Escola Politécnica da Universidade de São Paulo

WANG CONGSHI

Proposta de Algoritmo para Redução da Malha de Polígonos em Modelos Sólidos B-Rep

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para a obtenção do Título de Mestre em Engenharia

Área de Concentração: Engenharia Mecânica

Orientador: Prof. Dr. Marcos de Sales Guerra Tsuzuki

São Paulo
2005

30

UNIVERSIDADE DE SÃO PAULO

Relatório de Defesa

Relatório de defesa pública de Dissertação do(a) Senhor(a) Wang Congshi no Programa: Engenharia Mecânica, do(a) Escola Politécnica da Universidade de São Paulo.

Aos 24 dias do mês de outubro de 2005, realizou-se a Defesa da Dissertação do(a) Senhor(a) Wang Congshi, apresentada para a obtenção do título de Mestre em Engenharia - Área: Engenharia Mecânica - Opção: Mecatrônica, intitulada:

"Proposta de algoritmo para redução da malha de polígonos em modelos sólidos B-Rep"

Após declarada aberta a sessão, o(a) Sr(a) Presidente passa a palavra aos examinadores para as devidas argüições que se desenvolvem nos termos regimentais. Em seguida, a Comissão Julgadora proclama o resultado:

eodpes/01: 63453 PMR

Nome dos Participantes da Banca	Vínculo do Docente	Sigla da Unidade	Resultado
Marcos de Sales Guerra Tsuzuki	Presidente	EP - USP	Aprovado.
Fabio Kawaoka Takase	Titular	EP - USP	APROVADO
Júlio Arakaki	Titular	PUC-SP - Externo	APROVADO

Resultado Final: APROVADO.

Parecer da Comissão Julgadora *

Comentários da Defesa (opcional)

Eu, Elisabete Aparecida F da Silva Ramos Elisabete Ramos, Técnico Acadêmico, lavrei a presente ata, que assino juntamente com os(as) Senhores(as). São Paulo, aos 24 dias do mês de outubro de 2005.

Fabio Kawaoka Takase

Júlio Arakaki

Marcos de Sales Guerra Tsuzuki
Orientador(a)

* Obs: Se o candidato for reprovado por algum dos membros, o preenchimento do parecer é obrigatório.

Nos termos do artigo 110, do RG-USP, encaminhe-se o presente relatório à CPG, para homologação.

Impresso em: 21/10/2005

Agradecimentos

Agradeço ao Prof. Dr. Marcos de Sales Guerra Tsuzuki pela orientação.

Índice

Lista de Figuras

Lista de Tabelas

Resumo

Algoritmos para realizar a simplificação de malhas poligonais foram propostos na literatura. A grande maioria dos algoritmos faz uso intenso de informação sobre a adjacência entre elementos primitivos (face, aresta e vértice). Este tipo de informação está presente de forma explícita em Modeladores de Sólido B-Rep, o que os torna vantajosos para o desenvolvimento de algoritmos para realizar a simplificação de malhas poliedrais. Todos os vértices originalmente fornecidos devem ser incidentes a pelo menos uma face do sólido simplificado, sendo que a incidência será determinada utilizando-se aritmética intervalar. Assim, cada face possui uma lista de vértices incidentes. Cada vértice do sólido original é analisado e considerada a sua exclusão. O vértice possui uma lista de faces adjacentes, que por sua vez possuem uma lista de vértices incidentes. Para que um vértice seja removido, é necessário que uma nova forma de triangulação possa ser feita e que os vértices incidentes às faces anteriormente existentes sejam incidentes a pelo menos um dos triângulos recém criados. Em caso negativo, este vértice não poderá ser removido. O algoritmo foi implementado e testado com três sólidos diferentes: esfera, toróide e coelho. Por meio de uma análise visual é possível concluir que o algoritmo desenvolvido produziu resultados satisfatórios.

Abstract

In 3D computer graphics, polygonal models are often used to represent individual objects. Planar polygons, especially triangles, are used primarily because they are easy and afficient to render. Large polygon meshes are a common entity in scientific and engineering science. Mesh reduction algorithms are employed to reduce the mesh size. This kind of algorithms use adjacency information among the topological primitives (face, edge and vertex). This kind of information is explicitly present in B-Rep Solid Modelers. Then, B-Rep Solid Modelers are suited for the development of mesh reduction algorithms. The simplification process is controled by a geometrical criterion. The original vertices must be incident to at least one face from the simplified solid. The incidence is calculated using interval arithmetic. Each face has a list of incident vertices. Each vertex from the original solid is considered for deletion. The vertex has neighboring faces, and each face has a list of incident vertices. A vertex can be removed if a new triangulation can be defined and the geometrical criterion is satisfied. If this is not possible, the vertex is not removed.

1. Introdução

Modelos poliedrais com uma grande quantidade de polígonos são comuns em computação gráfica, por exemplo quando são utilizados dispositivos como Tomografia Computadorizada, Ressonância Magnética e Varredura Tridimensional. Como modelos poliedrais com grande quantidade de polígonos fazem uso demasiado de memória, comunicação e hardware para renderização, os algoritmos para redução da malha de polígonos devem ser aplicados. Algumas técnicas procuram eliminar polígonos pequenos ou mal formados, e outras técnicas tentam obter o máximo possível em simplificação.

Neste trabalho será proposto um algoritmo para redução da malha poligonal triangular de um Modelo Sólido B-Rep. A grande maioria dos algoritmos faz uso intenso de informação sobre a adjacência entre elementos primitivos (face, aresta e vértice). Este tipo de informação está presente de forma explícita em Modeladores de Sólido B-Rep, o que os torna vantajosos para o desenvolvimento de algoritmos para realizar a simplificação de malhas poliedrais.

A simplificação é obtida realizando-se uma série de operações locais. Cada uma destas operações objetiva reduzir a quantidade de vértices, arestas ou faces do modelo poliedral. Os algoritmos de simplificação geralmente escolhem uma destas operações e a aplica repetidamente ao modelo poliedral até que o nível de complexidade desejado seja obtido.

Wuensche [21] identificou seis metodologias utilizadas para reduzir o modelo poliedral:

1. Aglutinação de Polígonos: combina polígonos coplanares ou quase coplanares em polígonos maiores. A topologia original da malha não é modificada;

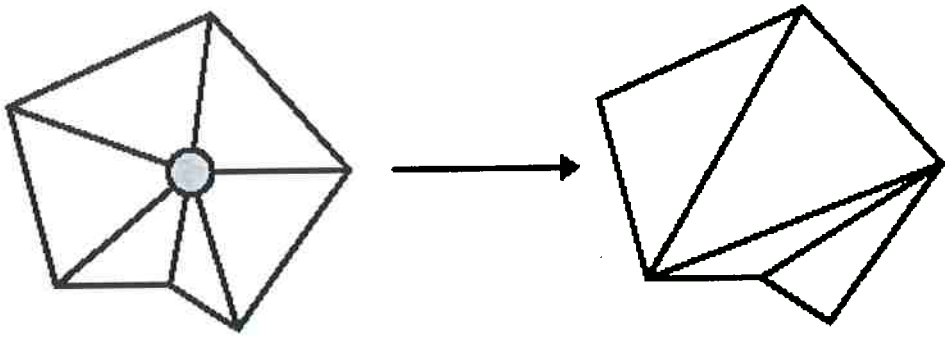


Figura 1.1. Operação de elimição de um vértice.

2. Eliminação de Vértices: esta operação remove do modelo poliedral um único vértice e todos os seus triângulos adjacentes. Este processo cria um furo que deve ser preenchido por um novo conjunto de triângulos. Um vértice com *n* triângulos adjacentes, ao ser removido cria um furo com *n* lados. O problema de preenchimento do furo envolve a seleção de uma opção entre um número finito de possíveis triangulações. Os *n* triângulos ao redor do vértice são substituídos por uma nova triangulação com *n-2* triângulos. Esta operação está detalhada na Figura 1.1;

3. Eliminação de Aresta: esta operação se tornou muito comum na

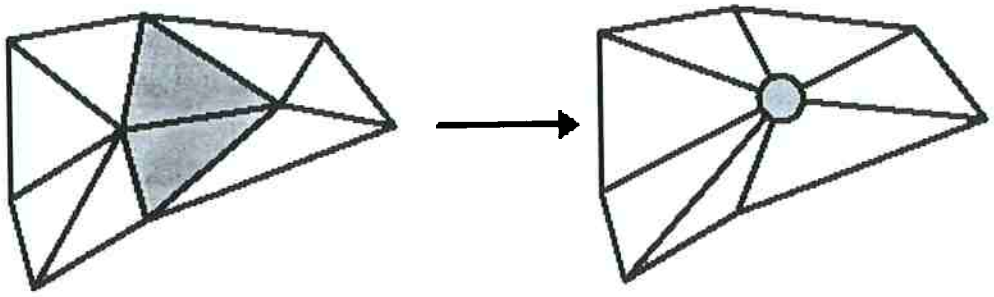


Figura 1.2. Operação de eliminação de uma aresta.

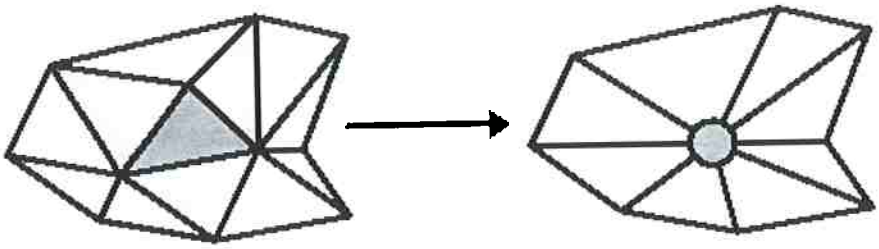


Figura 1.3. Operação para eliminação de face.

comunidade de computação gráfica nos últimos anos. Os dois vértices da aresta são aglutinados em um único vértice. Esta operação distorce todos os triângulos adjacentes à aresta. Os triângulos adjacentes à aresta se degeneram para arestas unidimensionais e são removidos do modelo poliedral. Esta operação requer que as coordenadas do novo vértice sejam escolhidas a partir de um domínio contínuo. Entre as escolhas comuns para o valor das coordendas incluem: coordenadas de um dos vértices originais, coordenadas do ponto médio da aresta removida, pontos arbitrários sobre a aresta eliminada, ou pontos arbitrários na vizinhança da aresta eliminada. Esta operação está ilustrada na Figura 1.2;

4. Eliminação de Face: a operação de remoção de face é similar à operação de eliminação de aresta, exceto que ela realiza uma simplificação em maior escala. Todos os vértices da face são aglutinados em um único vértice. Isto faz com que a face original se degenere a um vértice e as faces adjacentes à face se degeneram para segmentos lineares, removendo no total 4 triângulos do modelo poliedral. Esta operação permite que o processo de simplificação se realize mais rapidamente. A Figura 1.3 ilustra esta operação;

5. Repoligonalização: substitui a malha triangular utilizando algum critério de erro. Um exemplo é dado por Turk [17] que distribui um conjunto de

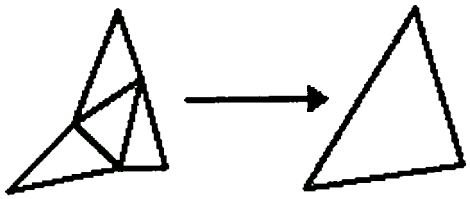


Figura 1.4. Operação de agrupamento.

pontos em uma malha por repulsão de pontos, com densidade ponderada por estimativa da curvatura local e utiliza estes pontos para uma nova triangulação da malha;

6. Agrupamento: uma operação comum no refinamento de um modelo poliedral é a divisão de um triângulo em outros quatro. A operação de agrupamento é o inverso desta operação. Então uma operação de agrupamento aglutina quatro triângulos de uma configuração particular em um único triângulo, reduzindo em três o número de triângulos (vide Figura 1.4).

A proposta apresentada por Léon e Véron [18,19] se denomina estática porque procura por um subconjunto de vértices a partir de um conjunto dado de vértices. É importante salientar que os vértices originais não são movimentados. Para realizar a redução, o algoritmo associa a cada vértice do poliedro original uma tolerância geométrica. Esta tolerância é representada por uma esfera associada a cada vértice do poliedro original e ela possui duas interpretações distintas (dependendo da utilização do poliedro simplificado):

1. A primeira interpretação associa a tolerância ao erro de medida relacionado ao sistema de digitalização utilizado. Este sistema pode ser um sensor mecânico, scanner a laser ou algum dispositivo similar. Este erro é avaliado simultaneamente a partir de característica do sistema de

medição e dos parâmetros utilizados durante o processo de digitalização.

2. A segunda interpretação associa a tolerância à mínima distância perceptível segundo parâmetros de visualização. Assim, elementos que não podem ser distinguíveis não serão exibidos, reduzindo os elementos a serem exibidos e em conseqüência acelerando o processo de visualização.

Hussain et al. [8] propuseram um algoritmo pelo qual a malha de triângulos é simplificada, Hussain et al. salientam que as características geométricas relevantes da malha de triângulos são preservadas. Eles não utilizam nenhuma métrica para quantificar as características geométricas relevantes, sendo a sua análise puramente visual. Para este fim, cada vértice possui um valor quantificando a sua importância visual. A proposta feita por Léon e Véron [18,19] possui uma qualificação semelhante denominada por curvatura gaussiana.

Li e Watson [11] criam uma árvore para representar os vértices hierarquicamente. As características geométricas são representadas por meio de agrupamentos de vértices. Esta é uma proposta semi-automática que exige a interação com o usuário. Schroeder et at. [15] propuseram um algoritmo que determina se vértices podem ser ou não removidos pela sua incidência com faces e arestas. Entretanto, este método não faz uso de nenhuma forma de controle, sendo que o sólido final pode estar distante do sólido original. Eastlick e Maddock [4,5] propuseram um algoritmo de redução de malhas que avalia a diferença de volume entre poliedros, evitando que o resultado se afaste do sólido original.

Neste trabalho, consideramos uma adaptação da proposta feita por Léon e Véron [18,19]. Consideramos a possibilidade de utilizar a aritimética intervalar [16] para representar a tolerância utilizada na proposta de Léon e Véron [18,19]. Utilizamos o modelador de sólidos USPDesigner [16], desenvolvido pelo grupo da LGC – Laboratório de Geometria Computacional do PMR-EPUSP. Todos os algoritmos para simplificação de malha poligonal utilizam em larga escala as informações de adjacência entre elementos geométricos primitivos (vértice, aresta e face), e estas informações estão presentes explicitamente nos modeladores B-Rep acelerando o processamento destes algoritmos.

O algoritmo desenvolvido será testado em três sólidos distintos: esfera, toróide e o coelho de Stanford. O modelo poliedral do coelho é fornecido pela Universidade de Stanford e diversos autores o utilizam para testar os seus algoritmos (`http://www-graphics.stanford.edu/data/3Dscanrep/`).

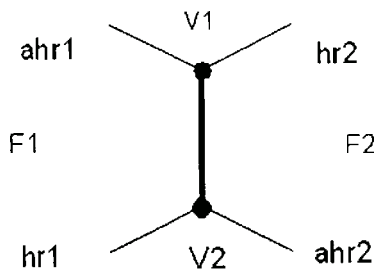
Este trabalho está estruturado da seguinte maneira, no capítulo 2, apresentamos alguns conceitos sobre Modelagem de Sólidos B-Rep, no capítulo 3, apresentamos conceitos sobre Aritmética Intervalar. No capítulo 4, apresentamos o algoritmo para simplificação de malha poligonal que está sendo proposto. No capítulo 5, apresentamos alguns resultados. No capítulo 6, apresentamos as conclusões.

2. Modelagem de Sólidos B-Rep

Neste capítulo será apresentada a técnica de modelagem de sólidos conhecida como B-Rep (*Boundary Representation*) . Inicialmente, as propostas de estrutura de dados para Modeladores de Sólidos serão detalhadas. Em seguida, os Operadores de Euler que foram definidos para facilitar a manipulação da estrutura de dados serão apresentados. E por último, algumas funções de alto nível serão detalhadas, conhecidas como Operadores Locais. Os operadores locais fazem uso exclusivo de Operadores de Euler para editar a estrutura de dados.

2.1. Estrutura de Dados

Nesta parte, será explicado como os elementos primitivos são representados de modo a definir um sólido B-Rep. Um modelo B-Rep possui sete tipos de elementos primitivos (vértice, meia-aresta, aresta, laço, face, shell e região). Existem três estruturas de dados propostas na literatura que implementam o



v1	v2
hr1	hr2
ahr1	ahr2
f1	f2

hr = aresta no sentido horário
ahr = aresta no sentido anti-horário

Figura 2.1. Estrutura *winged-edge*.

Modelador de Sólidos B-Rep. Todas baseadas na aresta como elemento de referência [1,9,12,20]: estrutura *winged-edge,* estrutura meia-aresta e estrutura unificada. Neste projeto, será utilizada a estrutura unificada.

A estrutura w*inged-edge* [1] mantém as informações de adjacência por meio de ponteiros a vários elementos adjacentes à aresta de referência: duas faces, dois vértices e quatro arestas. Cada uma das quatro arestas compartilha com a aresta de referência um vértice e uma face (vide Figura 2.1). Onde v1 e v2 são ponteiros para os dois vértices adjacentes, hr1, ahr1, hr2 e ahr2 são ponteiros para as quatro arestas adjacentes, e f1 e f2 são ponteiros para as duas faces adjacentes.

A estrutura meia-aresta [12,20] representa a metade das informações de adjacência da estrutura *winged-edge*. Cada *meia-aresta* possui apenas uma orientação, e cada face possui um circuito direcional de *meia-arestas* (vide Figura 2.2). Onde v é ponteiro para o vértice da meia-aresta, hr e ahr são ponteiros para as duas meia-arestas adjacentes, e f é ponteiro para a face adjacente à meia-aresta.

A estrutura unificada [9] foi proposta para evitar a duplicidade das informações associadas à aresta nas duas meia-arestas, foi proposta a criação de

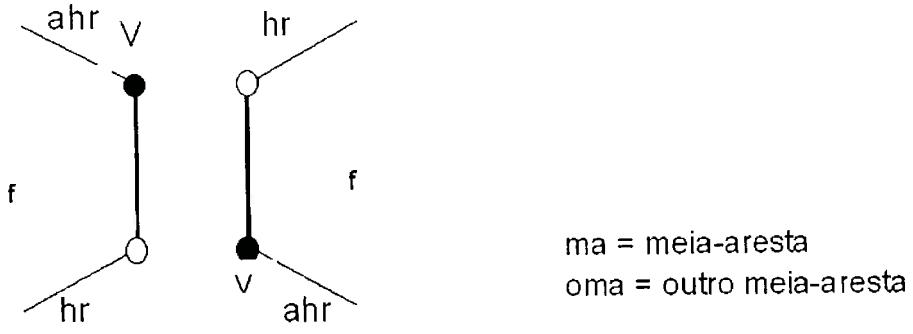


Figura 2.2. Estrutura Meia-aresta.

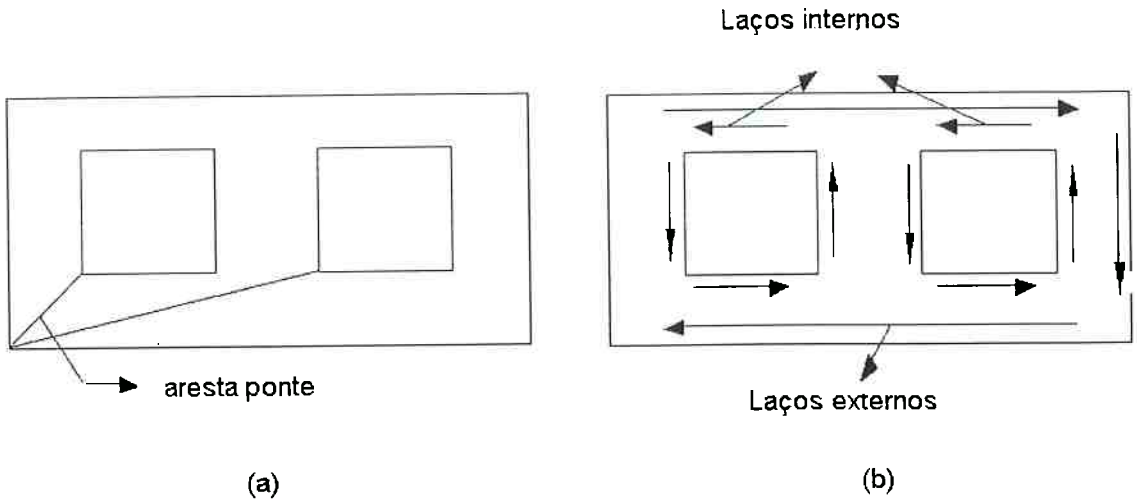


Figura 2.3. Extensão para faces com múltiplos contornos.

uma nova estrutura para a aresta que associa as duas meia-arestas que definem a aresta de referência, evitando assim o ponteiro para a outra meia-aresta.

Até este momento, foi assumido que cada face possui apenas um contorno. Entretanto, casos práticos requerem que uma face possua mais de um contorno (como uma face com furos). Faces com mais de um contorno podem ser simuladas pela técnica de aresta-ponte (*bridge-edge*) [12] no qual uma aresta une os contornos de uma face entre si. A aresta-ponte, portanto, possui a mesma face adjacente em ambas as laterais (vide Figura 2.3 (a)).

Entretanto, a técnica aresta-ponte não é muito eficiente porque será necessário determinar como os contornos devem ser conectados pelas arestas-ponte, o que criará a necessidade de complexos algoritmos para implementar operações de modelagem. Como exemplo, podemos citar os Operações Booleanas que provavelmente interseccionarão as arestas-ponte. Alterações na estrutura de dados unificada de maneira a suportar faces com mais de um contorno não afetarão a estrutura B-Rep ao nível de aresta, mas sim, ao nível de face. Uma técnica muito comum é adicionar uma estrutura de tamanho fixo chamada laço (loop) que é associada a cada contorno da face. A estrutura laço

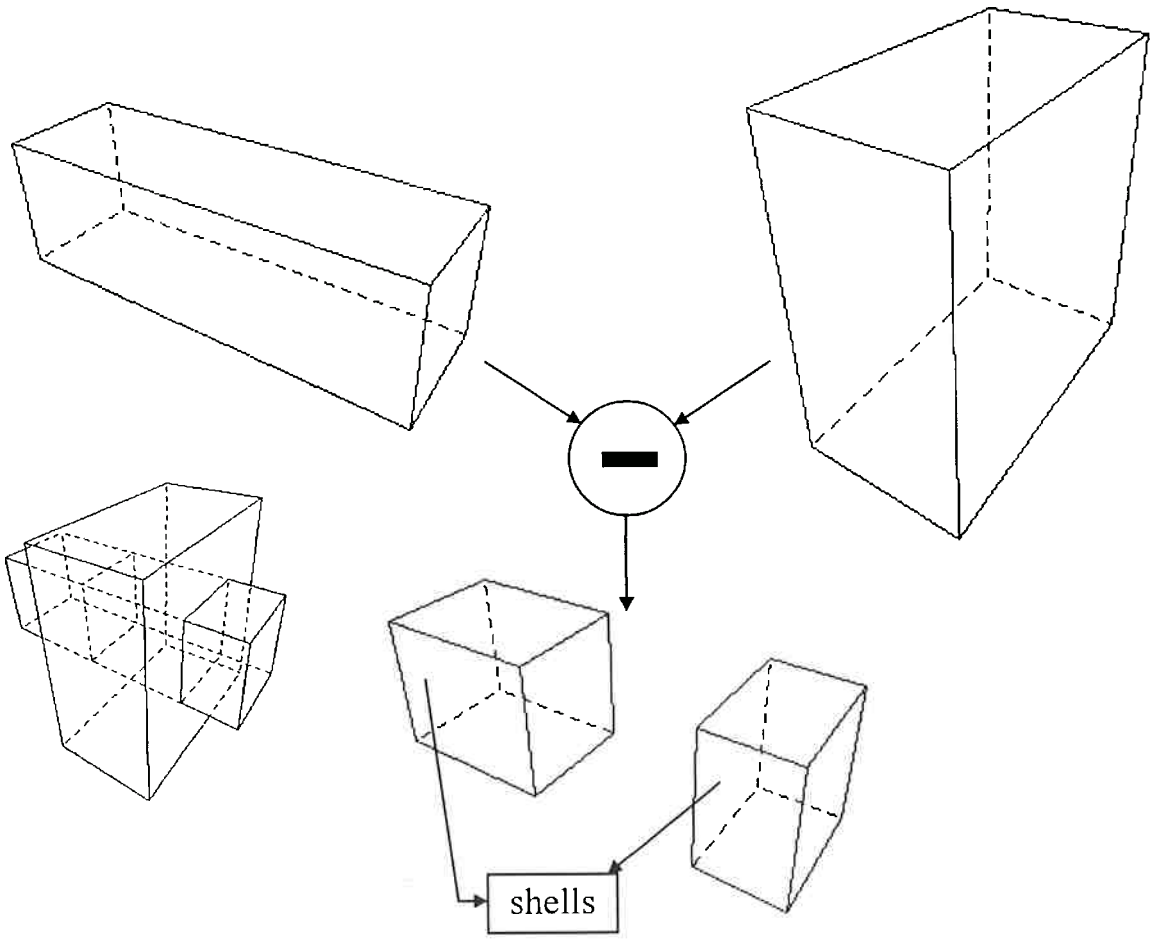


Figura 2.4. Sólido com duas regiões e cada região com um shell.

fornece à estrutura face um mecanismo para manter uma lista ligada de ponteiros para os seus múltiplos contornos. Cada face possui um laço externo e zero ou mais laços internos (vide Figura 2.3.(b)).

Devido ao formalismo das Operações Booleanas [7], ao serem combinados dois sólidos por uma Operação Booleana, o resultado sempre será um sólido. Entretanto, mesmo em situações especiais, como a situação exemplificada na Figura 2.4, onde o resultado da Operação Booleana aparenta apresentar dois sólidos, o resultado é considerado como sendo apenas um sólido. Entretanto, nesta situação em especial, considera-se que o sólido resultante possui duas regiões. O elemento representa conjuntos desconexos de faces no espaço. A

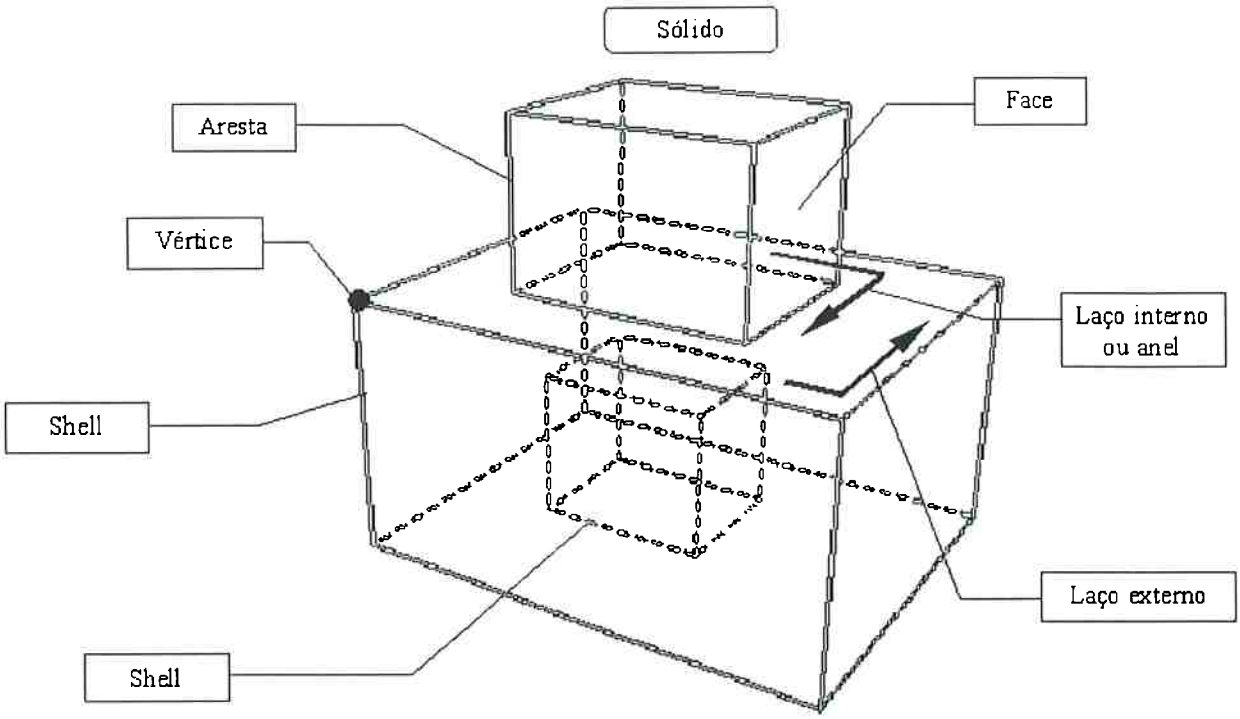


Figura 2.5. Elementos topológicos de um modelo sólido.

Figura 2.5 ilustra os elementos primitivos de um modelo sólido. A Figura 2.6 ilustra a hierarquia da estrutura unificada resultante.

Neste trabalho, foi utilizada a estrutura unificada para representar sólidos poliedrais.

2.2. Operadores de Euler

Por conterem informações sobre as adjacências entre os elementos primitivos, as estruturas computacionais anteriormente expostas são bastante complexas e a sua manipulação exige muitos cuidados para que a consistência dos dados seja mantida. Para contornar este problema, um conjunto de operadores foi desenvolvido com o objetivo de tornar a manipulação da estrutura de dados da representação B-Rep mais intuitiva. Eles permitem que a construção do sólido possa ser executada passo a passo, escondendo todos os detalhes de

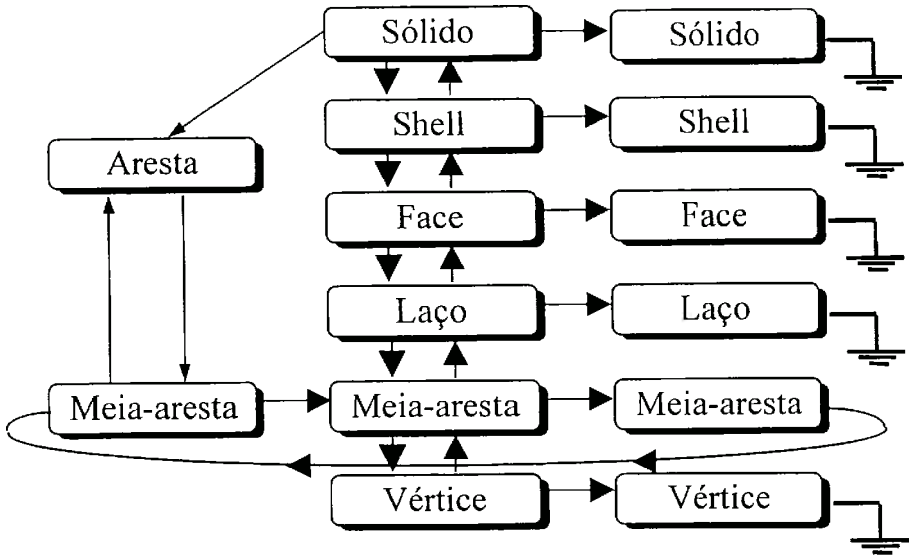


Figura 2.6. Hierarquia de elemento da estrutura unificada.

implementação da estrutura de dados. Estes operadores formam o segundo nível de representação do modelador B-Rep.

A equação de Euler-Poincaré [12] diz que um sólido poliédrico é topologicamente válido se a seguinte relação entre as suas quantidades de elementos for verificada:

$$v - e + 2f = 2(s - h) + l \qquad (1)$$

onde *v* é o número de vértices do sólido, *e* o número de arestas, *f* o número de faces, *s* o número de shells, *h* o número de furos e *l* o número de laços. A forma mais conhecida na literatura [12] da equação de Euler-Poincaré supõe ainda a existência de *r* anéis no sólido, onde $r = l - f$. Ficando a equação da seguinte forma:

$$v - e + f = 2(s - h) + r \qquad (2)$$

Vários autores demonstram que sete operadores são suficientes para construir todos os sólidos. Enquanto estes sete operadores podem ser escolhidos

de várias maneiras, considerações de modularidade e independência criaram apenas pequenas variações na coleção encontrada na literatura [12]. Com a finalidade de facilitar a memorização, os Operadores de Euler estão definidos utilizando a nomenclatura da Tabela 1.

Tabela 1. Nomenclatura dos Operadores de Euler

Símbolo	Significado		Símbolo	Significado
M	make		F	face
K	kill		S	shell ou solid
V	vertex		H	hole
E	edge		R	ring ou region

Por exemplo, o operador **MEV** deve ser traduzido por *Make Edge, Vertex* (Crie uma aresta e um vértice). A seguir são descritos os sete operadores mais comumente utilizados na literatura e uma representação gráfica para cada operador pode ser observada na Figura 2.7.

- **MVSF** (*Make Vertex Solid Face)* : este operador cria um sólido inicial com apenas uma face e um vértice;

- **MEV** (*Make Edge Vertex)* : este operador adiciona a um sólido uma aresta e um vértice. A aresta é criada conectando-se um vértice já existente ao novo vértice criado;

- **MEF** (*Make Edge Face)* : este operador adiciona ao sólido uma aresta e uma face. A face é criada pela divisão de uma face já existente acrescentando-se a nova aresta;

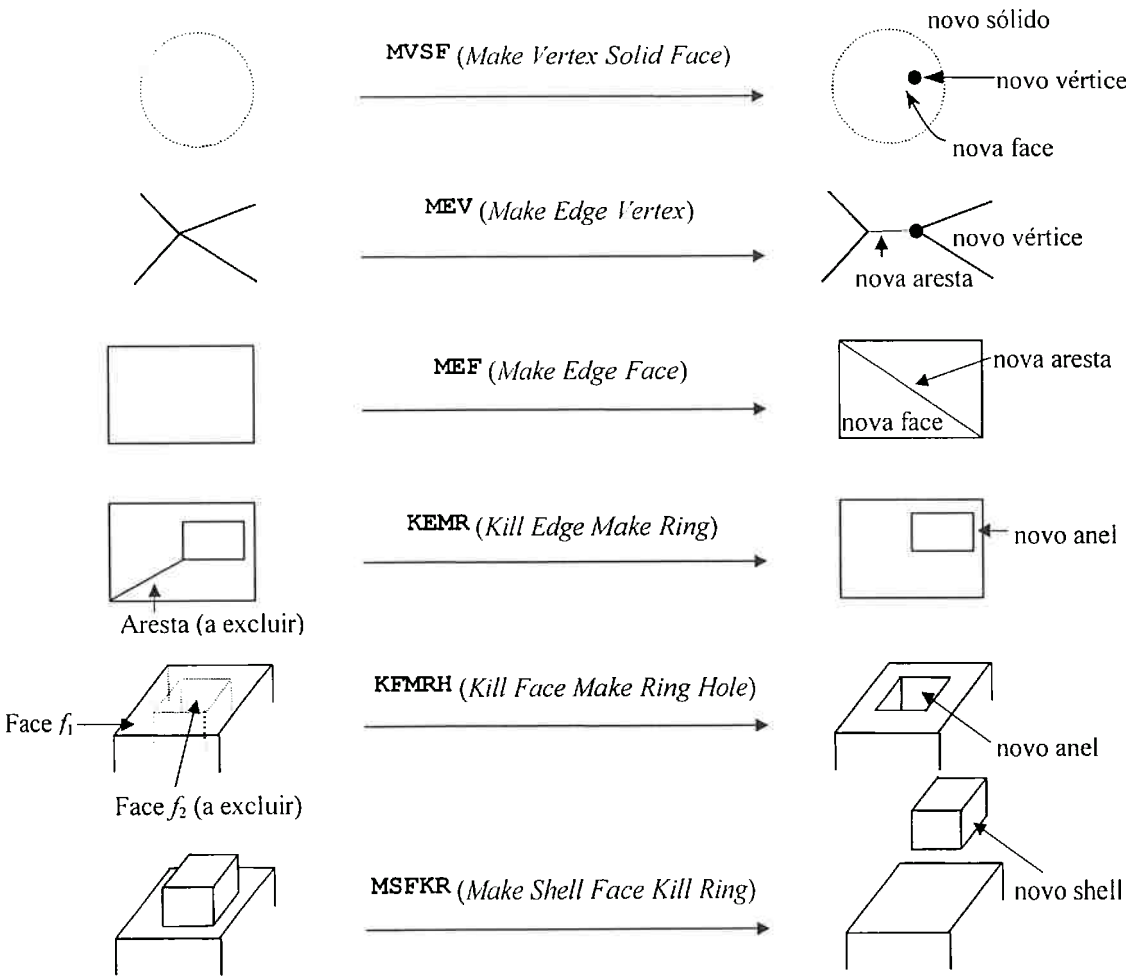


Figura 2.7. Representação gráfica dos operadores de Euler.

- **KEMR** (*Kill Edge Make Ring*) : este operador divide o contorno de uma face em dois laços pela remoção de uma aresta-ponte;

- **KFMRH** (*Kill Face Make Ring Hole*) : nenhum dos operadores discutidos anteriormente é capaz de modificar as propriedades topológicas globais da estrutura de dados, como dividir um sólido em dois componentes ou criar um furo passante. O operador **KFMRH** possui este objetivo;

- **MSFKR** (*Make Shell Face Kill Ring*) : este é outro operador que manipula informações globalmente. Ele transforma o anel de uma face em uma nova face, e todo o conjunto de faces associadas à nova face constituirá um novo *shell*;

- **MRSFKR** (Make Region Shell Face Kill Ring): este é outro operador que manipula informações globalmente. Ele transforma o anel de uma face em uma nova face, e todo o conjunto de faces associadas à nova face constituirá uma nova região.

Os sete operadores detalhados acima são todos construtivos entretanto, para que o sólido possa ser alterado satisfatoriamente ele precisa também de algumas operações destrutivas. Por isto, cada operador construtivo possui um operador correspondente destrutivo. Eles estão descritos abaixo:

KVSF (*Kill Vertex Solid Face*) ←→ **MVSF** (*Make Vertex Solid Face*)
KEV (*Kill Edge Vertex*) ←→ **MEV** (*Make Edge Vertex*)
KEF (*Kill Edge Face*) ←→ **MEF** (*Make Edge Face*)
MEKR (*Make Edge Kill Ring*) ←→ **KEMR** (*Kill Edge Make Ring*)
MFKRH (*Make Face Kill Ring Hole*) ←→ **KFMRH** (*Kill Face Make Ring Hole*)
KSFMR (*Kill Shell Face Make Ring*) ←→ **MSFKR** (*Make Shell Face Kill Ring*)

Durante o processo de construção de um sólido pela utilização de Operadores de Euler, a validade topológica do mesmo é mantida, observando-se a equação de Euler-Poincaré. Entretanto, é comum o agrupamento de Operadores de Euler em uma certa seqüência para mantermos também a geometria válida. É importante observar que não há como manter a geometria válida em todos os estágios da construção. Por isto, os Operadores de Euler devem ser agrupados em seqüências que possuam algum significado. Através de um exemplo simples, um bloco retangular com um furo passante retangular, é possível ilustrar a utilização dos Operadores de Euler (Figura 2.9).

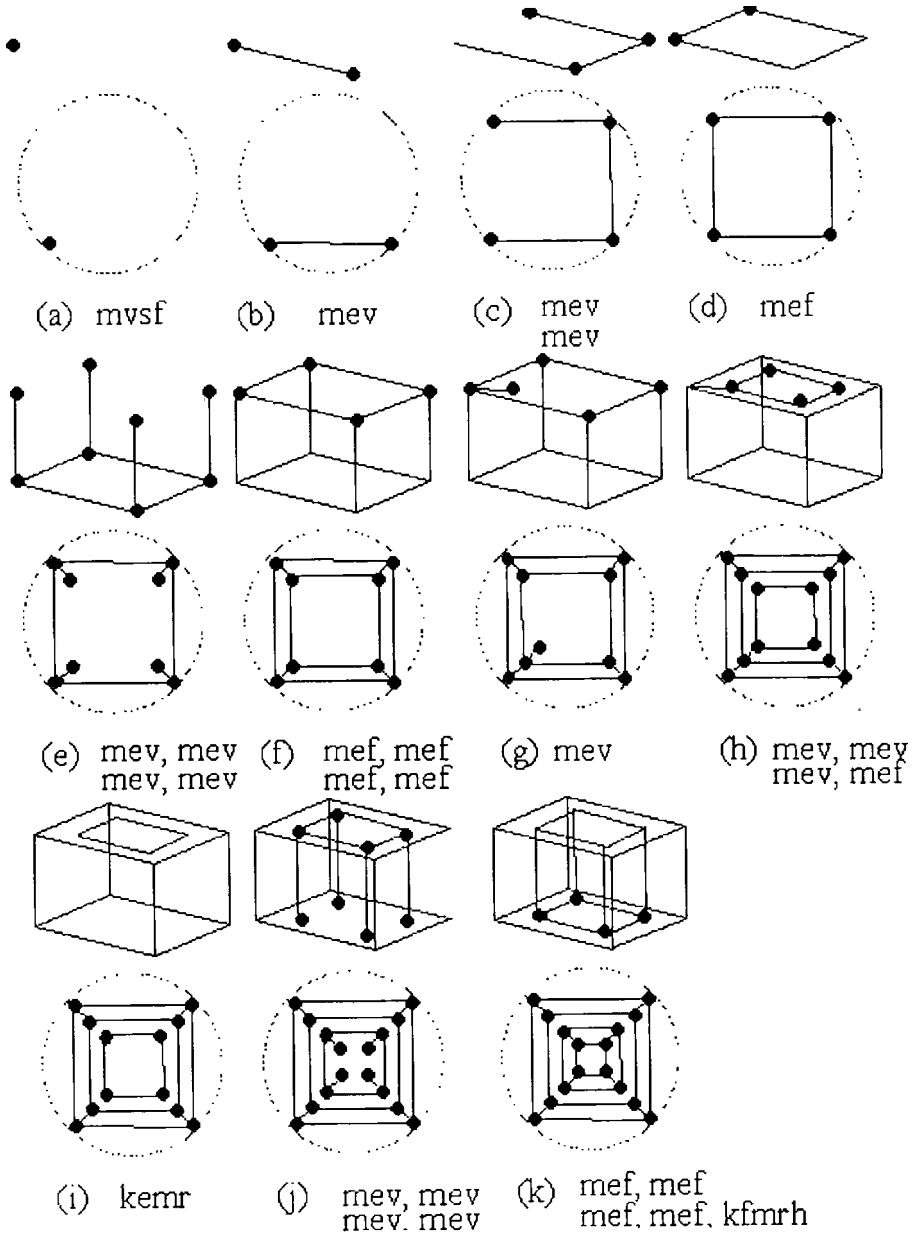


Figura 2.8. Seqüência de Operadores de Euler para criação de um cubo com furo passante.

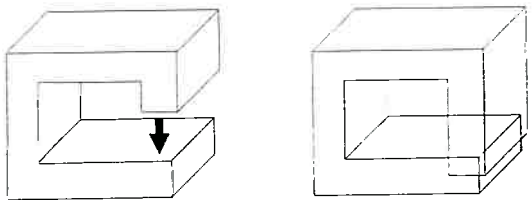


Figura 2.9. Um sólido auto-interceptante.

2.3. Operadores Locais

Os operadores locais permitem que algumas características dos sólidos possam ser modificadas diretamente pelo usuário preservando as consistências topológica e geométrica do local modificado. Por não realizarem alterações em áreas externas à localidade em que eles se aplicam, não são realizadas verificações sobre o sólido após a realização de uma operação local. Entretanto, existe a possibilidade de se criar um sólido inválido utilizando-se operadores locais, como por exemplo, um sólido auto-interceptante (Figura 2.9). Não é uma tarefa trivial verificar se um sólido é auto-interceptante; geralmente, depende da correta utilização destas operações pelo usuário.

Um operador local pode ser considerado uma extensão dos Operadores de Euler. Um operador local, ao ser acionado pelo usuário, definirá uma seqüência de Operadores de Euler que a implemente. A seguir, será apresentado o Operador para criar um arco de circunferência.

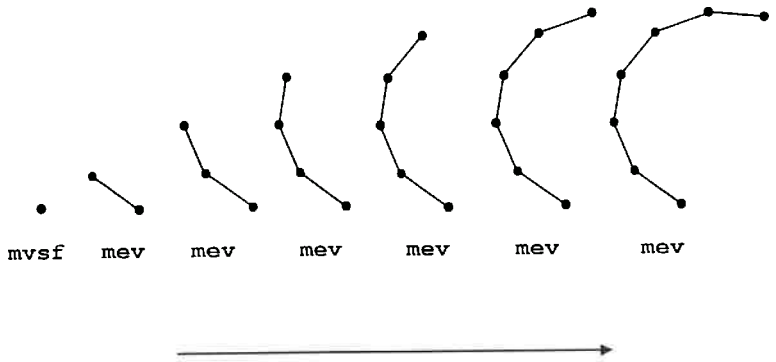


Figura 2.10. Seqüência de Operadores de Euler que implementam a criação de um arco.

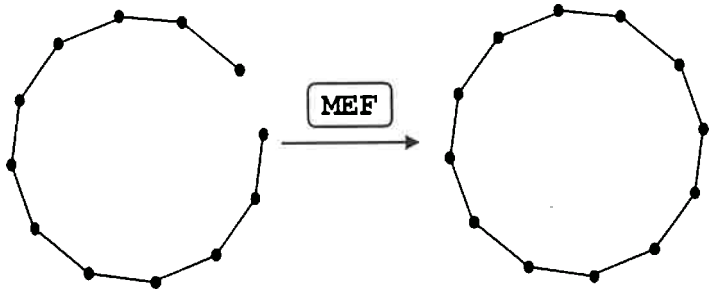


Figura 2.11. Finalização da criação de uma circunferência.

Este Operador define uma aproximação poligonal de um arco de circunferência, a partir de um ponto fornecido. Assim, o algoritmo que implementa este Operador pode ser definido utilizando-se apenas o Operador MEV (vide Figura 2.10). Para criar uma circunferência, será necessário definir um arco de 360° e o último Operador de Euler MEV deve ser substituído por um operador MEF (vide Figura 2.11).

3. Aritmética Intervalar e sua Aplicação em Modelagem de Sólidos

Nesta seção será apresentada a Aritmética Intervalar, e uma possível aplicação para Modelagem de Sólidos. Shimada [16] estudou a rerpresentação e os casos de incidência entre elementos geométricos primitivos. Ele aplicou os casos de incidência para implementar Operações Booleanas [7]. Neste trabalho, os casos de incidência serão aplicados para implementar o critério de simplificação para o algoritmo de simplificação de malhas.

Na primeira parte desta seção será apresentada a representação dos elementos geométricos vértice, linha e vetor na aritmética intervalar. Desta forma, será possível compreender o significado geométrico da aritmética intervalar.

Na segunda parte, os testes de incidência são analisados. Os testes de incidência são importantes na implementação dos Operadores Booleanos [7], onde é necessário verificar a incidência dos elementos geométricos de um sólido em relação aos elementos geométricos do outro sólido. Neste trabalho, os testes de incidência serão utilizados para verificar se existe alguma auto-intersecção no modelo sólido fornecido para ser simplificado. Os casos considerados são ponto e ponto, ponto e linha e ponto e plano.

3.1 Aritmética Intervalar arredondada

O intervalo $[a,b]$ é um conjunto de números reais definido por:

$$[a,b]=\{x \mid a \le x \le b\} \tag{3}$$

Onde as seguintes operações de aritmética intervalar são definidas:

$$
\begin{aligned}
&[a,b]+[c,d]=[a+c,b+d]\\
&[a,b]-[c,d]=[a-d,b-c]\\
&[a,b]\cdot[c,d]=[\min(a\cdot c,a\cdot d,b\cdot c,b\cdot d),\max(a\cdot c,a\cdot d,b\cdot c,b\cdot d)]\\
&[a,b]/[c,d]=[\min(a/c,a/d,b/c,b/d),\max(a/c,a/d,b/c,b/d)]
\end{aligned}
\qquad (4)
$$

Se a aritmética de ponto flutuante é utilizada para implementar estas operações de aritmética intervalar, então não é garantido que o arredondamento dos contornos esteja sendo realizado corretamente. Em contrapartida, a aritmética intervalar arredondada assegura que os contornos são determinados de modo a conter o intervalo exato, assim definem-se as seguintes operações para álgebra intervalar arredondada [16]:

$$
\begin{aligned}
&[a,b]+[c,d]=[a+c-\varepsilon,b+d+\varepsilon]\\
&[a,b]-[c,d]=[a-d-\varepsilon,b-c+\varepsilon]\\
&[a,b]\cdot[c,d]=[\min(a\cdot c,a\cdot d,b\cdot c,b\cdot d)-\varepsilon,\max(a\cdot c,a\cdot d,b\cdot c,b\cdot d)+\varepsilon]\\
&[a,b]/[c,d]=[\min(a/c,a/d,b/c,b/d)-\varepsilon,\max(a/c,a/d,b/c,b/d)+\varepsilon]
\end{aligned}
\qquad (5)
$$

Onde ε representa a diferença entre o número em ponto flutuante determinado e o número em ponto flutuante imediatamente superior. Quando forem realizadas operações padrões utilizando-se de números intervalares, as extremidades inferior e superior são estendidas para incluir o seu número em ponto flutuante imediatamente anterior e posterior, respectivamente. Então, o comprimento do resultado é aumentado de $2\cdot\varepsilon$ e o resultado será confiável nas operações subseqüentes. O valor de ε depende do número que está sendo representado, sendo que ele deve ser determinado a cada operação. Maekawa (apud Abrams et al. [2]) utilizou apenas funções matemáticas da linguagem C padrão [10] para calcular o valor de ε. Apenas para ilustração, o algoritmo utilizado está listado abaixo:

```
double ulp(const double x)
{
    double ulp;
    int exp;

    frexp(x, &exp);
    ulp = ldexp(0.5, exp-52);

    return ulp;
}
```

Abrams et al. [2] aprimoraram o algoritmo de modo que ele utilizasse apenas operações com bits. Apenas para ilustração, o algoritmo é apresentado abaixo (entretanto, para maiores detalhes sobre o seu funcionamento, recomenda-se referenciar a literatura):

```
typedef union {
    double dp;
    unsigned short sh[4];
} Double;

#define MSW 3 /* 0 if the left-most 16-bit is
sh[0] */
              /* 3 if the left-most 16-bit is
sh[3] */

static unsigned short mask[16] = {
    0x0001, 0x0002, 0x0004, 0x0008, 0x0010,
    0x0020, 0x0040, 0x0080, 0x0100, 0x0200,
    0x0400,  0x0800,  0x1000,  0x2000,  0x4000,
0x8000
};

double ulp(double x)
{
    Double X, U;
    int bit, e1, word;

    X.dp = x;
    X.sh[MSW] &= 0x7ff0;
    U.dp = 0.0;
    if (X.sh[MSW] > 0x0340)
        U.sh[MSW] = X.sh[MSW] - 0x0340;
    else {
        e1 = (X.sh[MSW] >> 4) - 1;
        word = e1 >> 4;
        if (MSW == 0) word = 3 - word;
        bit = e1 % 16;
        U.sh[word] |= mask[bit];
    }

    return U.dp;
}
```

A maioria dos ambientes de programação intervalar possui operadores lógicos do tipo certamente e possivelmente [1]. Onde certamente significa que o operador é verdadeiro para cada par de elementos. Os exemplos abaixo são fornecidos para clarificar estes conceitos:

- $[a_1,b_1]$ é certamente menor que $[a_2,b_2]$ se $x_1 < x_2$ para todo $x_1 \in [a_1,b_1]$ e $x_2 \in [a_2,b_2]$, $\Rightarrow b_1 < a_2$;
- $[a_1,b_1]$ é certamente igual a $[a_2,b_2]$ se $x_1 = x_2$ para todo $x_1 \in [a_1,b_1]$ e $x_2 \in [a_2,b_2]$, $\Rightarrow a_1 = b_1 = a_2 = b_2$.

Possivelmente significa que o operador é verdadeiro para algum par de elementos:

- $[a_1,b_1]$ é possivelmente menor que $[a_2,b_2]$ se $x_1 < x_2$ para algum $x_1 \in [a_1,b_1]$ e $x_2 \in [a_2,b_2]$, $\Rightarrow a_1 < b_2$;
- $[a_1,b_1]$ é possivelmente igual a $[a_2,b_2]$ se $x_1 = x_2$ para algum $x_1 \in [a_1,b_1]$ e $x_2 \in [a_2,b_2]$, $\Rightarrow a_1 \leq a_2 \leq b_1$ ou $a_1 \leq b_2 \leq b_1$.

Um requisito importante, é que o intervalo seja mantido sempre o mais estreito possível. Um dos teoremas fundamentais da aritmética intervalar expressa que duas expressões algebricamente equivalentes não são necessariamente equivalentes em aritmética intervalar. Por exemplo, considere o polinômio $f(x) = 2x^3 - 3x^2 + 1$ ao ser desenvolvido em $\mathbf{X} = [1,2]$ fornece $[0,5]$, enquanto que a expressão equivalente $f(x) = x^2 \cdot (2x - 3) + 1$ fornece $[-3,5]$. A razão para isto está baseada no fato que aritmética de intervalar não segue as mesmas regras como a aritmética para reais números.

3.2. Vértice Intervalar

Cada coordenada do vértice é representada por um valor intervalar. E cada valor intervalar possui: extremo inferior e extremo superior. A representação bidimensional do vértice intervalar é um retângulo, e a tridimensional, um bloco (Figura 3.1).

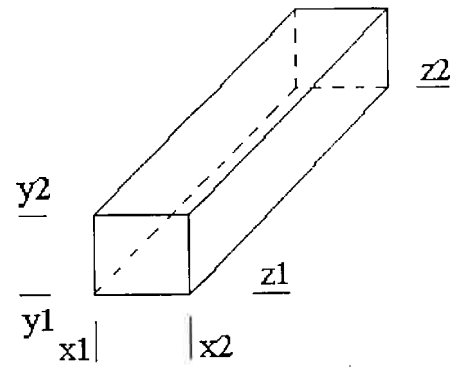


Figura 3.1. Representação tridimensional de um vértice intervalar.

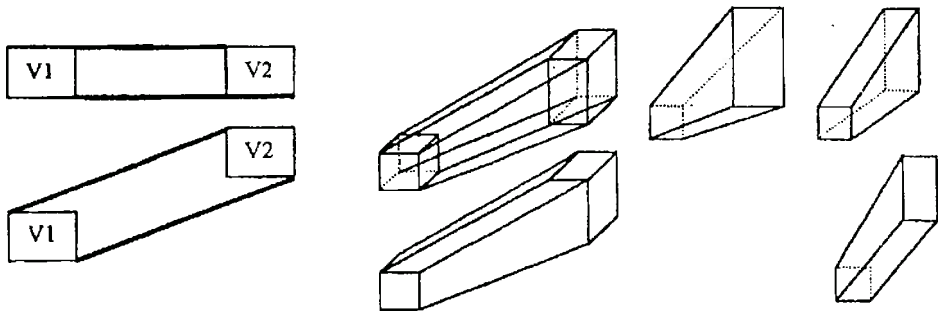


Figura 3.2. Linha intervalar bidimensional e tridimensional. Vistas ortográficas da linha tridimensional.

3.3. Linha Intervalar

A linha intervalar é formada por dois vértices intervalares. A representação bidimensional da linha intervalar é um polígono de seis lados; e a tridimensional, um poliedro (Figura 3.2).

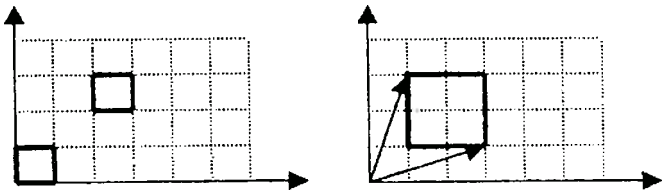


Figura 3.3. Gráfico do vetor v - ([1,3], [1,3]).

3.4. Vetor Intervalar

O vetor é um elemento aparentemente simples, mas que permite ilustrar a possibilidade do intervalo aumentar em demasia. Um vetor é definido como:

$$v = \overline{P1P2} = P2 - P1 \qquad (6)$$

sendo

$$P1 = ([x1, x2], [y1, y2], [z1, z2])$$
$$P2 = ([x3, x4], [y3, y4], [z3, z4]) \qquad (7)$$

Logo, utilizando as operações básicas definidas para aritmética intervalar, o vetor v é obtido como sendo:

$$v = ([x3 - x2, x4 - x1], [y3 - y2, y4 - y1], [z3 - z2, z4 - z1]) \qquad (8)$$

Um exemplo 2D é apresentado na Figura 3.3 :

$$P1 = ([0,1], [0,1])$$
$$P2 = ([2,3], [2,3]) \qquad (9)$$
$$v = ([2-1, 3-0], [2-1, 3-0]) = ([1,3], [1,3])$$

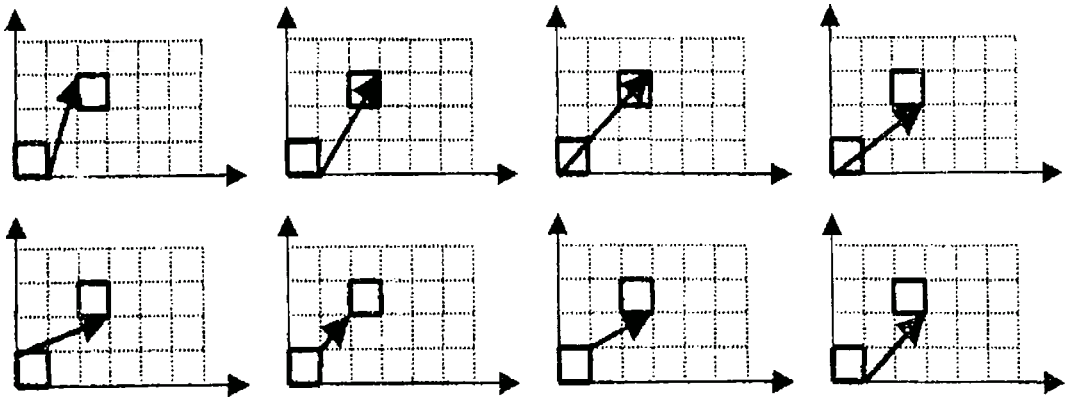


Figura 3.4. Exemplos do alcance do vetor v =([1,3],[1,3]).

$$P1 = ([0,1],[0,1])$$
$$P2 = ([2,3],[2,3])$$
$$\overline{P1P2} = ([1,3],[1,3])$$
$$P1 + \overline{P1P2} = ([0,1],[0,1]) + ([1,3],[1,3]) = ([1,4],[1,4])$$

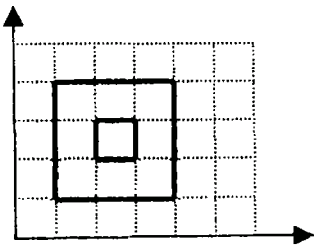


Figura 3.5. Exemplo de aumento de intervalo.

A Figura 3.4. ilustra o vetor intervalar ([1,3],[1,3]) considerando que ele é a união entre os pontos P1 e P2.

Ao calcular-se $P1 + \overline{P1P2}$ obtém-se ([1,4],[1,4]), que deveria ser na realidade o mesmo que P2. Apesar de serem valores possivelmente iguais, este resultado ([1,4],[1,4]) possui intervalos maiores do que o inicial ([2,3],[2,3]) (vide Figura 3.5). Desta maneira, diz-se que a Aritmética Intervalar representa o histórico de operações. Assim, quanto mais operações aritméticas forem feitas para se determinar um valor, maior será o seu intervalo associado.

$P_1([x_1, x_2],[y_1, y_2],[z_1, z_2])$
$P_2([x_3, x_4],[y_3, y_4],[z_3, z_4])$

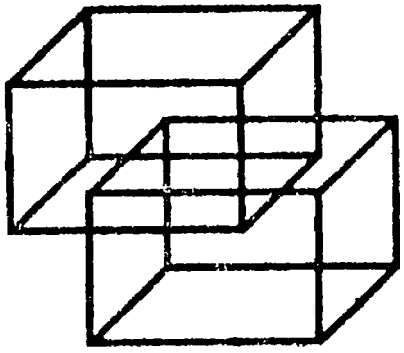


Figura 3.6. Pontos intervalos P1 e P2.

3.5. Teste de Incidência - Ponto e Ponto

Dados dois pontos P1 e P2 (Figura 3.6). Para que o ponto P_1 seja considerado incidente ao P_2 é necessário que todas estas condições estejam satisfeitas:

- Intervalo $[x_1,x_2]$ tenha algum valor de intersecção com o intervalo $[x_3,x_4]$
- Intervalo $[y_1,y_2]$ tenha algum valor de intersecção com o intervalo $[y_3,y_4]$
- Intervalo $[z_1,z_2]$ tenha algum valor de intersecção com o intervalo $[z_3,z_4]$

Se for confirmada a incidência, um novo ponto P_3 é criado, que engloba os dois pontos P1 e P2, conforme representado na Figura 3.7.

$$P_3 = ([\min(x_1, x_3), \max(x_2, x_4)],[\min(y_1, y_3), \max(y_2, y_4)],[\min(z_1, z_3), \max(z_2, z_4)])$$

3.6 Teste de Incidência - Ponto e Linha

Shimada [16] demonstrou que é possível determinar se um ponto é incidente a um segmento de aresta pela seguinte sequência de operações:

$$
\begin{aligned}
E_1 &= [(P_2 - P_1) \bullet (P_3 - P_1)] \\
E_2 &= [(P_1 - P_2) \bullet (P_3 - P_2)] \\
E_3 &= [(P_1 - P_3) \bullet (P_2 - P_3)] \\
A_3 &= [(P_1 - P_3) \times (P_2 - P_3)]
\end{aligned}
\qquad (10)
$$

Por estas operações será possível determinar:

1. se a área do triângulo formado por P1, P2 e P3 é possivelmente igual a zero;
2. se o ponto P3 está entre os pontos P1 e P2 – se E1 for possivelmente maior que zero, e E2 for possivelmente maior que zero e E3 for possivelmente menor que zero (vide Figura 3.7 (a)).

3.7. Teste de Incidência - Ponto e Plano

É analisado se um ponto P4 pertence a um plano formado pelos pontos P1, P2 e P3. Shimada [16] demonstrou que é possível determinar se um ponto é incidente a um plano pela seguinte sequência de operações:

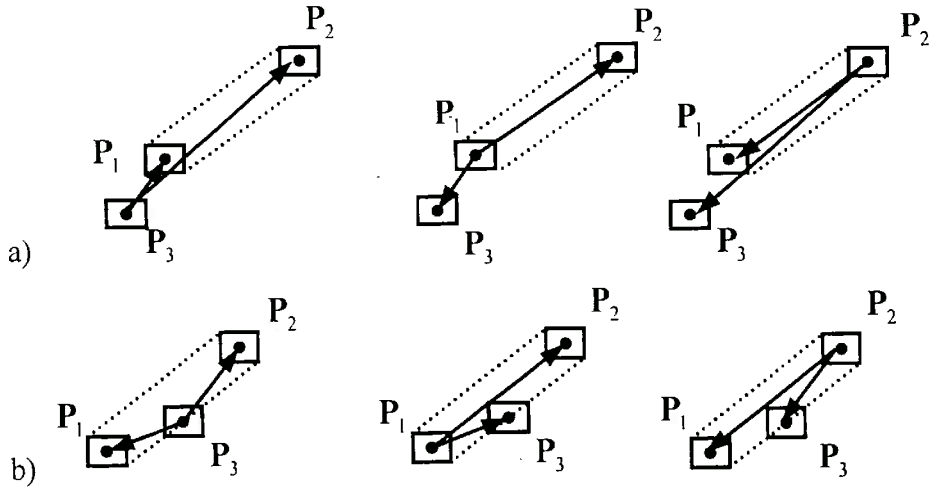


Figura 3.7. a) Ponto P3 não é incidente à linha intervalar P1P2; b) Ponto P3 é incidente à linha intervalar P1P2.

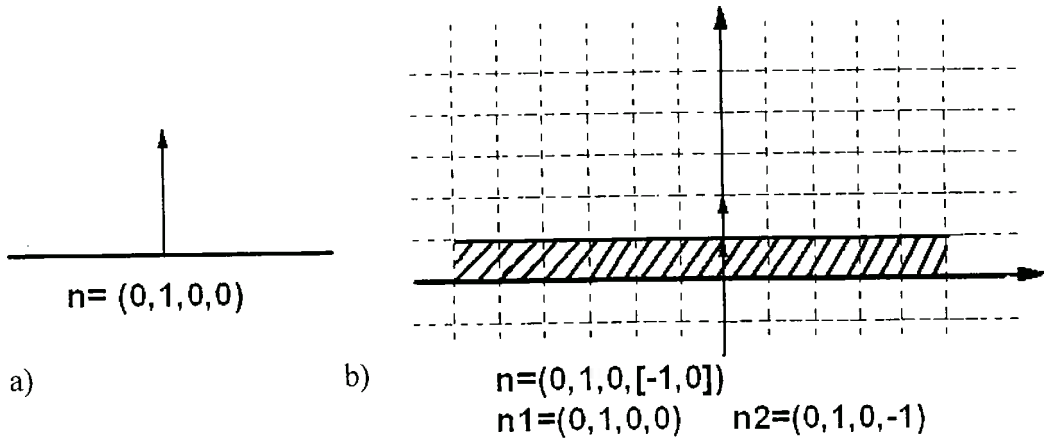


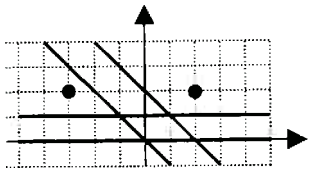
Figura 3.8. a) Plano n; b) Plano Intervalar.

$$n = (x, y, z, w)$$
$$P_4 = (x_4, y_4, z_4) \qquad (11)$$
$$x * x_4 + y * y_4 + z * z_4 + w = 0$$

Assim, estamos calculando a equação da face que contém os pontos P1, P2 e P3. Em seguida, aplica-se a equação para o ponto P4 e se o valor é possivelmente igual a zero, então o ponto P4 está sobre o plano definido por P1, P2 e P3.

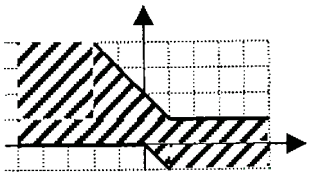
No exemplo da Figura 3.8.(a) é apresentado um plano e na Figura 3.8.(b), um plano intervalar, que pode ser subdivido em dois planos n1 e n2. A área hachurada entre estes dois planos está possivelmente sobre o plano intervalar n.

Na Figura 3.9.(a) temos um plano intervalar n formado pelos planos n1, n2, n3 e n4. A área hachurada está possivelmente sobre o plano intervalar n. Na Figura 3.9.(b), o ponto P = (-3,2,0) está na área hachurada e o teste indica que o valor é possivelmente igual a zero, no exemplo o resultado foi [-2,2]. Para o ponto R, o resultado foi [1,4], não está sobre o plano n.



```
n=([0,1],1,0,[-1,0])
n1=(0,1,0,0)
n2=(0,1,0,-1)
n3=(1,1,0,0)
n4=(1,1,0,-1)
```

(a)



```
n=([0,1],1,0,[-1,0])
P=(-3,2,0)
R=(2,2,0)

Teste de incidência para o ponto P:
[0,1]*(-3)+1*2+
[-1,0]*0+[-1,0]=[-2,2]
Teste de incidência para o ponto R:
[0,1]*2+1*2+
[-1,0]*0+[-1,0]=[1,4]
```

(b)

Figura 3.9. a) Plano intervalar formado por n1, n2, n3 e n4; b) Exemplo de verificação do pontos P e R sobre o plano n.

4. Algoritmo para Simplificação da Malha Poliedral

Neste capítulo será apresentado o algoritmo para simplificação da malha poligonal. O algoritmo proposto é a composição de algumas propostas existentes na literatura, sendo que alguns conceitos serão adaptados para aplicar a Aritmética Intervalar.

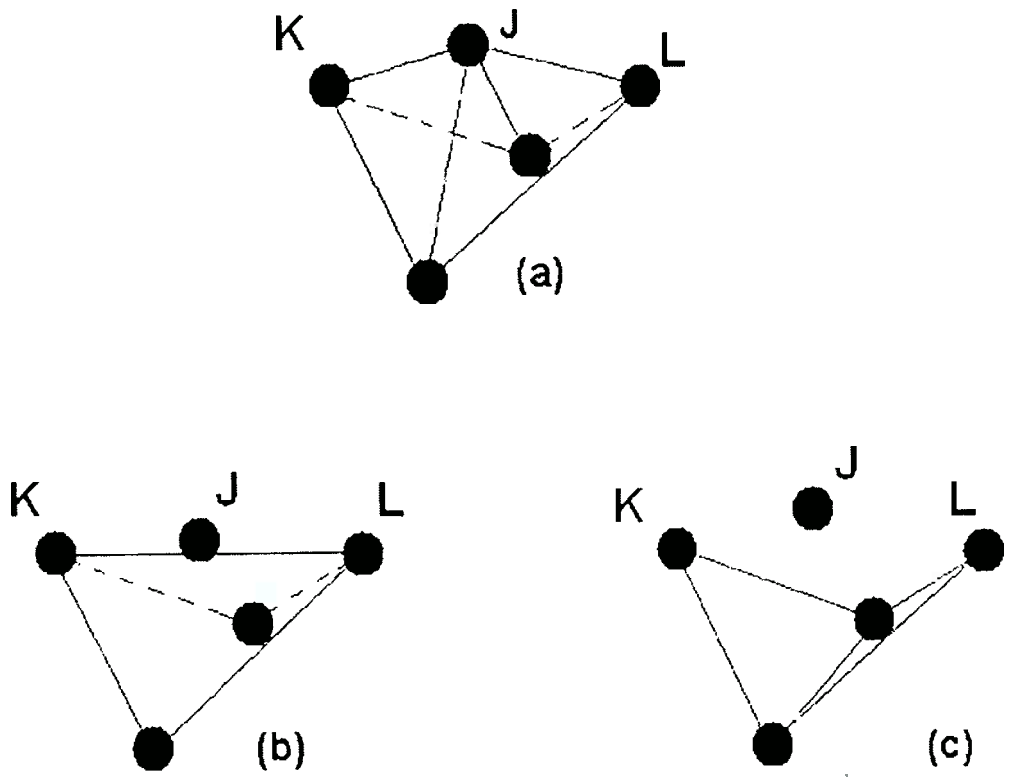


Figura 4.1. Critério geométrico aplicado para controlar a simplificação da malha poliedral: poliedro inicial onde o vértice J será removido; poliedro simplificado válido; poliedro simplificado inválido.

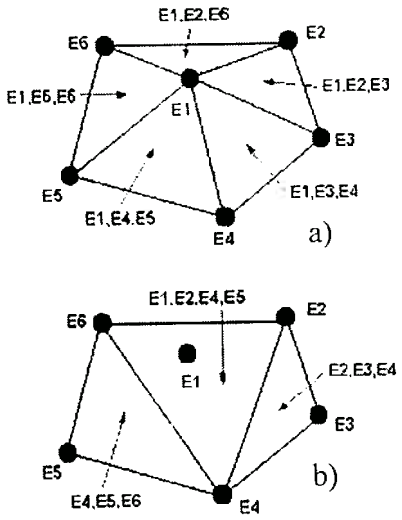


Figura 4.2. Lista de vértices associados às faces.

Serão definidos alguns operadores locais, definidos como sequência de operadores de Euler, para realizar a simplificação da malha poliedral.

Será adaptado o conceito de hierarquia de tolerância proposto por Véron e Léon [18,19]. Segundo este conceito, é necessário verificar se o resultado final da aplicação de um operador de simplificação satisfaz o critério de simplificação. Em caso afirmativo, o operador de simplificação pode ser aplicado. Os vértices eliminados devem ser associados às faces da malha poliedral resultante.

Como explicado por Véron e Léon [18,19], a Figura 4.1.(a) ilustra a malha poliedral original. Está sendo considerado que o vértice J será removido. A situação após a remoção do vértice J deverá ser a representada na Figura 4.1.(b) ou na Figura 4.1.(c)? O correto é a representada na Figura 4.1.(b), pois o vértice J será associado à aresta LK. Na Figura 4.1.(c) o vértice J não está associado a nenhum elemento geométrico primitivo. Após a simplificação o vértice J estará associado à aresta LK.

A Figura 4.2. ilustra um outro caso de remoção de vértice. A Figura 4.2.(a) exibe a malha poliedral original, e a Figura 4.2.(b) ilustra a situação após a remoção do vértice E1. Cada face possui uma lista de vértices associados. É

importante observar que a lista de vértices inclui também os vértices removidos. Desta maneira, após aplicar um operador de simplificação, é necessário associar os vértices removidos a elementos geométricos primitivos.

O critério de simplificação, que deve ser mantido como verdadeiro durante todo o processo de simplificação, é que todos os vértices originais do sólido devem ser incidentes a pelo menos uma face do sólido simplificado. Assim, o critério de simplificação deve ser verificado após a aplicação de cada operador de simplificação. Caso, o critério de simplificação não possa ser mantido verdadeiro, o operador de simplificação deverá ser desfeito.

Antes de iniciar a simplificação propriamente dita, todos os vértices são classificados e ordenados. A ordenação permite processar primeiro os vértices que possuem alta probabilidade de serem removidos.

4.1 Curvatura Gaussiana

Os vértices a serem removidos são ordenados por prioridade. O critério para ordenar os vértices a serem removidos é baseado na curvatura gaussiana do poliedro. A avaliação deste parâmetro é definida por:

$$K = Gaussian\ \ curvature = \frac{A_{Vertice}}{S_{Vertice}} \qquad (12)$$

Onde $A_{Vertice}$ é definido como 2π - soma dos ângulos das faces adjacentes ao vértice e $S_{vertice}$ é igual soma das áreas dos triângulos adjacentes ao vértice.

Os vértices são ordenados segundo o valor absoluto da curvatura gaussiana, do menor até o maior. Isto permite remover primeiro os vértices com curvatura gaussiana nula ou aproximadamente nula como planos, áreas aproximadamente planas ou aproximação de superfícies com pequena curvatura. Durante o

processo de simplificação, a remoção dos vértices modifica a curvatura gaussiana dos vértices adjacentes ao vértice removido. Conseqüentemente, é necessário atualizar a lista ordenada de vértices a serem removidos para refletir a nova realidade.

4.2 Operadores

A seguir serão apresentados os operadores implementados:

- remove o vértice se ele for incidente a um vértice adjacente;
- remove o vértice se ele for incidente a uma aresta definida pelos seus vértices adjacentes;
- remove o vértice se for incidente ao plano definido pelos seus vértices adjacentes.

4.2.1 Remove Vértice – Incidência Vértice-Vértice

Dado um vértice, se ele for incidente a um dos seus vértices adjacentes então este vértice será removido. O algoritmo abaixo verifica se este operador deve ser aplicado:

```
PtrHalfEdge itHE1 = vtx->getHalfEdge();
tnVector<T,4> tV1 = vtx->getCoord(), tV2;
bool flagRemover = false;
PtrHalfEdge itHE2 = itHE1;
do {
    tV2 = itHE2->mate()->Vtx()->getCoord();
    double v1 = fabs(tV1[0] - tV2[0]);
    double v2 = fabs(tV1[1] - tV2[1]);
    double v3 = fabs(tV1[2] - tV2[2]);
    if (v1 <= DetPP && v2 <= DetPP && v3 <= DetPP)
        flagRemover = true;
} while ((itHE2 = itHE2->mate()->Nxt()) != itHE1);
```

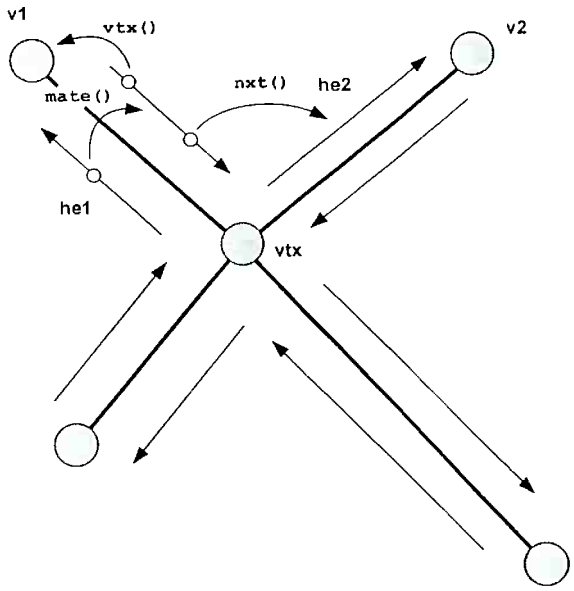


Figura 4.3. Vértices adjacentes a um dado vértice.

Onde para um dado vértice `vtx`, recuperamos a sua halfedge associada (por meio do método `getHalfEdge()`) para realizar uma varredura nos vértices adjacentes ao vértice `vtx`. A Figura 4.3 ilustra o significado dos métodos `mate()`, `nxt()` e `vtx()`. O método `vtx()` retorna o ponteiro para o vértice associado à halfedge, que é o vértice de onde sai a halfedge. O método `mate()` retorna o ponteiro para a halfedge oposta em relação à aresta. O método `nxt()` retorna a próxima halfedge no circuito de halfedges ao redor da face. O método `getCoord()` retorna o vetor de coordenadas do vértice. Caso a variável `flagRemover` seja verdadeira, então o operador deve ser aplicado.

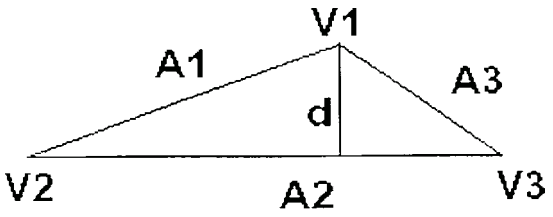


Figura 4.4 Cálculo da incidência vértice-aresta.

4.2.1 Remove Vértice – Incidência Vértice-Aresta

Considere-se um vértice candidato a ser removido, analisam-se os vértices adjacentes a este vértice. Caso existam dois vértices adjacentes a este vértice candidato, tal que o vértice candidato seja incidente à aresta definida por estes dois, então o vértice candidato pode ser removido (vide Figura 4.4). O algoritmo abaixo verifica se este operador deve ser aplicado:

```
itHE2 = itHE1;
do {
   PtrHalfEdge itHE3 = itHE1;
   do {
       if (itHE2 != itHE3)
           if (CheckCoordInEdge(tV1,
                   itHE2->mate()->Vtx()->getCoord(),
                   itHE3->mate()->Vtx()->getCoord(),
                   DetPP))
               flagRemover = true;
   } while ((itHE3 = itHE3->mate()->Nxt()) != itHE1);
} while ((itHE2 = itHE2->mate()->Nxt()) != itHE1);
```

Neste algoritmo são realizadas duas varreduras pelos vértices que estão ao redor do vértice candidato. O método `CheckCoordInEdge()` verifica se ocorre a incidência vértice-aresta entre o trio de coordenadas. Caso a variável `flagRemover` seja verdadeira, então o operador deve ser aplicado.

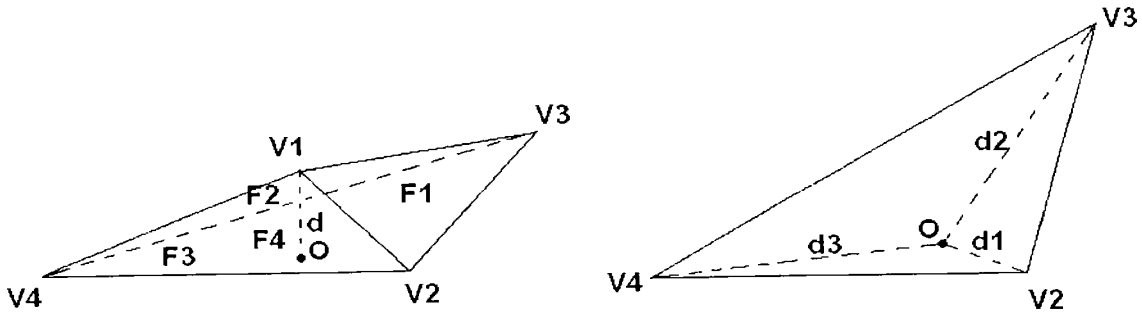


Figura. 4.5 Cálculo da distância para o caso 3.

4.2.2 Remove Vértice – Incidência Vértice-Plano

Considerando-se um vértice candidato a ser removido, determina-se se o vértice candidato é incidente à face média definida por seus vértices adjacentes. Se o vértice candidato for incidente, então este vértice pode ser removido (vide Figura 4.5).

```
vector<tnVector<T,4> > vcoord;
PtrHalfEdge itHE1 = vtx->getHalfEdge();
bool flagRemover = false;
PtrHalfEdge itHE2 = itHE1;
do {
    tV2 = itHE2->mate()->Vtx()->getCoord();
    vcoord.push_back(tV2);
} while ((itHE2 = itHE2->mate()->Nxt()) != itHE1);

if (CheckCoordInFace(vcoord, tV1, DetPP))
    flagRemover = true;
```

Neste algoritmo as coordenadas dos vértices adjacentes ao vértice candidato são armazenadas em um vetor, de modo a definir um polígono. O método `CheckCoordInFace()` verifica se ocorre a incidência entre o vértice candidato e o polígono definido pelo vetor de coordenadas.

4.3 Remoção do Vértice

Em qualquer um dos operadores, o vértice é removido eliminando-se todas as arestas que emanam do vértice candidato. O algoritmo que realiza a remoção desta arestas está descrito abaixo:

```
PtrHalfEdge phe = vtxI->getHalfEdge()->Nxt();
TLoop<T> *wloop;
while (true) {
    PtrHalfEdge ite = vtxI->getHalfEdge();
    if (ite->getLoop() == ite->mate()->getLoop()) {
        wloop = ite->getLoop();
        TKEV<T>::instance().low(ite, ite->mate());
        break;
    }
    else
        TKEF<T>::instance().low(ite, ite->mate());
}
```

A Figura 4.6 ilustra esta operação, todas as arestas (exceto a última) são removidas pela aplicação do operador de Euler KEF, a última aresta é removida pela aplicação do operador de Euler KEV.

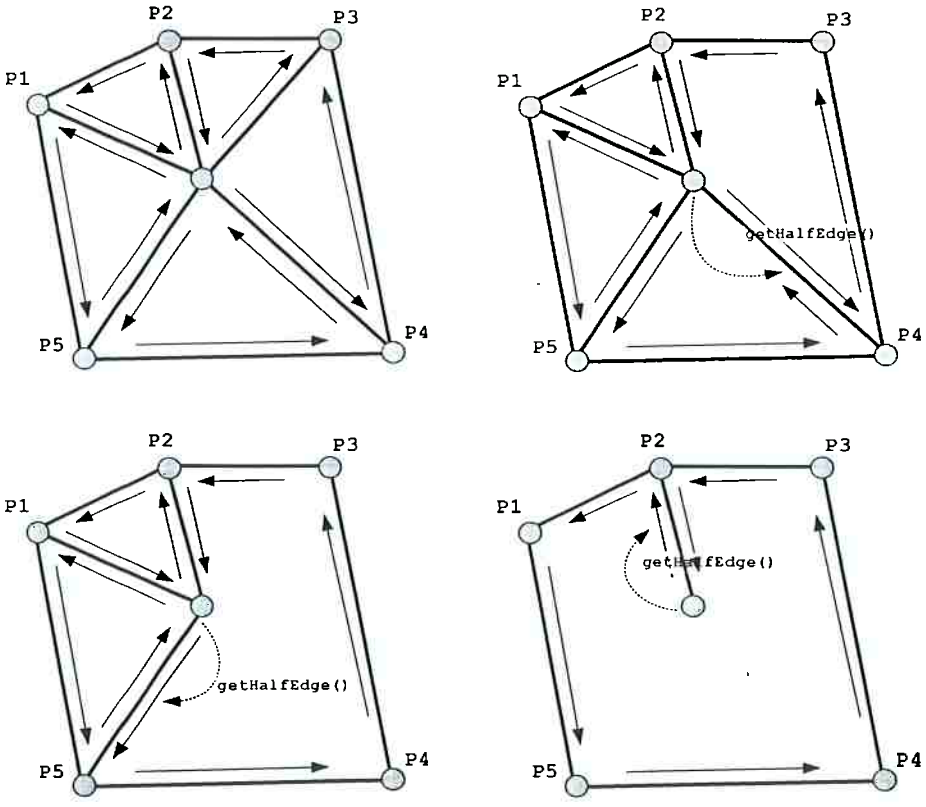


Figura 4.6. Exemplo de remoção das arestas ao redor do vértice.

4.3 Retriangulação Poligonal

Após remover as arestas ao redor do vértice candidato, é necessário realizar a retriangulação. Será utilizada a proposta de retriangulação feita por Véron e Léon [18,19]. O primeiro passo neste algoritmo é determinar o vetor normal médio das faces ao redor do vértice candidato:

$$N = \frac{\sum_{i=1}^{m} n_i}{\left\| \sum_{i=1}^{m} n_i \right\|} \qquad (13)$$

onde n_i é a normal unitária da i-ésima face adjacente ao vértice a ser removido e m é o número de faces adjacentes.

Em seguida o algoritmo faz uso de uma iteração que cria novos triângulos utilizando-se de três vértices consecutivos ordenados segundo o contorno

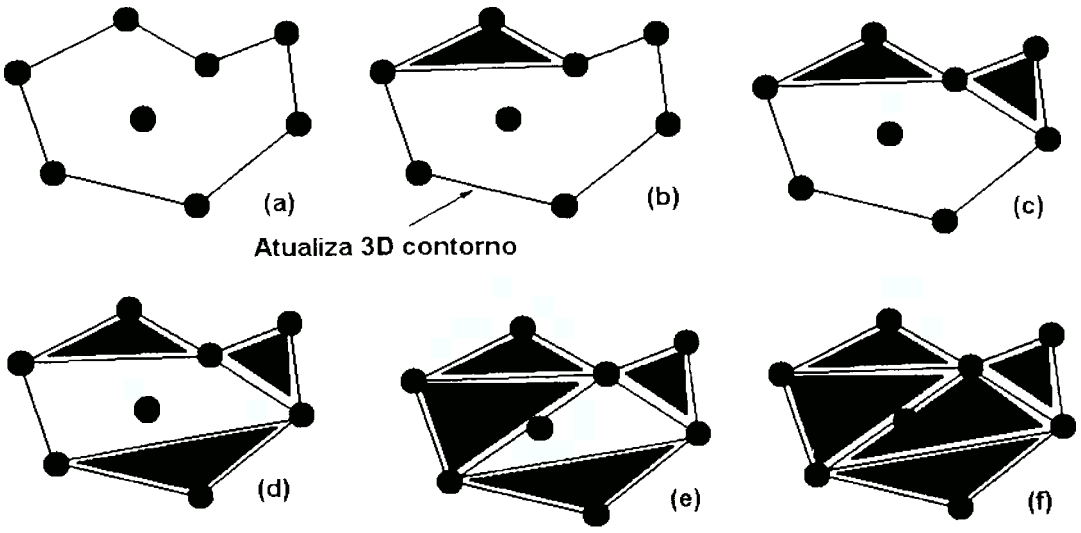


Figura 4.7. O iteração de processo de enredando de 3D poliedro usando o critério geométrico.

poligonal. A cada triângulo criado, o contorno poligonal é atualizado, pois um vértice é removido de sua lista de vértices. A Figura 4.7 ilustra este processo iterativo.

A iteração começa pela seleção de três vértices consecutivos do contorno poligonal. Um vértice é escolhido arbitrariamente no contorno (vértice J da Figura 4.7). O último vértice desta iteração, será o primeiro vértice da próxima iteração (que no caso da Figura 4.7 corresponde ao vértice L).

Uma superfície plana P é definida pelo vértice J e sua normal N_p. (vide Figura 4.8) O vetor N_p, normal à aresta JK, é definido como:

$$N_p = N \times JK \qquad (14)$$

É necessário, então, verificar se o vértice L está no mesmo semi-plano que o vértice removido I. Isto é verificado por meio das expressões abaixo:

$$\begin{aligned} test &= N_p \cdot JL = N(JK \times JL) \\ referência &= N_p \cdot JI = N(JK \times JI) \end{aligned} \qquad (15)$$

se ambos os produtos escalares acima produzirem resultado positivo ou negativo, é porque o vértice L está no mesmo semi-espaço que o vértice removido.

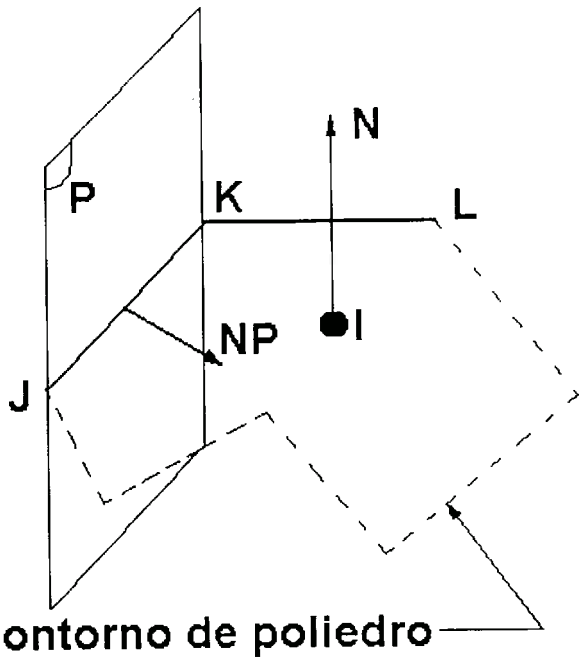


Figura 4.8. Primeiro critério para controlar o processo de triangulação.

Existe uma segunda verificação que será executada, caso a primeira seja satisfeita. Três planos P1, P2 e P3 são definidos (vide Figura 4.9). Estes planos possuem normais N_{P1}, N_{P2} e N_{P3} respectivamente. As normais são definidas pelas equações abaixo:

$$N_{P1} = N \times JK, \qquad N_{P2} = N \times KL, \qquad N_{P3} = N \times LJ \tag{16}$$

Este teste verifica se existe algum vértice do contorno que seja interno ao triângulo definido por J, K e L. Assim, este teste é aplicado a cada vértice do contorno, exceto os três vértices J, K e L. Este teste é representado pelas equações abaixo:

$$\forall n\, o\, vértice\, do\, contorno. \quad N \neq J. \quad N \neq K. \quad N \neq L.$$

$$\begin{cases} teste1 = N_{p1} \cdot JN \\ teste2 = N_{p2} \cdot KN \\ teste3 = N_{p3} \cdot LN \end{cases} \qquad (17)$$

Se (teste1,teste2 e teste3) são todos positivos, então este critério não foi satisfeito.

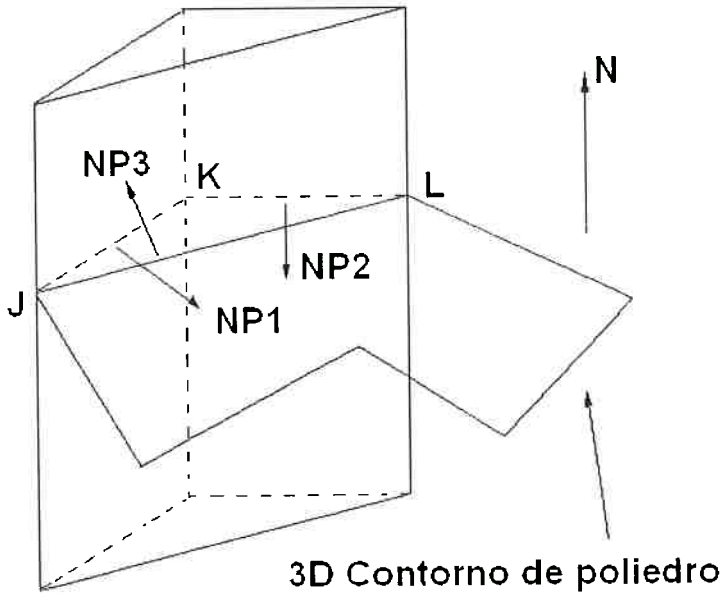


Figura 4.9. Segundo critério para controlar o processo de triangulação.

4.4 Critério de Simplificação

Será utilizado um critério de simplificação para controlar o avanço do algoritmo, evitando que o algoritmo se afaste do sólido original. Todos os vértices do sólido original são associados a vértices intervalares que devem ser incidentes a pelo menos uma face do sólido simplificado. E isto deve ser verdadeiro em todas as etapas do processo iterativo. Cada face possui a sua lista de vértices incidentes.

Inicialmente, as faces não possuem nenhum vértice incidente; entretanto, conforme o processo de remoção ocorre, o número de vértices incidentes cresce. A cada iteração, um vértice candidato ao ser removido, cria-se uma lista contendo todos os vértices incidentes às faces adjacentes ao vértice candidato e que foram removidas. Realiza-se a triangulação do polígono criado pela remoção do vértice candidato.

Para que a remoção deste vértice candidato seja aceita, é necessário que todos os vértices desta lista sejam incidentes a pelo menos um triângulo definido pelo processo de retriangulação. Em caso contrário, as operações são desfeitas e o vértice candidato não é removido.

5. Resultados

Neste capítulo, são apresentados os resultados da aplicação dos operadores de simplificação implementados no Modelador de Sólidos USPDesigner. Foram utilizados três sólidos diferentes: esfera, toróide e coelho. Estes três sólidos são utilizados com frequência para testar algoritmos de redução de malhas. A Figura 5.1 exibe uma esfera de raio 40 com 80 divisões, contendo 3.122 vértices, 6.240 faces e 9.360 arestas. Esta esfera foi utilizada para testar os algoritmos propostos.

Cada operador foi aplicado separadamente para identificar a ação de cada um deles: incidência vértice-vértice, incidência aresta-vértice e incidência face-vértice. A aplicação do operador de incidência vértice-vértice resultou em um sólido com 2.214 vértices, 4.424 faces e 6.636 arestas (vide Figura 5.2) – 70% de simplificação. Na Figura 5.2, é possível observar que a aplicação do operador de

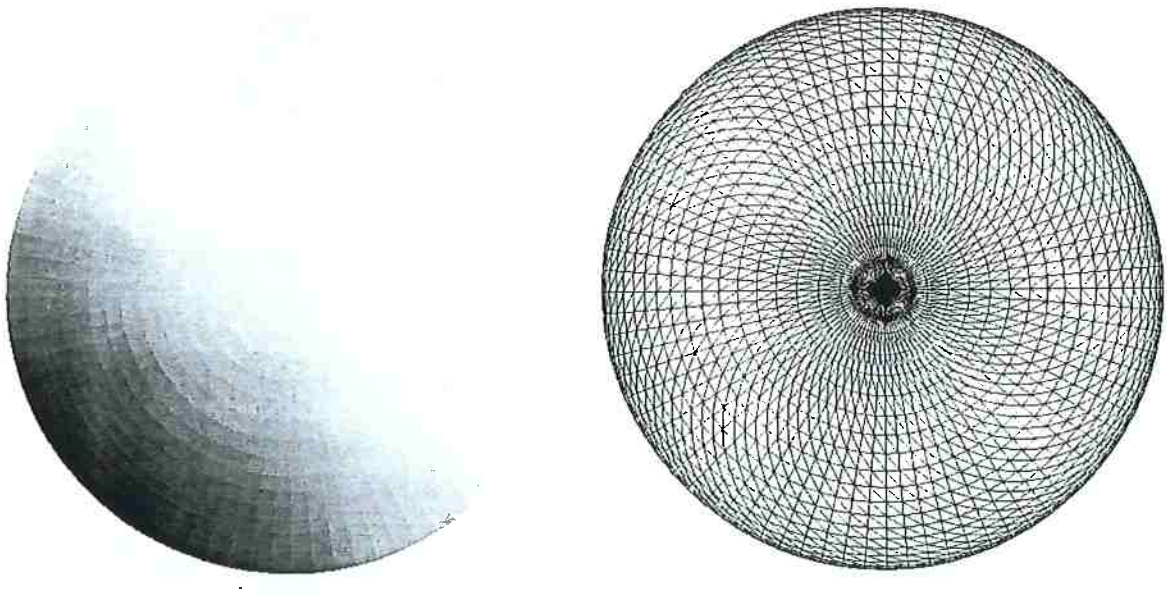


Figura 5.1. Esfera contendo 3.122 vértices, 6.240 faces e 9.360 arestas.

incidência vértice-vértice removeu principalmente os vértices próximos aos polos da esfera.

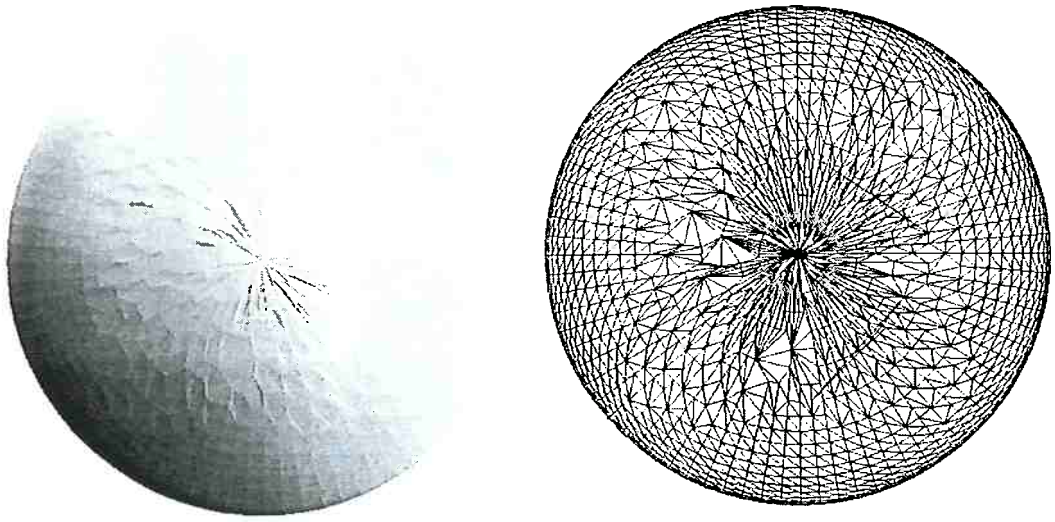


Figura 5.2. Aplicação do operador de incidência vértice-vértice à esfera.

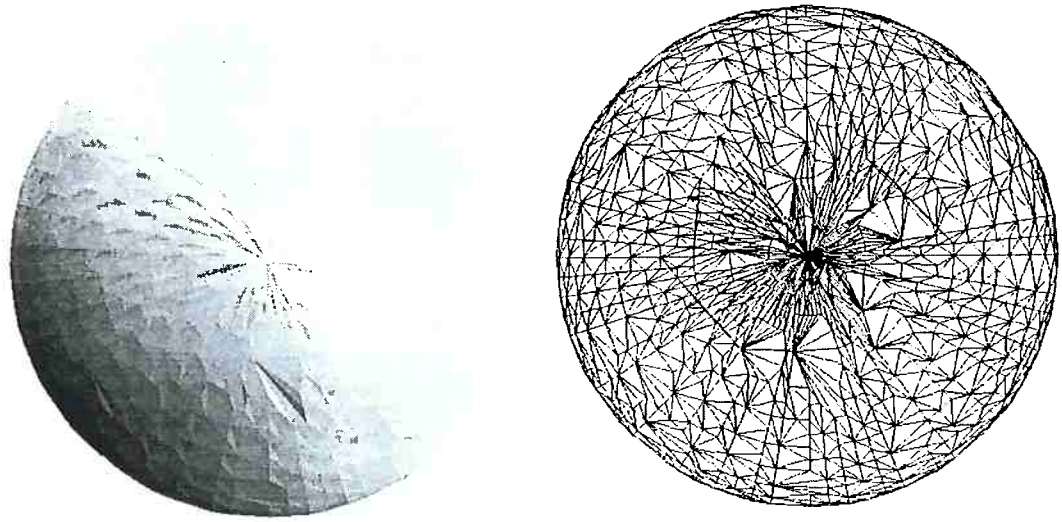


Figura 5.3. Aplicação do operador de incidência aresta-vértice à esfera.

A aplicação do operador de incidência aresta-vértice resultou em um sólido com 1.258 vértices, 2.512 faces e 3.768 arestas (vide Figura 5.3) – 40% de simplificação. É possível observar que a aplicação deste operador não se concentrou a uma única região como o operador anterior.

A aplicação do operador de incidência face-vértice resultou em um sólido com 1.258 vértices, 2.512 faces e 3.768 arestas – 40% de simplificação. É praticamente o mesmo resultado obtido pelo operador de incidência aresta-vértice. A aplicação de todos os operadores em conjunto resultou em um sólido com 1.258 vértices, 2.512 faces e 3.768 arestas (vide Figura 5.4) – 40% de

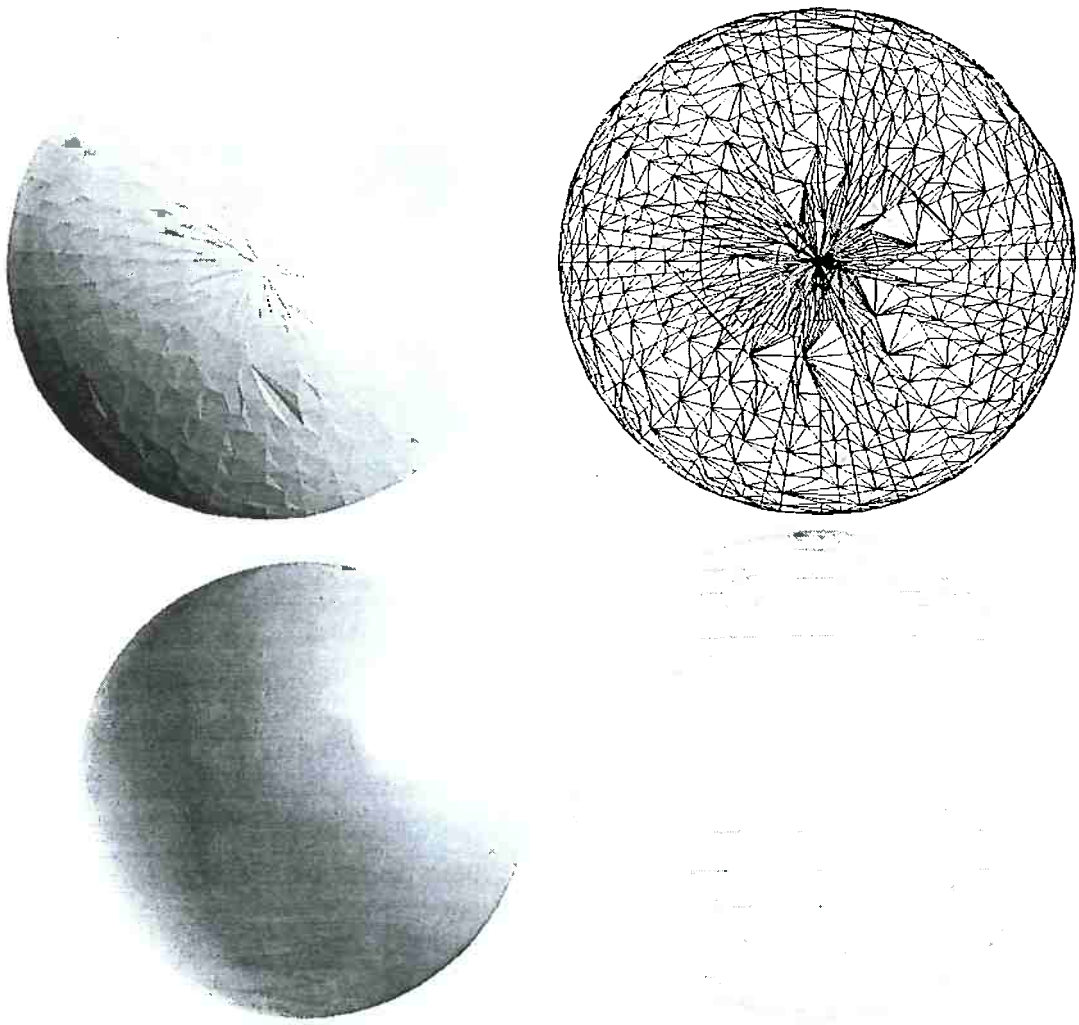


Figura 5.4. Aplicação dos três operadores de incidência à esfera.

simplificação. A tolerância utilizada foi igual a 1.0 em todas as situações consideradas. É possível observar que a aplicação dos operadores de incidência aresta-vértice e face-vértice reduziram mais a malha quando comparado ao operador de incidência vértice-vértice. Os resultados foram exportados para o formato 3DS (Autodesk 3D Studio). As duas imagens inferiores da Figura 5.4 foram feitas em um software de renderização (Phong e eliminação de superfícies escondidas).

A aplicação dos três operadores de incidência, aparentemente resultaram em pequenos arranhões no modelo. A Figura 5.5 exibe uma ampliação de uma região que apresenta um aparente arranhão. Nesta ampliação, na imagem renderizada, é possível observar que o arranhão aparentemente fica reduzido.

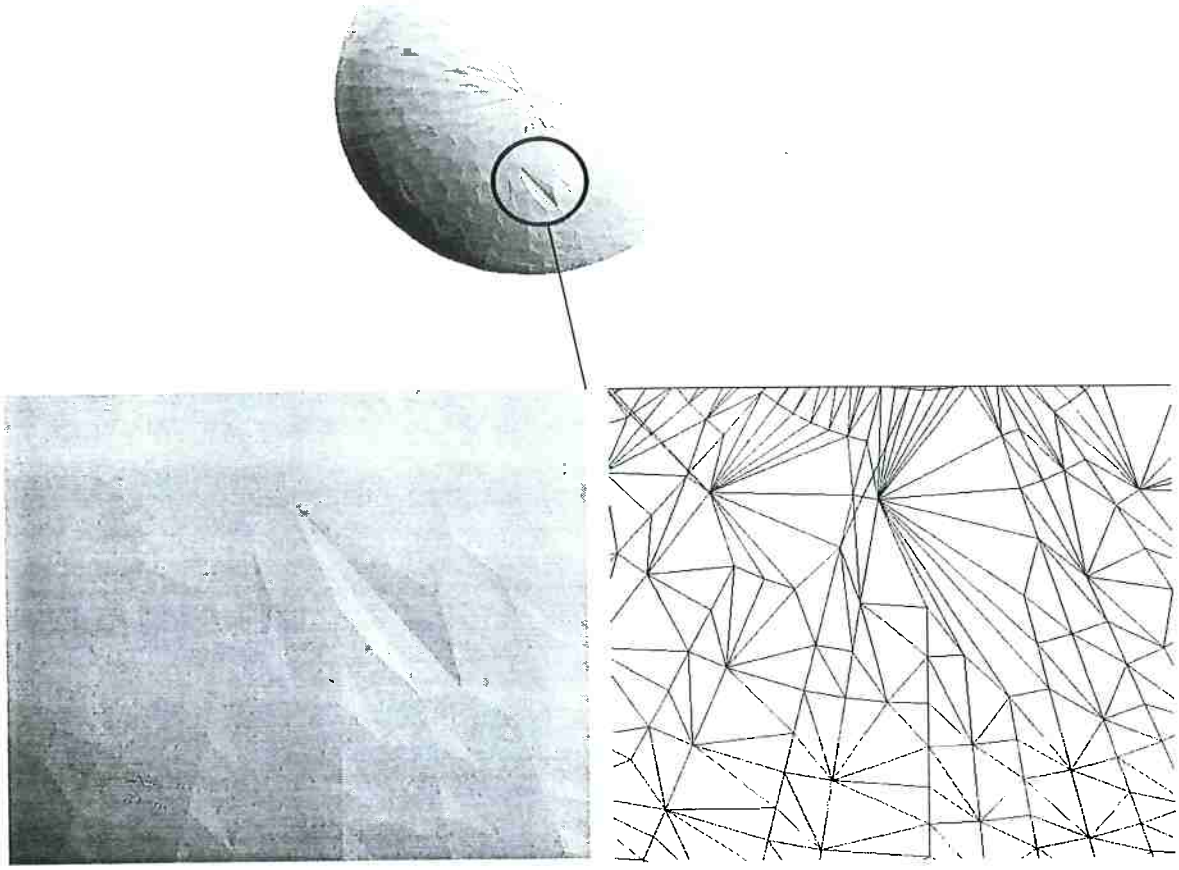


Figura 5.5. Ampliação de detalhe da esfera simplificada.

A Figura 5.6 exibe um toróide com raio maior de 40 e raio menor de 10 cm 80 divisões, contendo 6.400 vértices, 12.800 faces e 19.198 arestas. Este toróide foi utilizado para testar os algoritmos propostos. A aplicação do operador de incidência vértice-vértice resultou em um sólido com 2.551 vértices, 5.102 faces e 7.651 arestas (vide Figura 5.7) – 40% de simplificação.

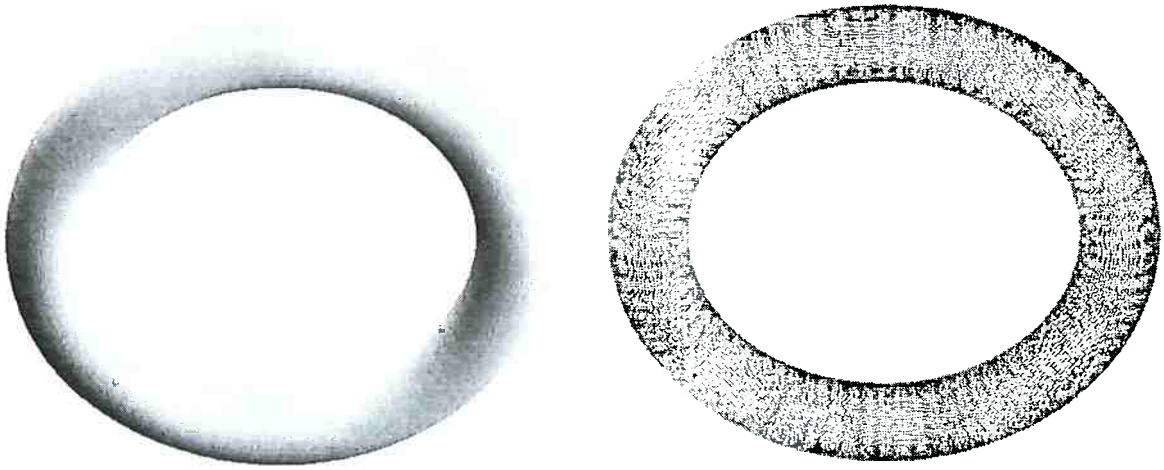


Figura 5.6. Toróide com 6.400 vértices, 12.800 faces e 19.198 arestas.

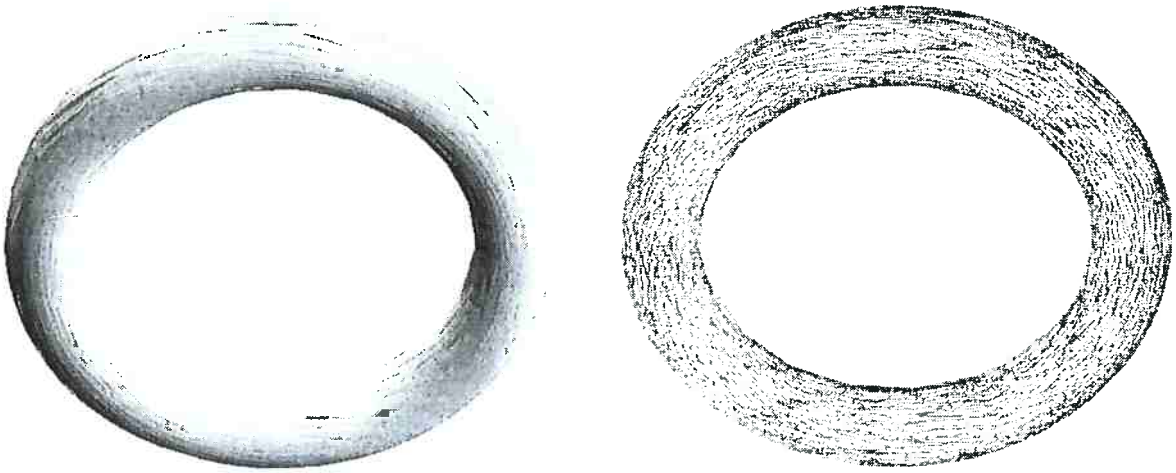


Figura 5.7. Aplicação do operador de incidência vértice-vértice ao toróide.

A aplicação do operador de incidência aresta-vértice resultou em um sólido com 2.551 vértices, 5.102 faces e 7.651 arestas que é praticamente o mesmo resultado obtido pelo operador de incidência vértice-vértice. A aplicação do operador de incidência face-vértice resultou em um sólido com 2.539 vértices, 5.078 faces e 7.615 arestas (vide Figura 5.8) – 40% de simplificação. É possível observar que os triângulos do toróide simplificado se alongam no sentido do maior raio. Este é um resultado esperado, visto que este é o sentido de menor

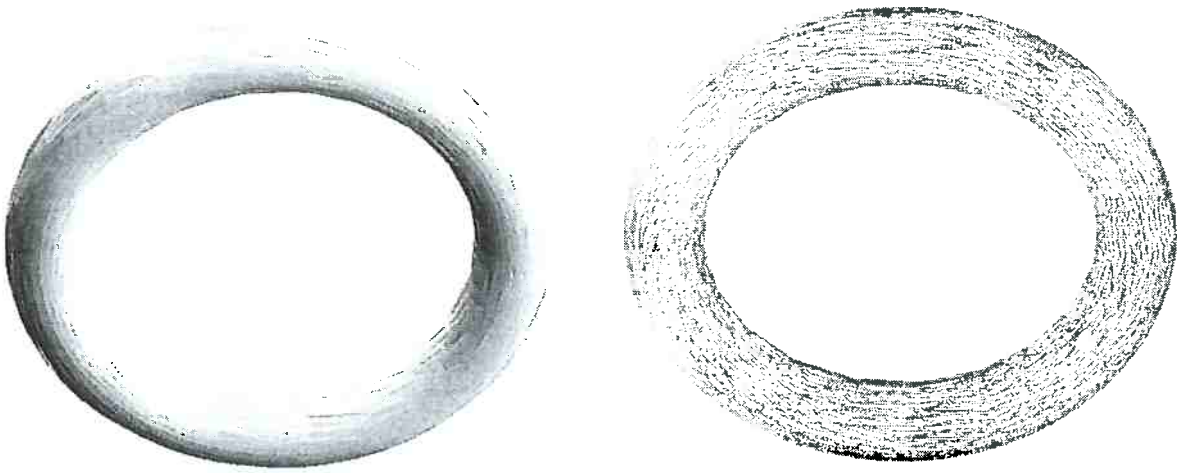


Figura 5.8. Aplicação do operador de incidência face-vértice ao toróide.

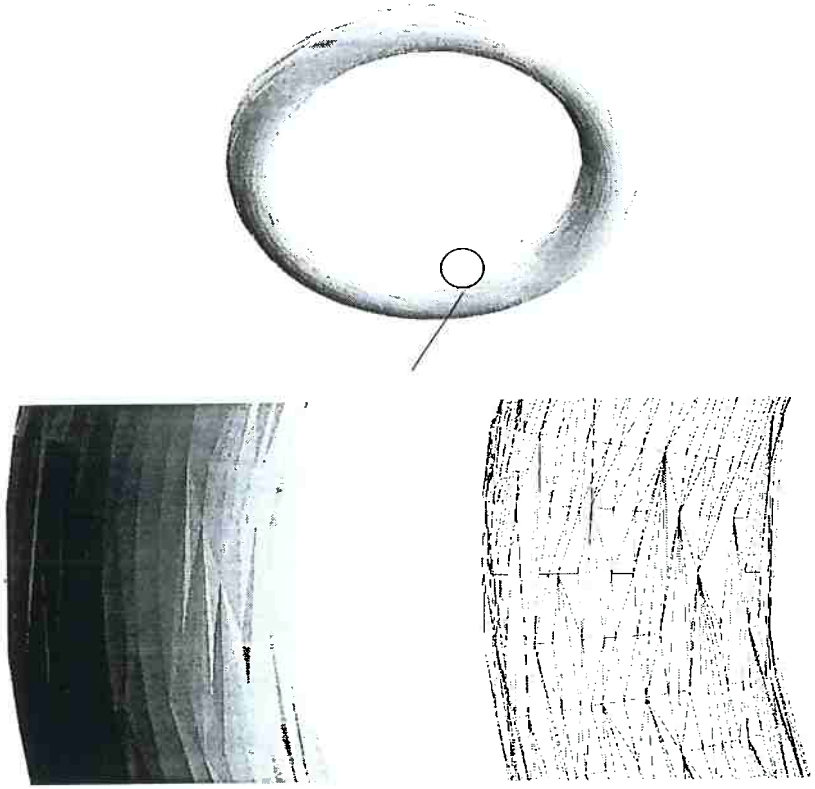


Figura 5.9. Ampliação de detalhe do toróide simplificado.

curvatura. Novamente, é possível observar pelas imagens renderizadas que existem pequenos arranhões no modelo simplificado. A Figura 5.9 exibe uma ampliação do toróide simplificado. Pela imagem renderizada as deformações aparentam não estar tão salientes como na imagem reduzida. A tolerância utilizada foi igual a 1.0 em todos os testes realizados.

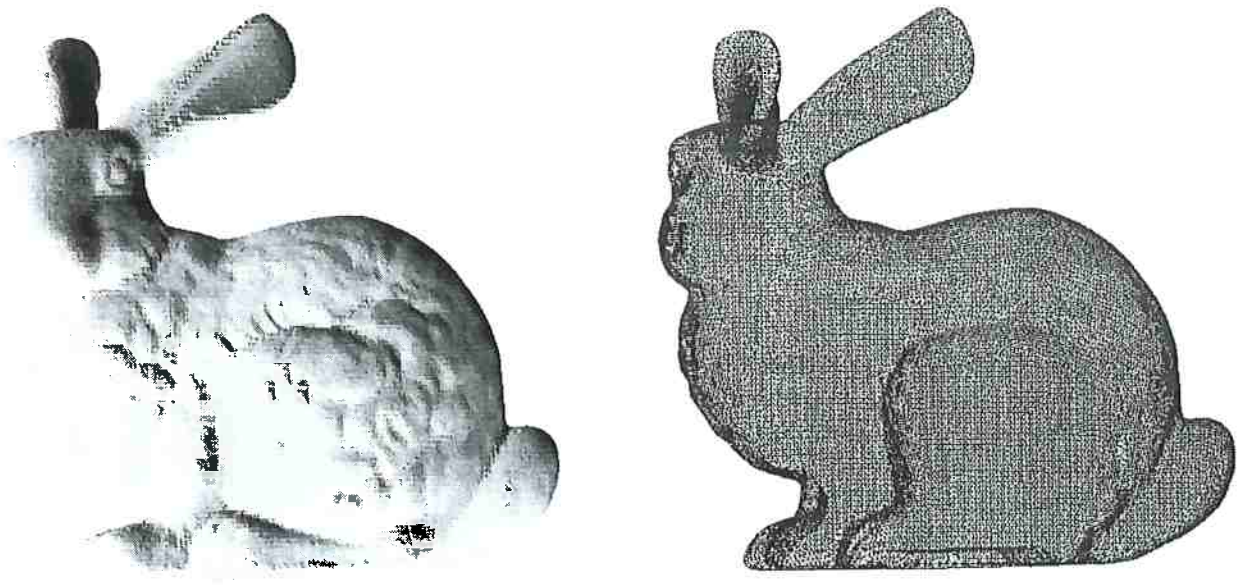


Figura 5.10. Coelho com 34.834 vértices, 69.456 faces e 104.288 arestas.

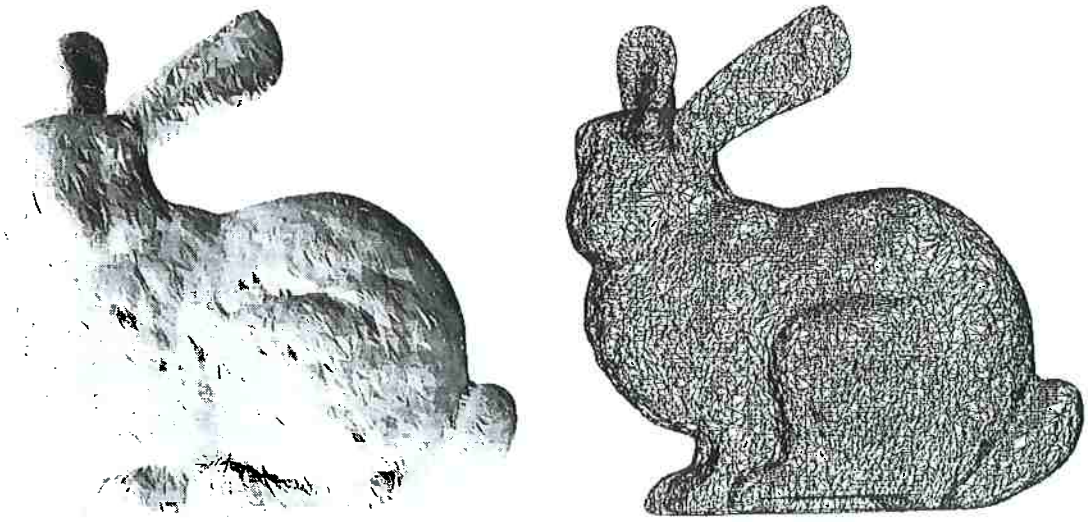


Figura 5.11. Aplicação do operador de incidência vértice-vértice ao coelho.

A Figura 5.10 exibe um coelho com 34.834 vértices, 69.456 faces e 104.288 arestas. Este coelho foi utilizado para testar os algoritmos propostos. O coelho possui dimensões muito reduzidas – sua altura não atinge 0,1. A aplicação do operador de incidência vértice-vértice resultou em um sólido com 17.836 vértices, 35.466 faces e 53.300 arestas (vide Figura 5.11) – 51% de simplificação. A Figura 5.12 exibe um detalhe das orelhas do coelho após a aplicação do operador de incidência vértice-vértice. Nesta imagem é possível observar que a malha original está bem distribuída, e a malha simplificada possui regiões com igual distribuição e regiões desorganizadas.

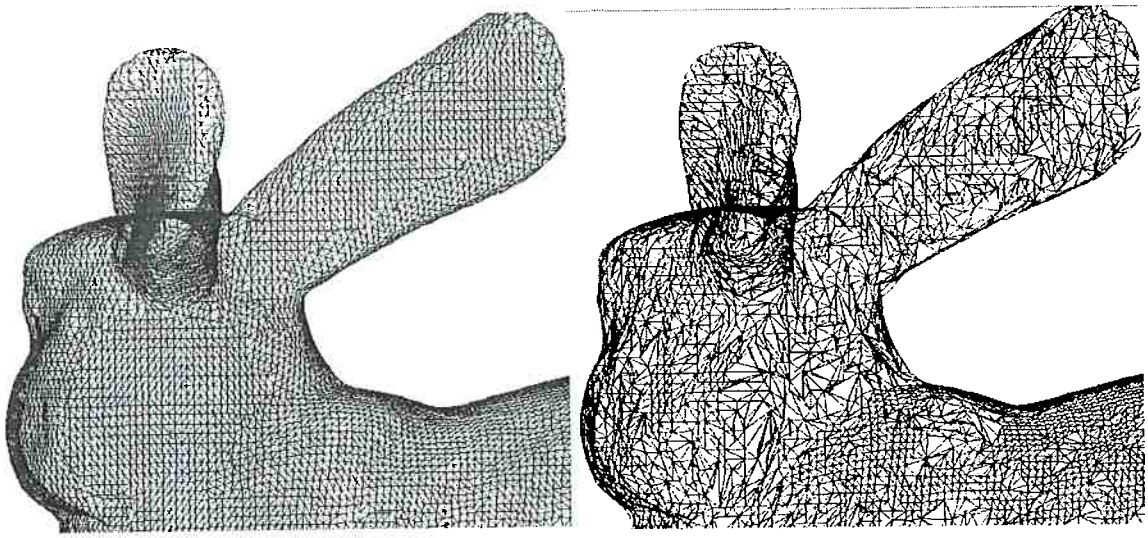


Figura 5.12. Detalhe da cabeça do coelho após a aplicação do operador de incidência vértice-vértice.

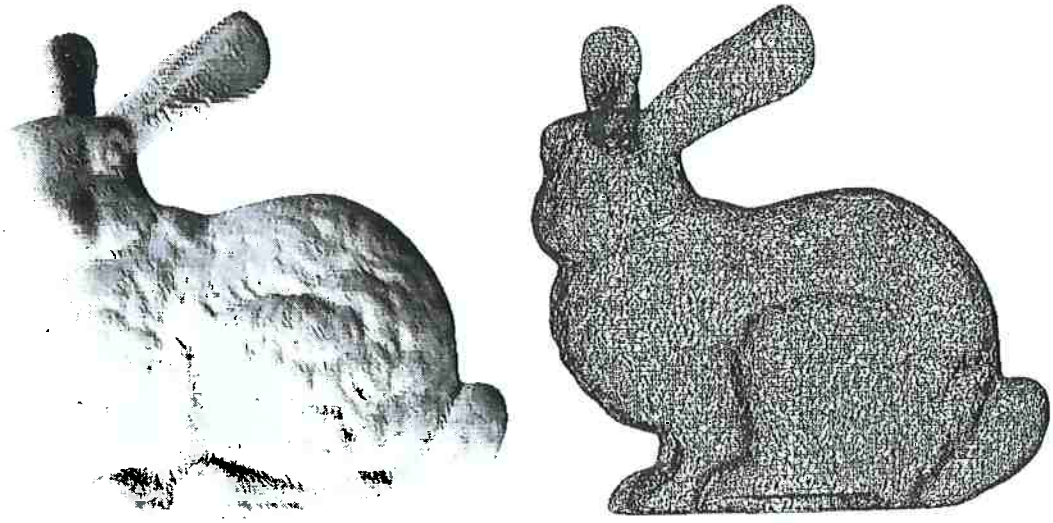


Figura 5.13. Aplicação do operador de incidência aresta-vértice ao coelho.

A aplicação do operador de incidência aresta-vértice resultou em um sólido com 25.549 vértices, 50.886 faces e 76.433 arestas (vide Figura 5.13) – 73% de simplificação. A aplicação do operador de incidência face-vértice resultou em um sólido com 17.150 vértices, 34.088 faces e 51.236 arestas (vide Figura 5.14) – 49% de simplificação. A aplicação de todos os operadores em conjunto resultou em um sólido com 7.987 vértices, 15.766 faces e 23.751 arestas (vide Figura 5.15) – 23% de simplificação.

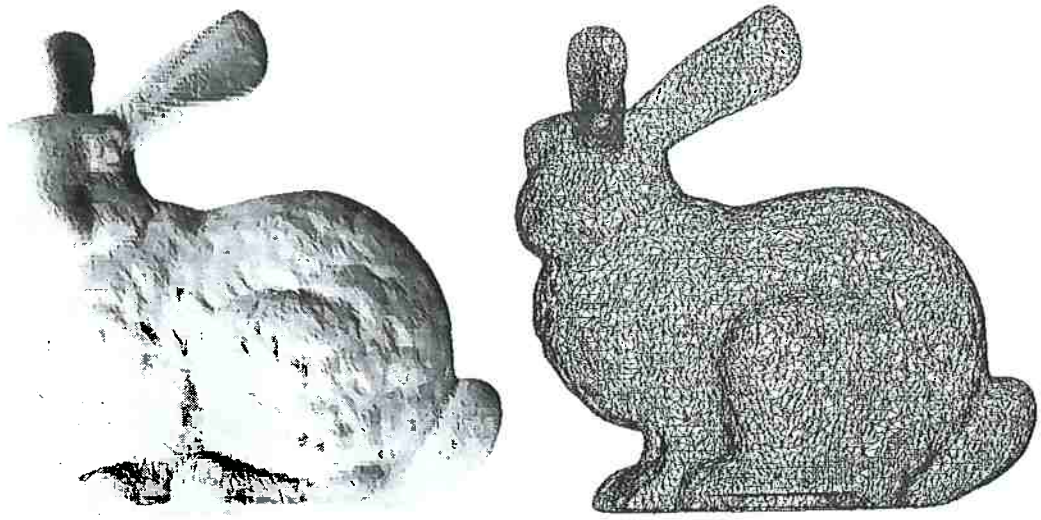


Figura 5.14. Aplicação do operador de incidência face-vértice ao coelho.

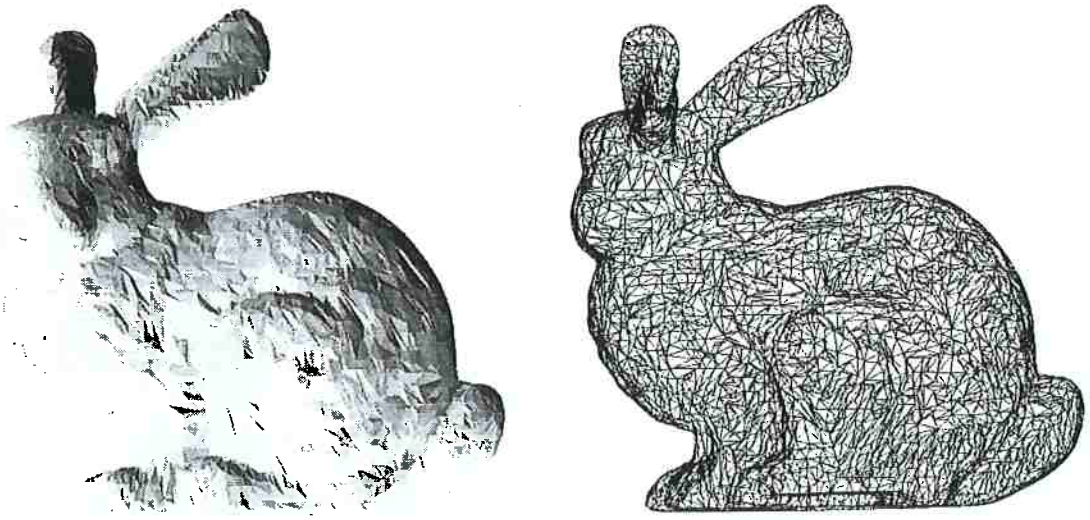


Figura 5.15. Aplicação dos três operadores de incidência ao coelho.

A tolerância utilizada por cada um dos três operadores de incidência foi diferente. O operador de incidência vértice-vértice utilizou a tolerância de 0,01. O operador de incidência aresta-vértice utilizou a tolerância de 0,001. O operador de incidência face-vértice utilizou a tolerância de 0,00000001. A determinação destes valores para cada um dos operadores foi uma atividade interativa.

No coelho, pelas imagens renderizadas, é possível observar pequenos arranhões nos modelos simplificados. A Figura 5.16 exibe um detalhe da cabeça do coelho após a aplicação do operador de incidência aresta-vértice. A Figura 5.17 exibe um detalhe da cabeça do coelho após a aplicação do operador de

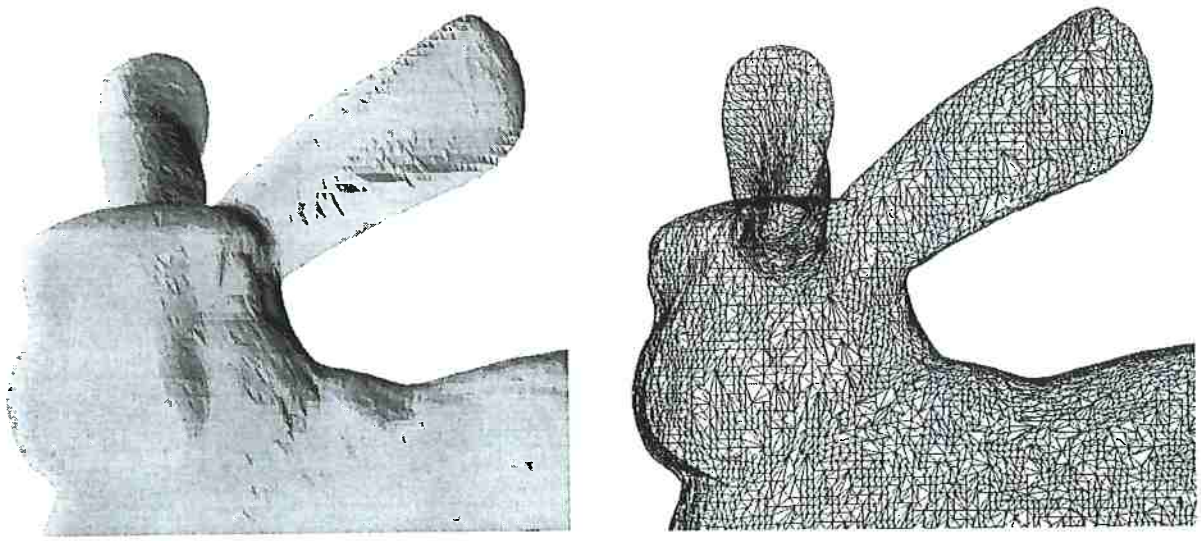


Figura 5.16. Detalhe da cabeça do coelho após a aplicação do operador de incidência aresta-vértice.

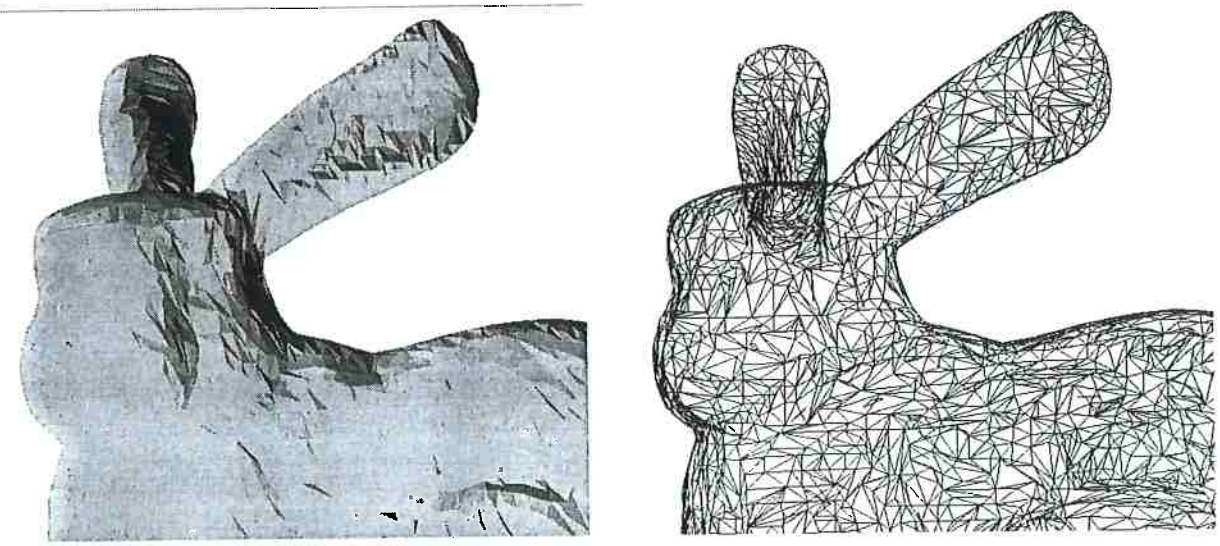


Figura 5.17. Detalhe da cabeça do coelho após a aplicação do operador de incidência face-vértice.

incidência face-vértice. Pelas imagens renderizadas as deformações aparentam não estar tão salientes como na imagem reduzida.

6. Conclusões

Foi desenvolvido um algoritmo para simplificação de malhas triangulares em um Modelador de Sólidos B-Rep. Foram propostos três operadores de simplificação baseados na incidência entre elementos primitivos: vértice, aresta e face. Estes operadores de simplificação foram implementados utilizando-se de aritmética intervalar. Todos os vértices originalmente fornecidos devem ser incidentes a pelo menos uma face do sólido simplificado resultante. Desta maneira, o critério de simplificação também foi implementado utilizando-se de aritmética intervalar.

Os operadores de simplificação determinam vértices candidatos a serem removidos. Ao se remover o vértice, foi definida uma nova triangulação para o polígono resultante. É necessário verificar se o critério de simplificação é satisfeito, em caso afirmativo o vértice candidato é removido. Em caso negativo ele é mantido.

Pela análise visual é possível confirmar que os sólidos mantiveram características geométricas. Regiões nos modelos simplificados com arranhões aparentes foram ampliadas, e os arranhões não aparentam ser tão salientes. Uma análise visual das imagens renderizadas, entretanto, pode ser afetada por efeitos de iluminação. O critério de simplificação funcionou como um controle global no processo de simplificação da malha. No exemplo do coelho é possível observar que o seu olho ficou prejudicado no resultado exibido na Figura 5.17. Entretanto, os resultados podem ser considerados satisfatórios.

As pesquisas devem avançar no sentido de definir um critério de simplificação que realize um controle local.

Referência

1 O. Aberth. *Precise Numerical Methods Using C++*. Academic Press, 1998.

2 S. L. Abrams; W. Cho; C. Y. Hu; T. Maekawa; N. M. Patrikalakis, E. C. Sherbrooke; X. Ye. "Efficient and Reliable Methods for Rounded-Interval Arithmetic". *Computer Aided Design*. Volume 30, number 8, pages 657-665, 1998.

3 B. G. Baumgart. "A polyhedron representation for computer vision". In *Proc. AFIPS Natl. Comput. Conf.* AFIPS Press, Alrington, Va., 1975. Volume 44, pages 589-596.

4 M. Eastlick e S. Maddock. "Triangle-mesh simplification using error polyhedra", *Proc. 19th Eurographics UK Chapter Annual Conference.* UCL, 3-5 April, 2001, pp.1-10.

5 M. Eastlick e S. Maddock. "Triangle-mesh simplification using error polyhedra", Department of Computer Science Technical Report CS-01-07. 2001.

6 T. Gerstner e R. Pajarola. "Topology Preserving and Controlled Topology Simplifying Multiresolution Isosurface Extraction". *VISUALIZATION '00: Proceedings of the 11th IEEE Visualization 2000 Conference (VIS 2000),* Salt Lake City, October 8-13, 2000.

7 C. Hoffmann. *Geometric and Solid Modeling*. Morgan-Kaufmann, San Mateo, CA, 1989.

8 M.. Hussain, Y. Okada and K. Niijima, "Efficient and feature-preserving triangular mesh decimation", *Journal of WSCG*, Vol. 12, No. 1, pp. 167-174, 2004.

9 Y. E. Kalay. "The Hybrid Edge: A Topological Data Structure forVertically Integrated Geometric Modelling". *Computer Aided Design*, pages 130–40, April 1989.

10 B. W. Kernighan; D. M. Ritchie *The C Programming Language*. Second Edition, Prentice Hall, 1988.

11 G. Li e B. Watson. "Semiautomatic simplification", *3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics.* Pages 43-48, ACM PressNew York, NY, USA, 2001.

12 M. Mäntylä. *An Introduction to Solid Modeling*. Computer Science Press, Rockville, MD, 1988.

13 M. Roy, S. Foufou, e F. Truchetet. "Generic Attribute Deviation Metric for Assessing Mesh Simplification Algorithm Quality", In Proceedings of *IEEE*

International Conference on Image Processing. Pages 817-820, September 2002, Rochester, USA.

14 M. Roy, F. Nicolier, S. Foufou, F. Truchetet, A. Koschan, and M. Abidi. "Assessment of Mesh Simplification Algorithm Quality", In Proceedings of *SPIE Electronic Imaging.* Vol. 4661, Pages. 128-137, January 2002, San Jose, USA.

15 W. J. Schroeder, J. A. Zarge, e W. E. Lorensen. "Decimation of Triangle Meshes", *Computer Graphics* (SIGGRAPH '92 Proceedings). Vol. 26, No. 2, July 1992, pages 65-70

16 M. Shimada. "Aritmética Intervalar Aplicada em um Modelador de Sólidos B-Rep". Dissertação de Mestrado, PMR-EPUSP, 2002.

17 G. Turk. "Re-tiling polygonal surfaces". In Edwin E. Catmull, editor, Computer Graphics (SIGGRAPH'92 Proceedings). Volume 26, Pages 55-64, July 1992.

18 P. Véron, J.-C. Léon. Static "Polyhedron Simplification using Error Measurements". *Computer-Aided Design.* Volume 29, Number 4, pages 287-298, 1997.

19 P. Véron, J.-C. Léon. "Shape preserving polyhedral simplification with bounded error", *Computers and Graphics.* Volume 22, number 5, pages 565-585, 1998.

20 K. Weiler. "Edge-based data structures for solid modeling in a curved surface environment". *IEEE Comput. Graph. Appl.* Volume 5, Number 1, pages 21-40, 1985.

21 B. Wuensche. "A Survey and Evaluation of Mesh Reduction Techniques", *Proceedings of IVCNZ '98*, Auckland University, Auckland, November 1998, pages 393-398.