelos de superfícies moleculares.

2.3.4 Aglomeração de vértices

O algoritmo descrito por Rossignac [45] é um dos poucos capazes de trabalhar sobre modelos poligonais arbitrários.

Define-se a região que envolve o modelo original e subdivide-se este espaço uniformemente, criando uma grade sobre o modelo. Cada região da subdivisão é chamada de célula. Todos os vértices que se encontram dentro de uma mesma célula são agrupados em um único vértice, dado pelo centróide, e as faces do modelo são atualizadas adequadamente.

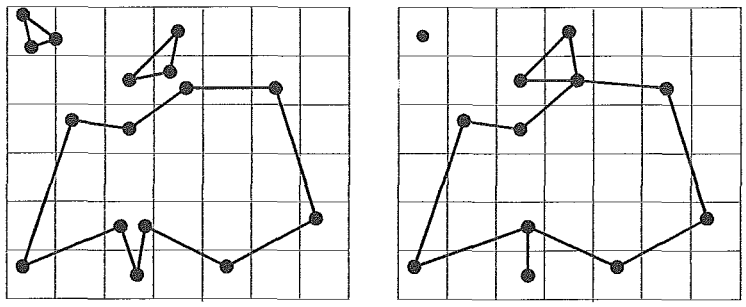


Figura 2.15: Aglomeração de vértices (*Vertex Clustering*).

Este processo pode ser muito rápido e pode fazer drásticas alterações topológicas sobre o modelo. Entretanto, enquanto o tamanho da célula garante um limite para o erro geométrico, a qualidade do modelo aproximado é, freqüentemente, absolutamente baixa. Além disso, é difícil construir uma aproximação com determinado número de faces, uma vez que o número de faces é indiretamente determinado pelas dimensões da grade. A aproximação gerada é também dependente da posição e orientação exatas do modelo original, em função da grade que o envolve.

Este método uniforme pode facilmente ser generalizado para usar uma estrutura adaptativa, como uma *octree* [30]. Isto pode aprimorar os resultados da simplificação, mas não garante a melhoria na qualidade e o controle desejados.

2.3.5 Análise em multi-resolução por *wavelets*

O método de análise em multi-resolução (MRA) está baseado numa proposta de multi-resolução por *wavelets*, isto é, sobre o uso de uma malha base simples mais uma seqüência de termos de correção local, chamados de coeficientes *wavelets*, os quais capturam em diferentes resoluções os detalhes presentes no objeto representado.

Lounsbery et al. [35, 36] generalizaram o conceito de análise de multi-resolução para superfícies de tipos topológicos arbitrários e desenvolveram um método para geração da decomposição *wavelet* com conectividade na subdivisão. Eck et al. [12] descreveram como a MRA pode ser aplicada para a aproximação de uma malha qualquer e constrói as aproximações MRA garantindo erro máximo limitado.

Esses métodos têm três passos principais:

1. Particionamento;
2. Parametrização;
3. Reamostragem.

No particionamento, a malha de entrada é dividida em um número pequeno de regiões triangulares $T_1, T_2, ..., T_n$ e essa triangulação de baixa resolução construída sobre ela é chamada de malha base K^0.

No passo de parametrização, para cada região T_i, uma parametrização local é construída sobre a face correspondente na malha base K^0.

Na reamostragem, são efetuados j passos recursivos de subdivisão quaternária das faces da malha base para construção da malha K^j. As coordenadas da nova malha K^j são obtidas pelo mapeamento dos vértices de K^j em $\Re^3$ usando a parametrização construída na fase anterior.

Este método provê, então, uma solução para o problema de reamostragem: dada uma malha de entrada M arbitrária, construir uma malha simplificada K^j que esteja garantidamente dentro uma tolerância ϵ_1 e a qual satisfaz a restrição de conectividade de subdivisão (necessária na decomposição *wavelet*).

Por prover uma parametrização entre as malhas de diferentes níveis de detalhe, a estrutura MRA torna possível a edição da superfície em multiresolução.

Conforme observado por Hoppe [26], as decomposições *wavelet* são geralmente incapazes de representar vincos sobre a superfície, a menos que elas estejam sobre arestas da malha base (veja Fig. 2.16).

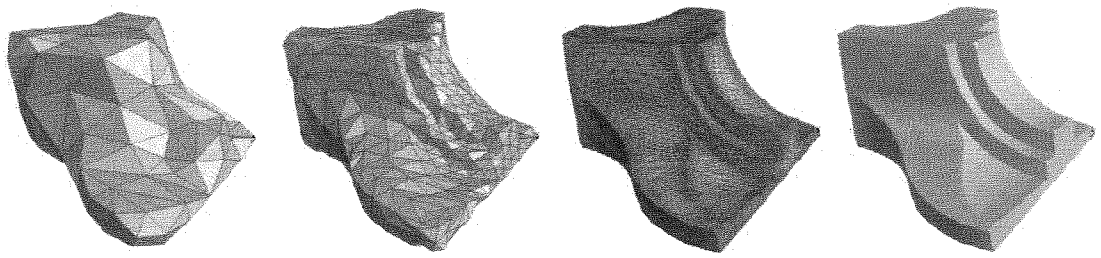


Figura 2.16: Os três primeiros modelos são aproximações do modelo da extrema direita geradas pelo esquema MRA. Observa-se que a MRA não consegue recuperar vincos e produz aproximações de baixa qualidade.

Certain et al. [5] extenderam a MRA para gerenciar a aproximação tanto com relação à geometria quanto com relação à cor da superfície, permitindo que elas possam ser comprimidas independentemente, e apresentaram um visualizador Web de multi-resolução suportando transmissão progressiva.

Assim como outros esquemas baseados em subdivisão, os métodos baseados em wavelets não conseguem construir facilmente aproximações com topologia diferente da superfície original [17].

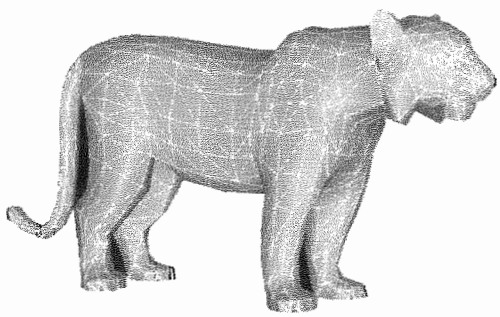
2.4 LOD e Multi-resolução

A idéia de representar um modelo em vários níveis de detalhe (*Levels of Detail* - LOD) é uma estratégia comum em vários trabalhos de computação gráfica e processamento de imagens.

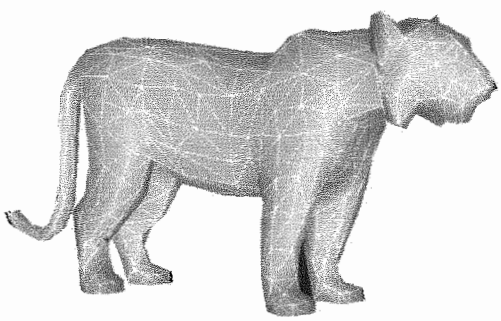
Existem muitos benefícios em se ter mais de uma representação de um mesmo objeto. Um primeiro exemplo é que, freqüentemente, é desnecessário usarmos um modelo completamente detalhado de um objeto durante a sua renderização se este objeto só cobre uma pequena região da tela. O uso de um modelo menor, ou seja, com poucos detalhes ou pequena quantidade de polígonos, pode diminuir significativamente o tempo de renderização da sua imagem. Uma outra vantagem do uso de vários níveis de detalhe, é a sua utilização para reconhecimento de características de um objeto através de representações sucessivamente mais grosseiras, técnica empregada em diversos trabalhos da área de processamento de imagens e reconhecimento de padrões [49].

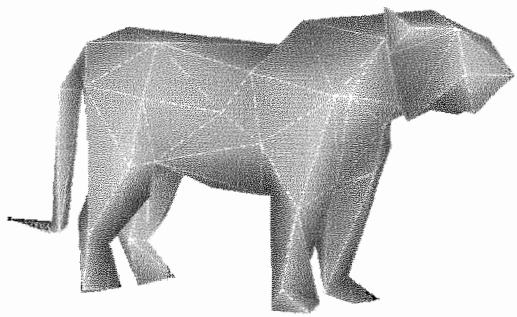
No nosso contexto, a representação LOD de um objeto consiste numa coleção de malhas independentes de diferentes tamanhos, onde cada uma delas representa esse objeto numa dada resolução, conforme ilustrado na Figura 2.17.

Uma representação LOD pode ser facilmente construída armazenando-se os resultados



(a)





(b) (c)

Figura 2.17: Modelo representado em 3 resoluções diferentes: (a) 956 vértices e 1.908 faces (b) 500 vértices e 996 faces (c) 100 vértices e 196 faces

obtidos pela aplicação repetida de algum algoritmo de simplificação variando-se apenas o parâmetro de simplificação.

Pela forma como essa representação é construída, existem alguns problemas que podem torná-la inadequada para determinadas aplicações. Por exemplo, a troca de um nível de detalhe por outro (mesmo consecutivo) é frequentemente feita de forma visualmente descontínua e nem sempre as resoluções existentes na representação são adequadas à necessidade da aplicação.

Para superar essas limitações surgiram os modelos em multi-resolução, que podem prover um grande número de malhas representando um único objeto em diferentes resoluções.

2.5 Modelos em multi-resolução

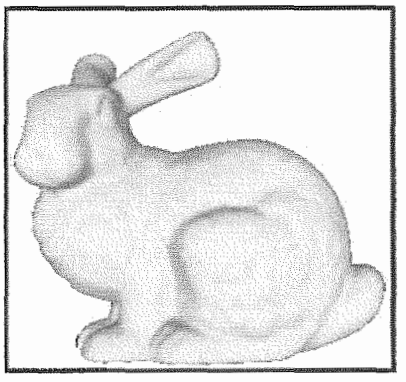
Um modelo em multi-resolução é uma representação de superfície que suporta a extração ou reconstrução de várias aproximações as quais podem acomodar um intervalo amplo de contextos de visão [18].

Algumas propriedades que são desejadas para um modelo em multi-resolução são [40]:

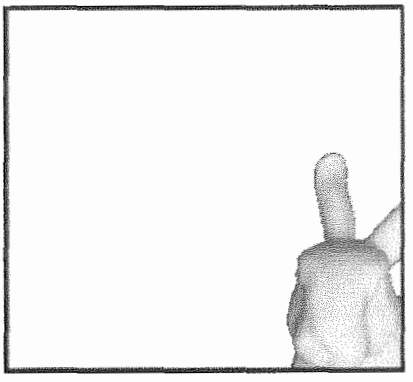
- O modelo deve possibilitar que uma malha com uma dada resolução possa ser recuperada em espaço curto de tempo.
- O tamanho do modelo não pode ser consideravelmente maior do que o tamanho da malha de resolução mais alta que ele pode prover.
- Qualquer malha extraída do modelo deve ser contínua, ou seja, deve-se evitar buracos devido à mudança do nível de detalhe dentro de uma mesma malha.
- A transição entre diferentes malhas extraídas do modelo deve ser feita de forma suave, evitando o efeito de "*popping*" entre as malhas de níveis de detalhes consecutivos.

Como exemplo da aplicação de um modelo em multi-resolução, consideremos a Figura 2.18 que mostra um objeto em três diferentes contextos (exemplo extraído de [18] e [17]). Não existe um único conjunto de polígonos que seja apropriado para essas diferentes condições de visão ao mesmo tempo. Ao invés disso, desejamos ter várias aproximações diferentes disponíveis, para que possamos selecionar a melhor de acordo com a condição de visão. Neste caso, um modelo em multi-resolução é mais apropriado do que um modelo de resolução fixa.

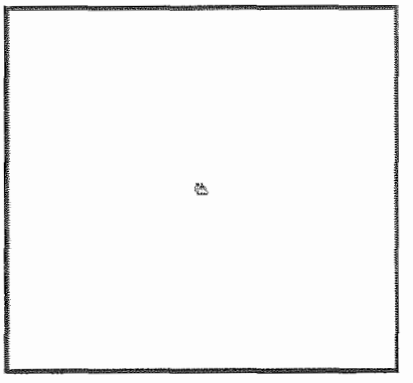
A Figura 2.18(a) exibe um modelo de superfície contendo cerca 69.000 faces triangulares. Consideremos o caso em que o observador esteja examinando de perto uma região dessa superfície, conforme mostra a Figura (b). A tela é, então, preenchida por uma pequena porção



(a)



(b)



(c)

Figura 2.18: Modelo com 34.834 vértices e 69.664 faces: (a) Visão completa com distância adequada (b) Visão seletiva (c) Visão completa à longa distância

da superfície total. Sob essas condições, a área sendo examinada pode estar usando uma quantidade relativamente pequena de triângulos enquanto que o resto do modelo, que está fora do volume de visão, é ignorado. Agora consideremos a figura (c), onde o modelo preenche uma pequena porção da tela. Neste caso, o número de triângulos sendo renderizados é desproporcional ao número de pixels efetivamente produzidos. Dada a resolução de saída na qual o objeto está sendo renderizado, é bem provável que se consiga resultados idênticos com menos de 100 triângulos.

Assim, a aproximação apropriada de um modelo original depende das condições de visualização (por exemplo, distância do objeto ao observador). Ou seja, o nível de detalhe adequado da malha varia. Mais precisamente, o nível de detalhe adequado pode ainda variar dentro de uma mesma aproximação (malha).

Os modelos em multi-resolução que suportam extração de malhas com resolução uniforme sobre todo o objeto representado são denominados *modelos de domínio uniforme*, enquanto os modelos que suportam a extração de uma malha cuja resolução varia em diferentes regiões do objeto representado são chamados *modelos de domínio variável* [40].

Os modelos em multi-resolução devem não só permitir a extração de aproximações adequadas a diferentes circunstâncias de visão, mas também permitir a troca do nível de detalhe sem custo excessivo. Se o tempo necessário para mudar e renderizar um nível de detalhe mais baixo é muito maior do que o tempo necessário para simplesmente renderizar o mesmo objeto um nível de detalhe mais alto, então não estamos tendo nenhuma vantagem com esse modelo em multi-resolução [17].

De acordo com a sua estrutura, os modelos em multi-resolução podem ser divididos em dois grupos: representação por árvore e representação por histórico.

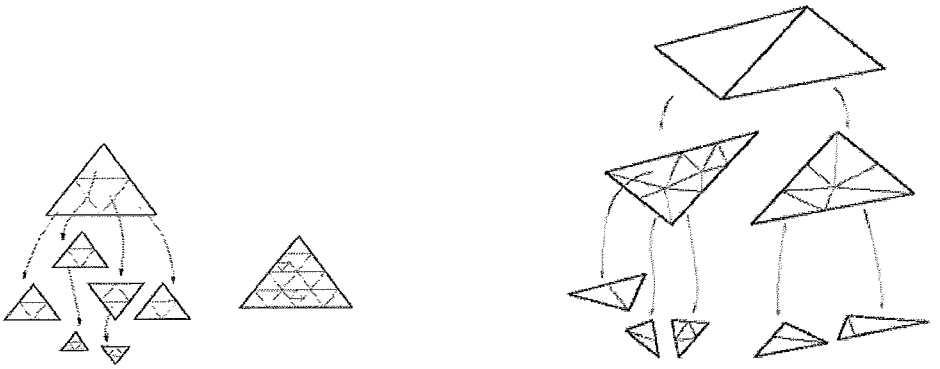
2.5.1 Modelos em árvore

Os modelos em multi-resolução de representação em árvore são aqueles baseados numa subdivisão aninhada: a resolução do modelo pode ser aumentada (refinada) pela subdivisão recursiva de uma região, criando um conjunto de regiões menores que cobrem a original. Esta hierarquia de regiões pode ser descrita por uma estrutura de árvore.

Um exemplo de deste tipo de modelo pode ser visto na Figura 2.19.

O nó raíz da árvore de representação corresponde à malha grosseira. A existência de filhos para um nó indica que a região representada por esse nó ainda pode ser refinada, ou seja, subdividida em k regiões, onde k é o número de filhos do nó.

O modelo apresentado na Figura 2.19(a) é representado por uma árvore onde cada nó pode



(a) (b)

Figura 2.19: (a) Triangulação quaternária (b) Triangulação adaptativa (Triangulação hierárquica de Delaunay).

ter até 4 filhos, uma vez que ela é construída a partir de uma triangulação quaternária. Já a figura 2.19(b) ilustra um modelo onde a árvore suporta uma subdivisão adaptativa, ou seja, os nós da sua árvore de representação podem ter um número qualquer de filhos.

2.5.2 Modelos por histórico

Os modelos por histórico são malhas em multi-resolução cuja estrutura é obtida através do armazenamento da evolução de uma malha através de um seqüência de modificações locais.

Assim, cada vez que um passo de um método de simplificação baseado em atualizações locais é executado, a operação efetuada é armazenada. Esta seqüência somada à malha original definem um modelo com níveis de detalhe, ou resolução, decrescente. Um modelo crescente pode ser obtido invertendo-se a estrutura definida pelo algoritmo de simplificação e armazenando-se a malha simplificada ao invés da malha original; ou simplesmente armazenando os passos de um algoritmo de refinamento.

2.6 Transmissão progressiva

Quando uma malha é transmitida por uma linha de comunicação, é de grande interesse que possamos ver aproximações cada vez melhores à medida que os dados vão sendo recebidos, ao invés de termos que esperar o recebimento de todo o conteúdo para, aí sim, visualizarmos o objeto transmitido. Esta forma de transmissão desejada é denominada transmissão progressiva.

Malhas progressivas são uma representação natural para transmissão progressiva [26]. Esse processo se dá através da transmissão, em primeiro lugar, de uma malha grosseira M^0, usando

um formato convencional de resolução uniforme, seguida de uma seqüência de registros de atualização da malha. O processo que recebe os dados reconstrói incrementalmente a malha original, à medida que cada registro chega. Para evitar que na mudança de malha apareçam descontinuidades visuais, pode-se utilizar o recurso de *morphing* entre os elementos geométricos das malhas em questão (*geomorphing*) [26] [28].

2.7 Refinamento seletivo

O simples armazenamento de uma malha simplificada acompanhada da seqüência de passos efetuados sobre a malha original detalhada durante o processo de simplificação permitem a extração direta de uma malha em qualquer nível de detalhe mas com resolução uniforme. Ou seja, essa estrutura de representação só permite reconstruir modelos que foram gerados durante o processo de simplificação original.

Como já foi observado, em muitas aplicações é extremamente importante que se possa reconstruir uma aproximação com resolução adaptada às condições de visualização desse modelo.

Um primeiro esquema de refinamento dependente de visão para malhas progressivas, proposto por Hoppe em [26], baseava-se numa função de refinamento que avaliava critérios de volume de visão, distância a silhuetas e áreas de faces projetadas na tela.

A idéia básica era percorrer a seqüência de registros *vsplit* de acordo com a sua ordem de definição, executando a operação somente se duas condições fossem satisfeitas:

1. os vértices necessários para execução de *vsplit* devem estar presentes na malha corrente; e,

2. a função de refinamento aplicada ao vértice a ser particionado deve retornar um valor verdadeiro.

O problema deste esquema de refinamento seletivo é que ele não é eficiente para ser executado em tempo real, uma vez que para cada mudança dos parâmetros de visualização toda a seqüência de operações é percorrida.

A solução proposta pelo próprio Hoppe em [27] foi utilizar uma hierarquia de vértices que permite fazer refinamento seletivo em tempo real. Esta hierarquia foi introduzida inicialmente por Xia e Varshney [50] e pode ser construída a partir da própria seqüência de operações *vsplit* presente na representação por malhas progressivas.

2.7.1 Hierarquia de vértices

Xia e Varshney [50] usaram as transformações *ecol* e *vsplit* (ver seção 2.3.1) para criar uma hierarquia de simplificação que permite refinamento seletivo em tempo real. Nesta abordagem, é construída uma árvore, mais precisamente uma "floresta", contendo todos os vértices da malha original M^n mais todos os vértices gerados durante o processo de simplificação dessa malha.

Voltando a Figura 2.10, uma operação *ecol* efetua a substituição de uma aresta por um vértice, conforme ilustrado na Figura 2.20.

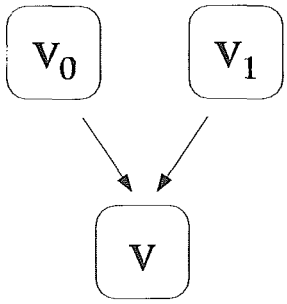


Figura 2.20: Representação de *ecol*.

Inversamente à operação *ecol*, uma operação *vsplit* substitui um vértice por uma aresta, conforme ilustrado na Figura 2.21, definindo uma relação *pai-filhos*.

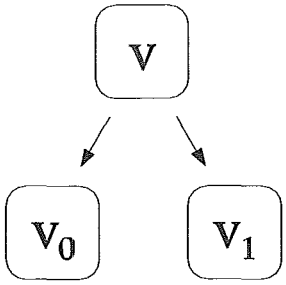


Figura 2.21: Representação de *vsplit*.

Assim, uma hierarquia de vértices é definida armazenando-se essas relações definidas pela operação *vplit* (ver Figura 2.22). Os nós raízes correspondem aos vértices da malha grosseira M^0, enquanto que os nós folhas correspondem aos vértices da malha M^n detalhada original.

Essa hierarquia de vértices pode ser construída tanto de baixo para cima, ou seja, a partir da malha detalhada, de acordo com Xia e Varshney [50], quanto de cima para baixo conforme proposto por Hoppe [27]. Neste último caso, tendo-se uma representação por malha progressiva $(M^0, \{vsplit_0, ..., vsplit_{n-1}\})$, a hierarquia é construída percorrendo-se todos os registros

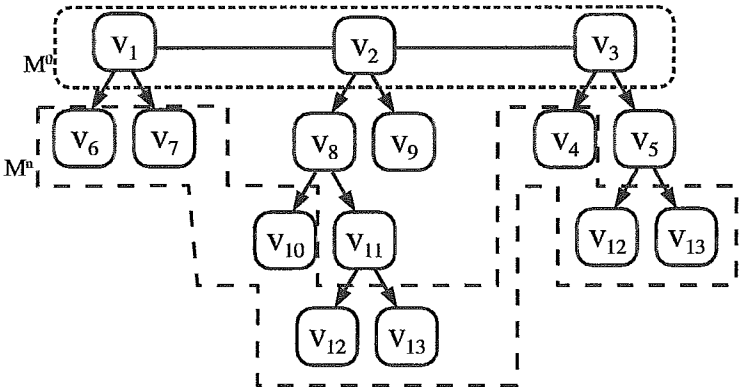


Figura 2.22: Hierarquia de vértices: M^0 é a malha grosseira simplificada e M^n é a malha original detalhada.

vsplit. Assim, os vértices de M^0 são inseridos no topo da estrutura que vai "crescendo" à medida que se percorre cada registro *vsplit*.

Capítulo 3

Plataforma para construção de malhas progressivas

3.1 Introdução

Na intenção de gerar um modelo em multi-resolução de forma que o processo de transmissão do mesmo possa ser feito de maneira eficiente, vamos adotar um processo de simplificação utilizando um método incremental baseado em uma única atualização local que possa ser facilmente invertida.

O processo de simplificação se dará, então, através da aplicação iterativa de operações de contração de aresta, onde a seleção da aresta será feita segundo uma fila de prioridade (veja capítulo 2).

A fila de prioridade será preenchida inicialmente com todas as arestas do modelo que estão livres para contração seguindo um critério de ordenação por peso das arestas.

Uma aresta é classificada como *livre* para contração quando pelo menos um de seus vértices delimitadores pode ter as suas coordenadas alteradas. Neste caso, o vértice é também denominado livre.

O cálculo do peso de cada aresta é feito segundo uma métrica pré-definida. Ao invés de estabelecer uma única métrica fixa, optamos por implementar algumas já bastante conhecidas, de forma que o usuário fique livre para fazer a seleção e avaliar os diferentes resultados obtidos a partir de cada uma delas.

Para que a operação de contração da aresta selecionada possa ser realmente efetuada sobre a malha, são feitos alguns testes de verificação para garantir a consistência topológica e geométrica do resultado.

A cada passo desse processo de simplificação, informações sobre a operação executada são

guardadas numa lista de operações. As informações armazenadas são as necessárias para que se possa efetuar a operação em si e, principalmente, a sua inversa, que será executada num processo de refinamento do modelo simplificado.

Ao final desse processo, tem-se um modelo simplificado que é armazenado em 2 arquivos: um contendo a descrição e geometria do modelo simplificado e o outro contendo a seqüência de operações de contração de aresta executadas.

Esses arquivos nos dão uma representação em multi-resolução, permitindo a exibição do modelo em diferentes níveis de detalhe.

Neste capítulo vamos detalhar a operação de contração de aresta e a sua inversa, abordar os critérios de cálculo do peso das arestas do modelo (métricas implementadas) e as estruturas de armazenamento de informações do modelo para que se possa ter uma representação em multi-resolução que permita também uma visualização detalhada dependente do observador.

3.2 A operação de contração de aresta e a sua inversa

Seja uma malha M_i e uma aresta $hCol$ que conecta dois vértices v_0 e v_1, ou seja, v_0 e v_1 são os vértices delimitadores de $hCol$. A operação de contração de aresta implica na substituição de $hCol$ pelo vértice v, atualizando-se as arestas incidentes em v_0 e v_1 nesse novo vértice e removendo-se as faces que se tornam degeneradas e as arestas que ficam duplicadas, gerando assim a malha M_{i+1} (veja Fig. 3.1).

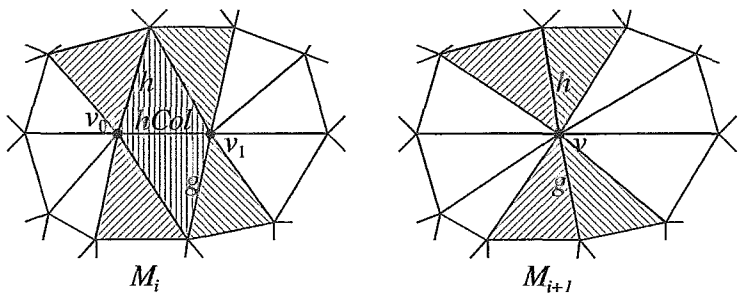


Figura 3.1: À esquerda uma malha M_i e à direita uma malha M_{i+1} obtida a partir de M_i através de uma operação de contração de aresta.

Ou seja, uma operação de contração de aresta como a ilustrada na Figura 3.1 consiste em:

- Remover os elementos:
  - Aresta $hCol$;

- Vértices delimitadores v_0 e v_1;
- Faces incidentes em $hCol$;

- Criar um novo vértice v;
- Atualizar as relações de incidência sobre v.

É importante notar que este processo de simplificação está limitado a uma porção local da malha M_i: o conjunto de elementos incidentes em v_0 ou v_1 [9].

Para que se possa recuperar M_i a partir de M_{i+1} precisamos de uma operação inversa à de contração de aresta. Esta operação é definida pela substituição de um vértice por uma aresta e a denominaremos *partição de vértice*.

Como podemos ver na Figura 3.1, ao efetuar a partição de um vértice v para gerar uma aresta, duas das arestas incidentes sobre v são duplicadas e dois novos triângulos são criados. Os triângulos já existentes e incidentes em v se mantêm, porém sua forma é modificada, uma vez que agora incidem sobre vértices em novas posições. Eventualmente, as coordenadas de um dos novos vértices podem coincidir com as coordenadas de v e, neste caso, alguns desses triângulos manterão suas formas.

Assim, para que a operação inversa possa ser efetuada sobre M_{i+1} ou, mais precisamente, sobre v, são necessárias informações que permitam recuperar de forma correta a malha original M_i, em particular as relações de incidência sobre os vértices delimitadores da aresta reconstruída $hCol$ (ver Figura 3.2).

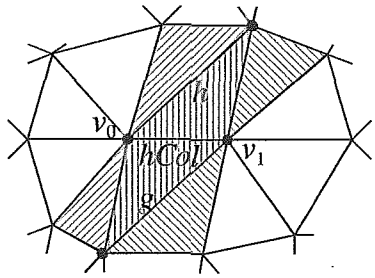


Figura 3.2: Reconstrução inválida da malha mostrada na Fig. 3.1.

Seja E o conjunto de arestas incidentes em v. Para que se possa recuperar de forma eficiente as relações de incidência sobre v_0 e v_1 precisamos definir sobre o conjunto E quais arestas deverão incidir sobre v_0 e quais arestas deverão incidir sobre v_1. Então, é necessário que saibamos quem são as duas arestas a serem duplicadas na execução da operação de partição do vértice v. Em outras palavras, é preciso determinar as arestas h e g que se tornaram duplicadas na execução da operação de contração da aresta $hCol$.

3.3 Métricas para seleção de aresta

3.3.1 Comprimento da aresta

Esta métrica baseia-se na distância euclideana entre os vértices delimitadores da aresta a ser contraída. Partindo-se da idéia de que a remoção de elementos menores é menos perceptível do que a remoção de elementos maiores, a escolha da aresta a ser removida é feita pelo seu comprimento (ver Figura 3.3). Sua vantagem está no baixo custo computacional.

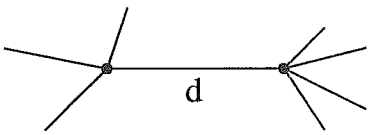


Figura 3.3: *d*: Comprimento de uma aresta.

3.3.2 Área do triângulo

Ainda seguindo o princípio de remover primeiro os elementos menos perceptíveis, esta métrica seleciona a aresta cuja soma das áreas dos triângulos adjacentes é mínima, conforme ilustrado na Figura 3.4. A vantagem também está no baixo custo computacional, uma vez que o acesso aos triângulos adjacentes a uma dada aresta pode ser feito em tempo constante desde que se use uma estrutura de dados adequada. Como exemplo, podemos citar as estruturas: *Winged-edge* [3], *Quad-edge* [20] e *Half-edge* [38].

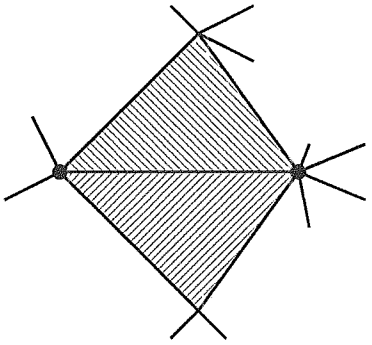


Figura 3.4: Área dos triângulos adjacentes a uma aresta.

3.3.3 Erro quádrico

A métrica do erro quádrico foi proposta por Garland [16]. Nela o erro associado a cada vértice é dependente dos planos dos seus triângulos incidentes. O erro de um vértice $v = [v_x\ v_y\ v_z\ 1]^T$ é definido pela soma dos quadrados das distâncias a esses planos, que pode ser reescrita numa forma quadrática

$$\delta v = v^T Q v, \tag{3.1}$$

onde Q é uma matriz simétrica 4×4.

Para uma dada contração $(v_1, v_2) \rightarrow \bar{v}$, a matriz $\bar{Q}$ que aproxima o erro no vértice $\bar{v}$ é dada pela soma $\bar{Q} = Q_1 + Q_2$. Como o erro associado ao novo vértice depende da sua posição geométrica, é importante que essa escolha seja feita de forma a minimizá-lo. Como a função de erro δ é uma função quadrática, encontrar esse mínimo é um problema linear, bastando para isso resolver o sistema de equações

$$\frac{\partial \delta}{\partial x} = \frac{\partial \delta}{\partial y} = \frac{\partial \delta}{\partial z} = 0. \tag{3.2}$$

Vale ressaltar que o erro inicial estimado para cada vértice da superfície original é nulo, pois cada vértice pertence aos planos de todos os seus triângulos incidentes.

3.3.4 Baseada em Spline

Esta métrica proposta por El-Sana e Varshney [14] combina normal e posição do par de vértices a ser contraído usando splines cúbicas.

O trabalho em que ela foi proposta permite a remoção de pares de vértices desconexos a fim de reduzir a complexidade do conjunto de dados. Para tanto, os autores propõem usar uma métrica baseada em curvas spline cúbicas. Esta métrica aproxima bem o comprimento de uma aresta de uma superfície e, ao mesmo tempo, estima razoavelmente bem a distância entre dois vértices da superfície não-conectados por uma aresta.

Uma curva cúbica paramétrica é construída usando interpolação Hermiteana 3.3,

$$P(t) = \sum_{i=0}^{3} H_i t^i, t_a \leq t \leq t_b, \tag{3.3}$$

onde H_i é determinado pela posição e pela tangente dos dois vértices, conforme ilustrado na Figura 3.5.

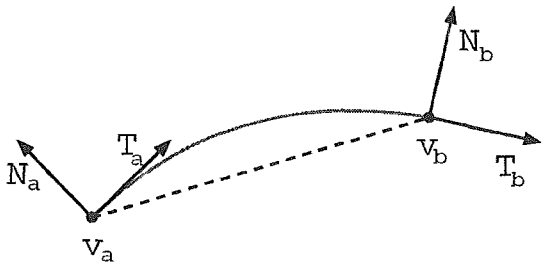


Figura 3.5: Curva cúbica de Hermite

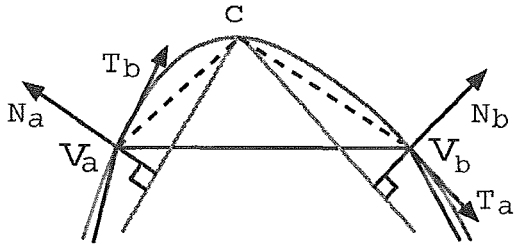


Figura 3.6: Métrica para contração baseada em spline.

Os modelos em geral não têm a informação das tangentes nos seus vértices. Contudo, a direção dessas tangentes pode ser calculada em função das normais em cada vértice. O módulo dessas tangentes deve ser levado em consideração, uma vez que esses valores controlam o comprimento da curva como um todo. A escolha desses valores deve garantir que a curva no intervalo $[v_a,v_b]$ seja suave e não possua pontos de inflexão, "laços" (*loops*) ou cúspides. Uma estimativa de comprimento razoável para que essas singularidades não aconteçam é que o módulo dos vetores tangentes devem ser menores do que o comprimento da aresta $v_a v_b$.

O erro associado a uma contração de dois vértices é dado, então, pelo comprimento da curva spline cúbica $P(t)$ 3.4.

$$Comprimento(P(t), v_a, v_b) = \int_{t_a}^{t_b} \| \frac{\partial P(t)}{\partial t} \| dt \tag{3.4}$$

É claro que o erro introduzido é afetado pela posição do novo vértice. Então, é importante que a escolha da nova posição seja de tal forma que esse erro seja minimizado.

Os autores perceberam que, freqüentemente, quando a posição do vértice resultante c é dada pelo ponto que tem a média das tangentes T_a e T_b, a distância entre os dois vértices e a curva resultante é minimizada (ver Figura 3.3.4).

3.4 Consistência topológica e geométrica

Na intenção de sempre obtermos modelos válidos tanto topológica quanto geometricamente, as operações de contração de aresta devem ser precedidas de testes de viabilidade.

Uma operação de contração pode resultar em um modelo não válido, por exemplo um modelo com faces degeneradas (de área nula) ou que delimitem uma região de volume zero, ou ainda uma não-variedade. Assim, as atualizações necessárias por uma operação de contração de aresta devem ser feitas através de operadores que possuam a característica de manter a topologia do modelo original. Além disso, para efetuar uma operação de contração de aresta, são empregados alguns testes que verificam se a consistência geométrica local será garantida.

3.4.1 Os Operadores de Euler

A necessidade de edição de um modelo sólido poliédrico nos leva à utilização de operações que adicionam ou removem vértices, arestas ou faces.

Em um processo de edição de um poliedro, alguns resultados intermediários podem não representar sólidos válidos.

Usualmente, a construção de modelos "B-rep" válidos é feita usando os assim chamados *operadores de Euler*. Estes operadores se caracterizam por alterar o número de elementos topológicos do modelo mantendo a relação definida na fórmula de Euler-Poincaré (Equação 2.3). Pode ser mostrado que com a utilização desses operadores, somente modelos realizáveis podem ser construídos. Os operadores de Euler formam um conjunto completo de primitivas de modelagem para uma variedade. Mais precisamente, todo poliedro válido topologicamente, pode ser construído a partir de um poliedro inicial através da aplicação de uma seqüencia finita de operadores de Euler[38].

Esses operadores dividem-se em dois grupos: um de criação e um de remoção [38].

Operadores de criação:

1. MVFS - Cria vértice, face e superfície.
2. MEV - Cria aresta e vértice.
3. MEF - Cria aresta e face.
4. MEKR - Cria aresta e remove anel.
5. MFKRH - Cria face e remove anel e buraco.

Operadores de remoção:

1. KVFS - Remove vértice, face e superfície.

2. KEV - Remove aresta e vértice.

3. KEF - Remove aresta e face.

4. KEMR - Remove aresta e cria anel.

5. KFMRH - Remove face e cria anel e buraco.

Observe que o nome dos operadores são codificados da seguinte forma:

M - *Make*	V - *Vertex*	S - *Solid*
K - *Kill*	E - *Edge*	H - *Hole*
	F - *Face*	R - *Ring*

A Figura 3.7 ilustra exemplos da aplicação desses operadores sobre modelos.

Com o uso desses operadores, podemos manipular os modelos de borda de forma que os novos modelos sejam topologicamente consistentes com o original. Entretanto, modelos inválidos podem ser construídos associando-se informações geométricas inadequadas às entidades topológicas consistentes [38].

3.4.2 Contração de aresta preservando topologia

Dey et al. [11] determinaram condições necessárias e suficientes para que uma operação de contração de aresta possa ser efetuada sobre um modelo 2-variedade sem mudança da topologia.

Para nosso uso, este critério pode ser melhor entendido se nos referirmos a um exemplo. Contraste a malha mostrada na Fig. 3.8(a), onde a aresta ab pode ser contraída, com a malha mostrada na Fig. 3.8(b), onde a contração da aresta ab implicaria na criação dos triângulos degenerados xaz e xzb.

Dey et al. [11] propuseram este critério no contexto de complexos simpliciais. Para compreendê-lo melhor, precisamos definir alguns termos.

Seja σ um k-simplexo, isto é, a combinação convexa afim de $k+1$ pontos. Uma face de σ é um simplexo τ definido por um subconjunto não vazio desses $k+1$ pontos. σ é então chamado de co-face de τ e escreve-se $\tau \leq \sigma$. Um complexo simplicial K é uma coleção finita de simplexos tal que se $\sigma \in K$ e $\tau \leq \sigma$ então $\tau \in K$ e se $\sigma, \sigma' \in K$ então $\sigma \cap \sigma'$ é vazio ou uma face em comum.

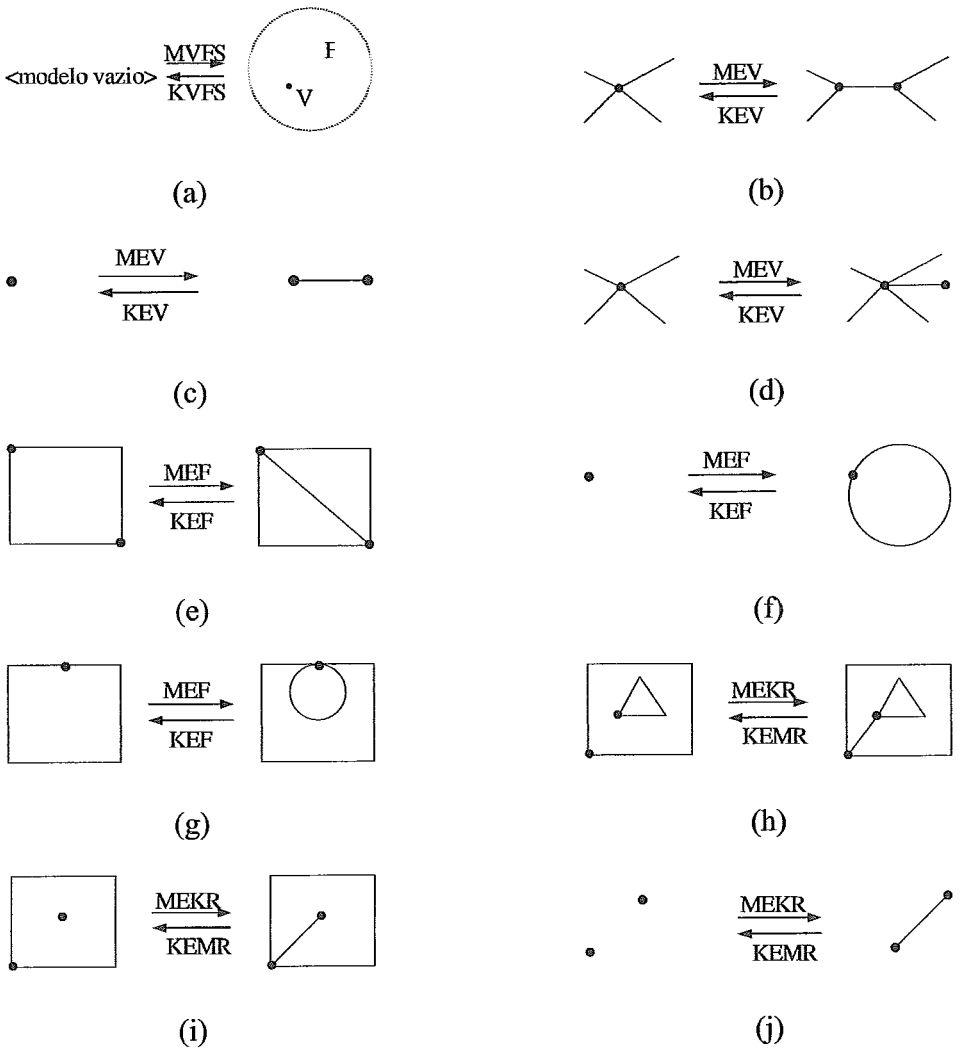


Figura 3.7: Exemplos de uso dos operadores de Euler.

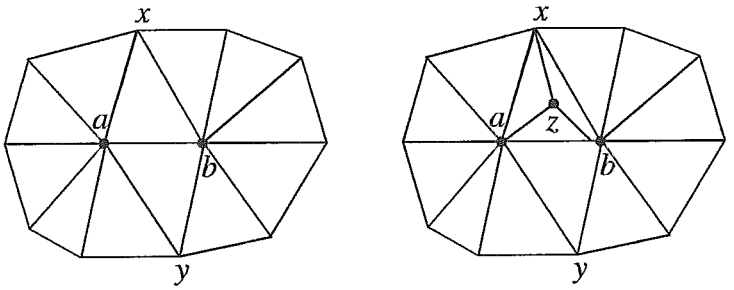


Figura 3.8: Verificação de consistência topológica: à esquerda, a condição $Lk(a) \cap Lk(b) = \{x, y\} = Lk(ab)$ indica uma contração válida, o que não ocorre à direita, onde detecta-se uma contração inválida pois $Lk(a) \cap Lk(b) = \{x, y, z, zx\} \neq Lk(ab)$.

Seja $St(\sigma)$ o conjunto de todas as co-faces de σ e $Lk(\sigma)$ o conjunto de todas as faces pertencentes ao conjunto $St(\sigma)$ mas que não incidem sobre σ (isto é, o conjunto de todas as faces das co-faces de σ disjuntas de σ. Para ilustrar estas definições, considere a Figura 3.9. Então, para o vértice x temos

$$St(x) = \{xa, xb, xy, xg, xh, xi, xj, xab, xby, xyg, xgh, xhi, xij, xja\}, \text{ e}$$

$$Lk(x) = \{a, b, y, g, h, i, j, ab, by, yg, gh, hi, ij, ja\},$$

e para a aresta xy,

$$St(xy) = \{xab, xby, ybc, ycd, yde, yef, yfg, xyg, xgh, xhi, xij, xja\}, \text{ e}$$

$$Lk(xy) = \{a, b, c, d, e, f, g, h, i, j, ab, bc, cd, de, ef, fg, gh, hi, ij, ja\}.$$

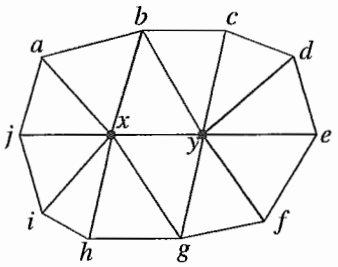


Figura 3.9: Complexo simplicial K.

Seja S_i uma superfície poliédrica sem borda, e S_{i+1} a superfície poliédrica depois de executada uma contração de aresta. De acordo com [11], as seguintes sentenças são equivalentes:

1. $Lk(v_0) \cap Lk(v_1) = Lk(hCol)$
2. S_i, S_{i+1} são homeomorfos

Assim, é suficiente verificar a condição 1 para que a condição 2 seja satisfeita, o que garante a coerência topológica do passo de simplificação que está sendo executado.

Este critério valida ou não a operação de contração de aresta antes mesmo que ela seja executada, ou seja, se a condição 1 não for satisfeita, a aresta não será contraída (ver Figura 3.8).

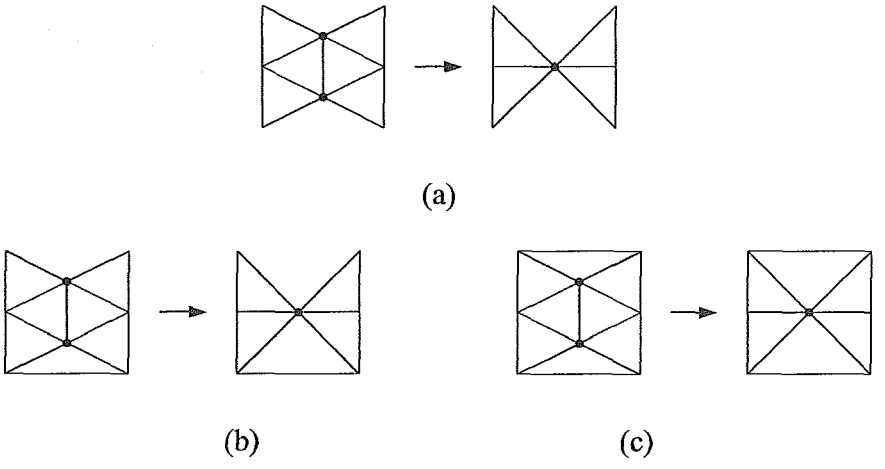


Figura 3.10: Exemplos de casos de contração de aresta com diferentes relações de incidência sobre seus vértices delimitadores.

3.4.3 Teste de variedade

Em modelos com borda, é necessário inspecionar a vizinhança da aresta a ser contraída para evitar que o modelo se torne uma não variedade depois da execução da contração de aresta.

Como podemos verificar na Figura 3.10, quando uma aresta possui os seus 2 vértices delimitadores sobre a borda da superfície, ela não poderá ser contraída para não gerar um vértice cuja vizinhança não é topologicamente equivalente a um disco aberto do E^2.

3.4.4 Consistência geométrica

Nesta seção, vamos descrever os testes efetuados sobre o modelo para que se possa garantir a consistência geométrica local.

Quatro possíveis situações perigosas devem ser evitadas no processo de simplificação:

- Dobra ou inversão de tetraedro;
- Faces adjacentes delimitando volume zero;
- Faces muito finas;
- Auto-interseção.

Os três primeiros casos são fáceis de verificar, porém a detecção de auto-interseção é uma tarefa mais complexa, porque esta é a única situação onde os efeitos de uma operação de contração de aresta podem ser não-locais [9]. Depois da execução de uma contração de aresta, algumas faces topologicamente não adjacentes mas geometricamente muito próximas

podem se interceptar. Em geral, a prevenção deste tipo de degeneração pode envolver o uso de técnicas mais elaboradas [10] e é difícil de ser implementada de forma eficiente e correta sem a utilização de estruturas de dados espaciais específicas [9]. Estas estruturas usam o princípio de particionar o espaço ou o modelo de forma a fazer testes globais de auto-interseção mais eficientes e mais rápidos. As estruturas de particionamento do espaço mais conhecidas são: grade de *voxels*, *octrees*, árvores *k*-d e árvores BSP(*Binary Space Partitioning*). Exemplos de estruturas de particionamento do modelo são: esferas limitante, árvores AABB (*Axis-Aligned Bounding Boxes*), árvores *k*-DOP (Discrete-Orientation Polytopes) e árvores OBB (*Oriented Bounding Boxes*).

Antes de detalharmos os testes em si, vamos fazer aqui algumas definições que serão úteis. Seja $hCol$ a aresta a ser contraída no ponto p e v_0 e v_1 seus dois vértices delimitadores.

Região de influência de $hCol$: conjunto de triângulos incidentes em v_0 ou v_1 (ver Fig. 3.11).

Limite de contração de $hCol$: conjunto de arestas que formam a borda da região de influência de $hCol$ (ver Fig. 3.11).

Fecho de v_0 (ou v_1): arestas do limite de contração de $hCol$ que delimitam faces não incidentes em v_1 (ou v_0).

Triângulos auxiliares de $hCol$: Para cada aresta do limite de contração de $hCol$, triângulos definidos pelos dois vértices delimitadores dessa aresta e um vértice na posição p.

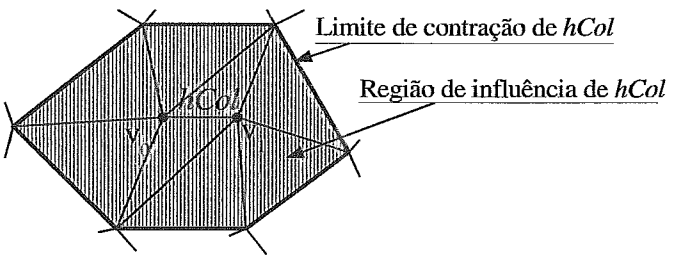


Figura 3.11: Região de influência e limite de contração de $hCol$.

Verificação de dobra

O teste mais comum de consistência está relacionado ao problema de inversão da malha, ou o que simplesmente chamamos de *dobra*. Considere a contração ilustrada na Figura 3.12, onde as coordenadas do ponto p coincidem com as coordenadas do vértice v_1.

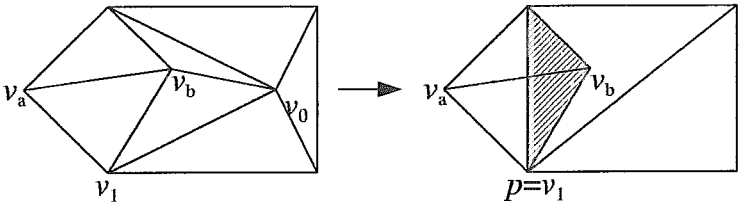


Figura 3.12: Contração de aresta resultando em dobra.

Esta escolha de p faz com que a malha dobre sobre si mesma (área escura). A prevenção deste tipo de problema será feita adotando o critério utilizado por Garland em [18], o qual faz uma verificação cuidadosa e oferece mais confiança na prática [8] [13] do que outros critérios anteriormente propostos.

Para cada aresta do fecho de v_0 (v_1) definimos um plano que a contém e é perpendicular à face correspondente incidente em v_0 (v_1). O ponto p e o vértice v_0 (v_1) devem estar do mesmo lado de cada um desses planos. A figura 3.13 ilustra esses planos e as setas indicam a região sobre a qual p deve recair.

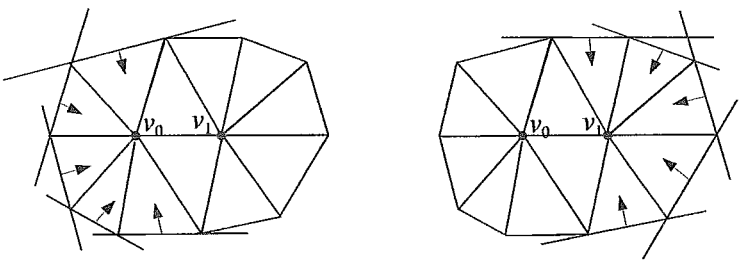


Figura 3.13: Planos perpendiculares através das arestas ao redor de v1 e v2 definem a zona na qual p deve estar para prevenir uma dobra.

Assim, se para algum desses planos o teste falhar, a contração da aresta delimitada por v_0 e v_1 não é efetuada.

Verificação de volume zero

Sobre o conjunto de triângulos auxiliares, verificamos se existe algum par adjacente desses triângulos tal que o volume definido por eles seja próximo de zero, ou seja, o ângulo entre eles é muito pequeno (menor que algum valor limite estipulado). Caso isto aconteça, a contração não é permitida.

Verificação de triângulo fino

Triângulos que possuem ângulos muito pequenos, ou seja, triângulos finos são, muitas vezes, indesejáveis em algumas aplicações.

Conforme proposto por Guéziec [21] [22], a qualidade γ de um triângulo de área a e comprimentos de lados l_0, l_1 e l_2 é dada por

$$\gamma = \frac{4\sqrt[2]{3}a}{l_0^2 + l_1^2 + l_2^2}. \tag{3.5}$$

Esta qualidade associa o valor 1 a um triângulo equilátero e 0 a um triângulo cujos vértices são colineares, sugerindo uma medida para a compacidade de um triângulo.

Assim, medimos a qualidade de todos os triângulos do conjunto de triângulos auxiliares de $hCol$ e verificamos se γ é maior do que um limite pré estabelecido γ_{min} para que a operação de contração possa ser efetuada. Se $\gamma < \gamma_{min}$ então a operação de contração da aresta $hCol$ é abortada.

Verificação de auto-interseção

Sobre o conjunto de triângulos auxiliares de $hCol$, fazemos uma varredura comparando cada um contra todos os outros, verificando num primeiro momento a orientação dos vértices e, se preciso, realizando testes de interseção geométrica.

Dados dois triângulos, t_1 e t_2, o primeiro teste é feito verificando-se se todos os vértices de um dos triângulos estão de um mesmo lado do plano definido pelo outro triângulo. Em caso positivo, não temos nenhum problema, já que t_1 e t_2 não se interceptam. Caso contrário, ou seja, t_1 contém vértices em lados opostos (ou sobre) do plano de t_2 (ou vice-versa), é preciso verificar se os triângulos se interceptam. Caso os triângulos se interceptem, a contração não é permitida.

3.5 O esquema de representação

Uma maneira comum de se representar uma malha poligonal é através de uma lista de vértices e uma lista de faces onde cada face é descrita como uma circulação de ponteiros para a lista de vértices.

Abaixo temos a lista de vértices e faces do cubo ilustrado na Figura 3.14.

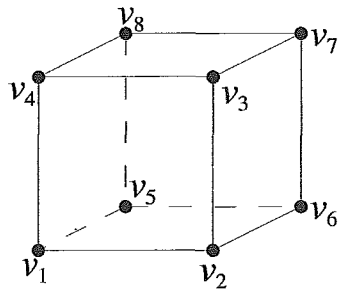


Figura 3.14: Representação de um cubo por lista de vértices e faces.

Vértice	Coordenadas
v_1	$x_1\ y_1\ z_1$
v_2	$x_2\ y_2\ z_2$
v_3	$x_3\ y_3\ z_3$
v_4	$x_4\ y_4\ z_4$
v_5	$x_5\ y_5\ z_5$
v_6	$x_6\ y_6\ z_6$
v_7	$x_7\ y_7\ z_7$
v_8	$x_8\ y_8\ z_8$

Face	Vértices
f_1	$v_1\ v_2\ v_3\ v_4$
f_2	$v_6\ v_2\ v_1\ v_5$
f_3	$v_1\ v_4\ v_8\ v_5$
f_4	$v_2\ v_6\ v_7\ v_3$
f_5	$v_4\ v_3\ v_7\ v_8$
f_6	$v_8\ v_7\ v_6\ v_5$

Esta representação é conveniente e eficiente em muitas aplicações, porém em alguns domínios ela pode não ser útil. Numa operação de contração de aresta, por exemplo, as faces incidentes à aresta em questão são removidas e as faces que compartilham os vértices dessa aresta precisam ser atualizadas. Este tipo de "cirurgia" sobre a malha requer que se descubra relações de adjacência entre componentes da malha, como faces e vértices incidentes. Embora essa busca possa ser feita sobre a representação mencionada acima, ela seria custosa, uma vez que requer uma varredura sobre toda a lista de faces e/ou lista de vértices.

Outros tipos de consultas de adjacências sobre malhas poligonais (ver Figura 3.15) incluem: Quais faces incidem sobre um vértice? Quais arestas incidem sobre um vértice? Quais faces incidem sobre uma aresta? Quais arestas delimitam uma face? Quais as faces adjacentes a uma dada face? Para implementar eficientemente tais consultas, foram desenvolvidas representações por borda mais sofisticadas, as quais modelam explicitamente vértices, arestas e faces da malha com informações adicionais de adjacência armazenadas nesses elementos.

Uma das implementações mais comuns desse tipo de representação é a estrutura de dados *Winged-Edge* [3] [4], onde as arestas contêm ponteiros para os seus dois vértices, ponteiros para as duas faces que incidem sobre ela e ponteiros para as arestas que incidem sobre os seus

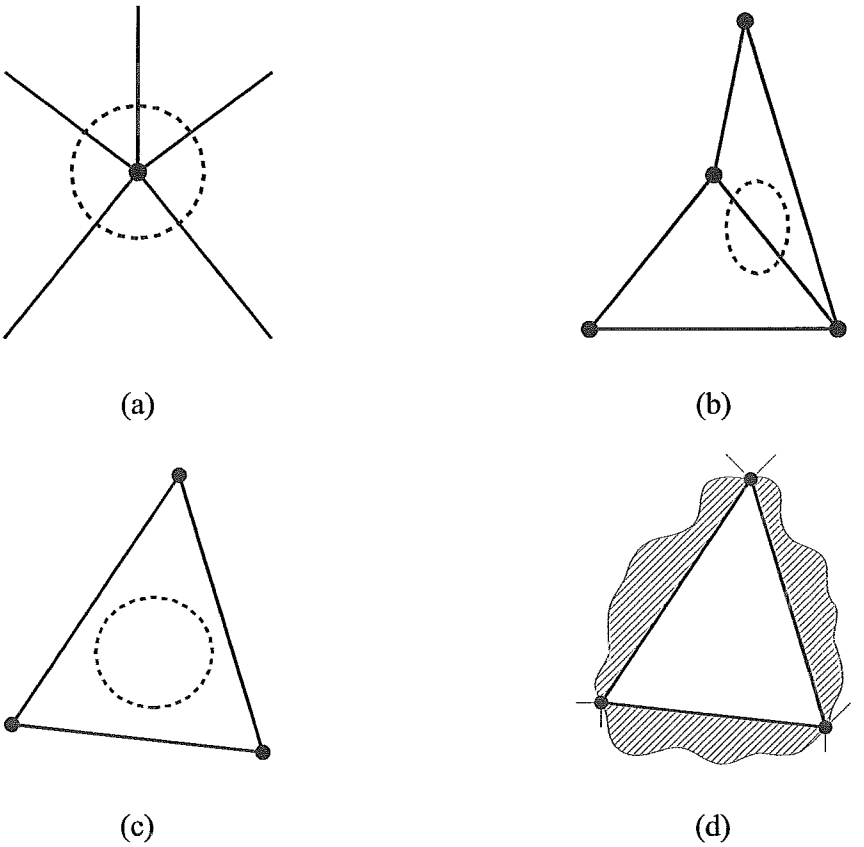


Figura 3.15: Exemplos de relações de incidência numa malha: (a) Arestas ao redor de um vértice (b) Faces ao redor de uma aresta (c) Arestas ao redor de uma face (d) Faces ao redor de uma face (por adjacência de aresta).

vértices delimitadores. Esta estrutura permite que se encontre as faces ou vértices incidentes sobre uma aresta em tempo constante. É claro que isso não exclui a possibilidade de que outros tipos de consultas venham a ter um custo de processamento mais alto.

Como variação da *Winged-Edge*, surgiu a estrutura de dados *Half-Edge* que é uma representação por fronteira um pouco mais sofisticada. Ao invés de ter o seu foco sobre as arestas, seu elemento principal é uma semi-aresta. Além disso, permite também todas as consultas listadas acima (assim como outras) em tempo constante, ou melhor, $O(n)$.

Esta propriedade faz da *Half-Edge* uma excelente escolha para muitas aplicações, embora ela só seja capaz de representar variedades. No nosso caso, essa restrição não é um problema, uma vez que os objetos representados são sólidos poliédricos válidos.

3.5.1 A estrutura *Half-Edge*

A estrutura *Half-Edge* é uma variação da estrutura de dados *Winged-Edge*, na qual cada aresta é desdobrada em duas *semi-arestas* de sentidos opostos. Assim, cada semi-aresta tem um sentido definido, determinando sobre os seus vértices delimitadores um de origem e um de destino.

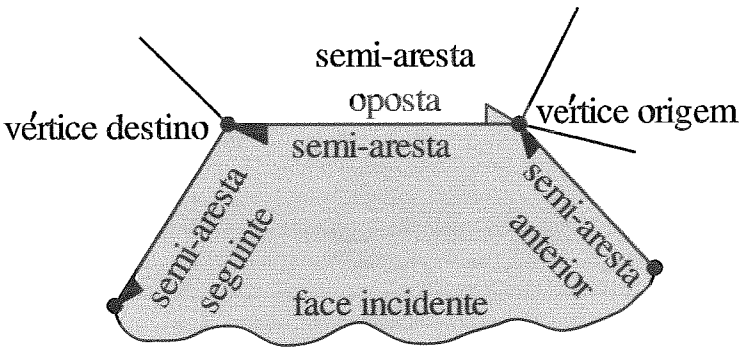


Figura 3.16: Esquema da estrutura *Half-edge*.

Como pode ser visto na figura 3.16, as semi-arestas que delimitam uma face formam uma lista encadeada circular sobre todo o perímetro desta face. No nosso caso, essa lista estará sempre orientada no sentido anti-horário. Cada uma das semi-arestas desse ciclo armazenam um ponteiro para a face incidente, um ponteiro para sua semi-aresta oposta e um ponteiro para seus vértices inicial e final.

Então, dada uma semi-aresta h, vamos adotar a seguinte notação:

$h \rightarrow oposta$: semi-aresta oposta à h;

$h \rightarrow prev$: na mesma face incidente em h, é a semi-aresta anterior.

$h \rightarrow prox$: na mesma face incidente em h, é a semi-aresta seguinte.

Existem algumas bibliotecas que implementaram a estrutura *Half-Edge*, como por exemplo a biblioteca CGAL [6], que utilizamos no desenvolvimento deste trabalho.

3.6 A estrutura de dados proposta

O processo de simplificação de malhas aqui proposto é feito através da aplicação iterativa de operações sobre arestas, $oper_i$, que efetuam uma modificação local sobre a malha inicial:

$$(\hat{M} = M^n) \overset{oper_{n-1}}{\rightarrow} \ldots \overset{oper_1}{\rightarrow} M^1 \overset{oper_0}{\rightarrow} M^0. \tag{3.6}$$

Mais precisamente, $oper_i$ define uma operação de contração de aresta, embora possam ser utilizadas outras operações como troca (*flip*) e partição de aresta (veja Fig. 2.9).

Seja M_n uma malha inicial. Nosso esquema de representação em multi-resolução é constituído de uma malha simplificada M_0 e uma seqüência de informações a respeito das operações executadas no processo de simplificação da malha M_n. Essas informações permitem a execução da operação inversa àquela utilizada no processo de simplificação, ou seja, no caso específico de contração de arestas seria a partição dos vértices da malha simplificada de forma que se recupere corretamente a malha original:

$$M^0 \overset{oper_0^{-1}}{\rightarrow} \ldots \overset{oper_1^{-1}}{\rightarrow} M^1 \overset{oper_{n-1}^{-1}}{\rightarrow} (M^n = \hat{M}). \tag{3.7}$$

Essa estrutura de representação foi inicialmente proposta por Hoppe em [26], definindo o que foi chamado de representação por *malhas progressivas* (PM). A representação PM de uma malha M^n define uma seqüência contínua de malhas M^0, M^1, ..., M^n representando aproximações cada vez melhores. Ou seja, pode-se extrair aproximações em níveis de detalhe de forma eficiente, a partir da execução de uma seqüência contínua de operações armazenadas. Porém a grande vantagem que queremos explorar utilizando este tipo de representação é que além de permitir a extração de aproximações em níveis de detalhe de forma eficiente, ela permite também transmissão progressiva e refinamento seletivo.

Assim, durante o processo de simplificação de uma malha inicial M_n, para cada operação de contração de aresta executada sobre M_n, armazenamos as informações que identificam o novo vértice, as arestas que permitem a inversão correta dessa operação (seção 3.2), além de algumas propriedades da aresta contraída.

No nosso esquema de representação, cada vértice v_i de uma malha é identificado por um número inteiro que é associado seqüencialmente de acordo com a ordem de aparecimento

(criação) do vértice. Cada semi-aresta será identificada pelos seus vértices delimitadores seguindo sua orientação: v_0 indica o vértice de origem e v_1 representa o vértice de destino.

Então, em cada elemento $oper_i$ da seqüência de operações são armazenadas as seguintes informações (ver Figura 3.17):

- Inteiro identificador do tipo de operação efetuada;
- Identificador do vértice v gerado pela contração de $hCol$;
- Identificadores dos vértices delimitadores de h: v_{0h} e v_{1h};
- Identificadores dos vértices delimitadores de g: v_{0g} e v_{1g};
- Coordenadas e propriedade livre dos vértices v_0 e v_1 gerados pela partição de v;

Os identificadores dos vértices delimitadores da aresta $hCol$ contraída, v_0 e v_1, estão explicitamente representados por v_1g e v_1h, respectivamente.

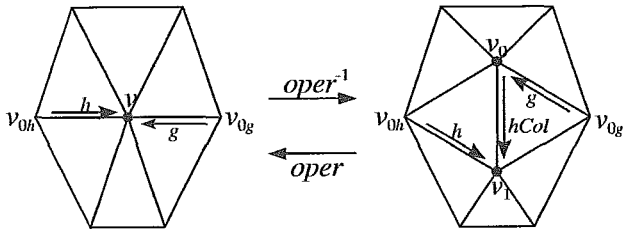


Figura 3.17: Operação de partição de vértice e contração de aresta com as informações que precisam ser armazenadas.

3.6.1 Comparação com a operação *vsplit*

De acordo com a seção 3.2 cada operação de partição de vértice (veja Fig. 3.18) modifica a malha introduzindo um novo vértice v e duas novas faces $f_l = \{v_0, v_l, v_1\}$ e $f_r = \{v_1, v_r, v_0\}$.

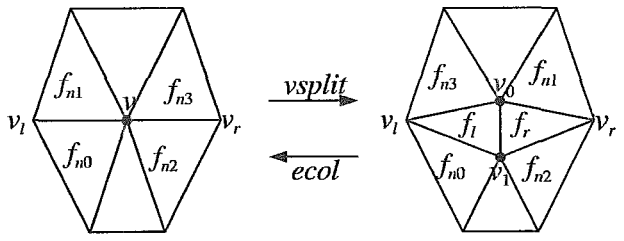


Figura 3.18: Operação de partição de vértice *vsplit* definida por Hoppe.

A operação *vsplit* inicialmente proposta por Hoppe em [26] era parametrizada da seguinte forma: $vsplit(v, v_l, v_r, v_0, f_l, f_r)$. De forma a se poder fazer refinamento seletivo sobre uma malha progressiva, Hoppe introduziu uma nova parametrização:

$$vsplit(v, v_1, v_0, f_l, f_r, f_{n0}, f_{n1}, f_{n2}, f_{n3}). \tag{3.8}$$

A operação *oper* aqui definida é semanticamente equivalente à operação de partição de vértice *vsplit* definida por Hoppe, mas sua parametrização é escrita da seguinte forma:

$$oper(v, e_h, e_g), \tag{3.9}$$

onde e_h e e_g são as arestas respectivamente definidas pelos pares de vértices (v_l, v_1) e (v_r, v_0). Mais precisamente, e_h e e_g são semi-arestas, uma vez que têm um sentido definido que deve ser respeitado. Essa parametrização com menos informações só é possível porque a estrutura de dados utilizada para representação de um modelo encapsula as relações de incidência necessárias para que a operação de partição de vértice possa ser executada.

3.7 Refinamento seletivo

A simples execução de uma lista de operações em ordem inversa só permite a visualização dos níveis de detalhe definidos pelo próprio processo de simplificação. Ou seja, uma malha simplificada acompanhada de uma seqüência de n operações permitem diretamente a extração ou visualização de várias representações da malha original em diferentes níveis de detalhe (no máximo n), mas todas são modelos de domínio uniforme.

Quando estamos trabalhando com refinamento de um modelo grosseiro, é desejável realizar o refinamento de forma seletiva. Assim, de acordo com a posição do observador, que é definida pelo usuário, somente a região do modelo que se encontra dentro do volume de visão determinado deve ser detalhada.

Para que isso possa ser implementado eficientemente, não podemos nos basear na lista de operações, mas sim utilizar uma outra estrutura de dados que permita a execução de um passo de refinamento sem que para isso seja necessária a execução de todos os anteriores, que muitas vezes podem estar refinando regiões que não nos interessam, mas precisam ser executados para que já possam ser criados os vértices requisitados por passos subseqüentes.

Desta forma, nossa proposta consiste em construir a hierarquia de vértices discutida na seção 2.7.1 a partir da nossa lista de operações. Detalhes desta construção serão feitos mais adiante na seção 4.5.

Capítulo 4

Implementação

4.1 Introdução

Desenvolvemos três aplicações, onde cada uma delas aborda:

- Simplificação;
- Visualização em níveis de detalhe;
- Refinamento seletivo.

Os programas foram desenvolvidos em linguagem C utilizando-se a biblioteca OpenGL [39] [37] para a visualização gráfica e GLUI [41] para construção das interfaces.

Para auxílio na implementação de algumas operações e estruturas de dados utilizadas, usamos as bibliotecas STL[48] e CGAL[6].

A biblioteca STL (*Standard Template Library*) é uma biblioteca C++ padrão que implementa algumas estruturas de dados fundamentais tais como listas, filas de prioridade, tabelas de espalhamento (*hash*), etc.

A biblioteca CGAL provê a implementação de uma série de estruturas de dados e algoritmos básicos da geometria computacional, além das próprias primitivas geométricas tais como pontos, vetores e linhas.

4.2 A Estrutura de representação

A representação de um modelo na nossa implementação utiliza a classe *Polyhedron* da biblioteca CGAL. Esta classe encapsula uma implementação da estrutura de dados *Half-Edge*.

Desta forma, cada aresta é representada por duas semi-arestas com orientações opostas. As faces são polígonos planares, sem buracos, definidas pela seqüencia circular de semi-arestas

ao longo de sua borda. A superfície poliédrica em si pode conter buracos. As semi-arestas ao redor de um buraco são chamadas arestas de borda e não têm face incidente. Uma aresta é de borda se uma de suas semi-arestas é de borda. Quando uma superfície não contém nenhuma semi-aresta de borda, ela é classificada como fechada. Por convenção, as semi-arestas de uma face são sempre orientadas no sentido anti-horário.

4.3 Os Operadores de Euler

Como numa operação de contração de aresta faz-se apenas a remoção ou adição de vértices, arestas ou faces, apenas quatro operadores de Euler são utilizados para efetivar uma operação de contração de aresta: MEV, KEV, MEF, KEF.

A utilização dessas operações foi através da funções já implementadas no CGAL (ver Figura 4.1):

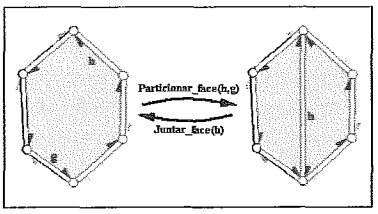
Particionar_vértice(Semi-aresta h, Semi-aresta g): Divide em dois o vértice ao qual h e g incidem, criando uma nova aresta entre eles. Corresponde ao operador de Euler MEV.

Juntar_vértice(Semi-aresta h): junta os dois vértices incidentes a h em um só, removendo a aresta da semi-aresta h. É a função inversa à de particionar vértice, ou seja, corresponde ao operador de Euler KEV.

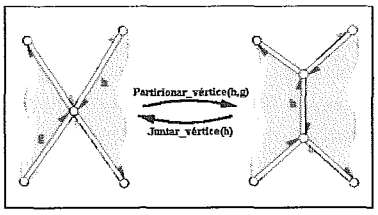
Particionar_face (Semi-aresta h, Semi-aresta g): Divide a face incidente a h e g em duas, criando uma diagonal entre os vértices das duas arestas informadas. Corresponde ao operador de Euler MEF.

Juntar_face (Semi-aresta h): junta as faces incidentes a semi-aresta h e a sua semi-aresta oposta, removendo a aresta definida por estas duas semi-arestas. Executa a operação inversa à de particionar face, correspondendo assim ao operador KEF.

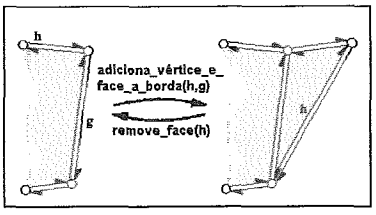
Adicionar_vértice_e_face_à_borda (Semi-aresta h, Semi-aresta g): Cria uma nova face incidindo em g tal que as outras duas arestas da nova face passam a compor a borda do modelo em seqüência à aresta h. Corresponde a uma combinação dos operadores de Euler MEV e MEF.



(a)



(b)



(c)

Figura 4.1: Funções de implementação dos operadores de Euler. Figuras extraídas do manual da biblioteca CGAL [6].

4.4 Classificação de casos de contração de arestas e seus inversos

Num caso genérico, utilizando-se malhas triangulares, quando uma aresta é contraída são removidos três arestas, dois vértices e duas faces e um novo vértice é criado.

Em função de nos propormos a trabalhar com modelos abertos, ou seja, que têm arestas de borda, a operação de contração de aresta pode ser dividida em três casos, conforme a configuração de seus vértices: ambos de borda; um vértice de borda e o outro não (vértice interno); e nenhum de borda ou ambos internos. É claro que os últimos casos poderiam ser classificados num só: pelo menos um vértice de borda. Mas no passo adiante de refinamento, as operações inversas desses dois casos devem ser tratadas de forma diferente, por isso o desmembramento.

Cada um desses casos deve ser ainda desmembrado, levando-se em consideração que a estrutura de dados utilizada é baseada em semi-arestas, ao invés de arestas.

Assim, a operação de contração de aresta foi dividida nos seguintes casos, de acordo com a classificação da aresta $hCol$ a ser contraída:

1. h pertence borda:

   (a) $hCol \rightarrow prox$ não é borda;

   (b) $hCol \rightarrow prev$ não é borda;

2. h não pertence à borda:

   (a) Não existem arestas de borda incidindo sobre v_0 ou v_1;

   (b) Não existem arestas de borda ao redor de v_0:

      i. $hCol{\rightarrow}prev$ e $hCol{\rightarrow}oposto{\rightarrow}prox$ NÃO têm faces em comum;

      ii. $hCol{\rightarrow}prev$ e $hCol{\rightarrow}oposto{\rightarrow}prox$ TÊM faces em comum;

   (c) Não existem arestas de borda ao redor de v_1:

      i. $hCol{\rightarrow}prox$ e $hCol{\rightarrow}oposto{\rightarrow}prev$ NÃO têm faces em comum;

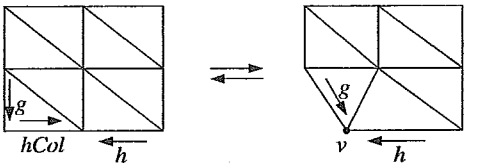
      ii. $hCol{\rightarrow}prox$ e $hCol{\rightarrow}oposto{\rightarrow}prev$ TÊM faces em comum.

A Figura 4.2 exibe cada um desses casos, identificando as respectivas arestas h e g que devem ser armazenadas.

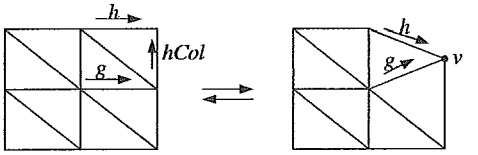
4.4.1 Combinação dos operadores de Euler

Na operação de contração de aresta, os operadores de Euler são utilizados para manipular a malha dando o resultado de contração esperado. Assim, para cada um dos casos de contração de aresta classificado temos combinações de diferentes desses operadores a serem usadas.

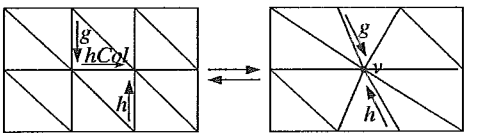
A tabela a seguir indica para cada caso as operações que são utilizadas, assim como também as operações do processo inverso (partição de vértice), uma vez que este também será executado segundo o caso da operação original.



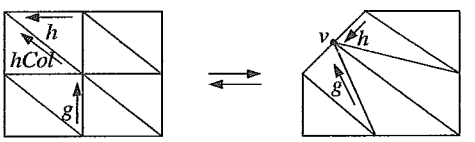
(Caso 1.a)



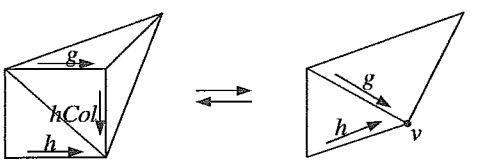
(Caso 1.b)



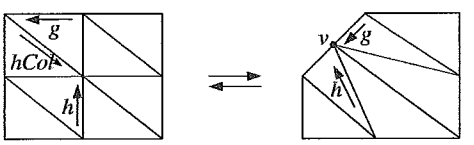
(Caso 2.a)



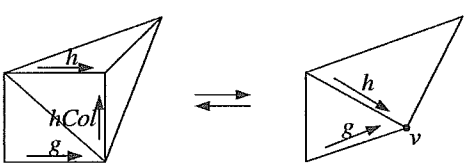
(Caso 2.b.i)



(Caso 2.b.ii)



(Caso 2.c.i)



(Caso 2.c.ii)

Figura 4.2: Mapa dos casos de contração de aresta.

Caso	Operações para efetuar contração de aresta	Operações para efetuar partição de vértice
1.(a)	Juntar_face($hCol \rightarrow prox$) Juntar_vértice($hCol$)	Particionar_vértice(h, g) Particionar_face($g \rightarrow prev$, $g \rightarrow prox$)
1.(b)	Juntar_face($hCol \rightarrow prev$) Juntar_vértice($hCol$)	Particionar_vértice(h, g) Particionar_face(g, $g \rightarrow prox \rightarrow prox$)
2.(a)	Juntar_face($hCol \rightarrow prev \rightarrow oposta$) Juntar_face($hCol \rightarrow oposta \rightarrow prev \rightarrow oposta$) Juntar_vértice($hCol$)	Particionar_vértice(h, g) Particionar_face($h \rightarrow prev$, $h \rightarrow prox$) Particionar_face($g \rightarrow prev$, $g \rightarrow prox$)
2.(b).i	Juntar_face($hCol \rightarrow prev \rightarrow oposta$) Juntar_face($hCol \rightarrow oposta \rightarrow prox \rightarrow oposta$) Juntar_vértice($hCol$)	Parcionar_vértice(h, g) Particionar_face($h \rightarrow prev$, $h \rightarrow prox$) Particionar_face(g, $g \rightarrow prox \rightarrow prox$)
2.(b).ii	Juntar_face($hCol \rightarrow prev \rightarrow oposta$) Juntar_face($hCol \rightarrow oposta \rightarrow prev \rightarrow oposta$) Juntar_vértice($hCol$)	
2.(c).i	Juntar_face($hCol \rightarrow prox$) Juntar_face($hCol \rightarrow oposta \rightarrow prev \rightarrow oposta$) Juntar_vértice($hCol$)	Parcionar_vértice(h, g) Particionar_face(h, $h \rightarrow prox \rightarrow prox$) Particionar_face($g$, $g \rightarrow prox \rightarrow prox$)
2.(c).ii	Juntar_face($hCol \rightarrow prev \rightarrow oposta$) Juntar_face($hCol \rightarrow oposta \rightarrow prev \rightarrow oposta$) Juntar_vértice($hCol$)	

As operações efetuadas para partição de vértice nos casos 2.(b).ii e 2.(c).ii são dependentes de condições de vizinhança:

1. Se h é aresta de borda, então utilizamos o procedimento Adicionar_face_a_borda(h, $h \rightarrow prox \rightarrow prox$). Caso contrário, a operação Particionar_face(h, $h \rightarrow prox \rightarrow prox$) é executada.

2. Se g é aresta de borda, então efetuamos o procedimento Adicionar_face_a_borda(g, $g \rightarrow prox \rightarrow prox$). Caso contrário, utilizamos a operação Particionar_face(g, $g \rightarrow prox \rightarrow prox$).

4.5 Construção da hierarquia de vértices para refinamento seletivo

A hierarquia de vértices que permite fazer refinamento seletivo é implementada como um vetor de elementos tVertexHierarchy:

```
estrutura tVertexHierarchy
{
    unsigned int id_v;  // inteiro identificador do vértice
    Point3 geom;        // coordenadas (x, y, z)
                        // do vértice
    int pos_id_pai;     // posição do vértice pai
                        // na estrutura
    int pos_id_vh[2];   // posição dos vértices da aresta
                        // h na estrutura
    int pos_id_vg[2];   // posição dos vértices da aresta
                        // g na estrutura
    unsigned int caso;  // inteiro identificador do caso
                        // de contração que gerou este vértice
    bool livre;         // indica se o vértice tem
                        // coordenadas livre
    bool refinado;      // indica se o vértice já foi
                        // refinado (percorrido na estrutura)
};
```

Sua construção é feita a partir do modelo simplificado e da seqüencia de operações de contração de aresta. Essa construção começa pelos vértices do modelo simplificado, que vão preenchendo as primeiras posições do vetor. Neste momento, os identificadores de pai e filhos são configurados para nulo. Em seguida, para cada registro do arquivo de operações, que contém os vértices v, v_{0h}, v_{1h}, v_{0g}, v_{1g}, os vértices que ainda não existem no vetor são inseridos e as relações de parentesco (pai/filhos) e as propriedades dos vértices (coordenadas e propriedade livre) são atualizadas.

Como exemplo, considere a Figura 4.3 que ilustra um processo de simplificação de uma malha contendo 20 vértices.

O arquivo de operações gerado contém os seguintes registros:

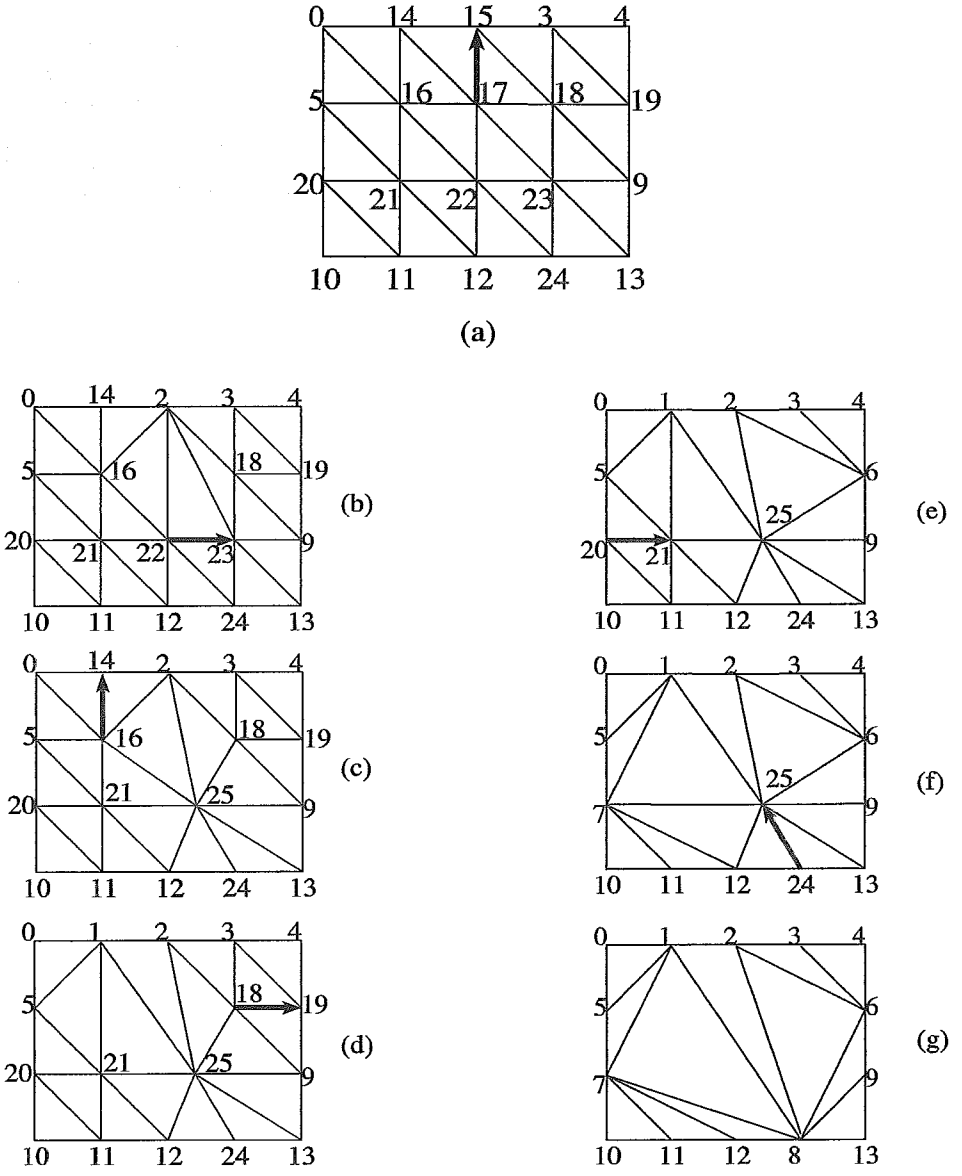


Figura 4.3: Processo de simplificação por contração de arestas.

Tipo	Caso	v	v_{0h}	v_{1h}	v_{0g}	v_{1g}	p_{0x}	p_{0y}	p_{0z}	p_0 livre?	p_{1x}	p_{1y}	p_{1z}	p_1 livre?
1	2.(c).i	8	9	25	12	24	150	2	150	0	125	12	100	1
1	2.(c).i	7	12	21	5	20	0	32	100	0	50	24	100	1
1	2.(b).i	6	9	19	2	18	150	22	50	1	200	27	50	0
1	2.(b).i	1	2	14	5	16	50	34	50	1	50	30	0	0
1	2.(a)	25	24	23	2	22	100	14	100	1	150	10	100	1
1	2.(b).i	2	18	15	16	17	100	19	50	1	100	22	0	0

O primeiro passo da construção da hierarquia de vértices, gera os nós raízes, conforme ilustrado na Fig. 4.4.

Figura 4.4: Hierarquia de vértices: vértices da malha grosseira são as raízes.

Em seguida, de acordo com o 1° registro do arquivo de operações, a árvore é atualizada e configura-se de acordo com a ilustração da Fig. 4.5.

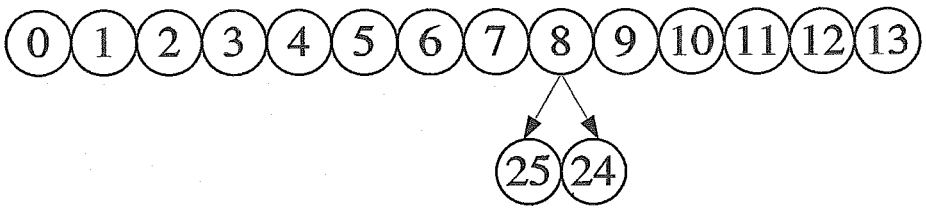


Figura 4.5: Hierarquia de vértices: Nível criado a partir da 1ª operação.

Seqüencialmente cada registro lido atualiza a árvore que no final terá a configuração ilustrada na Fig. 4.6.

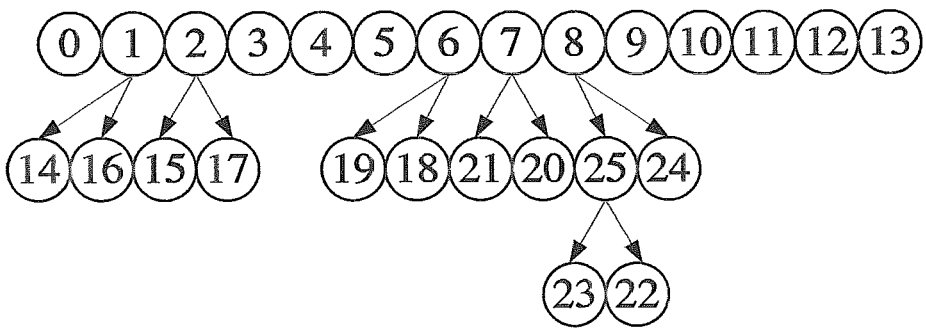


Figura 4.6: Hierarquia de vértices: configuração final.

Uma vez construída a hierarquia de vértices, as pré-condições para execução de um passo de refinamento são:

1. v está ativo;
2. v_{0h} e v_{0g} ativos ou algum ancestral ativo;
3. v não figura como v_{0h} ou v_{0g} de nenhum outro nó ainda não refinado.

A primeira condição é verificada imediatamente, uma vez que não se pode operar sobre um vértice que não esteja presente na malha corrente. A segunda condição, é necessária para que se possa fazer a partição do vértice de forma correta (discutido na seção 3.2). Se v_{0h} e/ou v_{0g} não são encontrados como vértices ativos, podemos usar o seu ancestral ativo, que vai estar conectado a v. Se ainda neste caso não houver nenhum ancestral ativo na malha corrente, significa que este vértice está particionado e, desta forma, não se pode fazer o refinamento de v. Por fim, a última condição existe para garantir que um vértice só será particionado se nenhum outro "precisa dele" para ainda ser refinado. Se não tivéssemos esta restrição, uma vez que um vértice v_1 ainda requerido por outro vértice v_2 fosse refinado, isto implicaria que v_2 nunca seria refinado, pela condição 2.

4.6 As aplicações desenvolvidas

As três aplicações desenvolvidas seguem um padrão na sua interface. Na direita sempre tem um painel com configurações e operações que podem ser efetuadas sobre o modelo. Em-baixo tem-se o painel de visualização, com elementos para manipulação da posição do objeto, configurações de forma de desenho dos polígonos da malha (aramado ou preenchido), entre outros. O modelo em si é ilustrado na parte principal dentro de uma janela GLUT [32].

Nesta seção vamos descrever cada uma dessas aplicações desenvolvidas.

4.6.1 Simplificação

O programa de simplificação tem como entrada um arquivo contendo a descrição de um modelo 3D (formato OFF) e, a partir de alguns parâmetros configurados na interface, efetua o processo de simplificação. O processo de simplificação de forma automática utiliza uma fila de prioridade para selecionar as arestas a serem contraídas segundo uma ordem definida pela métrica de contração escolhida pelo usuário (ver Fig. 4.7). Pode-se também efetuar um processo manual, contraindo-se uma aresta selecionada sem o auxílio da fila de prioridade (ver Fig. 4.8).

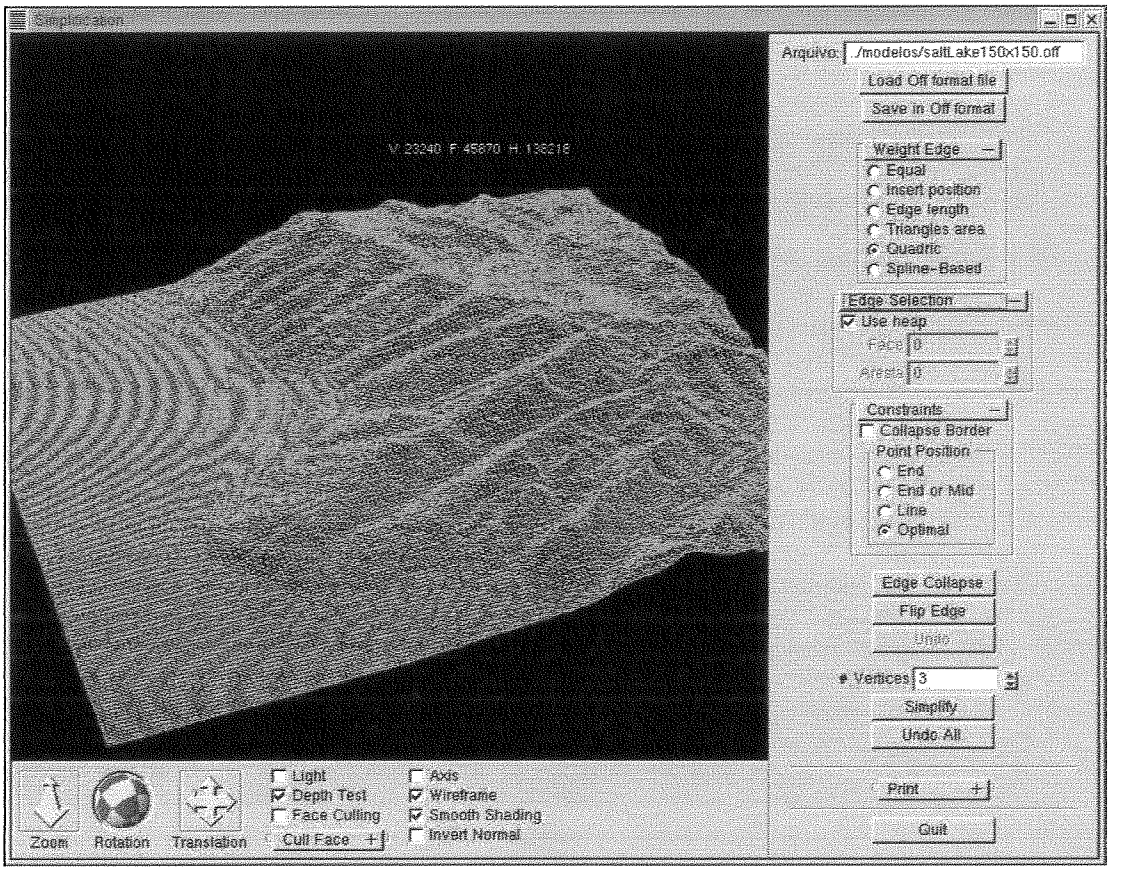


Figura 4.7: Interface da aplicação de simplificação.

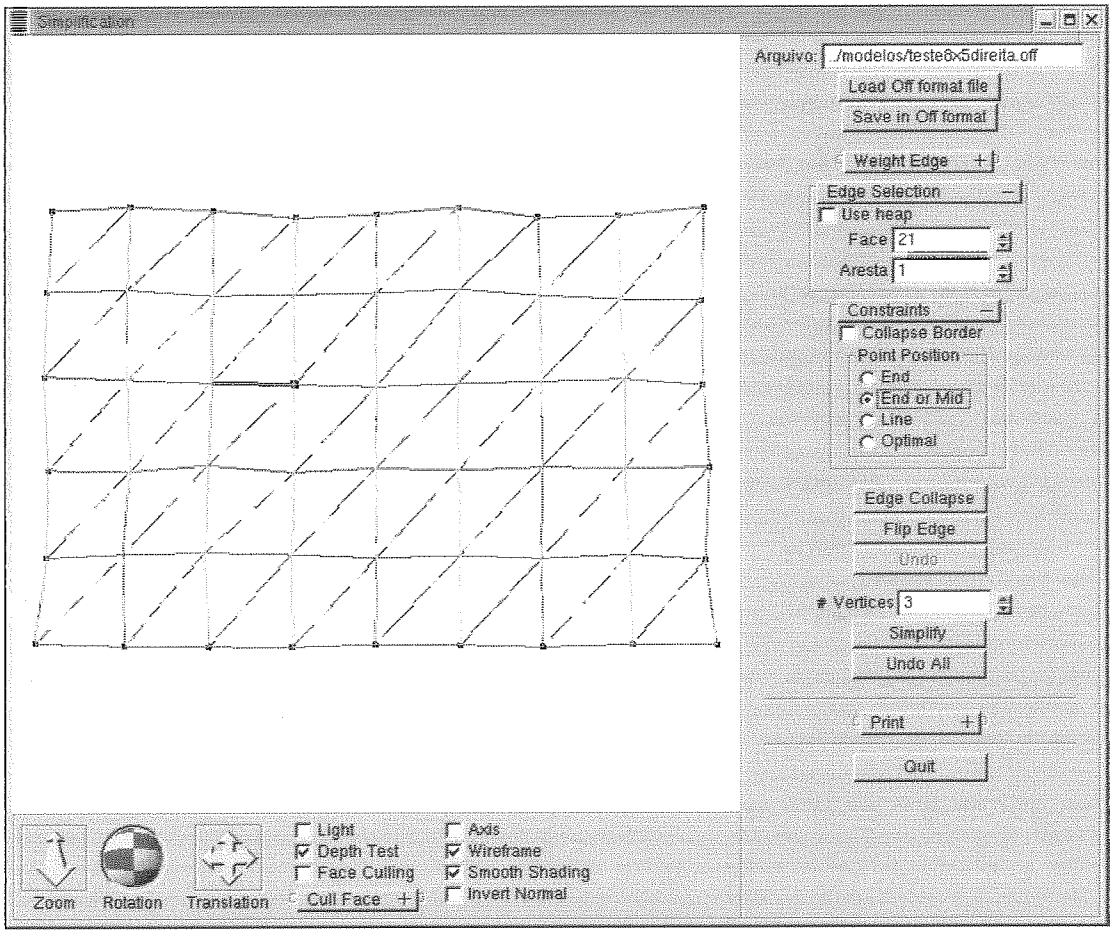


Figura 4.8: Interface da aplicação de simplificação indicando seleção manual da aresta a ser contraída.

A Fig. 4.7 exibe a aplicação efetuando processo de simplificação automático (opção "Use heap" ativa), definido pela métrica de erro quádrico (métrica definida no opção "Weight Edge"). Neste caso, as arestas que têm pelo menos um vértice de borda não podem ser contraídas (opção "Collapse Border" não está selecionada).

Os botões "Edge Collapse" e "Flip Edge" efetuam as operações de contração e troca, respectivamente, sobre a aresta selecionada, que está sempre indicada em verde com o seu vértice destino em azul. O botão "Undo" desfaz a última operação executada. Conforme ilustrado na Fig. 4.9, este botão só fica ativo depois que um operação tenha sido efetuada sobre a malha.

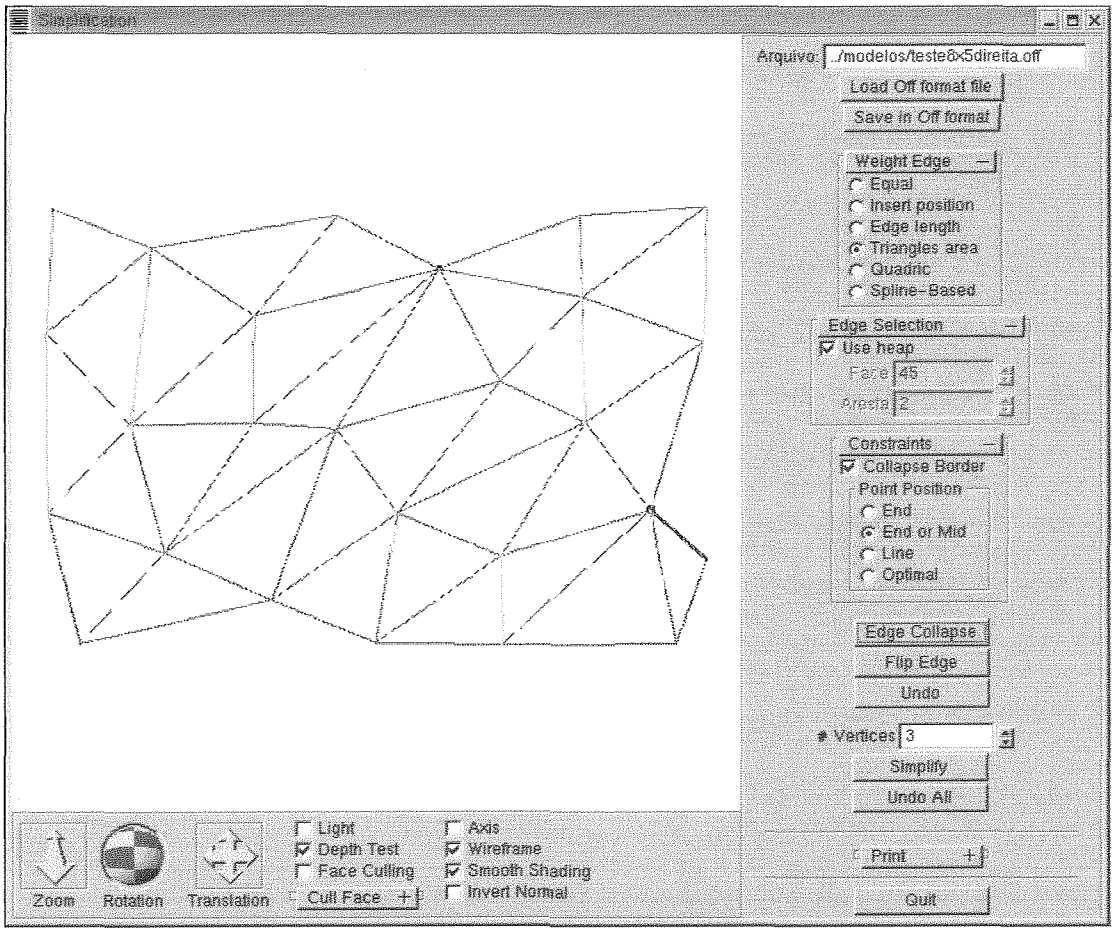


Figura 4.9: Interface da aplicação de simplificação onde os vértices sobre a borda do modelo estão tendo suas posições alteradas por operações de contração de arestas incidentes neles. Neste momento o modelo já sofreu algumas contrações e por isso o botão "Undo" está ativo, para eventual restauração de estados anteriores.

Relembrando, a propriedade livre de um vértice indica que a posição geométrica dele pode ser alterada. Normalmente, os vértices assim configurados são os que estão sobre a borda do modelo. O usuário pode ou não permitir que os vértices de borda sejam movidos quando uma contração é efetuada selecionando a opção "Collapse Border" presente na interface. Esta opção

configura a propriedade livre desses vértices com valor falso. Os vértices cuja propriedade livre é falsa, ficam indicados em vermelho na visualização do modelo. Um exemplo de modelo cujos vértices de borda não têm nenhuma restrição sobre suas posições (isto é, estão livres) pode ser visto na Fig. 4.9. Repare que, neste caso, a opção "Collapse Border" está selecionada.

Num processo de simplificação automática, o usuário deve configurar o peso da aresta (métrica para seleção de contração de aresta) e a restrição de posicionamento do vértice gerado pela contração. Em seguida, define-se o número de vértices que se deseja que o modelo simplificado tenha no final do processo. Feitas essas configurações, basta clicar no botão "Simplify" que o programa vai simplificar o modelo até que o número de vértices seja alcançado ou, se não for possível, até que não exista mais nenhuma aresta possível de ser contraída.

A saída do modelo simplificado em multi-resolução pode ser armazenada pressionando-se o botão "Save". Esse modelo simplificado é armazenado em dois arquivos: um de formato OFF (descrição do modelo) e um de formato OPL, que contém as informações das operações executadas em seqüência.

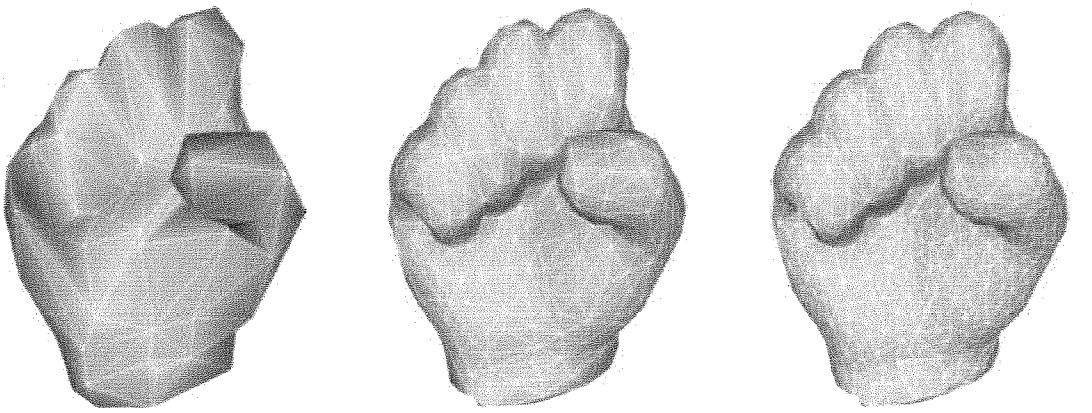
4.6.2 Visualização de níveis de detalhe

A fim de se visualizar um modelo nos diferentes níveis de detalhe extraídos diretamente da representação por malhas progressivas, desenvolvemos uma aplicação que carrega um modelo assim representado e permite que o usuário informe um número $m \leq n$ de operações a serem efetuadas sobre o modelo. n é o número de operações presente na malha progressiva.

A aplicação, então, executa os m registros da lista de operações ($m \leq n$ definido pelo usuário) tanto no sentido de refinamento quanto no sentido de simplificação, o que nos permite ver o modelo nos vários níveis de detalhe, porém lembramos que sempre de acordo com o pré-processo de simplificação executado.

A Figura 4.10 ilustra o processo de visualização de um modelo em diferentes níveis de detalhe dependente do processo de simplificação. O modelo grosseiro exibido na Fig. 4.10(a) contém 100 vértices. A partir da execução de 400 registros seqüenciais do arquivo de operações de simplificação obtemos o modelo da Fig. 4.10(b), que contém 500 vértices. A Fig. 4.10(c) exibe o modelo no nível de detalhe mais alto, com 1.340 vértices, recuperado a partir da execução de todos as n=1.240 operações.

A aplicação de visualização (ver Fig. 4.11) mostra o número de elementos topológicos da malha corrente em um painel de detalhes. O número de operações a serem executadas para se trocar um nível de detalhe por outro deve ser informado no parâmetro "Step". A troca do nível de detalhe de um modelo é feita utilizando-se os botões "+" e "-" do painel "Resolução", que



(a) (b) (c)

Figura 4.10: Através da seqüência de operações associada a um modelo grosseiro (esquerda), pode-se percorrê-la tanto no sentido de refinamento (da esquerda para a direita), quando no sentido de simplificação (da direita para a esquerda), que permite a visualização de aproximações mais grosseiras.

indica também o nível corrente em função do número de vértices atual com relação a número de vértices da malha detalhada.

4.6.3 Refinamento seletivo

A aplicação de refinamento seletivo, carrega um modelo em multi-resolução (arquivos .off e .opl) e efetua o refinamento dos vértices que estão dentro de uma região definida por um observador, conforme ilustram as Figuras 4.12 e 4.13.

Como já analisamos, no caso de refinamento seletivo a partição dos vértices de uma malha está dependente da estrutura de hierarquia de vértices já discutida nas seções 3.7 e 4.5. Em função disso, a aplicação pode fazer o refinamento de duas formas. A primeira delas é percorrendo uma única vez os vértices ativos para efetuar as partições possíveis. Isto é feito utilizando-se o botão "Single Pass" do painel "Refine Active Vertices". Porém isto não garante que todos os vértices sejam particionados, por causa da terceira pré-condição definida para a execução de uma passo de refinamento (ver seção 4.5). Assim, o botão "While Possible" do mesmo painel mencionado efetua a partição de todos os vértices ativos enquanto cada um deles ainda tiver "filhos" que podem ser refinados.

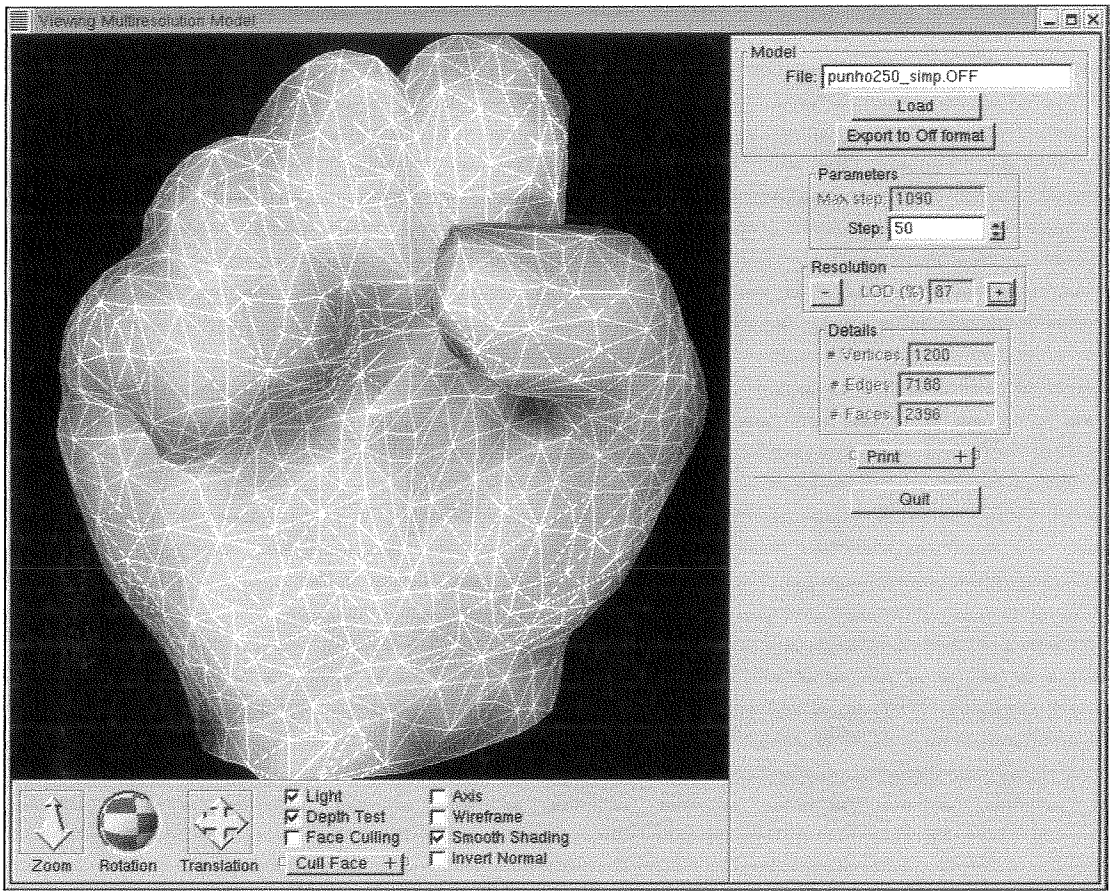


Figura 4.11: Aplicação de visualização dos níveis de detalhe de um modelo em multi-resolução.

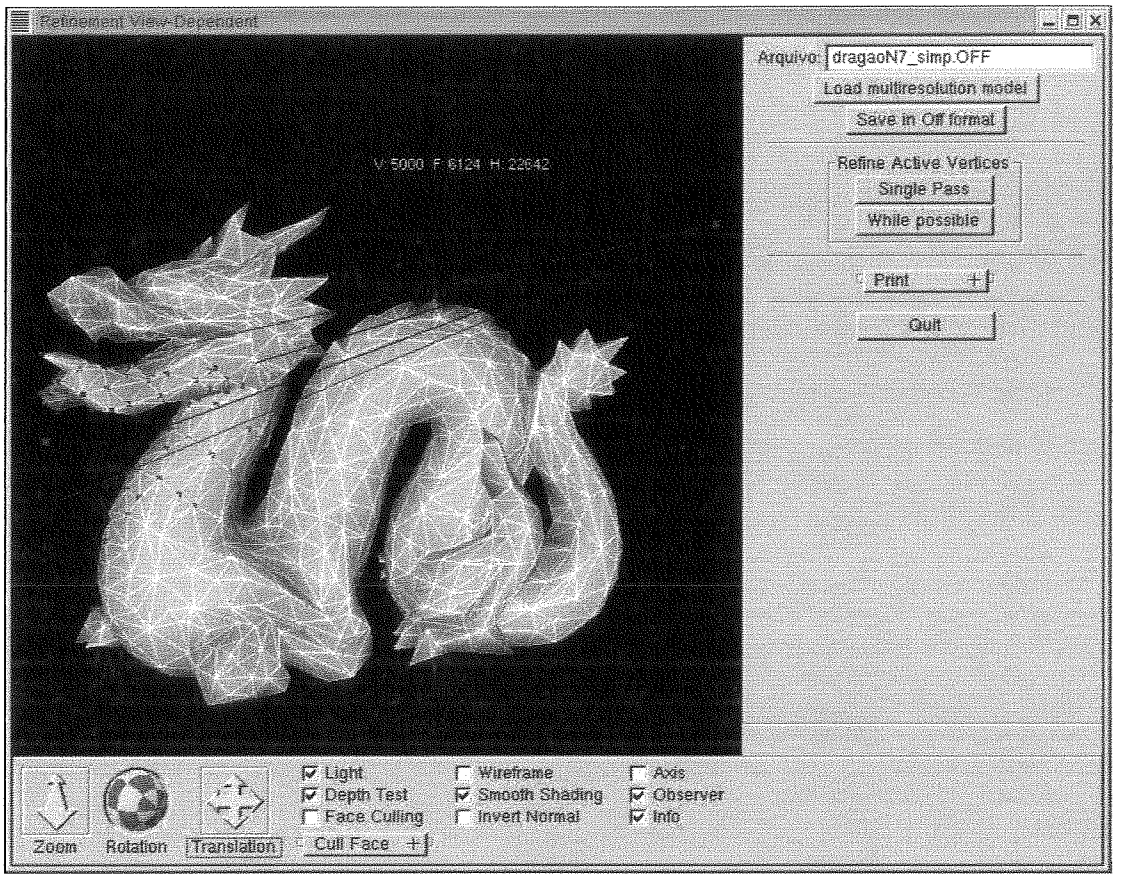


Figura 4.12: Aplicação de visualização em níveis de detalhe de forma seletiva: A posição do observador determina a região a ser detalhada, ou seja, os vértices a serem particionados.

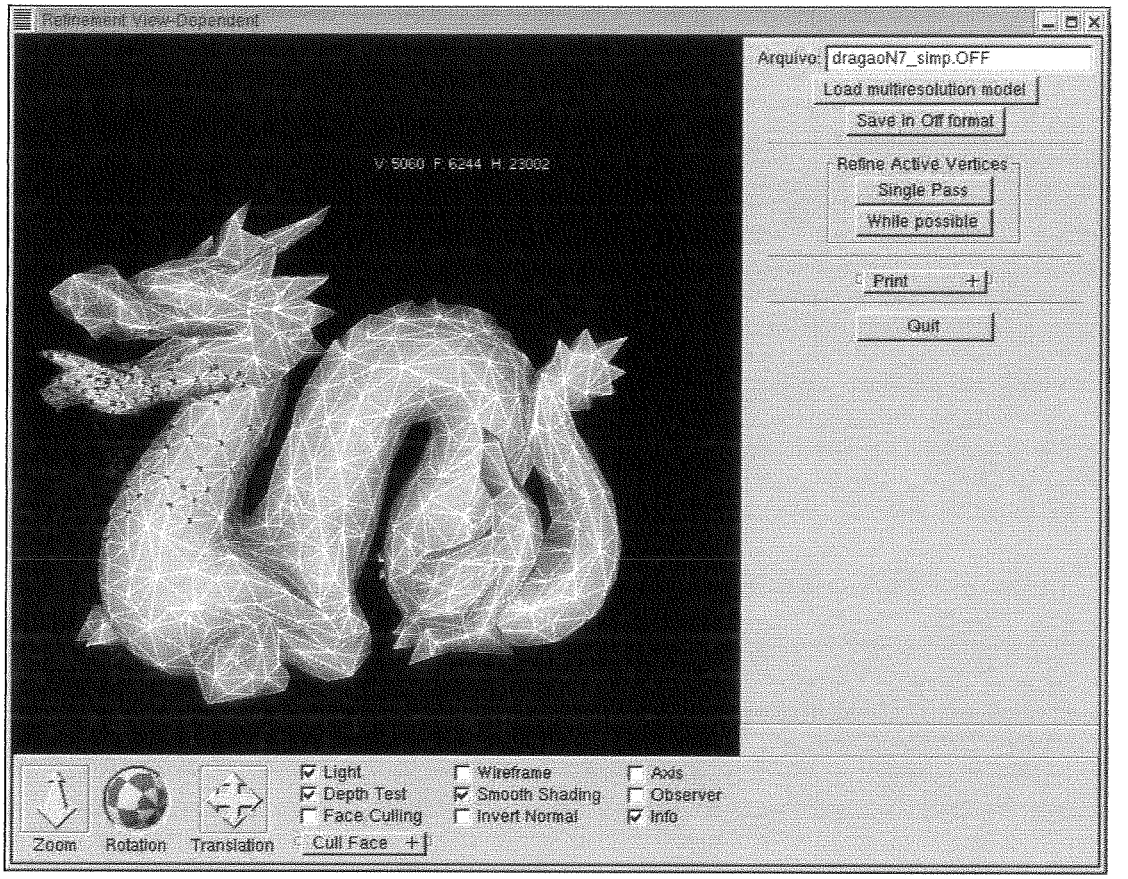


Figura 4.13: Aplicação de visualização em níveis de detalhe de forma seletiva: região detalhada em função da posição do observador.

Capítulo 5

Resultados

A ordem de seleção de arestas a serem contraídas afeta diretamente o resultado final da simplificação. Por isso, num processo de simplificação deve-se definir uma métrica para a contração de arestas.

Considere a Fig. 5.1 ilustrando o modelo de um toro que se deseja simplificar.

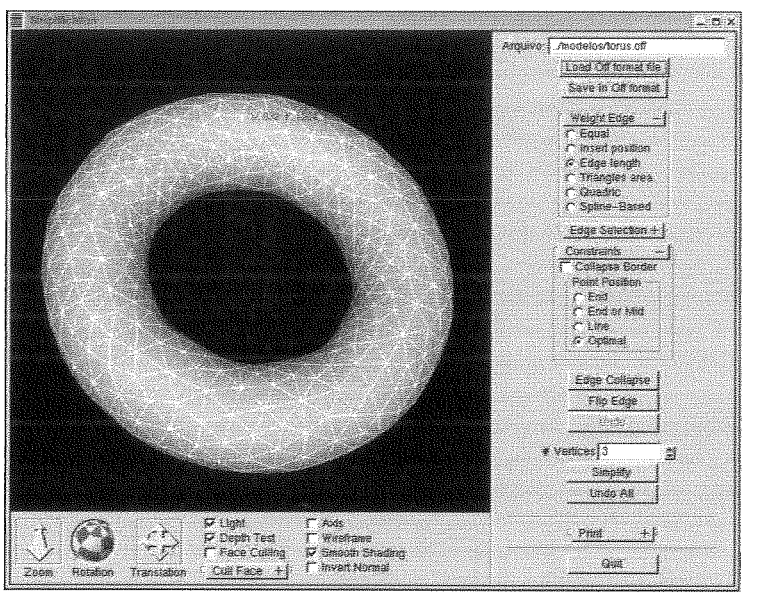


Figura 5.1: Modelo original de um toro contendo 632 vértices.

Se a escolha das arestas a serem contraídas não obedecer algum critério específico, pode-se obter um resultado não desejado, conforme ilustrado na Fig. 5.2.

Para que se possa obter modelos simplificados com forma aproximadamente igual à do modelo original, é preciso empregar um critério que eleja como aresta a ser contraída aquela que, de alguma maneira, possa ser considerada pouco importante. Algumas métricas que tentam avaliar essa importância já foram estabelecidas na literatura (vide 4). Algumas das mais

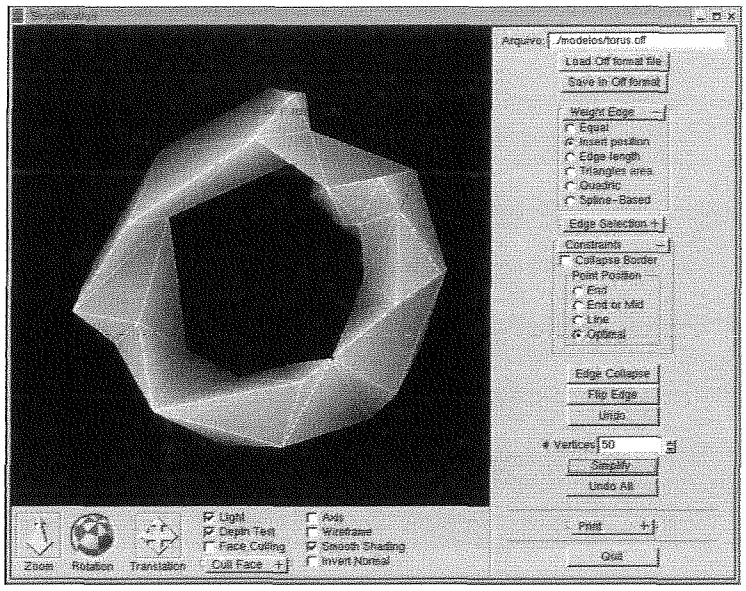


Figura 5.2: Modelo simplificado do toro ilustrado na Fig. 5.1 contendo 50 vértices.

conhecidas foram implementadas em nosso protótipo.

A fim de compararmos as diferentes métricas implementadas, as Figuras 5.3, 5.4 e 5.5 exibem os diferentes resultados gerados pela aplicação de cada uma dessas métricas sobre um mesmo modelo.

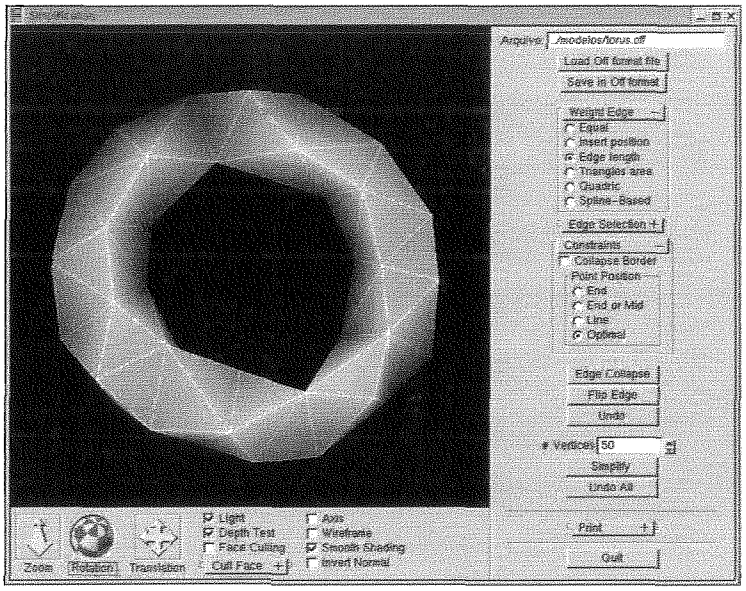


Figura 5.3: Modelo simplificado do toro ilustrado na Fig. 5.1 contendo 50 vértices. Aproximação obtida utilizando-se métrica de comprimento de aresta.

As Figuras 5.6, 5.7 e 5.8 ilustram diferentes aproximações de modelos obtidas pela aplicação da métrica de comprimento de aresta, área de triângulos e erro quádrico, respectivamente.

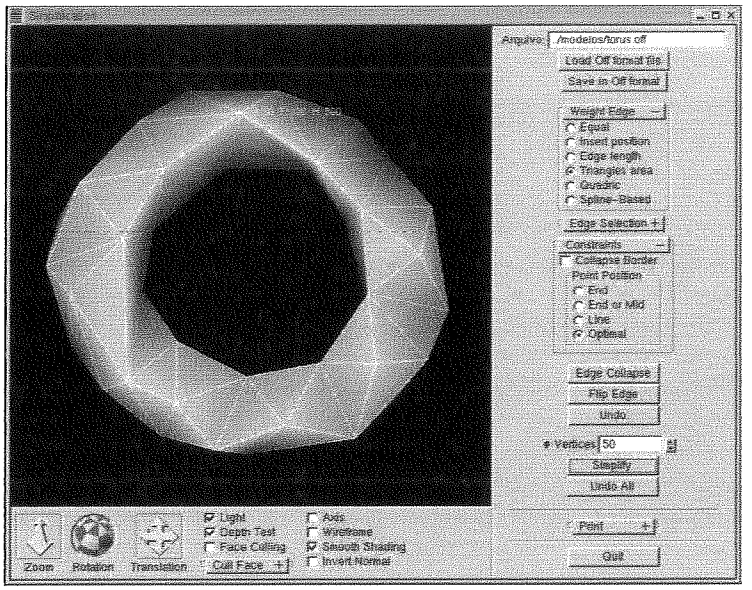


Figura 5.4: Modelo simplificado do toro ilustrado na Fig. 5.1 contendo 50 vértices. Aproximação gerada utilizando-se métrica de área dos triângulos incidentes.

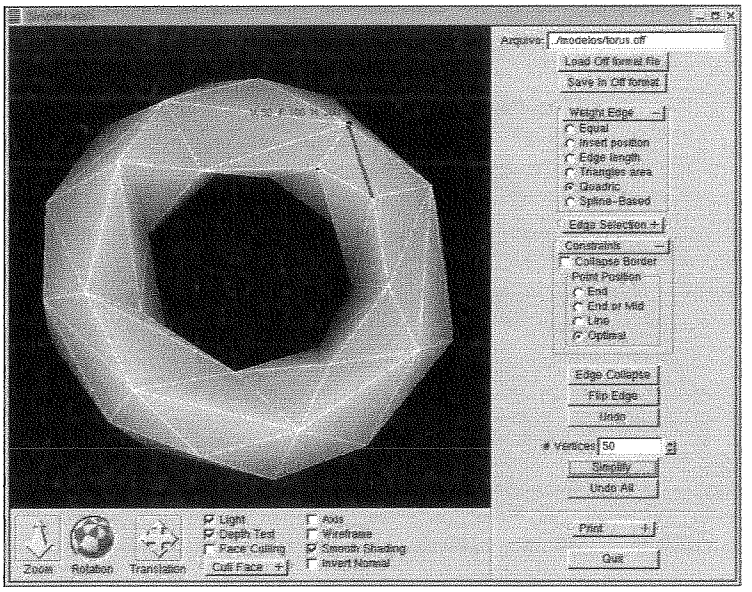


Figura 5.5: Modelo simplificado do toro ilustrado na Fig. 5.1 contendo 50 vértices. Aproximação gerada utilizando-se métrica de erro quádrico.

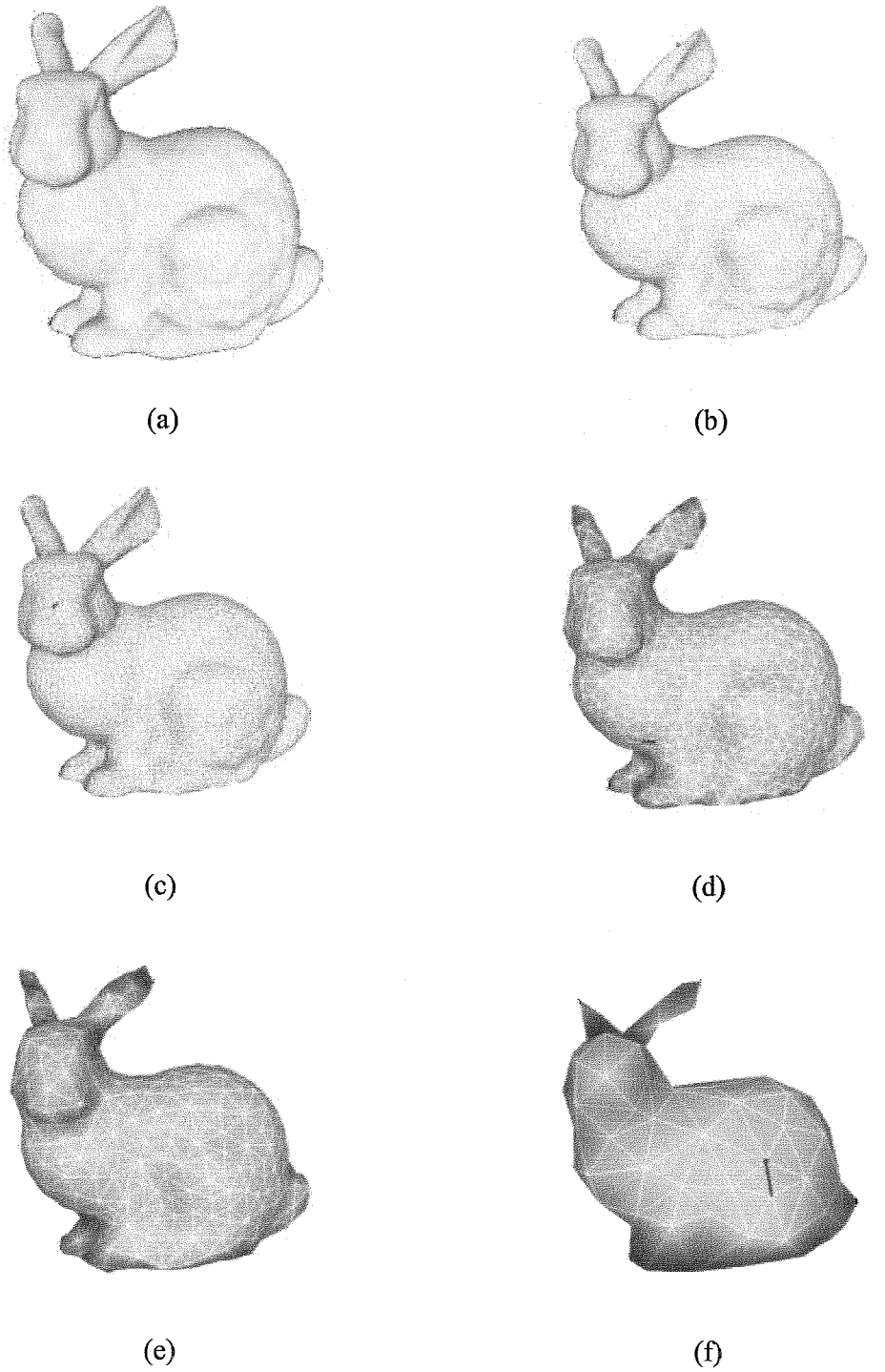


Figura 5.6: Simplificação usando métrica de comprimento de aresta: (a) modelo original com 34.834 vértices (b) Aproximação com 10000 vértices (c) Aproximação com 5000 vértices (d) Aproximação com 1000 vértices (e) Aproximação com 500 vértices (f) Aproximação com 100 vértices

(a)

(b)

(c)

(d)

(e)

Figura 5.7: Simplificação usando métrica de área dos triângulos incidentes a aresta: (a) Modelo original com 1.250 vértices (b) Aproximação com 1.000 vértices (c) Aproximação com 700 vértices (d) Aproximação com 500 vértices (e) Aproximação com 100 vértices

(a)

(b)

(c)

(d)

(e)

Figura 5.8: Simplificação usando métrica de erro quádrico: (a) Modelo original com 2.752 vértices (b) Aproximação com 2.000 vértices (c) Aproximação com 1.000 vértices (d) Aproximação com 500 vértices (e) Aproximação com 100 vértices

Com relação ao custo de armazenamento em disco de um modelo utilizando o esquema de representação proposto, consideremos as Figuras 4.11 e 4.12, que ilustram modelos de um punho e de um dragão, respectivamente. O arquivo de descrição do modelo original detalhado do punho, com 1.340 vértices e 2.676 faces, contém 78Kb. Uma aproximação com 250 vértices e 496 faces - mais de 80% de redução com relação ao modelo original - ocupa 14,3Kb para a descrição geométrica do modelo e 96,2Kb para a seqüência de operações (neste caso, foram efetuadas 1.090 operações). Embora este custo seja mais alto do que o de armazenamento do modelo original, em torno de 40%, devemos levar em consideração que num processo de transmissão progressiva, o tempo de envio do modelo grosseiro é bem menor do que o de transmitir o modelo detalhado, permitindo assim que o usuário possa ter uma visualização inicial mais rápida. Além disso, os registros podem ser transmitidos um a um, onde serão enviados aproximadamente 90 bytes por vez, ou em pacotes contendo mais de uma operação. Em relação ao modelo do dragão, os números obtidos são: 918Kb para o modelo original, contendo 15.342 vértices e 26.808 faces, 253Kb para o modelo grosseiro, que contém 5.000 vértices e 6.124 faces, e 976Kb para a seqüência de operações que gerou esta aproximação. Estes números aqui exibidos não estão levando em consideração nenhum tipo de compressão sobre os dados armazenados, recurso este que pode melhorar ainda mais os resultados obtidos.

Capítulo 6

Conclusão e Trabalhos Futuros

Malhas poligonais são estruturas de dados de larga utilização na representação tanto de superfícies quanto de objetos tridimensionais em geral. Nos últimos anos, vimos crescer o interesse de pesquisadores em computação gráfica no sentido de conciliar o uso de malhas cada vez mais detalhadas com as limitações de memória e de processamento dos mais variados sistemas computacionais. Neste contexto, as diversas técnicas que permitem obter malhas em diversos níveis de detalhe (LOD) ou que permitem utilizar resolução variável ou progressiva têm especial interesse.

Neste trabalho foi proposta uma estrutura de dados que suporta níveis de detalhe, transmissão progressiva e refinamento seletivo de malhas. Embora baseada no trabalho de Hoppe [27], a estrutura proposta apresenta a vantagem de representar operações de modificação local de forma mais compacta, o que pode levar a ganhos em termos de espaço de memória, embora a complexidade assintótica tanto de espaço quanto de tempo seja similar àquelas do trabalho de Hoppe.

Foram implementados todos os componentes fundamentais de uma plataforma de simplificação de superfícies poligonais, o que permite a experimentação com diversas métricas e critérios de simplificação. As três principais aplicações que compõem o software desenvolvido dispõem de interfaces simples, porém funcionais, o que permite sua utilização não só em ambientes de produção, mas também para fins educacionais. Pretendemos disponibilizar o engenho de simplificação e sua interface de programação de forma a permitir seu uso em outras aplicações, notadamente naquelas que possam se beneficiar das funcionalidades de refinamento adaptativo, como por exemplo aplicações de visualização de malhas via web.

O leitor interessado é instado a acessar o sítio `http://www.lcg.ufrj.br/members/meg` para obter mais detalhes sobre este trabalho e o software desenvolvido.

Referências Bibliográficas

[1] M.-E. Algorri and F. Schmitt. Mesh simplification. *Computer Graphics Forum*, 15(3), Aug. 1996. Proc. Eurographics '96.

[2] Chandrajit Bajaj and Daniel Schikore. Error-bounded reduction of triangle meshes with multivariate data. *SPIE*, 2656:34–45, 1996.

[3] B. Baumgart. *Geometric Modelling for Computer Vision*. PhD thesis, Stanford University, 1974. Tech. Rep. CS-463.

[4] B. Baumgart. A polyhedron representation for computer vision. In *AFIPS Conf. Proc.*, pages 589–596. National Computer Conference, 1975.

[5] A. Certain, J. Popovic, T. DeRose, T. Duchamp, D. Salesin, and W. Stuetzle. Interactive multiresolution surface viewing. In *Computer Graphics Proceedings, Annual Conf. Series (SIGGRAPG'96)*, pages 91–98. Siggraph, ACM Press, August 6-8 1996.

[6] CGAL - Computational Geometry Algorithms Library. `http://www.cgal.org`.

[7] A. Ciampalini, P. Cignoni, C. Montani, and R. Scopigno. Multiresolution decimation based on global error. Technical Report C96-021, CNUCE - C.N.R, Pisa, Italy, July 1996.

[8] P. Ciarlet and F. Lamour. Does contraction preserve triangular meshes? *Numerical Algorithms*, 13:201–223, 1996.

[9] P. Cignoni, D. Constanza, C. Montani, C. Rocchini, and R. Scopigno. Simplification of tetrahedral meshes with accurate error evaluation. In *Proceedings of the Conference on Visualization '00*, pages 85–92. IEEE Computer Society Press, 2000.

[10] Jonathan Cohen, Amitabh Varshney, Dinesh Manocha, Greg Turk, Hans Weber, Pankaj Agarwal, Frederick Brooks, and William Wright. Simplifica-

tion envelopes. In *SIGGRAPH '96 Proc.*, pages 119–128, August 6-8 1996. http://www.cs.unc.edu/ geom/envelope.html.

[11] T. K. Dey, H. Edelsbrunner, S. Guha, and D. V. Nekhayev. Topology preserving edge contraction. Technical report, RainDrop Geomagic Inc., Champaign IL., 1999. RGI-Tech-99.

[12] Matthias Eck, Tony DeRose, Tom Duchamp, Hugues Hoppe, Michael Lounsbery, and Werner Stuetzle. Multiresolution analysis of arbitrary meshes. In *SIGGRAPH '95 Proc.*, pages 173–182. ACM, Aug. 1995.

[13] Herbert Edelsbrunner and Dmitry V. Nekhayev. Topology and curvature preserving surface decimation. Technical Report Technical Report rgi-tech-97-010, Raindrop Geomagic, Inc., Champaign, IL, 1997.

[14] J. El-Sana and A. Varshney. Generalized view-dependent simplification. *EUROGRAPHICS'99*, 18(3), 1999.

[15] Leila De Floriani, Enrico Puppo, and Roberto Scopigno. Surface simplification algorithms overview. *IEEE Vis'98 Tutorial*, 1998.

[16] M. Garland and P.S. Heckbert. Surface simplification using quadric error metrics. *Proceedings of SIGGRAPH'97*, pages 209–216, 1997.

[17] Michael Garland. Multiresolution modeling: Survey and future opportunities. In *EUROGRAPHICS STAR - State of The Art Reports*. EUROGRAPHICS, 1999.

[18] Michael Garland. *Quadric-Based Polygonal Surface Simplification*. PhD thesis, School of Computer Science - Carnegie Mellon University, 5000 Fobes Avenue, Pittsburgh, PA 15213-3891, May 9 1999.

[19] Jonas Gomes and Luiz Velho. *Computação Gráfica - volume 1*. IMPA, Rio de Janeiro, RJ, 1998.

[20] Leonidas Guibas and Jorge Stolfi. Primitives for the manipulation of general subdivisions and the computation of voronoi diagrams. *ACM Transactions on Graphics*, 4(2):75–123, 1985.

[21] André Guéziec. Surface simplification with variable tolerance. In *Second Annual Intl. Symp. on Medical Robotics and Computer Assisted Surgery (MRCAS '95)*, pages 132–139, November 1995.

[22] André Guéziec. Surface simplification inside a tolerance volume. Technical report, Yorktown Heights, NY 10598, Mar. 1996. IBM Research Report RC 20440.

[23] Bernd Hamann. A data reduction scheme for triangulated surfaces. *Computer-Aided Geometric Design*, 11:197–214, 1994.

[24] Paul Heckbert and Michael Garland. Surface simplification survey slides, 1997.

[25] Paul Hinker and Charles Hansen. Geometric optimization. In *Proceedings of the 4th conference on Visualization '93*, pages 189–195, San Jose, CA, October 1993.

[26] H. Hoppe. Progressive meshes. *Proceedings of SIGGRAPH'96*, pages 99–108, 4-9 August 1996.

[27] H. Hoppe. View-dependent refinement of progressive meshes. *Proceedings of SIGGRAPH'97*, pages 189–197, August 1997.

[28] H. Hoppe. Efficient implementation of progressive meshes. *IEEE Visualization*, pages 35–42, October 1998.

[29] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. *Proceedings of SIGGRAPH'93*, pages 19–26, 1993.

[30] C. L. Jackins and S. L. Tanimoto. Octtrees and their use in representing three-dimensional objects. *Computer Graphics and Image Processing*, 14:249–270, 1980.

[31] Alan D. Kalvin and Russel H. Taylor. Superfaces:polygonal mesh simplification with bounded error. *IEEE Computer Graphics and Appl.*, 16(3), May 1996. http://www.computer.org/pubs/cg&a/cg&a.htm has corrected figures.

[32] Mark Kilgard. GLUT - OpenGL Utility Toolkit. `http://www.xmission.com/~nate/glut.html`.

[33] R. Klein, G. Liebich, and W. StraBer. Mesh reduction with error control. In R. Yagel and G. Nielson, editors, *Proceedings of Visualization'96*, pages 311–318, 1996.

[34] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface reconstruction algorithm. In *Computer Graphics - SIGGRAPH '87 Proceedings*, volume 21, pages 163–170. SIGGRAPH, July 1987.

[35] Michael Lounsbery. *Multiresolution Analysis for Surfaces of Arbitrary Topological Type*. PhD thesis, Dept. of Computer Science and Engineering, U. of Washington, 1997.

[36] Michael Lounsbery, Tony D. DeRose, and Joe Warren. Multiresolution analysis for surfaces of arbitrary topological type. *ACM Transactions on Graphics*, 16(1):34–73, 1997.

[37] Tom Davis Mason Woo, Jackie Neider. *OpenGL Programming Guide*. Addison-Wesley, 2nd edition, 1984.

[38] M. Mäntylä. *An Introduction to Solid Modeling*. Computer Science Press, Rockville, Maryland, 1988.

[39] OpenGL. `http://www.opengl.org`.

[40] E. Puppo and R. Scopigno. Simplification, lod and multiresolution - principles and applications, 1997.

[41] Paul Rademacher. GLUI - GLUT Based User Interface Library. `http://sourceforge.net/projects/glui`.

[42] K. J. Renze and J. H. Oliver. Generalized unstructured decimation. *IEEE C.G.&A.*, 16(6):24–32, 1996.

[43] A. Requicha and J. Rossignac. Representation for rigid solids: Theory, methods and systems. *ACM Computing Surveys*, 12(4), 1980.

[44] Rémi Ronfard and Jarek Rossignac. Full-range approximation of triangulated polyhedra. *Computer Graphics Forum*, 15(3), Aug. 1996. Proc. Eurographics '96.

[45] Jarek Rossignac and Paul Borrel. Multi-resolution 3D approximations for rendering complex scenes. In B. Falcidieno and T. Kunii, editors, *Modeling in Computer Graphics: Methods and Applications*, pages 455–465, Berlin, 1993. Springer-Verlag. Proc. of Conf., Genoa, Italy, June 1993. (Also available as IBM Research Report RC 17697, Feb. 1992, Yorktown Heights, NY 10598).

[46] William J. Schoroeder, Jonathan A. Zarge, and William E. Lorensen. Decimation of triangles meshes. *Proceedings of SIGGRAPH'92*, 26:65–70, July 1992.

[47] Marc Soucy and Denis Laurendeau. Multiresolution surface modeling based on hierarchical triangulation. *Computer Vision and Image Understanding*, 63(1):1–14, 1996.

[48] STL - Standard Template Library. `http://www.sgi.com/tech/stl/index.html`.

[49] Greg Turk. Re-tiling polygonal surfaces. *Computer Graphics (SIGGRAPH '92 Proc.)*, 26(2):55–64, July 1992.

[50] J. Xia and A. Varshney. Dynamic view-dependent simplification for polygonal models. In *IEEE Visualization '96 Proceedings*, pages 327–334, San Francisco, CA, Oct 27 - Nov 1 1996. IEEE.