Eduardo Sant'Ana da Silva

COMPACTAÇÃO EFICIENTE DA GEOMETRIA DE MALHAS TRIANGULARES

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.

Orientador: Prof. Hélio Pedrini

CURITIBA

2005

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

A disponibilidade e a complexidade de modelos de dados têm crescido rapidamente nos últimos anos devido às sofisticadas capacidades de aquisição e distribuição dos dados e devido à necessidade de resoluções cada vez mais altas.

Exemplos de áreas de aplicações envolvendo grandes volumes de dados em domínios de conhecimento complexos incluem sensoriamento remoto, exploração planetária, visão computacional, área de prototipagem e robótica. Informações redundantes ou procedimentos triviais de análise podem demandar recursos computacionais muito acima das capacidades atualmente existentes. Desse modo, técnicas de redução ou compressão de dados devem ser utilizadas para permitir que grandes volumes de dados sejam analisados e visualizados em tempo real.

Superfícies poligonais são, provavelmente, a representação mais freqüentemente utilizada para modelos geométricos, já que são flexíveis e suportadas pela maioria dos pacotes de modelagem e visualização. Uma superfície poligonal é uma superfície linear por partes definida por um conjunto de polígonos, tipicamente um conjunto de triângulos.

Uma técnica para codificação de malhas triangulares enumera os elementos da malha em uma seqüência de triângulos adjacentes para evitar repetição das coordenadas dos vértices de arestas comuns. Esta técnica é conhecida como *strips triangulares*.

Este trabalho descreve as principais técnicas utilizadas para a compactação de malhas triangulares e uma nova abordagem sobre o assunto. Um número baixo de *strips* é obtido pela manipulação de uma árvore geradora, reduzindo o número de caminhos sobre tal árvore. Um grande conjunto de modelos triangulares é utilizado para demonstrar a eficácia do método.

Glossário

API	*Application Programming Interface*	interface de programação de aplicações
BFS	*Breadth-First Search*	busca em largura
CLOD	*Custom Level Of Detail*	nível personalizado de detalhe
CSG	*Construtive Solid Geometry*	geometria sólida construtiva
DFS	*Depth-First Search*	busca em profundidade
IGL	*The Graphics Library for the i860*	biblioteca gráfica para o i860
Iris-GL	*Iris Graphics Library*	biblioteca gráfica Iris
LOD	*Level Of Detail*	nível de detalhe
MCAD	*Modeling Computer Aided Design*	modelagem auxiliada por computador
MST	*Minimum Spanning Tree*	árvore geradora mínima
NURBS	*Non Uniform Rational B-splines*	B-splines racionais não uniformes
OpenGL	*Open Graphics Library*	biblioteca gráfica OpenGL
PEX	*PHIGS Extension to X*	extensão PHIGS para X
PHIGS	*Programmer's Hierarchical Interactive Graphics System*	sistema gráfico PHIGS
VDPM	*View Dependent Progressive Mesh*	malha progressiva dependente da vista
TIN	*Triangulated Irregular Networks*	Redes Irregulares Trianguladas
XGL	*(Xlib — Sun's Foundation) Graphics Library*	biblioteca gráfica XGL

Capítulo 1

Introdução

1.1 Descrição do Problema

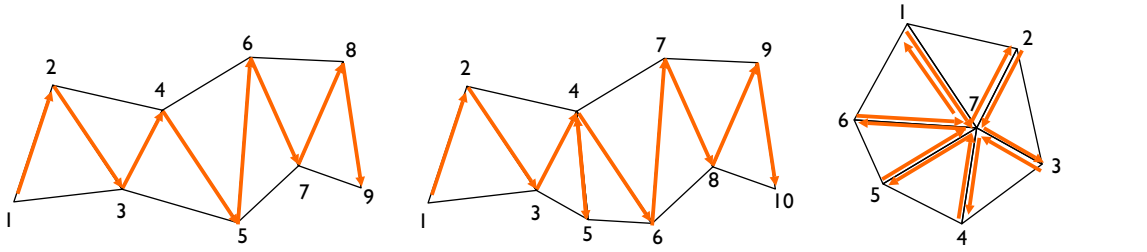
A representação, transmissão e visualização eficientes de um modelo de superfície poligonal é um problema clássico na área de computação gráfica. Superfícies poligonais são, provavelmente, as representações mais amplamente utilizadas para modelos geométricos, devido ao fato de serem flexíveis e suportadas pela maioria dos pacotes de modelagem e visualização. Uma superfície poligonal é uma superfície linear por partes definida por um conjunto de polígonos, sendo tipicamente um conjunto de triângulos.

Um dos esquemas de codificação comuns é baseado em *strips* triangulares que enumeram os elementos da malha em uma sucessão de triângulos adjacentes para evitar a repetição de coordenadas de vértices pertencentes a arestas compartilhadas.

Uma *strip triangular* é uma seqüência ordenada de vértices $(v_1, \ldots, v_m)$, admitindo repetições, que codifica um conjunto de $m-2$ triângulos. Portanto, uma *strip* triangular é uma seqüência na qual triângulos que compartilham uma aresta são denominados adjacentes, caso contrário são ditos disjuntos. Uma *strip seqüencial* codifica um conjunto de triângulos $(v_i, v_{i+1}, v_{i+2}), 1 \leq i \leq m-2$; uma *strip em leque* codifica o conjunto (v_1, v_{i+1}, v_{i+2}), todos os quais contêm v_1. Uma *strip generalizada* combina os dois tipos de *strips* definidos anteriormente. Para representar uma superfície poligonal, composta por uma malha triangular, podem ser usados qualquer um dos tipos de *strips* bem como a combinação entre eles.

Em uma codificação ideal, formada por uma única *strip*, apenas $n+2$ vértices teriam que ser representados para uma triangulação com n triângulos. Isto melhora o desempenho da transmissão, visto que o envio de vértices é o um fator limitante deste problema [6].

Uma operação *swap* pode ser definida como a repetição do vértice quando duas voltas sucessivas têm a mesma orientação, como usado no pacote OpenGL [27]. As *strips* triangulares são suportadas por várias bibliotecas gráficas incluindo IGL [5], PHIGS [22], Inventor [39], DirectX [25] e OpenGL [27]. Na figura 1.1, pode-se observar os três tipos de *strips* descritas anteriormente, podendo-se notar que na *strip* generalizada (figura 1.1(b)) foi necessária a repetição do vértice 4 para que a *strip* fosse definida. Nesse caso, utilizou-se o *swap* (4,5,*swap*(4)).



(a) *Strip* seqüencial (1,2,3,4,5,6,7,8,9)

(b) *Strip* generalizada (1,2,3,4,5,4,6,7,8,9,10)

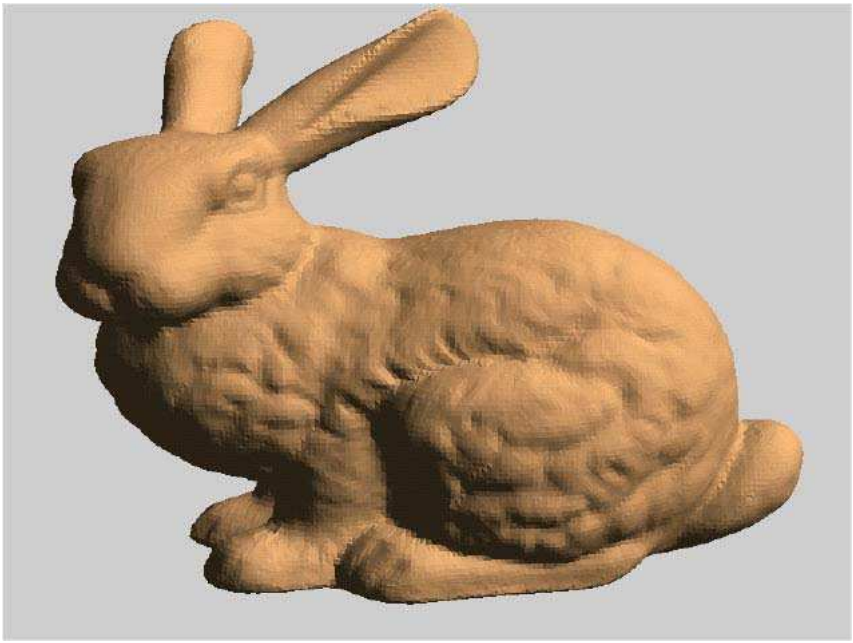
(c) *Strip* em leque (1,7,2,7,3,7,4,7,5,7,6,7,1)

Figura 1.1: Tipos de *strips*: (a) *strip* seqüencial; (b) *strip* generalizada; (c) *strip* em leque.

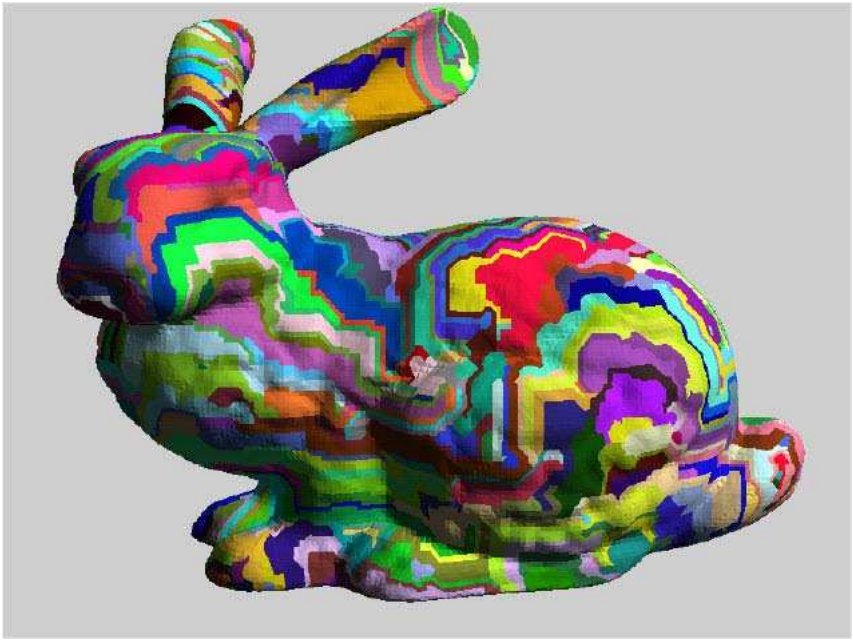
Atualmente, as placas gráficas 3D apresentam sérias limitações quanto à largura de banda entre a unidade de processamento gráfico (GPU, do inglês *graphical processing unit*) e a unidade central de processamento (CPU, *central processing unit*). O método baseado em *strips* triangulares permite uma *cache* de 2 vértices. Quando o vértice v_{i+2} é transmitido, o processador gráfico determina o triângulo com a *cache* de vértices v_i e v_{i+1}.

Uma *strip* seqüencial de $m-2$ triângulos é enviada ao processador gráfico como uma seqüência de m vértices, sendo três vértices para o primeiro triângulo e um para cada triângulo adicional. Um algoritmo eficiente para geração de *strips* deve calcular um pequeno conjunto de *strips* triangulares que cubra toda a malha. Poucas e longas *strips* permitem que menos vértices sejam enviados ao processador gráfico, reduzindo o tempo para a visulização dos modelos de dados.

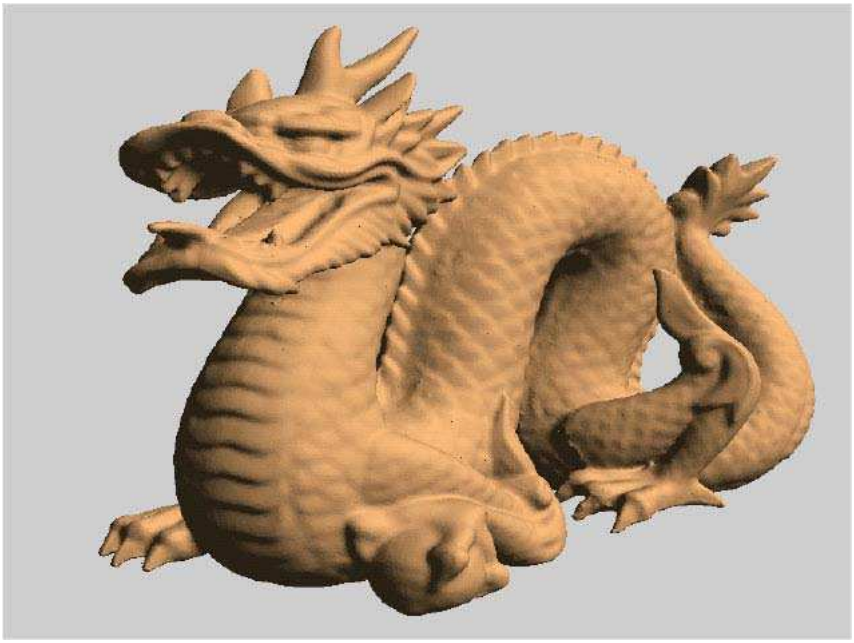
As figuras 1.2(a) e 1.2(c) apresentam alguns exemplos de modelos 3D e suas correspondentes versões com o uso de *strips* triangulares nas figuras 1.2(b) e 1.2(d). Cada *strip* é representada por uma faixa colorida.



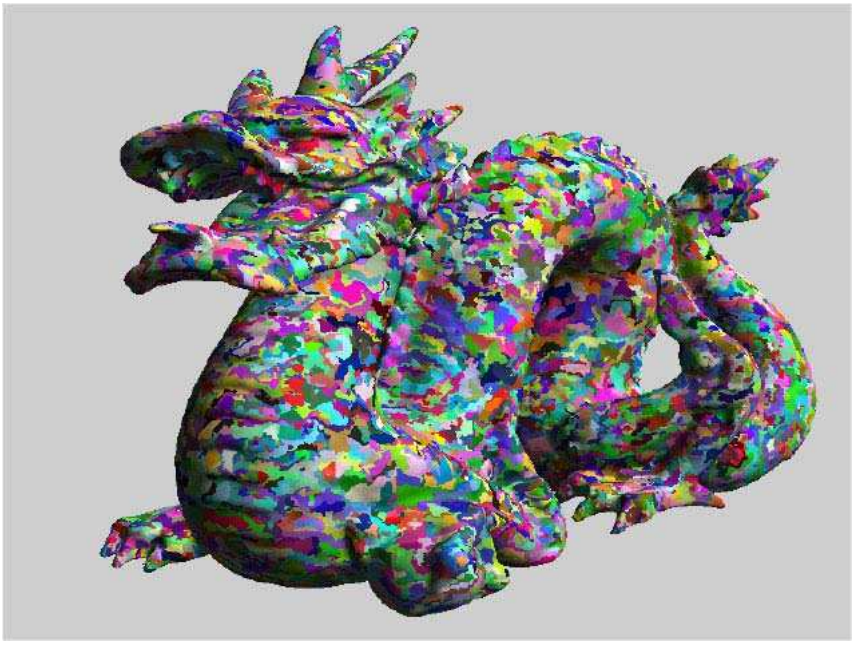
(a)



(b)



(c)



(d)

Figura 1.2: Modelos compostos por malhas triangulares e suas respectivas *strips*.

1.2 Objetivos e Contribuições

O principal objetivo deste trabalho é apresentar um método eficiente para a geração de *strips* a partir de malhas triangulares, tal que possa ser um método de otimização genérico que produza uma melhora significativa quando aplicado a um conjunto de *strips* gerado por qualquer outro método de compactação.

O método desenvolvido é composto por três etapas. A primeira etapa consiste na redução do número de *strips* pela eliminação, quando possível, de nodos que apresentem grau 3. A segunda, executada no passo de extração das *strips*, consiste em separar nodos vizinhos ao nodo de grau 3 analisado, excluindo arestas desnecessárias[1] do grafo. A terceira etapa consiste na eliminação da restrição em se formar *strips* circulares. As definições de superfície poligonal, grafo de adjacência e malhas triangulares encontram-se no capítulo 2.

Apesar da implementação conter limitações em relação ao tempo e à memória consumida para sua execução, o método pode ser modificado de modo a permitir compactação em tempo real. Diversos modelos foram utilizados para avaliar a metodologia proposta em relação aos trabalhos existentes na literatura.

1.3 Organização da Dissertação

Esta dissertação está organizada da seguinte forma:

- O capítulo 2 apresenta uma revisão bibliográfica com os principais trabalhos, de relevância para esta dissertação, realizados na área de compressão de malhas triangulares.

- O capítulo 3 apresenta o método desenvolvido para gerar as *strips* triangulares. Neste capítulo também é descrito o algoritmo usado para gerar o conjunto inicial de *strips* a ser otimizado.

- O capítulo 4 mostra os resultados obtidos pela aplicação do método desenvolvido a um conjunto de modelos poligonais. O comportamento de cada etapa do algoritmo

[1]Entende-se por arestas desnecessárias aquelas as quais podem ser removidas sem que os nodos ligados por elas percam sua referência no grafo, ou seja, deve haver outro caminho de acesso a estes nodos por outra aresta do grafo.

desenvolvido é analisado e discutido.

- O capítulo 5 apresenta as conclusões obtidas a partir deste trabalho, bem como sugestões para trabalhos futuros.

Capítulo 2

Revisão Bibliográfica

2.1 Definições Preliminares

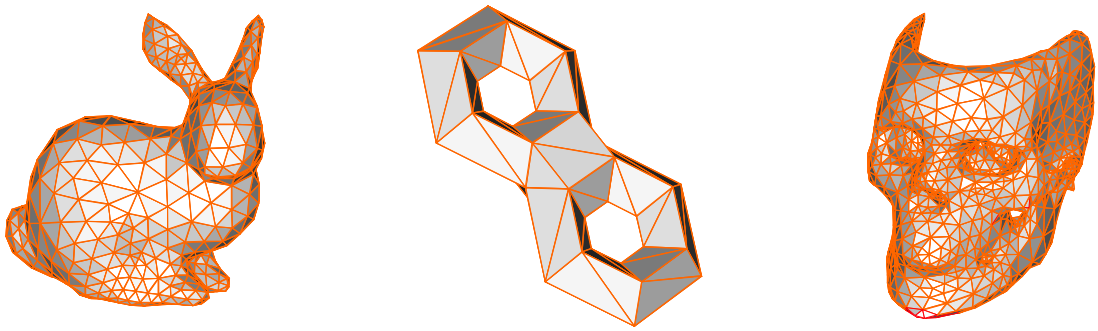
Este capítulo apresenta e discute as principais abordagens encontradas na literatura para compactação de malhas triangulares. Antes, porém, faz-se necessária a definição de alguns termos que serão utilizados ao longo deste trabalho.

2.1.1 Superfície poligonal

Uma superfície poligonal de um modelo é um conjunto de faces poligonais que representam os limites de um poliedro em um espaço tridimensional. Cada face poligonal é representada por uma lista circular de vértices que descrevem a parte externa da face, seguida por uma possível lista vazia de buracos.

Uma face poligonal sem buracos é dita simplesmente conectada, enquanto que outra, com um ou mais buracos, é dita multiplamente conectada. Esses conceitos são ilustrados na figura 2.1.

Uma aresta múltipla pode ser definida como uma aresta contendo exatamente dois triângulos incidentes, chamados de adjacentes. Por outro lado, uma aresta não múltipla contém mais de dois triângulos adjacentes.



(a) (b) (c)

Figura 2.1: Exemplos de malhas (a) simplesmente conectadas e (b,c) multiplamente conectadas.

2.1.2 Caminho Hamiltoniano

Um grafo de adjacência G, de um modelo triangulado possuindo somente arestas múltiplas ou de borda, é um grafo cujos nodos são as faces do modelo, no caso os triângulos, e as arestas deste grafo são as ligações entre estas faces. Por esta definição, tem-se que o grau máximo de um nodo do grafo é três.

Um caminho Hamiltoniano, em uma malha triangular, é uma seqüência de faces adjacentes, duas a duas, onde cada face é visitada apenas uma vez.

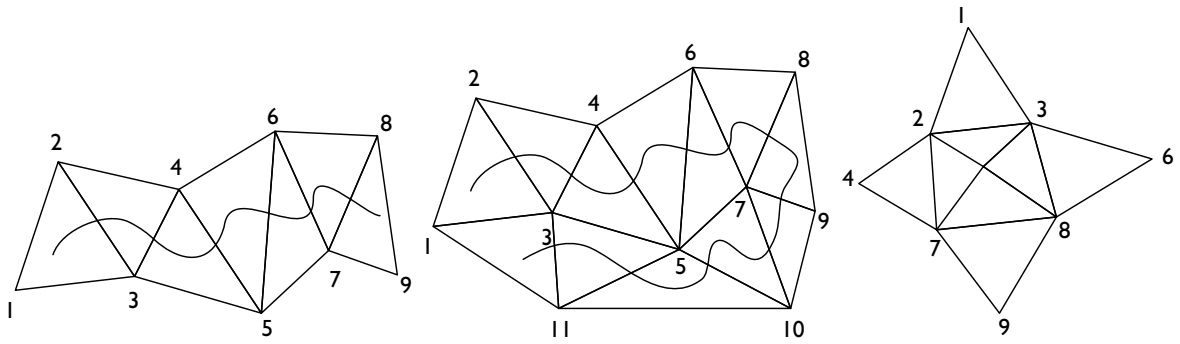
Adotando-se a definição de Arking *et al.* [2], uma triangulação S é dita Hamiltoniana se o grafo G possui um caminho Hamiltoniano, ou seja, se existe um caminho em G que visita cada nodo uma única vez.

2.1.3 Tipos de triangulação

Uma triangulação composta por uma seqüência de $n+2$ vértices utilizada para representar n triângulos é dita seqüencial.

Uma triangulação S é denominada em leque se o grafo G possui um caminho Hamiltoniano tal que todas as arestas têm um vértice incidente no seu caminho.

A figura 2.2 mostra os tipos de triangulação seqüencial, Hamiltoniana e não Hamiltoniana.



(a) seqüencial e Hamiltoniana (b) Hamiltoniana (c) não Hamiltoniana

Figura 2.2: Tipos de triangulação.

2.2 Trabalhos Relacionados

Várias abordagens têm sido propostas para o problema de compactação de malhas triangulares, algumas se assemelham no aspecto do componente a ser compactado e outras no modo de compactar, possibilitando assim uma categorização de tais abordagens.

Os métodos recentes de compressão 3D podem ser divididos em três categorias, sendo discutidos mais detalhadamente nas próximas seções:

- simplificação poliedral
- compressão de posições e propriedades
- codificação das informações de conectividade

2.2.1 Simplificação poliedral

Técnicas de simplificação poliedral reduzem o número de vértices na malha pela alteração dos modelos de conectividade ajustando as posições dos vértices restantes, quando possível, para minimizar o erro produzido pela simplificação com relação ao modelo original. Essas técnicas se concentram na geração de múltiplos níveis de detalhes (LOD - *level of detail*) para aceleração gráfica [14] ou redução de dados por amostragem das malhas [21].

Embora essas técnicas possam ser consideradas como compressão sem perda, elas não são adequadas para aplicações que requerem acesso exato à conectividade do modelo.

As técnicas de simplificação são, de fato, ortogonais às técnicas de compressão descritas em [33], porque a compressão geométrica pode ser aplicada a cada nível de detalhe.

2.2.2 Compressão de posições e propriedades

Métodos de compressão sem perda são usados para reduzir o espaço de armazenamento necessário para os dados geométricos associados com as localizações dos vértices e, possivelmente, para dados como normais, cores e coordenadas de texturas.

A aplicação de algoritmos de compressão de dados, de propósito geral, ao conjunto de dados geométricos leva a soluções sub-ótimas em relação ao espaço utilizado. Taubin [33] usa uma organização espacial dos vértices em uma árvore geradora e preditores geométricos para substitituir posições ou propriedades de coordenadas por pequenos termos corretivos, os quais podem ser codificados, sem perda de informação, com poucos bits. Modelos criados pelo processo de quantização em malhas compostas por um grande número de pequenos triângulos podem ser reduzidos usando técnicas de suavização como em [32] e [34].

Em [33], Taubin e Rossignac introduziram uma nova representação para modelos triangulados complexos e simples, embora eficiente, juntamente com algoritmos de compressão e descompressão para tal representação. Em sua proposta, as posições dos vértices são quantizadas com a precisão desejada. Uma árvore geradora de vértices é usada para predizer a posição de cada vértice de 2, 3 ou 4 de seus ancestrais na árvore. De forma similar, propriedades como cores, normais e coordenadas de textura são comprimidas. Ainda em [33], as informações sobre conectividade são codificadas, em média, com dois bits a menos por triângulo, sem perda de informação. Sua abordagem aperfeiçoa os resultados obtidos por Deering [9] pela exploração da coerência geométrica de vários ancestrais na árvore geradora de vértices, preservando as informações de conectividade, permitindo a repetição de vértices e, ainda, usando cerca de três vezes menos bits para a conectividade. Entretanto, uma desvantagem é que a descompressão requer acesso aleatório a todos os vértices. Sendo assim, o uso deste método tem que ser modificado para a renderização em hardware com memória limitada.

Resumindo o trabalho de Taubin e Rossignac [33], o formato comprimido e os algoritmos de compressão e descompressão introduzidos permitem:

- codificação sem perda e altas taxas de compressão para as informações de conectividade (em média, dois bits a menos por triângulo);
- melhor organização para compressão de coordenadas de vértices;
- métodos eficientes para compressão, próximos da solução ótima para modelos poliédricos de topologia arbitrária;
- técnicas de compressão e descompressão que produzem *strips* triangulares muito longas que são compatíveis com as mais novas gerações de adaptadores gráficos.

A figura 2.3 ilustra uma amostra do trabalho de Taubin e Rossignac.

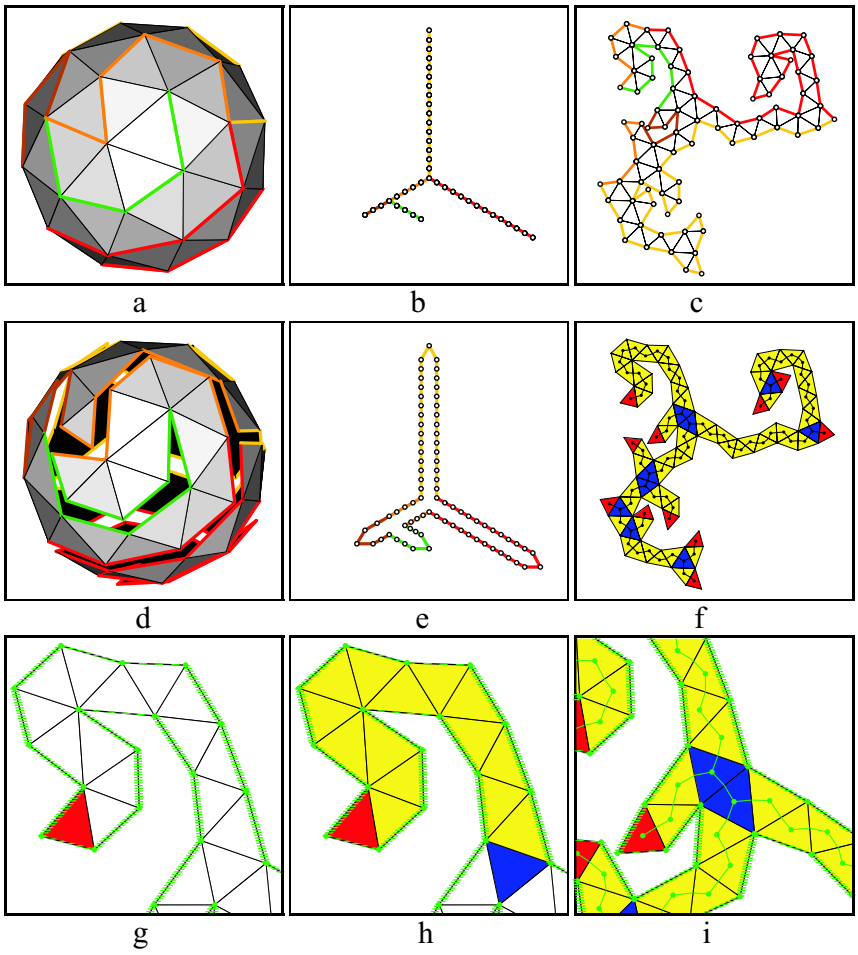


Figura 2.3: Strips de Taubin e Rossignac [33].

A árvore geradora de vértices é mostrada nas figuras (a) e (b). O grafo dual do polígono é a árvore geradora de triângulos em (f). Triângulos-folha são representados em vermelho, triângulos regulares representados em amarelo e os triângulos ramificadores

representados em azul. A árvore geradora de triângulos possui um triângulo raiz (figura (g)).

2.2.3 Codificação das informações de conectividade

A seguir, são descritos seis métodos para codificação das informações de conectividade.

Método SGI

O primeiro algoritmo para geração de *strips* foi desenvolvido por Akeley *et al.* [1], conhecido como SGI ou *tomesh*, o qual converte uma malha triangulada em *strips* triangulares. A heurística adotada é o algoritmo guloso simples baseado em um critério local.

O algoritmo SGI constrói *strips* adicionando os triângulos de forma incremental. Os triângulos adicionados são aqueles de menor valência, ou seja, a *strip* é iniciada com o triângulo com o menor número de vizinhos. A heurística gulosa é usada para adicionar à *strip* triângulos com a mesma característica do inicial, ou seja, o menor número de vizinhos. Quando houver mais triângulos com o mesmo número de vizinhos, o algoritmo se antecipa e, se não existir triângulo vizinho, uma nova *strip* é criada.

O algoritmo pára quando todos os triângulos forem adicionados às *strips*. Após começar em um triângulo arbitrário, o algoritmo estende suas *strips* em ambas as direções, sendo que, desta maneira, as *strips* são as mais longas possíveis.

Como a heurística para escolha do próximo triângulo pode ser facilmente modificada, várias versões deste algoritmo foram criadas. O algoritmo SGI procura criar *strips* que tendem a minimizar o número de triângulos isolados.

A complexidade do algoritmo é $O(n + s \cdot n)$, onde n é o número de triângulos e s o número *strips* triangulares. Uma versão com uma ordem de complexidade de $O(n)$ é obtida usando-se algumas estruturas de dados adicionais, como tabelas *hash* para armazenar as informações de adjacência.

Embora este algoritmo seja muito simples, ele apresenta uma boa geração de *strips* em um curto espaço de tempo. Como este algoritmo foi projetado para o pacote *Iris-GL*, com o qual se usa 1-*bit* de sinalização para o *swap*, não há impedimento em se produzir *swaps*.

O algoritmo usa uma função de heurística de abrangência local e, por este motivo,

produz um grande número de *strips* curtas. Uma abordagem mais global poderia melhorar a construção das *strips* triangulares.

Método SStrip

Um algoritmo baseado no método SGI foi desenvolvido por Silva *et al.* [7] e van Kaick *et al.* [36]. O método proposto procura minimizar o número de vértices a serem enviados ao *pipeline* gráfico, sendo que, para isto, duas heurísticas foram consideradas:

- a primeira tem como objetivo minimizar o número de *strips*, gerando a saída para um *hardware* e uma biblioteca gráfica que suporta *swaps*;
- a segunda heurística minimiza o número de vértices para modelos que simulam *swap* reenviando um vértice.

O algoritmo usa uma estratégia local baseada em uma construção simultânea de *strips*. Nesta abordagem, onde são mantidas s *strips* sob construção simultânea, um triângulo é adicionado a cada passo.

Um triângulo é dito *livre* se ele não pertencer a qualquer *strip*, sendo que o *grau* do triângulo é o número de triângulos adjacentes a ele.

O algoritmo escolhe s triângulos como início das s *strips*, seguindo o mesmo critério do método SGI, ou seja, com o menor número de vizinhos. Para evitar uma imediata concatenação de *strips*, uma nova restrição é adicionada, sendo que o começo de uma nova *strip* não pode ser adjacente às extremidades de uma *strip* existente.

A escolha do próximo triângulo a ser adicionado à *strip* é feita entre todos os triângulos candidatos, ou seja, todos os triângulos vizinhos a ambas as extremidades de todas as *strips* triangulares, conforme a ordem:

- se o triângulo vizinho tem grau 0, ele é adicionado imediatamente para evitar uma *strip* com um único triângulo.
- se não existe nenhum triângulo vizinho de grau 0, o vizinho com grau 1 é escolhido. Em caso de empate, um teste *look-ahead* é feito. Se o triângulo adjacente tem grau 1, ele é inserido automaticamente, caso contrário, o triângulo que não produz um *swap* (heurística de minimização de vértices) ou o triângulo com o vizinho de menor grau (heurística de minimização de *strips*) é escolhido.

- se todos os vizinhos têm grau 2, o triângulo escolhido é aquele que não produz o *swap*.

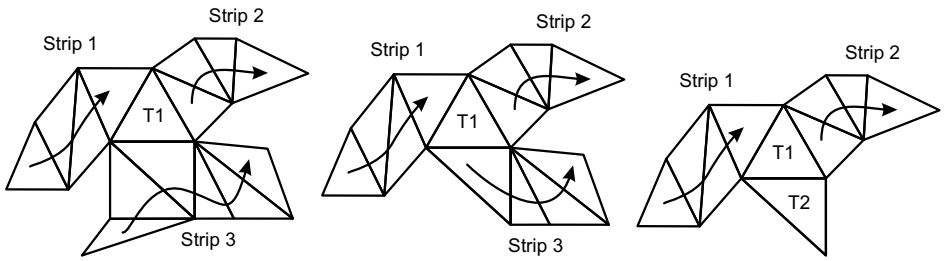
Em alguns casos, o acréscimo de um triângulo pode ocasionar que as extremidades das duas *strips* tornem-se adjacentes. Estas duas *strips* podem ser concatenadas e uma nova *strip* criada.

O acréscimo de um triângulo, em alguns casos, pode tornar adjacentes as extremidades de duas *strips* possibilitando uma concatenação que é feita da seguinte maneira:

- quando um triângulo T_1 de grau 0 é adjacente a duas extremidades de *strip*, ambas as *strips* são concatenadas. Na figura 2.4(a), o triângulo T_1 é adjacente a *strip* 1 e 2, e estas *strips* são concatenadas.

- quando um triângulo T_1 de grau 0 é adjacente a três extremidades de *strip*, uma concatenação que não produza um *swap* é escolhida. Na figura 2.4(b), as *strips* 1 e 2 são concatenadas.

- quando um triângulo T_1 de grau 1 é adjacente a uma *strip* e ao triângulo T_2, se o triângulo T_2 tem grau 1, ele é conectado a T_1 para evitar uma *strip* com um único triângulo; caso contrário as duas *strips* são concatenadas. Na figura 2.4(c), ambos os triângulos T_1 e T_2 são adicionados à *strip* 1.



(a) (b) (c)

Figura 2.4: Regras de contatenação do algoritmo *SStrip*.

Um pseudocódigo do método SStrip é apresentado na figura 2.5 e um exemplo de construção de *strips* simultâneas é apresentado na figura 2.6.

Este algoritmo é projetado para malhas estáticas e completamente trianguladas. Ele produz uma geração de *strips* com um número de *strips* e vértices menor que o método

```
iniciar s novas strips
enquanto existir algum triângulo na malha faça
   enquanto inserir o candidato de grau 0 faça
      tentar concatenar strips
      criar novas strips
   fim do enquanto
   se inserir o candidato de grau 1 então
      tentar concatenar strips
      criar novas strips
   senão inserir o candidato de grau 2 então
      tentar concatenar strips
      criar novas strips
   fim do se
fim do enquanto
```

Figura 2.5: Pseudocódigo para o método SStrip.

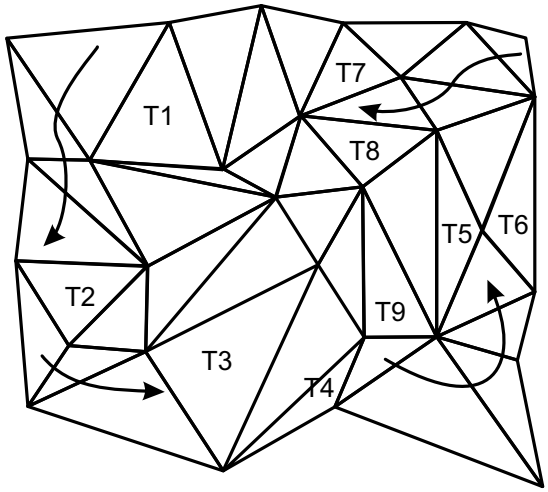


Figura 2.6: Exemplo de construção simultânea de *strips* no método *SStrip*.

SGI. Segundo Vanecek [37], esta melhora é devida, parcialmente, à múltipla construção de *strips*. De acordo com a medição feita por Vanececk, o número ideal de construção de *strips* simultâneas está entre 2 ou 4, entretanto, as diferenças não são significativas.

Método STRIPE

Para melhorar a qualidade da geração de *strips*, Evans *et al.* [13] desenvolveram um algoritmo que usa a análise global da estrutura poligonal dos modelos. Como tais modelos, freqüentemente, não são totalmente triangulados, é necessário que a malha seja triangulada antes de renderizar.

Em tais tipos de malhas existem, normalmente, muitas faces formadas por quadriláteros arranjadas em grandes regiões conectadas. A heurística concentra-se em encontrar grandes regiões retangulares consistindo somente em quadriláteros para formar caminhos. Estes caminhos são triangulados ao longo de cada linha ou coluna e, então, a *strip* é gerada.

Para calcular o número de polígonos em cada caminho é necessário examinar os quadriláteros em ambas as direções (leste-oeste e norte-sul). Também é necessário garantir que todos esses quadriláteros sejam adjacentes. Para evitar a geração de muitos caminhos, existe uma regra onde o tamanho mínimo do caminho é estipulado.

Duas abordagens diferentes foram utilizadas para a geração dos caminhos :

- *strips* de linhas/colunas: particiona os caminhos em *strips* seqüenciais ao longo de linhas/colunas dependendo do tamanho do caminho e da direção a ser tomada, sendo que, pela geração de uma *strip* ao longo de cada linha, o número de *swap* é minimizado;

- *strips* de caminho completo: converte cada caminho em uma *strip* generalizada ao custo de 3 *swaps* por rodada, sendo que, deste modo, cada *strip* é então estendida em ambas as extremidades até os quadriláteros vizinhos. Esta abordagem minimiza o número de *strips*.

Após a heurística global, um algoritmo baseado no SGI é usado para gerar *strips* a partir dos polígonos restantes.

Três abordagens diferentes são sugeridas para a triangulação de polígonos:

- a triangulação estática, onde todas as faces são trianguladas em um estágio pré-processado, usando voltas esquerda e direita alternadas. Esta triangulação é mais complexa do que a triangulação convencional em leque, entretanto, produz triângulos que podem ser definidos em uma *strip* seqüencial;

- a triangulação dinâmica de face total triangula uma face quando uma *strip* passa por ela;

- a triangulação dinâmica de face parcial cuja abordagem permite triangular somente a parte do polígono pela qual a *strip* passa, sendo que a parte restante do polígono pode ser triangulada posteriormente, o que permite uma melhor criação da *strip* triangular.

Considerando as três abordagens, a triangulação dinâmica de face parcial é a que obtém melhores resultados.

O *Stripe* gera *strips* triangulares eficientes, entretanto, requer mais tempo do que o simples algoritmo guloso usado pelo SGI. O algoritmo foi projetado para malhas estáticas, não completamente trianguladas com polígonos convexos. Para tais malhas, o algoritmo produz *strips* triangulares de alta qualidade em tempo linear. De acordo com testes realizados pelos autores [13], a melhor geração de *strips* é obtida pelas *strips* globais linha/coluna, com regra de tamanho mínimo de *strips* em 5, usando a triangulação de face total.

Para malhas completamente trianguladas, entretanto, ele produz um número alto de *strips* com um número de vértices um pouco menor que o método SGI. O tempo necessário para a execução do *Stripe* é superior ao do método SGI.

No artigo de Evans *et al.* [13] foi explorado um total de vinte algoritmos diferentes, locais e globais, avaliando-se o custo no modelo predominante, o OpenGL. Conseguiu-se um número significativamente menor de *strips* em comparação com o SGI, entre 60% e 80% a menos, resultando em uma melhora de 15% sobre o modelo OpenGL. A implementação do *Stripe* está disponível em [12].

Método FTSG (*Fast Triangle Strip Generator*)

Xiang *et al.* [42] desenvolveram um algoritmo de geração de *strips* baseado em um algoritmo de árvore geradora e um particionamento cuidadoso em um conjunto de caminhos. A partir de uma malha poligonal, o algoritmo pode gerar *strips* triangulares mesmo que a malha contenha polígonos não convexos. O objetivo do algoritmo é minimizar o número de vértices utilizando, para isto, principalmente *strips* seqüenciais.

O algoritmo pode ser dividido, basicamente, em cinco passos que podem ser explicados com o auxílio da figura 2.7 e que são descritos a seguir:

(a) gerar a triangulação das faces não trianguladas;

(b) construir uma árvore geradora com base no grafo dual da triangulação;

(c) particionar a árvore geradora em um conjunto de caminhos;

(d) decompor os caminhos em *strips* seqüenciais ou em leque;

(e) concatenar *strips* curtas em *strips* mais longas usando um conjunto de heurísticas de pós-processamento.

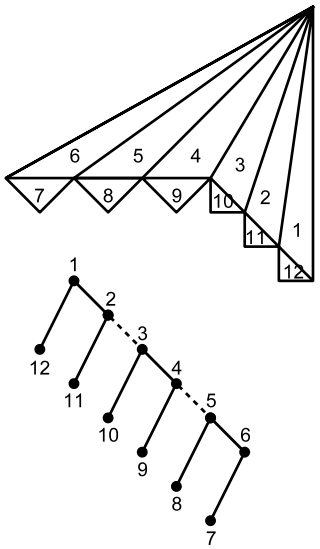


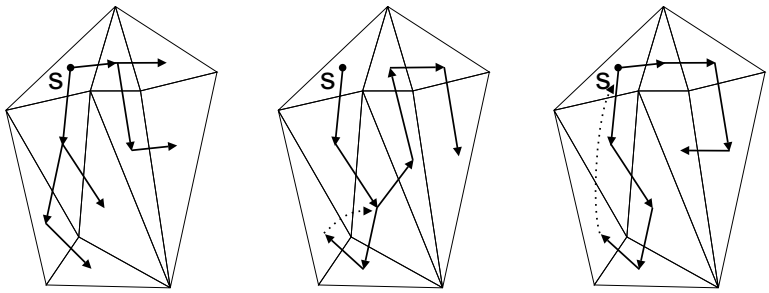
Figura 2.7: Exemplo de triangulação e o grafo dual correspondente.

Para gerar um modelo completamente triangulado, uma modificação do robusto algoritmo FIST [19] foi usada. O algoritmo FIST permite a triangulação de polígonos convexos e não convexos na malha, até mesmo quando estes estão corrompidos.

Esta parte do algoritmo, cuja saída da geração de *strips* são triangulações de faces convexas que são *strips* triangulares puras, é feita, no pior caso, em $O(n \log(n))$ (para faces com buracos). Na prática, como a maioria dos polígonos das malhas utilizadas tende a ser triângulos, quadriláteros ou polígonos de baixa cardinalidade, o algoritmo requer tempo linear.

Para a construção da árvore geradora, três abordagens diferentes foram implementadas (figura 2.8):

- busca em largura;
- busca em profundidade;
- uma variante híbrida que executa a busca em largura e retorna o nodo mais alto ainda não completamente explorado.



(a) (b) (c)

Figura 2.8: Três algoritmos de busca em grafos. (a) busca em largura; (b) busca em profundidade; (c) híbrido.

O objetivo dos algoritmos, mostrados na figura 2.8, é construir uma árvore geradora que tenha um pequeno número de nodos de grau 3, sendo que tais nodos requerem um grande número de caminhos.

A busca em largura tende a gerar árvores próximas às binárias e, por esta razão, o número de nodos de grau 3 pode ser muito grande.

A busca híbrida e a busca em profundidade tendem a produzir mais nodos de grau 2, portanto, o número de caminhos gerados é baixo.

De acordo com os resultados obtidos pelos autores, a busca em profundidade produz a melhor árvore geradora de todas as três abordagens.

O algoritmo FTSG tem como objetivo minimizar o número de vértices escolhendo, para isto, o triângulo que não produz um *swap*. Em seguida, o algoritmo utiliza uma otimização de programação dinâmica para particionar a árvore geradora em um conjunto de caminhos. A função objetivo desta programação dinâmica estabelece que cada nodo da árvore geradora tenha o número mínimo de *strips* seqüenciais derivadas da subárvore.

Percorrendo a árvore de baixo para cima e armazenando a melhor decomposição para cada nodo, o algoritmo pode alcançar a decomposição ótima. Como existe um número constante de casos por nodo e como cada nodo é visitado uma única vez, a computação deste algoritmo pode ser feita em tempo linear.

O uso exclusivo de *strips* seqüenciais é preferido para minimizar o número de vértices, neste caso, o passo de decomposição tem somente a tarefa de converter a lista de triângulos em uma lista de vértices. A decomposição usando *strips* em leque que, na prática, produz os piores resultados, também foi implementada juntamente com uma combinação de *strips* seqüenciais e em leque. No último caso, uma *strip* em leque somente é criada se a decomposição pelo algoritmo guloso encontra 4 triângulos consecutivos que não podem ser codificados em uma *strip* seqüencial.

Na última etapa, a concatenação da *strip* é feita, sendo que, para isto, deve ser permitida a concatenação de triângulos de área zero.

Existem várias configurações típicas no processo de geração de *strips* que podem reduzir o número de *strips*, pelo fator de 1, por um custo de 0, 1 ou 2 vértices. A abordagem usando a busca em profundidade otimiza localmente o processo de geração de *strips* somente para um par de *strips* vizinhas. Como a *strip* pode ter mais opções para a concatenação, a sua ordem é importante.

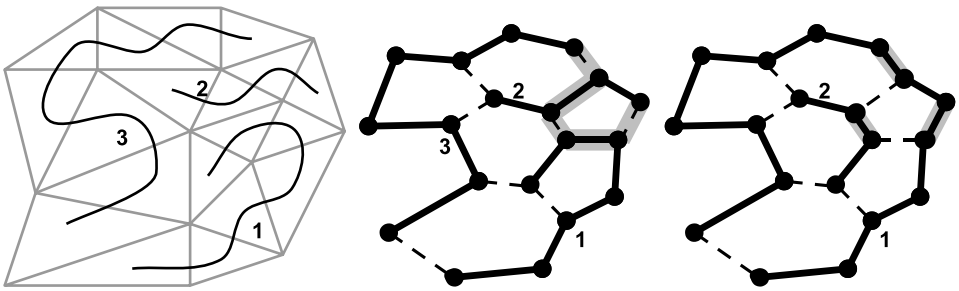
A otimização global pode ser obtida pelas múltiplas passagens do algoritmo, porém isto aumenta o seu tempo de execução e a memória a ser utilizada. O algoritmo é projetado para malhas estáticas não completamente trianguladas, sendo que a fase de triangulação é feita antes do início do processo de geração de *strips*. O número de vértices produzidos por este processo é cerca de 10% menor que o método SGI mas, em contrapartida, produz mais *strips*.

Método de Tunelamento

Stewart [31] desenvolveu um algoritmo global para malhas estáticas completamente trianguladas e malhas de nível de detalhe contínuo (*CLOD, continuos level of detail*). O método é baseado em um operador de grafo chamado tunelamento.

No grafo dual da triangulação gerada pelas *strips* podem ser encontradas dois tipos de arestas do grafo. A figura 2.9(a) mostra a geração de três *strips*. *Arestas-strip* unem nodos que correspondem a triângulos que são adjacentes à mesma *strip*. Todas as arestas

restantes são chamadas de arestas *não-strips*. Um túnel em um grafo dual é uma seqüência alternada de arestas *strips* e arestas não-*strips* que começam e terminam em arestas não-*strips* e conecta as extremidades de duas *strips* (túnel mostrado em cinza na figura 2.9(b)). Pelo complemento do estado de cada aresta em um túnel, isto é, mudando uma aresta *strip* para uma não-*strip* e vice-versa, o número de *strips* pode ser reduzido em 1 (figura 2.9(c)).



(a) (b) (c)

Figura 2.9: (a) Uma malha triangular composta por três *strips*; (b) o túnel, em cinza, é uma seqüência alternada de *strips* (linha preenchida) e não-*strips* (linha pontilhada); (c) complementando o estado das arestas no túnel, o número de *strips* é reduzido em um.

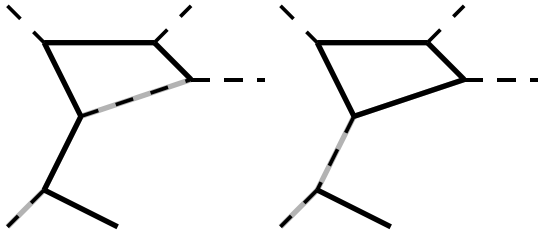
O algoritmo inicia pela escolha de algum nodo extremo da *strip*; pela busca em largura, o túnel mais curto no grafo é encontrado e o estado das arestas do túnel é mudado. Encontrar uma geração de *strips* ideal é um problema NP-difícil, por esta razão não é conhecido o número de *strips* que alcança o mínimo global. O algoritmo é repetido enquanto um túnel puder ser encontrado e o número de *strips* alcance o mínimo local. O número de *strips* no final do processo depende da ordem na qual os nodos são processados.

Existem, no entanto, duas limitações para que a busca em largura produza um túnel válido:

- a última aresta do túnel não pode conectar dois nodos pertencentes a uma mesma *strip* (figura 2.10(a)). Fazendo-se o complemento do estado das arestas em tal túnel, o número de *strips* não é reduzido e uma *strip* infinita é criada (figura 2.10(b)).

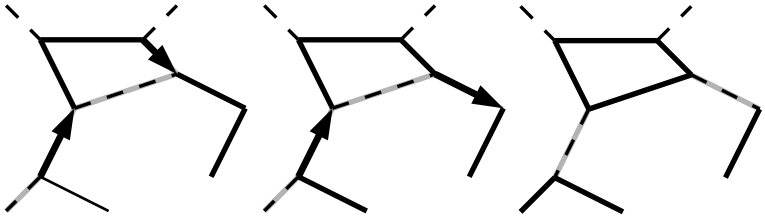
- uma aresta não-*strip* no túnel conecta dois nodos da mesma *strip*; as direções voltadas para o final das arestas adjacentes no túnel têm que ser opostas (figura 2.11(a)).

Se a segunda condição não for satisfeita (figura 2.11(b)), o número de *strips* não é reduzido e uma *strip* infinita é criada (figura 2.11(c)).



(a) (b)

Figura 2.10: (a) A última aresta do túnel não pode conectar dois nodos pertencentes a uma mesma *strip*; (b) ocorrendo a situação descrita, uma *strip* infinita surgirá.



(a) (b) (c)

Figura 2.11: (a) Uma aresta não-*strip* conecta dois nodos da mesma *strip*; as direções voltadas para o final das arestas adjacentes no túnel têm que ser opostas; (b) se esta condição não for satisfeita, uma *strip* infinita é criada (c).

Para que tais condições sejam verificadas, cada nodo tem que conter um identificador da *strip* mãe. Quando a *strip* é mudada pelo operador de tunelamento, todos estes identificadores têm que ser adaptados, o que requer uma visita a todas as *strips* afetadas (no pior caso). Esta operação pode consumir um tempo considerável.

O algoritmo de tunelamento pode ser utilizado de várias maneiras:

- em malhas estáticas: repetidas aplicações reduzem o número de *strips* otimizando a geração destas;
- em malhas CLOD: as transformações de nível de detalhe não requerem uma ordem específica. Estas transformações podem provocar falhas no modelo, as quais podem ser corrigidas pela aplicação do operador de tunelamento, mantendo uma geração de *strips* de alta qualidade;
- em malhas progressivas: age de forma similar ao utilizado em malhas CLOD, sendo

aplicado em níveis intermediários para melhorar a geração de *strips* da malha.

O tunelamento é o algoritmo que produz o menor número de *strips*, entretanto, o número de vértices é muito maior do que o produzido pelo método SGI.

A principal desvantagem do algoritmo de tunelamento, quando usado sobre malhas estáticas, é o tempo de execução em comparação ao método SGI. Quando aplicado a malhas CLOD, o tunelamento mantém uma boa geração de *strips* sobre um número de *strips* constantes.

Método baseado em Critério de Simplificação

Belmonte *et al.* [4] projetaram um algoritmo para a geração de *strips* a partir de malhas triangulares que é guiado por um critério de simplificação dado pelo erro quadrático mínimo associado à contração de uma aresta, conforme descrito em [15]. *Strips* triangulares criadas segundo esse critério são preservadas como no modelo que está sendo simplificado.

Enquanto a operação de contração de aresta está sendo executada podem ocorrer erros. O cálculo do erro é baseado na soma dos quadrados das distâncias dos vértices em relação a um plano médio. O algoritmo calcula esses erros e os associa com as arestas correspondentes, sendo que esses erros também determinam o peso da aresta no grafo dual da malha.

O algoritmo de busca da árvore geradora máxima é, então, aplicado ao grafo dual. *Strips* triangulares são criadas percorrendo-se a árvore geradora.

O algoritmo de Belmonte *et al.* foi projetado para malhas triangulares progressivas. Ele produz um alto número de *strips* e um número maior de vértices do que o método SGI, entretanto, o número de *strips* é preservado mesmo quando até 45% das arestas sofreram contração.

2.2.4 Métodos híbridos

Strips hierárquicas

Velho *et al.* [38] introduziram um método de refinamento para computar uma seqüência de triângulos em uma malha. Este método é aplicado para construir uma triangulação

em uma única *strip* que cobre uma superfície paramétrica ou implícita, sendo com isto obtida uma hierarquia de triângulos definida em cada nível de refinamento.

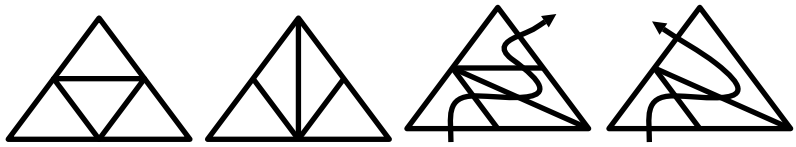
Como os dispositivos gráficos atualmente manipulam triângulos diretamente em hardware, as malhas triangulares são freqüentemente usadas para simular superfícies suaves. Para renderizar a malha em um nível de detalhe apropriado, um algoritmo de refinamento é aplicado. Geralmente, algoritmos de refinamento produzem modelos derivados pela subdivisão das faces originais. A face é subdividida de acordo com um modelo chamado esquema de subdivisão.

Este algoritmo produz uma seqüência refinada de triângulos, ou seja, uma seqüência na qual a ordem pode ser preservada quando os elementos são subdivididos. Tal propriedade depende exclusivamente do esquema de subdivisão.

O algoritmo consiste em duas partes:

- inicialização: que cria uma malha base e a *strip* triangular inicial correspondente.
- refinamento: que atua sobre a malha base preservando a única *strip* triangular.

Para fazer o refinamento é possível usar duas abordagens: a abordagem de subdivisão uniforme e a não uniforme. A abordagem uniforme subdivide recursivamente todos os triângulos da malha até que a resolução desejada seja obtida. Normalmente, a abordagem uniforme divide todas as três arestas no ponto médio de cada uma e subdivide o triângulo em 4 triângulos idênticos (figura 2.12(a,b)). A solução mostrada não é a única possível, há outros esquemas de subdivisão.

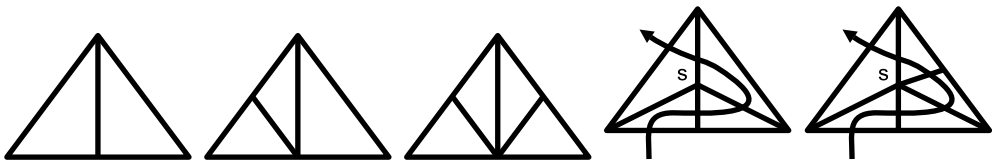


(a) (b) (c) (d)

Figura 2.12: (a,b) Esquemas de subdivisão; (c,d) seqüências de triângulos.

A abordagem de refinamento não uniforme (adaptativo) pode subdividir somente uma parte da malha. Esta abordagem divide as arestas do triângulo em duas partes, sem que necessariamente as três arestas tenham que ser subdivididas. Por isso, ao menos três diferentes esquemas devem existir.

A figura 2.13 mostra os esquemas de subdivisão: (a) uma aresta dividida; (b) duas arestas divididas e (c) três arestas divididas (este esquema é o mesmo da subdivisão uniforme). Para todas as situações é possível encontrar uma seqüência correspondente de triângulos a qual respeita a aresta de entrada e a aresta de saída. Na situação quando uma aresta de saída não dividida é adjacente a uma sub-aresta de entrada, não é possível encontrar uma seqüência, sendo assim, um novo ponto s é inserido ao esquema (figuras 2.13(d,e)).



(a) (b) (c) (d) (e)

Figura 2.13: Esquemas para subdivisões não uniformes utilizando (a) uma aresta, (b) duas arestas e (c) três arestas; (d,e) seqüências de triângulos.

O algoritmo produz uma única hierarquia de *strips* triangulares tanto para o esquema uniforme quanto para o esquema adaptativo. Para que se consiga uma única *strip* para toda a malha, um grande número de *swaps* é necessário. Por este motivo, o número de vértices no modelo composto por *strips* é alto.

Skip-strips

O algoritmo *skip-strip* [10] mantém uma geração de *strips* de um tipo de malha CLOD chamada VDPM (*View-dependent progressive mesh*). A maioria dos trabalhos anteriores referentes à geração de hierarquias de multi-resolução concentrava-se em computar um número fixo de conjunto de vistas independentes de níveis de detalhe. Em tempo de execução, um destes conjuntos era selecionado baseado em parâmetros do ponto de observação. Este tipo de abordagem pode ser muito restritiva, não levando em conta a posição da luz, acuidade visual, silhuetas e detalhes finos da imagem.

Houve recentes avanços neste tópico de modo a tomar vantagem da coerência temporal para, adaptativamente, refinar ou simplificar o ambiente poligonal de um *frame* para o próximo. Estes métodos, conhecidos como níveis de detalhe adaptativos, têm sido

usados em terrenos por Gross *et al.* [17] e Lindstrom *et al.* [23]. Várias técnicas de simplificação de malhas poligonais dependente de vista utilizam operações primitivas como contração de aresta e partição de vértice, como mostrado na figura 2.14. Simplificações dependentes de vista utilizando essas operações foram também desenvolvidas por Xia *et al.* [40], Hoppe [20], Guéziec *et al.* [18] e El-Sana e Varshney [11]. Outros tipos de operações para simplificação foram desenvolvidas por Luebke e Erikson [24] e De Floriani *et al.* [8].

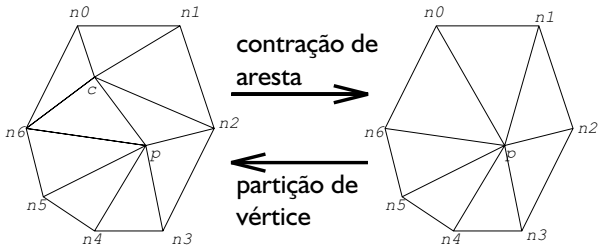


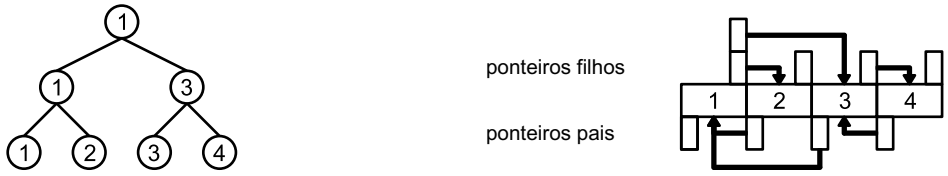
Figura 2.14: Operações em malhas.

O algoritmo *skip-strip* é baseado em uma estrutura de dados conhecida como *skip list* [28] e é possível usá-lo em combinação com qualquer outro algoritmo de geração de *strips*.

Para armazenar uma hierarquia da malha, utiliza-se uma estrutura chamada *merge tree* [41]. A *merge tree* é construída de baixo para cima na hierarquia, a partir de uma malha de alto nível de detalhe para uma malha de baixo nível, pelo armazenamento de operações de contração de aresta em uma estrutura hierárquica. Para construir um nível da árvore, seleciona-se o conjunto máximo de contrações de arestas, em ordem crescente de tamanho de aresta e com a limitação de não ter áreas sobrepostas. Os vértices restantes são promovidos ao próximo nível da árvore. O critério de não ter áreas sobrepostas permite mostrar vários detalhes dependendo dos parâmetros dependentes de vista, tais como luz ou orientação do polígono. Como a árvore não muda durante a visualização, ela é gerada em um estágio de pré-processamento.

Uma *skip strip* é um vetor de nodos onde cada nodo contém informação de vértice (coordenadas, cores, textura), uma lista de ponteiros de filhos e um ponteiro para o pai. Um nodo *skip strip* é alocado para cada nodo *merge tree*, ou seja, para cada vértice da malha base. O ponteiro pai e todos os ponteiros filhos copiam a hierarquia da *merge tree*. Na figura 2.15(a), um exemplo de *merge tree* é mostrada. Os níveis da árvore

correspondem ao nível de detalhe do modelo. Na figura, o modelo mais detalhado consiste em quatro vértices. Quando o modelo é simplificado, o vértice 2 se contrai no vértice 1 e o vértice 4 no vértice 3. O modelo menos detalhado é representado por um único vértice, o vértice 1. Na figura 2.15(b), uma estrutura *skip list* é apresentada.



(a) (b)

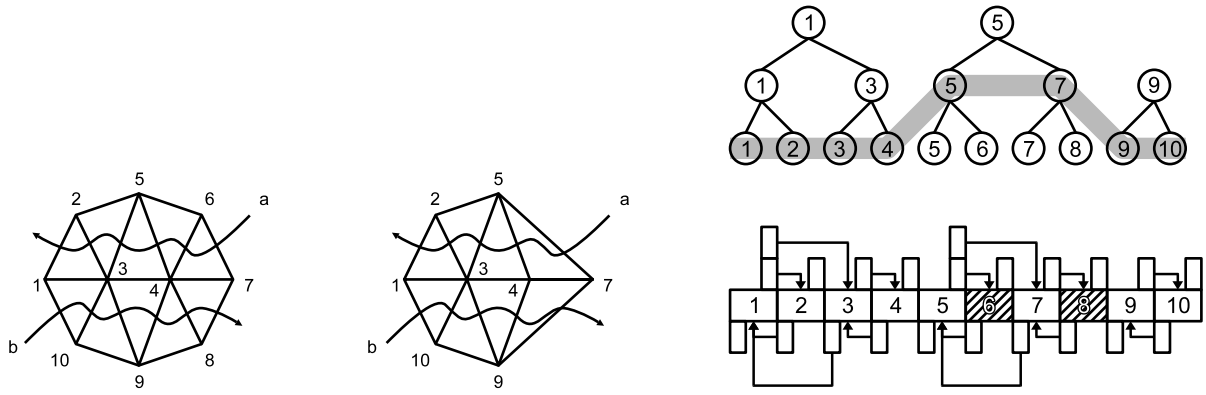
Figura 2.15: (a) Exemplo de *merge tree* e (b) sua estrutura *skip list* correspondente.

A figura 2.16(a) mostra uma malha triangular simples. Durante a visualização, algumas arestas podem sofrer contrações (figura 2.16(b)). Por meio da estrutura *skip strip*, as *strips* triangulares podem ser corretamente geradas pela troca de vértices inválidos por vértices adequados. Na figura, a *strip* original dada por a é definida por uma seqüência de vértices $7, 6, 4, 5, 3, 2, 1$. Como o vértice 6 contrai-se no vértice 5 em um nível de detalhe mais baixo, a *strip* é mostrada como a seqüência $7, 5, 4, 5, 3, 2, 1$. Para melhorar a geração de *strips*, o algoritmo percorre a *strip* triangular simples para detectar e substituir essas seqüencias repetidas.

Este algoritmo não produz *strips*, mas é projetado para manter as *strips* em malhas CLOD. Como a estrutura *skip strip* é genérica, ela pode ser usada em combinação com qualquer método de geração de *strips*. O aumento de velocidade usando a representação *skip-strip* é de 30 a 95% em comparação com a representação com triângulos.

Analisando-se o trabalho de El-Sana *et al.* [10], tem-se que a estrutura *skip strip*:

- oferece uma representação simples que integra estruturas como *strips* triangulares com um modo imediato de simplificação de vista dependente;
- oferece duas vantagens principais:
  - faz ligações de ponteiros eficientemente ao longo de ligações pais em qualquer esquema de contração de vértice hierárquico;



(a) (b) (c)

Figura 2.16: (a) Uma malha coberta por duas *strips*; (b) Uma malha simplificada; (c) suas correpondentes *merge tree* e *skip list*. A área cinza na *merge tree* mostra o nível de detalhe ativo para a malha.

  - simplifica a execução de operações como partição de vértice e contração de aresta a ponto de serem tão simples quanto à soma ou subtração de dois inteiros.

- oferece a vantagem da aceleração assistida por hardware para simplificações dependentes de vista.

Uma das desvantagens do algoritmo *skip strip* é que o desempenho não é alto para conjuntos de dados com várias descontinuidades nas superfícies, como falhas (*cracks*), junções T, normais, cores e texturas.

Cache de vértices

Deering [9] propôs o uso de um *stack-buffer* para o armazenamento dos últimos 16 vértices previamente utilizados, em vez de prover acesso aleatório a todos os vértices do modelo. Para placas com memória *on-board* limitada, esta se apresenta como uma solução satisfatória. Deering também generaliza a sintaxe da *strip* triangular provendo um controle mais geral sobre como o próximo vértice será usado, permitindo a inclusão temporária do vértice atual na pilha e o reuso de qualquer um dos 16 vértices do *stack-buffer*.

O custo de armazenamento para esta informação de conectividade é:

- 1 bit por vértice para indicar se o vértice deveria ser empilhado no *stack-buffer*;
- 2 bits por triângulo para indicar como continuar a *strip* atual;
- 1 bit por triângulo para indicar se um novo vértice deveria ser lido ou se um vértice do *stack-buffer* deveria ser utilizado;
- 4 bits de endereço para selecionar um vértice do *stack-buffer* para cada vez que um vértice velho é novamente utilizado.

Assumindo-se que cada vértice é reutilizado uma única vez, o custo total para a codificação da informação de conectividade é de 6 bits $(1 + 1 + 4)$ por vértice, mais 4 bits $(2 + 1 + 1)$ por triângulo. Assumindo-se ainda 2 triângulos por vértice, o custo total é de 11 bits por vértice.

É importante salientar que algoritmos para criar boas malhas transversais, ou gerais, usando-se a sintaxe generalizada de Deering, ainda não foram desenvolvidos. A criação trivial de malhas transversais pode resultar em muitos triângulos isolados ou corridas de pequeno comprimento, implicando que uma significativa porção dos vértices será enviada mais de uma vez e, com isto, aumentando o número de bits por triângulo.

Bar-Yehuda e Gotsman [3] estudaram o impacto do tamanho do *buffer* sobre o tempo de renderização. Mostrou-se que um *buffer* de tamanho $13.35\sqrt{n}$ é suficiente para renderizar qualquer malha poligonal definida sobre n vértices no tempo mínimo $O(n)$.

Hoppe [20] apresentou um algoritmo que otimiza as *strips* triangulares para um sistema com uma determinada quantidade de memória que, de forma simples, reduz a largura de banda da geometria. O algoritmo é baseado em uma simulação *look-ahead* do comportamento do *cache* de vértices. A estratégia básica do algoritmo é incrementalmente aumentar uma *strip* triangular e decidir, a cada passo, onde é melhor adicionar o novo triângulo à *strip* ou começar uma nova.

Para tomar essa decisão, o algoritmo faz uma simulação de *look-ahead* do cache de vértices. No início, o algoritmo marca todos os triângulos da malha como não visitados. Um triângulo com poucos vizinhos é escolhido para começar uma nova *strip*. Quando existir somente um triângulo vizinho não visitado, ele é conectado à *strip*. Quando existirem duas faces, o algoritmo continua a *strip* em sentido horário, mas o outro vizinho é colocado em uma fila de possíveis começos para novas *strips*. Quando não houver mais

triângulos não visitados, a *strip* não pode continuar e uma nova *strip* deve ser criada. Como triângulo inicial da *strip*, escolhe-se um triângulo da fila e, caso não haja nenhum, o algoritmo escolhe um triângulo no qual os vértices já estejam no *cache* e que possua um número baixo de vizinhos não visitados. Como a capacidade do *cache* de vértices é limitada, pode acontecer que a *strip* exceda esta capacidade e que não seja possível a reutilização dos vértices do *cache*.

Capítulo 3

Método Desenvolvido

Este capítulo descreve o método desenvolvido para a geração eficiente de *strips* a partir de malhas triangulares.

A técnica descrita a seguir é uma otimização do algoritmo apresentado em [7, 36] reduzindo a quantidade de *strips* geradas. Uma característica do algoritmo, desenvolvido neste trabalho, é a que, caso não seja possível diminuir o número de *strips* geradas, este deverá permanecer o mesmo que o fornecido pelo algoritmo de geração de *strips* na entrada.

O objetivo inicial deste trabalho consistia em se obter uma redução de cerca de 10% sobre o número de *strips* geradas pelo método SStrip [7, 36], classificado como o de melhor resultado em tempo real. A técnica apresentada neste trabalho obteve resultados, na maior parte dos testes, superiores à meta proposta inicialmente, além de requerer uma complexidade computacional significativamente reduzida. Esta redução possibilita que o algoritmo, inicialmente restrito a malhas estáticas devido ao fator tempo, possa ser aplicado a malhas dinâmicas, as quais necessitam de compactação em tempo real.

Este capítulo está dividido em quatro seções. A seção 3.1 provê um detalhamento maior do método desenvolvido explicando cada fase do algoritmo. A seção 3.2 descreve a fase de extração das *strips*, sendo seguida pela seção 3.3 que descreve as *strips* circulares, consideradas as maiores reponsáveis pelo aumento das taxas de compactação obtidas inicialmente. Finalmente, são apresentadas as vantagens e desvantagens do método desenvolvido.

3.1 Descrição do Método

O método SStrip, proposto em [7, 36], usado como ponto de partida do algoritmo desenvolvido neste trabalho, procura minimizar o número de vértices a serem enviados ao *pipeline* gráfico. Duas heurísticas foram consideradas. A primeira procura minimizar o número de vértices reduzindo o número de *strips*, gerando como saída um formato que suporta *swaps* sem que seja necessário o reenvio do vértice. O número necessário de vértices é definido como $t + 2k$, onde t é o número de triângulos na malha e k o número de *strips* geradas. A outra heurística minimiza o número de vértices evitando a geração de *swaps*, produzindo como saída um formato que simula o reenvio do vértice, como utilizado no pacote OpenGL.

Este trabalho também investigou a geração de múltiplas *strips* simultaneamente em vários lugares da malha, fazendo a concatenação quando possível.

3.1.1 Algoritmo local

O algoritmo para a escolha do próximo triângulo a ser inserido na *strip* é similar a outros algoritmos gulosos [1, 13]. O algoritmo proposto analisa o grafo dual (seção 3.1.2) da malha dando prioridade para inserir triângulos que tenham vários triângulos adjacentes na *strip*. Em caso de empate, o algoritmo usa uma estratégia de *look-ahead* diferente, dependendo da heurística utilizada.

Seja o grau de um triângulo o número de triângulos adjacentes a ele que não pertençam a nenhuma *strip*. A selecão do próximo triângulo é feita utilizando os seguintes passos:

- se um triângulo na lista de candidatos tem grau 0, ele é adicionado imediatamente para evitar a ocorrência de uma *strip* de tamanho 1.

- se não existe um candidato de grau 0, triângulos de grau 1 têm prioridade. Caso existam vários candidatos de mesmo grau, um teste é feito verificando-se o grau do triângulo adjacente ao candidato. Se o grau for 1 o triângulo é inserido imediatamente. Se todos os triângulos adjacentes tiverem grau 2 ou 3, o algoritmo procura inserir o candidato que não produza um *swap*, quando a heurística usada é a minimização de *swaps*, caso contrário, ele insere o triângulo com o menor grau no triângulo adjacente.

- se todos os candidatos tiverem grau 2, a escolha do próximo triângulo é feita de acordo com a heurística em consideração. No caso de minimização de *strips* é inserido o triângulo cujo adjacente possua menor grau. No caso de minimização de *swaps*, o próximo triângulo escolhido é aquele que não produza *swap*.

Quando uma nova *strip* é criada, o triângulo com menor grau é escolhido como o triângulo inicial.

3.1.2 Grafo Dual ou de Adjacências

Utilizou-se uma modificação da estrutura *Triangle* [29] para representar o grafo dual da triangulação, a qual é implicitamente dada pela lista de adjacência de cada triângulo. Esta lista é adaptada pela busca de triângulos adjacentes somente se eles têm, ao menos, um vértice em comum, como é descrito a seguir.

Primeiramente é criada a lista principal de vértices com uma entrada para cada vértice. Esta entrada será a segunda lista que retém a informação de quais triângulos estão ligados a este vértice (figura 3.1). Cada entrada da segunda lista é um item que contém um ponteiro para o triângulo associado com este vértice e o índice para o outro vértice que forma a aresta do triângulo apontado. O índice na lista principal de vértices e o índice na segunda lista de índices, implicitamente, compõem a informação da aresta do triângulo. Por exemplo, na figura 3.1, o índice a na lista principal e o índice b na lista de itens formam a aresta ab. Apenas três desses itens são inseridos, visto que estes representam as três arestas do triângulo, ordenadas pelos índices dos vértices.

Para cada triângulo, é verificado se existe uma referência para qualquer uma de suas arestas. Se existir, uma adjacência do triângulo é adaptada com o ponteiro para o triângulo do item, e uma adjacência do triângulo apontado é também adaptada com o triângulo corrente. Se não existir tal aresta, então os itens da aresta do triângulo corrente são inseridos na lista de vértice, onde as adjacências são adaptadas.

3.1.3 Estrutura de dados

Em conjunto com uma heurística simples, outra razão para a eficiência do algoritmo é o projeto de uma estrutura de dados eficiente. Esta estrutura permite uma indexação de triângulos direta em um vetor de triângulos e acesso através de uma lista de ponteiros

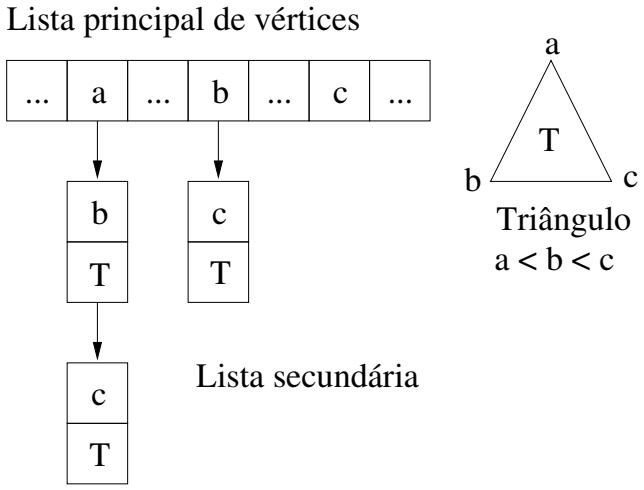


Figura 3.1: Estrutura de dados para a construção do grafo dual.

para triângulos ordenada por grau. Além disso, a estrutura de dados foi concebida de maneira que a reordenação da lista possa ser feita em tempo constante.

Além dos ponteiros para os triângulos adjacentes e seus vértices, cada triângulo da estrutura contém seu grau, uma referência à *strip* (se existir) na qual o triângulo está inserido e um ponteiro para o nodo da lista que aponta para o triângulo em consideração (figura 3.2). Esta informação é diretamente acessada através de um vetor de triângulos durante a execução da heurística. Quando for necessário começar a construção de uma nova *strip*, deve ser feita uma busca pelo triângulo com o menor grau na malha. Para isto, é utilizada uma lista de índices para os triângulos. Esta lista é ordenada pelo grau dos triângulos apontados pelos nodos. Sendo de 0 a 3 o intervalo possível de graus dos triângulos, a lista tem um ponteiro para o primeiro nodo de cada grau. Desta maneira, a reordenação da lista (que ocorre a cada passo de inserção de um triângulo) é trivial. Somente é necessária a remoção do nodo, para o caso em que o grau $k > 0$, e a sua inserção na lista usando o ponteiro para o primeiro nodo com grau $k - 1$.

No exemplo mostrado na figura 3.2, os nodos T_1 e T_2 da lista ordenada de triângulos apontam para triângulos com grau 0. Os nodos T_3 e T_4 apontam para um triângulo com grau 2 e os nodos $T_5 ... T_m$ apontam para triângulos com grau 3.

Uma vez que não existem triângulos com grau 1, $deg1$ aponta para nulo (figura 3.2). Se o grau do triângulo t_{k+1} muda de 3 para 2, o nodo T_6 é removido e reinserido entre T_2 e T_3 e é apontado por $deg2$. De outra forma, se o grau do triângulo t_{k+j} muda de 2 para

1, T_3 é removido, reinserido na mesma posição e será apontado por $deg1$ enquanto que $deg2$ apontará para T_4.

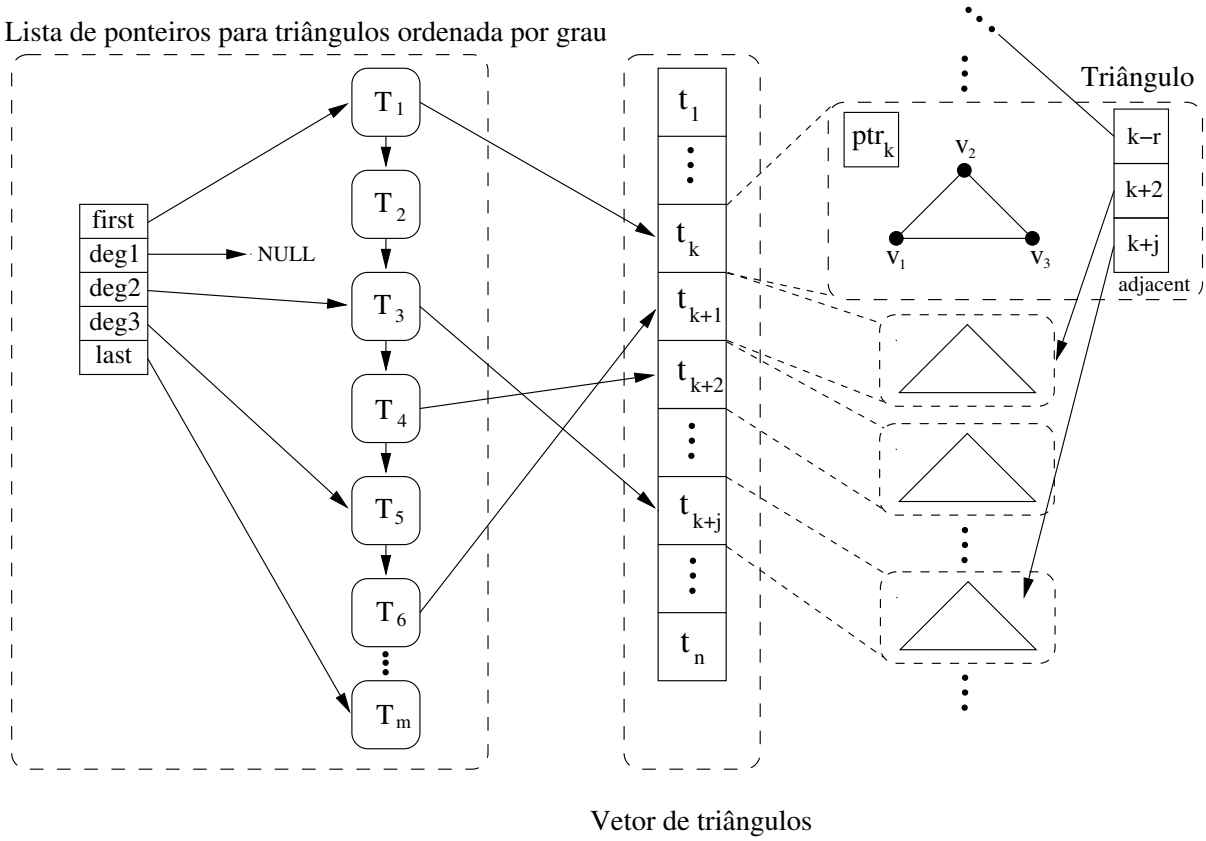


Figura 3.2: Estrutura de dados.

Após a geração das *strips*, na fase descrita acima, uma árvore é construída (figura 3.3) através da concatenação sucessiva de uma *strip* à outra. Na figura 3.4(a), 4 *strips* que são concatenadas por uma de suas extremidades para formar a árvore da figura 3.4(b).

Devido ao fato de nem todas as *strips* serem conectadas umas às outras, mais de uma árvore pode ser construída para cada malha. Por questões de simplificação, para se referir a tais árvores será utilizada a denominação de *árvore* simplesmente, sendo que esta englobará todas as sub-árvores geradas pelo algoritmo.

Após esta etapa, a construção da árvore, uma heurística é aplicada aos nodos de grau 3. A heurística utilizada consiste em uma análise dos nodos vizinhos ao nodo selecionado à procura de novas alternativas de conexão. Na figura 3.5(a), é mostrado um nodo de grau 3 e seus nodos vizinhos (destacados por círculos pontilhados). Caso algum vizinho do nodo analisado possua uma alternativa de conexão de grau 1, representando uma extremidade

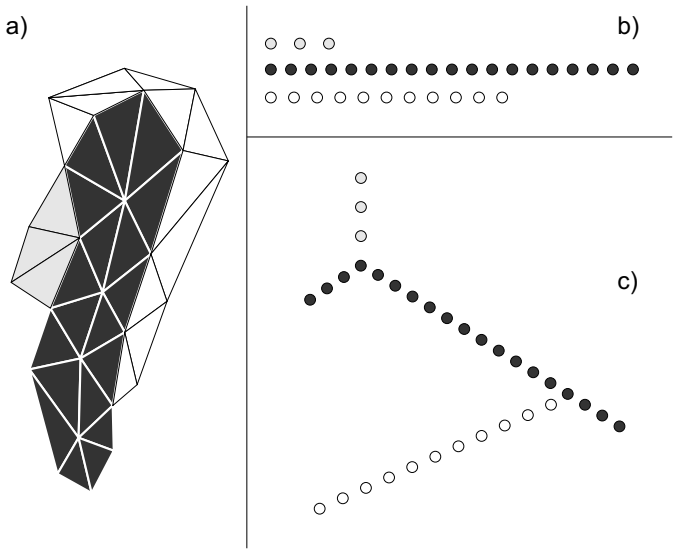
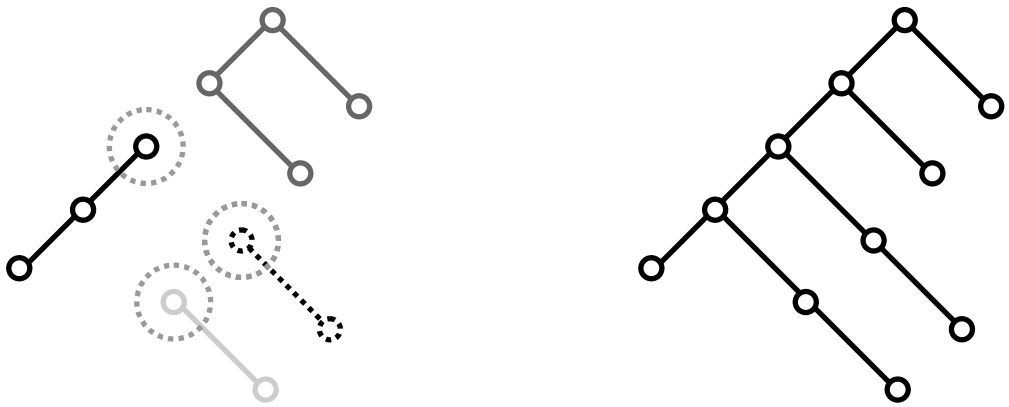


Figura 3.3: (a) Malha triangular; (b) strips da malha; (c) árvore com as *strips*.

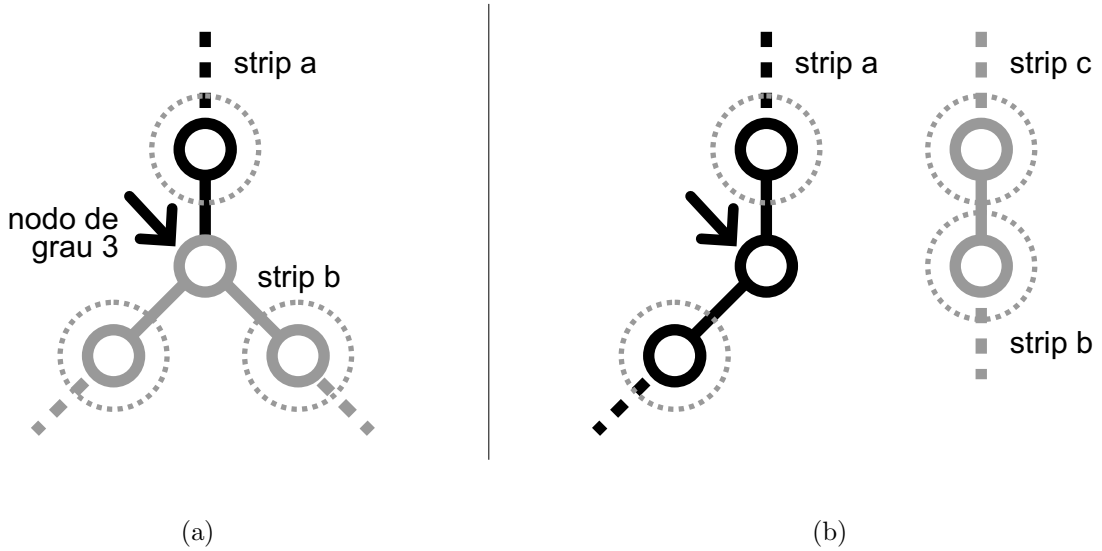


(a) (b)

Figura 3.4: Construção da árvore (a) quatro *strips* a serem conectadas (b) árvore construída a partir das strips anteriores.

de uma outra *strip*, este nodo é desconectado do nodo de grau 3 corrente (gerando uma redução de fator 1 no número de *strips*) e conectado ao nodo alternativo de grau 1. O nodo de grau 1 passará então a ter grau 2. Este aumento no grau do nodo só implicará em um aumento no número de *strips* caso o nodo passe a ter grau 3. As alternativas de redução do grau do nodo selecionado são mostradas na figura 3.6.

Resumindo-se a idéia principal do algoritmo, este consiste, basicamente, em particionar



(a) (b)

Figura 3.5: Nodo de grau 3 a ser analisado.

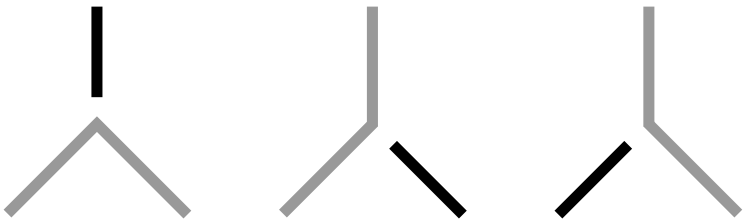


Figura 3.6: Possibilidades de desconexão e respectivas *strips* formadas.

as *strips* iniciais em porções as quais possam ser conectadas a outras *strips* existentes, reduzindo-se o número de *strips* finais. Caso isto não seja possível, a fase de extração das *strips* a partir da árvore gerará, no máximo, o mesmo número de *strips* que a entrada inicial.

Em uma análise mais detalhada do problema, verificou-se que não somente a abordagem sobre os vizinhos do nodo de grau 3 possibilitaria uma redução de grau do nodo selecionado. A figura 3.7 destaca os nodos analisados (envoltos por um círculo pontilhado) e os não levados em consideração (envoltos por um quadrado pontilhado). Caso qualquer um dos nodos, não levados em consideração, possa ser conectado a um nodo de grau 1, a redução do nodo de grau 3 pode ser obtida com poucas operações. Entretanto, para este tipo de redução de *strips*, a complexidade computacional pode crescer exponencialmente conforme o número de nodos que passarão a ser considerados.

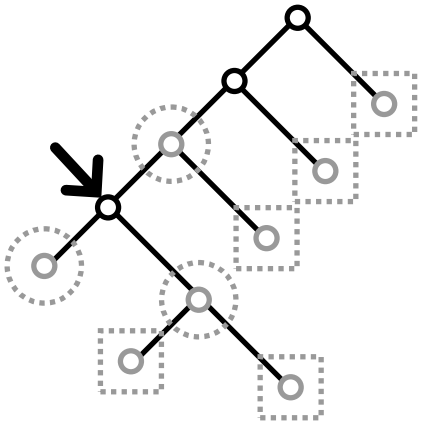
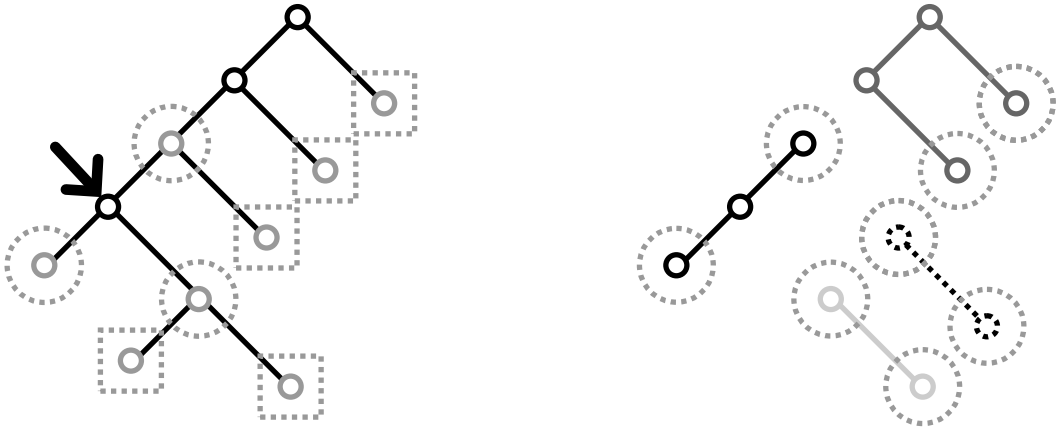


Figura 3.7: Nodo analisado (apontado pela seta), nodos levados em consideração (círculos pontilhados) e nodos não considerados (quadrados pontilhados).

Uma alternativa a esta abordagem, mantendo a complexidade em um limite conhecido, é a análise dos nodos das extremidades (envoltos por um quadrado pontilhado) na fase anterior, a fase de concatenação das *strips*. Se na fase de concatenação for definida a precedência em se utilizar nodos de grau 1 para os pontos de conexão, em ambas as extremidades, os casos mostrados na figura 3.8(a) já estarão cobertos. A complexidade desta etapa fica limitada à ordem de $2n$, sendo que n representa o número de *strips* e cada *strip* contendo um início e um fim.



(a) (b)

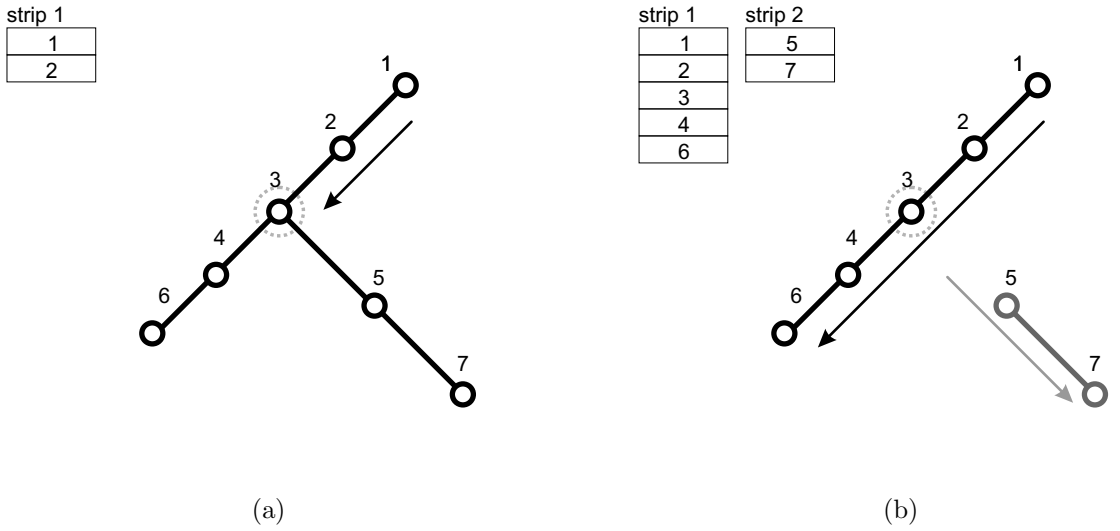
Figura 3.8: Averiguação de possibilidades de redução na fase 1.

A cada *strip* concatenada à outra por nodos de grau 1, uma redução de fator 1 é

obtida. Caso o nodo de grau 1 pertença à mesma *strip*, a redução não ocorre e uma *strip* circular é obtida. No ínicio do processo, esta condição era evitada, entretanto, a eliminação desta restrição incrementou o ganho na redução das *strips* de forma considerável. Esta característica deve-se ao fato de que as *strips* circulares têm mais chances de redução do que as não circulares, o que será melhor discutido na seção 3.3.

3.2 Extração das *Strips* da Árvore

O algoritmo de extração de *strips* funciona de maneira simples. Ao percorrer a árvore, o algoritmo utiliza uma fila para incrementalmente adicionar os nodos visitados até que encontre um nodo de grau 3. Quando o nodo de grau 3 é encontrado, o algoritmo cria uma nova fila vazia e chama o algoritmo de extração para um ramo do nodo encontrado, continuando a percorrer o outro ramo utilizando a fila antiga, conforme pode ser observado na figura 3.9.

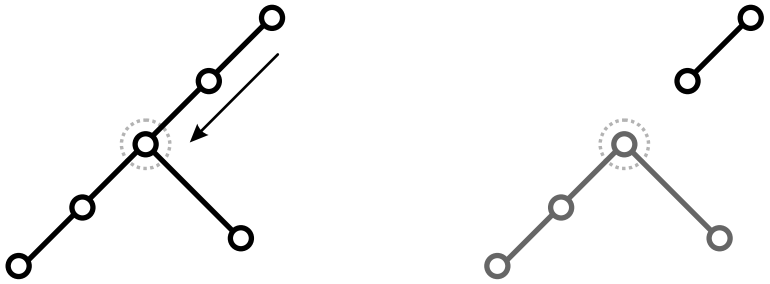


(a) (b)

Figura 3.9: Extração simples das *strips* da árvore.

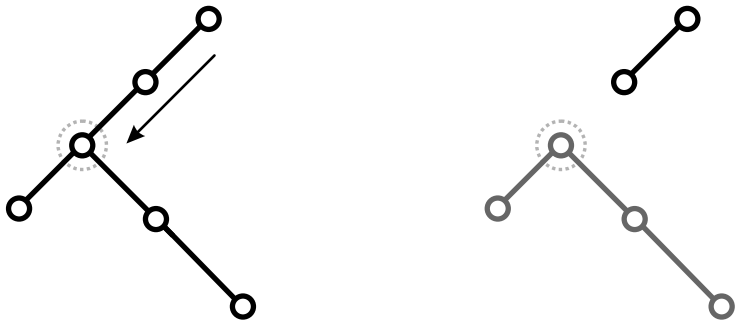
Realizando testes com algumas malhas, verificou-se que a malha final possuía um grande número de *strips* de comprimento 1, ou seja, um efeito indesejável do algoritmo. Este problema foi resolvido de forma simples, mostrado nas figuras 3.10 e 3.11. Caso o nodo de grau 1 seja o filho direito (esquerdo) do nodo de grau 3, figura 3.10 (3.11), a *strip* que estava sendo gerada anteriormente pára em um nodo acima do nodo de grau 3 e inicia outra com o filho de grau 1 do lado direito (esquerdo). Primeiramente é enfileirado

o nodo do lado direito (esquerdo) seguido pelo nodo de grau 3 e o filho esquerdo (direito) do nodo de grau 3 consecutivamente com seus filhos, caso existam. No caso em que o nodo de grau 1 é o nodo do topo, o algoritmo comporta-se da mesma maneira, fazendo-se as devidas considerações de topologia.



(a) (b)

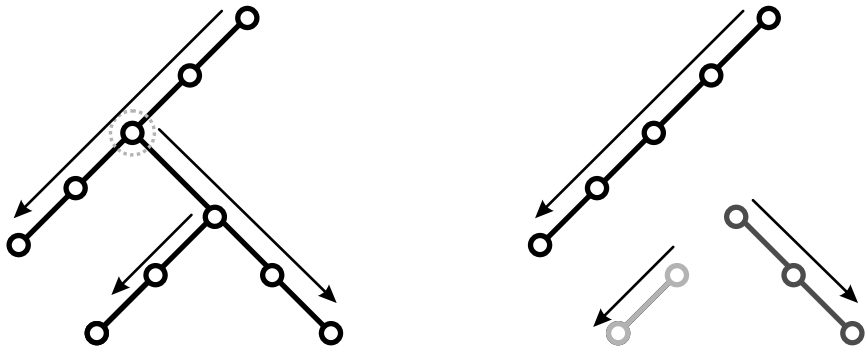
Figura 3.10: Tratamento para evitar *strips* de tamanho 1.



(a) (b)

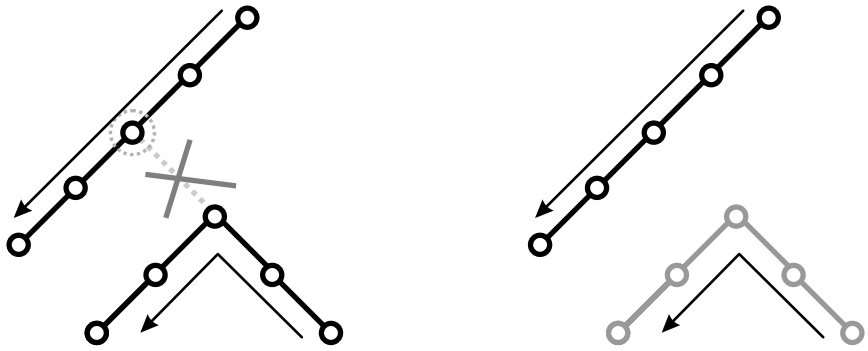
Figura 3.11: Tratamento para evitar *strips* de tamanho 1.

Um processo que se mostrou interessante para incrementar a taxa de redução das *strips* foi o eliminação de algumas arestas desnecessárias em uma fase anterior à extração. Na figura 3.13(a), a aresta eliminada não é necessária. Pelo processo normal de extração, sobre a árvore da figura 3.12 seriam geradas 3 *strips*, desnecessariamente. Com a modificação feita pela eliminação da aresta desnecessária, obteve-se 2 *strips*, conforme ilustrado na figura 3.13(b).



(a) (b)

Figura 3.12: Fase de extração gerando 3 *strips*.



(a) (b)

Figura 3.13: Eliminação da aresta desnecessária reduzindo a geração para 2 *strips*.

3.3 *Strips* Circulares

Retornando à figura 3.5, passada a fase inicial na qual já foram cobertas as outras possibilidades de redução (figura 3.8), a *strip a* (figura 3.5(a)), não circular, conectada à *strip b*, também não circular, possui somente três chances de redução (nodos circulados por linhas pontilhadas). Quando se passa a admitir o uso de *strips* circulares, as chances de redução são bem maiores, como foi comprovado pelos resultados obtidos.

Para se entender melhor o caso citado, suponha a existência de uma *strip* circular como a da figura 3.14. Qualquer *strip* não circular que seja conectada a um nodo qualquer de uma *strip* circular será reduzida a uma única *strip*.

Revisando o algoritmo de extração, explicado anteriormente, tem-se que a criação

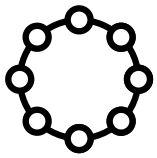


Figura 3.14: Uma *strip* circular inicialmente isolada.

de uma nova *strip* somente pode ser feita caso um nodo de grau 3 seja encontrado. Conectando-se uma *strip* não circular à *strip* anterior, obtém-se um cenário adequado para exemplificar a redução, mostrada na figura 3.15. O algoritmo irá percorrer a *strip* até chegar ao nodo de grau 3. Quando isto ocorrer, uma nova fila será criada para percorrer um ramo do nodo, enquanto que a fila antiga continuará a receber os nodos do ramo oposto. Como o algoritmo atua em profundidade, ele irá começar uma nova fila somente quando a fila anterior estiver preenchida. Como a segunda *strip* é circular, todos os nodos até o limite oposto ao nodo de grau 3 serão enfileirados. Desta maneira, quando a segunda fila for iniciada e o algoritmo testar o nodo para verificar se ele já foi percorrido, o teste resultará em verdadeiro e uma fila vazia será criada que, por sua vez, será descartada posteriormente. Este caso é ilustrado na figura 3.16. Ao invés de se ter somente 3 nodos candidatos à redução da *strip*, agora tem-se todos os nodos pertencentes às *strips* circulares como possíveis candidatos.

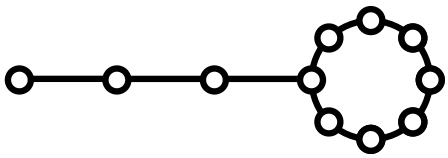


Figura 3.15: *Strip* circular conectada a uma *strip* simples.

Um fator de redução ainda maior pode ser obtido caso uma outra *strip* não circular seja conectada a um nodo vizinho ao nodo de grau 3 do caso anterior (figura 3.17). Neste caso, três *strips* passarão a ser uma única. A aresta marcada com um $\times$ será eliminada automaticamente na fase de pré-extração das *strips*, onde os nodos de grau 3 são desconectados de seus vizinhos de mesmo grau. Neste caso, o algoritmo nem se valerá

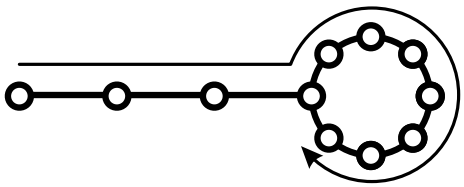


Figura 3.16: Formação de uma única *strip* percorrendo-se os nodos no sentido apontado pela seta.

da busca em profundidade e nem tampouco criará uma fila vazia, evitando processamento desnecessário.

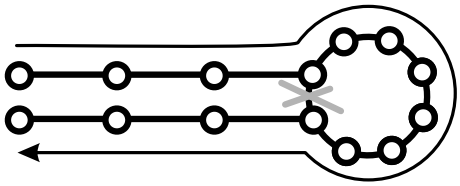
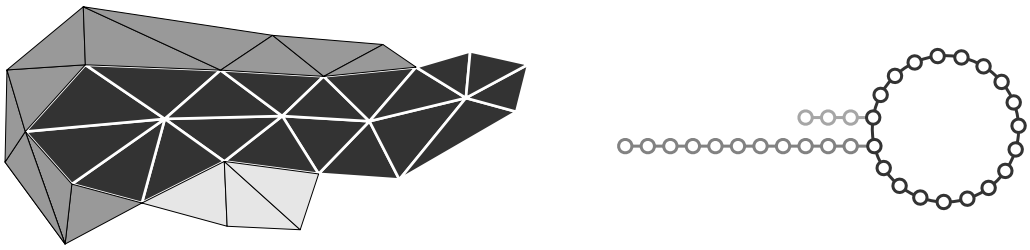


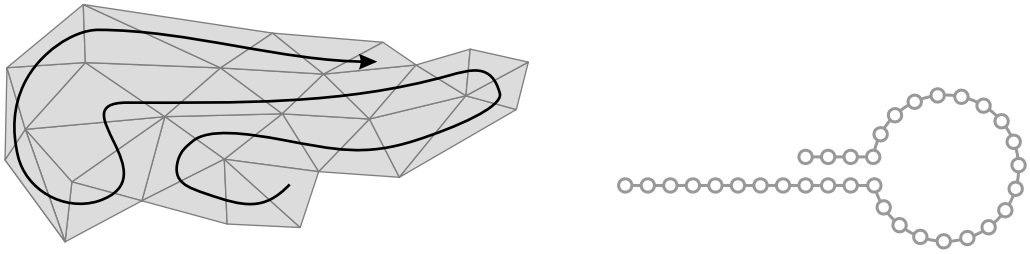
Figura 3.17: Três *strips* reduzidas a apenas uma.

A partir da figura 3.18(a), pode-se observar as 3 *strips* iniciais da malha na árvore da figura 3.18(b). Após a extração, uma única *strip* é criada como pode ser observado na figura 3.19.



(a) (b)

Figura 3.18: Malha reduzida a uma única *strip*.



(a) (b)

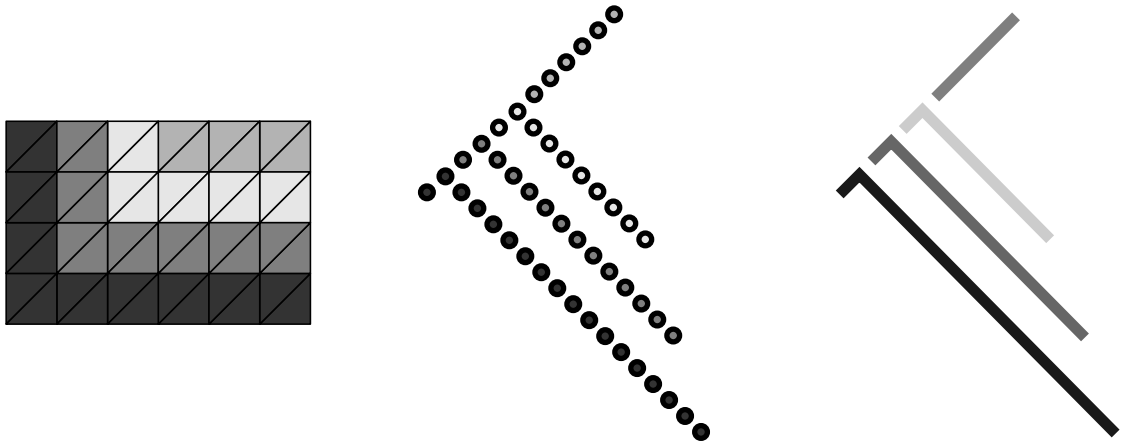
Figura 3.19: Redução a malha a uma única *strip*.

3.4 Vantagens e Desvantagens do Algoritmo Desenvolvido

O algoritmo desenvolvido requer acesso aleatório a uma parte dos nodos das *strips*, os nodos das extremidades e seus vizinhos.

Devido ao uso de uma heurística global, o algoritmo requer informações extras de topologia, consumindo mais memória.

Quando as *strips* de entrada são exclusivamente seqüenciais, ou seja, sem *swaps*, tem-se uma situação na qual o algoritmo deve manter a mesma configuração de *strips* que a de entrada, caso ilustrado na figura 3.20(a).



(a) (b) (c)

Figura 3.20: (a) Malha seqüencial; (b) árvore equivalente; (b) resultado da extração.

Nestas situações, o algoritmo se comporta exatamente como deveria. Relembrando o

passo inicial no qual o algoritmo tenta concatenar *strips* umas às outras pelas extremidades, a árvore mostrada na figura 3.20(b) é, então, criada. Na fase anterior, a extração das *strips*, quando a algoritmo desconecta vizinhos de grau 3, o resultado final são as mesmas *strips* que a entrada, mostradas na figura 3.20(c). É importante ressaltar que o mesmo resultado seria obtido pelo processo que evita a geração de *strips* de tamanho 1.

O algoritmo desenvolvido utiliza uma heurística global partindo de uma solução inicial que já é considerada uma geração de *strips* de boa qualidade. Não há nenhuma perda em relação à quantidade de *strips* gerada pelo algoritmo. No caso em que não há como melhorar a geração de *strips*, quando há somente *strips* seqüenciais, o algoritmo gera as mesmas *strips* como mostrado anteriormente. A complexidade computacional é relativamente baixa, visto que o algoritmo atua em somente uma das extremidades de cada *strip*. Desta maneira, o número de nodos analisados é $3n$ na fase de redução de *strips*, sendo n o número de *strips* e o fator 3 sendo o número de vizinhos ao nodo analisado. As fases de pré-extração e extração, atualmente separadas, poderiam ser unidas em uma única fase, adicionando mais $1n$ na complexidade final que ficaria em $4n$.

Apesar da implementação atual estar utilizando parte da estrutura do SStrip [7, 36], ela pode ser totalmente separada e utilizada como um processo de otimização de geração de *strips*.

Devido à baixa complexidade computacional, o método desenvolvido pode ser categorizado como compactação em tempo real, com resultados melhores que os métodos FTSG e SStrip na maioria dos casos, considerando-se o número de *strips*.

Capítulo 4

Resultados Experimentais

Neste capítulo são apresentados os resultados obtidos com a realização dos experimentos, juntamente com a discussão das contribuições advindas da aplicação das técnicas que compõem o método desenvolvido.

Os experimentos foram executados em um processador Athon XP 2500 de 1.8 GHz com 512 Mbytes de memória RAM, utilizando-se o sistema operacional Linux/Debian. As linguagens de programação utilizadas C/C++ e Java.

Nas figuras 4.1 e 4.2 são apresentadas amostras de alguns modelos utilizados nos testes. Esses modelos foram cedidos pelo *Stanford 3D Scanning Repository* [30], *United States Geological Survey* [35], *Georgia Institute of Technology* [16] e *Nasa's Planetary Data System* [26].

A seção 4.1 apresenta os resultados obtidos com o método desenvolvido, bem como uma comparação com outros dois métodos discutidos no capítulo 2: o FTSG [42], conhecido como o melhor programa disponível para compactação em tempo real e o SStrip [7, 36], método no qual este trabalho foi baseado.

Nos experimentos, a opção de redução de *strips* de tamanho 1, discutida na seção 3.2 do capítulo 3, é utilizada como padrão para o método de compactação, já que, como será visto mais adiante, ela introduz um acréscimo pouco significativo no número final de *strips* dos modelos.

Uma discussão dos resultados é apresentada na seção 4.2, avaliando-se o comportamento do método desenvolvido, suas vantagens e limitações.

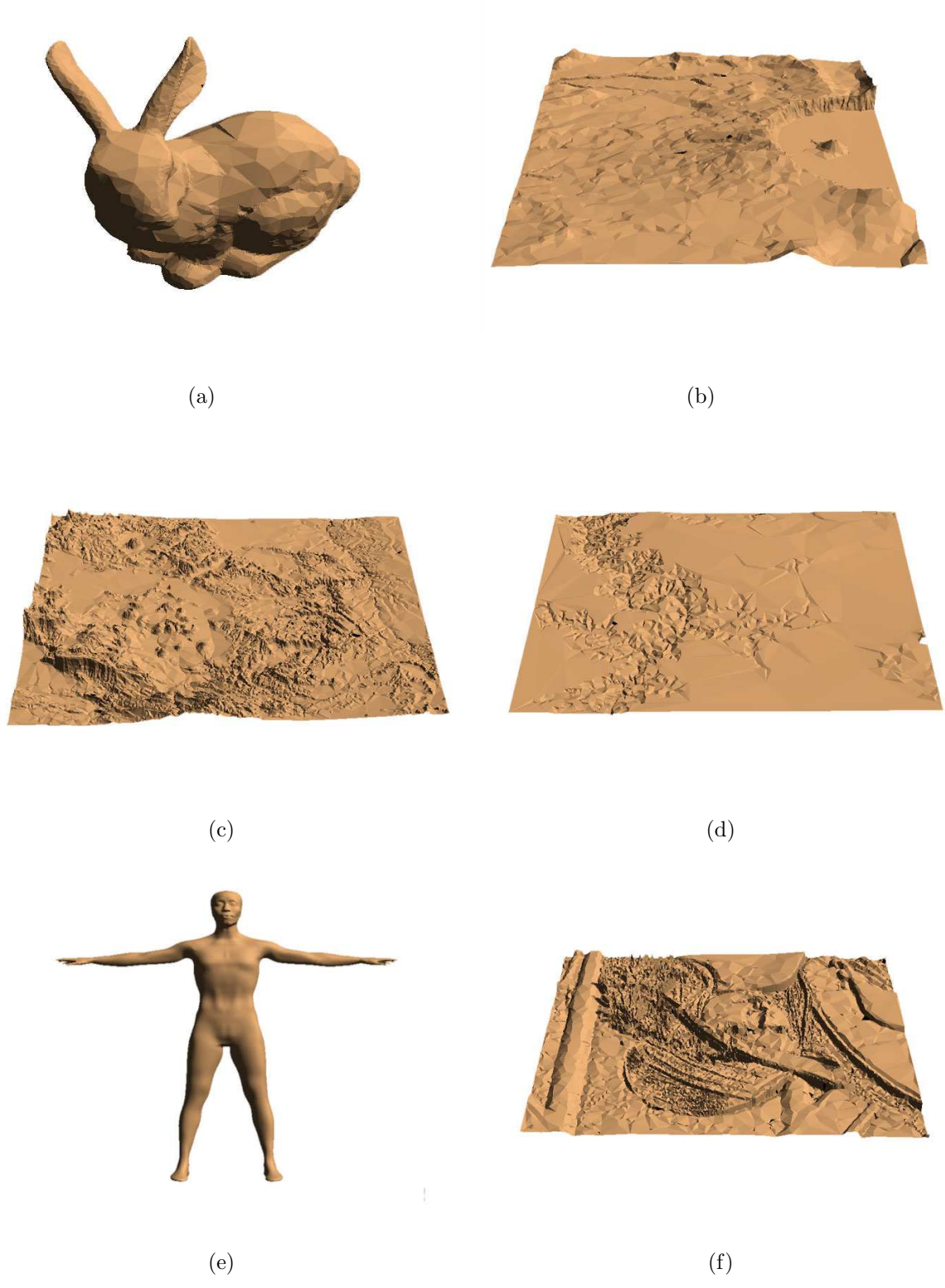
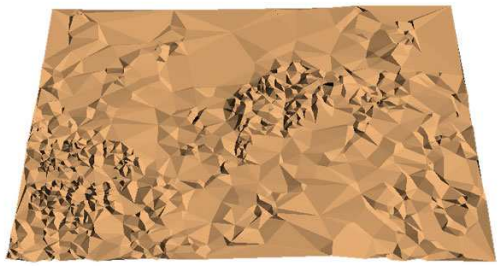


Figura 4.1: Alguns modelos utilizados nos experimentos. (a) *bunny4*; (b) *cl9852*; (c) *emorypeak*; (d) *grand canyon*; (e) *jackie*; (f) *lena*.

(a)



(b)

(c)

Figura 4.2: Alguns modelos utilizados nos experimentos. (a) *foot*; (b) *rl3955*; (c) *roseburg*.

4.1 Descrição dos Resultados

Esta seção descreve os resultados obtidos a partir dos experimentos realizados sobre diversos modelos, alguns deles mostrados anteriormente. A tabela 4.1 apresenta um resumo contendo os nomes dos modelos e suas respectivas quantidades de triângulos e vértices que os compõem, juntamente com a quantidade de *strips* geradas em cada método avaliado. Também é mostrado um gráfico na figura 4.3 para auxiliar a visualização dos resultados.

Como mencionado no capítulo 3, o algoritmo não fornece ganhos na taxa de redução para modelos compostos exclusivamente por *strips* seqüenciais, por exemplo o modelo *hawaii* mostrado na tabela 4.1. Entretanto, no pior caso, o algoritmo não aumenta o número de *strips* com relação ao modelo original.

Terrenos com grandes áreas planas, como os modelos *grand canyon* e *roseburg*, tendem

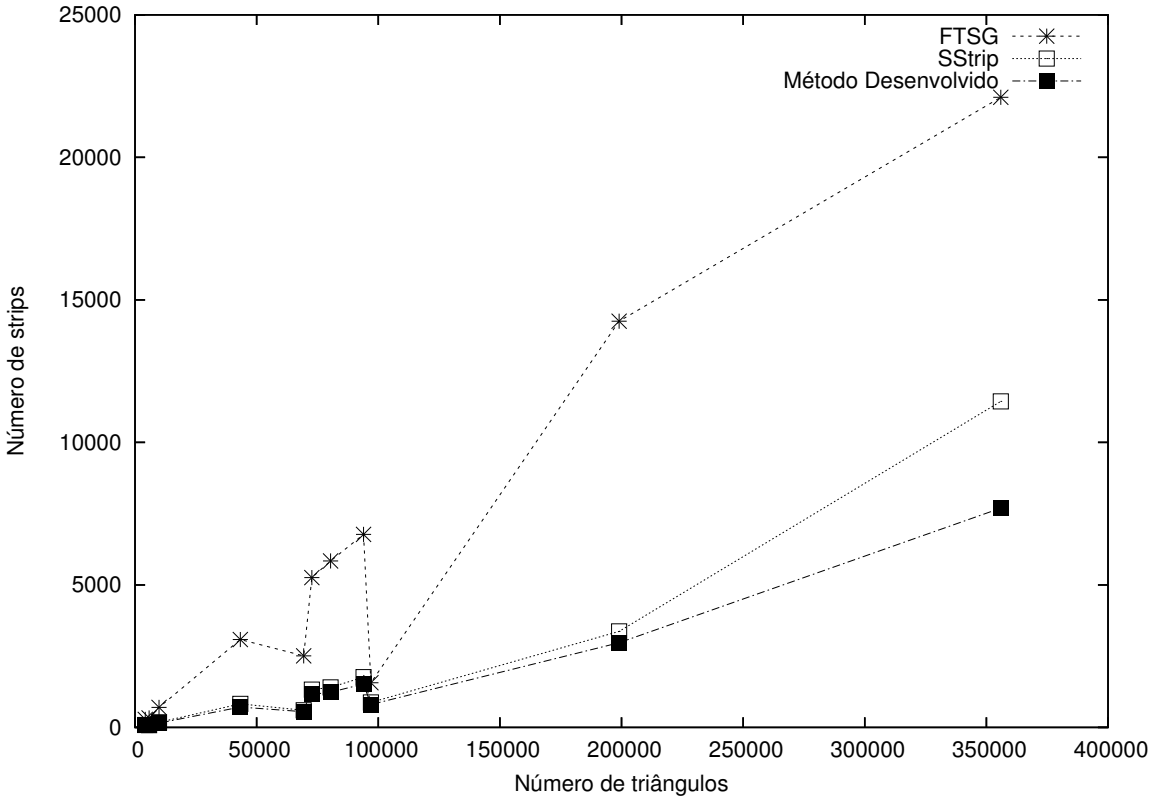


Figura 4.3: Gráfico do total de strips × total de triângulos dos métodos analisados.

a apresentar uma taxa de compactação menor se comparado a superfícies com várias imperfeições, geralmente compostas por *strips* que tendem a formar ciclos, como no modelo *jackie* da figura 4.1.

O método desenvolvido é composto por três etapas. A primeira etapa consiste na redução do número de *strips* pela eliminação de nodos que apresentem grau 3, quando possível. A segunda, executada no passo de extração das *strips*, consiste em separar nodos vizinhos ao nodo de grau 3 analisado, excluindo arestas desnecessárias do grafo. A terceira etapa consiste na eliminação da restrição em se formar *strips* circulares.

A tabela 4.3 mostra os valores de compactação em relação ao método *SStrip*, obtidos pela combinação das etapas anteriormente apresentadas.

As etapas são representadas por letras maiúsculas onde:

- R = redução dos nodos de grau 3.
- C = permissão da formação de *strips* circulares na fase de criação da árvore.
- P = pré-extração, separação dos vizinhos dos nodos de grau 3 que possuam o mesmo grau que o nodo analisado.

Modelo	Triângulos	Vértices	Quantidade de *Strips*		
			Mét. FTSG	Mét. SStrip	Mét. Desenv.
foot	4204	2154	282	82	67
cow	5804	2904	333	80	65
lena	43285	21737	3085	823	714
horse	96966	48485	1567	878	802
bunny	69356	34834	2508	599	549
epcot	1536	770	124	54	39
hawaii	19602	10000	99	99	99
jackie	355944	177974	22108	11441	7694
cl9852	9852	5000	700	182	153
roseburg	80423	40343	5841	1405	1236
emorypeak	72712	36500	5258	1328	1158
grand canyon	93980	47088	6775	1763	1519
lake champlain	198996	100000	14254	3372	2971

Tabela 4.1: Características dos modelos e quantidades de *strips*.

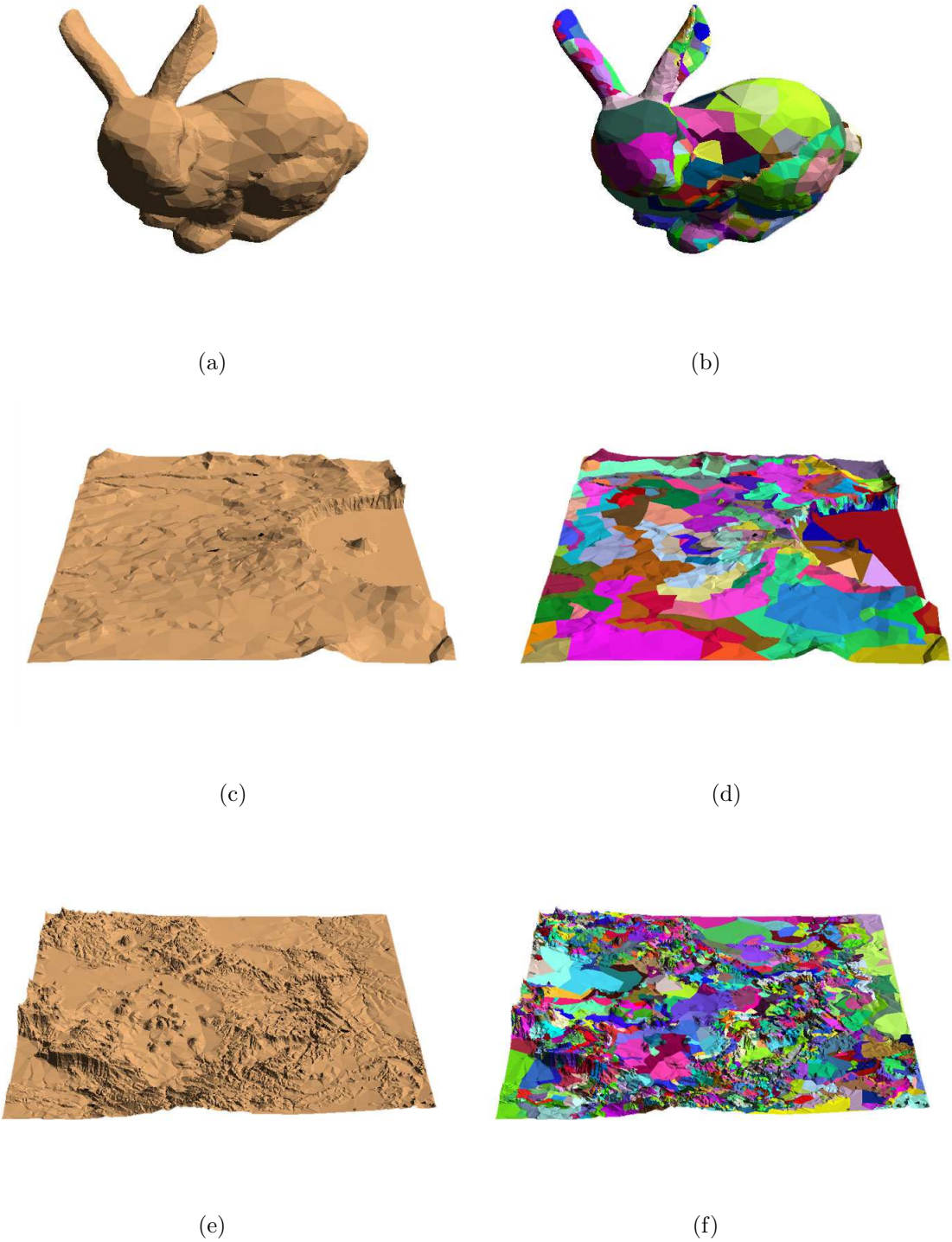
Modelo	Sem redução		Com redução	
	Total de *Strips*	*Strip* tam. 1	Total de *Strips*	*Strip* tam. 1
bunny4	206	5	206	1
cl9852	153	1	153	0
cow	65	2	65	0
emorypeak	1158	39	1160	5
grand canyon	185	7	185	0
canyon	1519	44	1520	7
lena	714	26	714	3
foot	67	1	67	0
rl3955	71	3	71	1
rl39788	690	22	690	3
roseburg	1236	38	1237	4

Tabela 4.2: Aplicação da técnica de redução de *strips* de tamanho 1 para o método desenvolvido.

Modelo	R( ) C( ) P( )	R(×) C( ) P( )	R(×) C( ) P(×)	R(×) C(×) P(×)
bunny4	03.84%	05.98%	07.26%	11.96%
cl9852	05.49%	10.43%	11.53%	15.93%
cow	02.50%	08.75%	08.75%	18.75%
emorypeak	04.74%	09.11%	09.93%	12.65%
grand canyon	03.77%	08.49%	09.43%	12.73%
canyon	05.84%	09.64%	10.49%	13.78%
lena	04.25%	08.26%	08.99%	13.24%
foot	06.09%	09.75%	10.97%	18.29%
rl3955	03.75%	08.75%	10.00%	11.25%
rl39788	06.43%	09.96%	10.71%	12.98%
roseburg	04.83%	08.82%	09.18%	11.95%

Tabela 4.3: Taxas de compactação, relativas ao método Sstrip, obtidas com a aplicação sucessiva das diferentes técnicas desenvolvidas.

Nas figuras 4.4 a 4.6 são apresentados alguns modelos testados juntamente com as *strips* geradas pelo método desenvolvido.

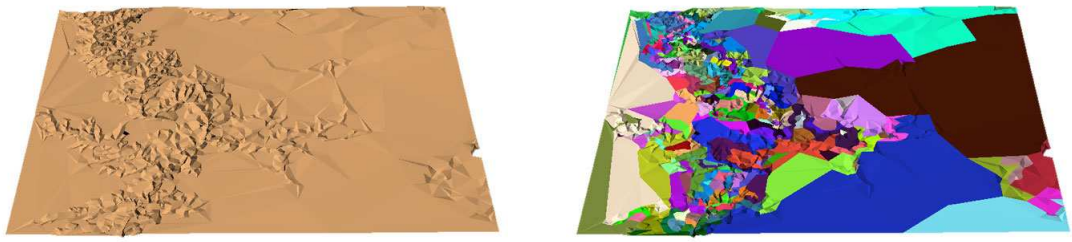


(a) (b)

(c) (d)

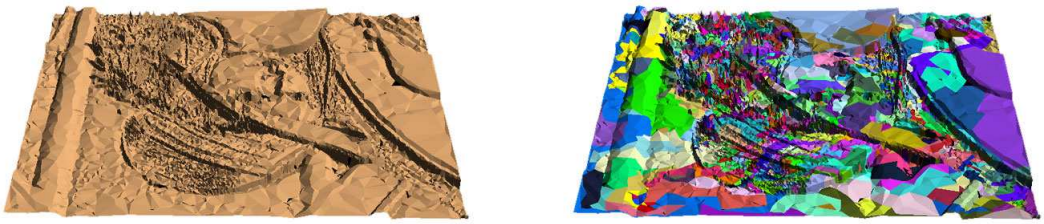
(e) (f)

Figura 4.4: Modelos com suas respectivas *strips*. (a,b) *bunny4*; (c,d) *cl9852*; (e,f) *emory-peak*.



(a) (b)

(c) (d)

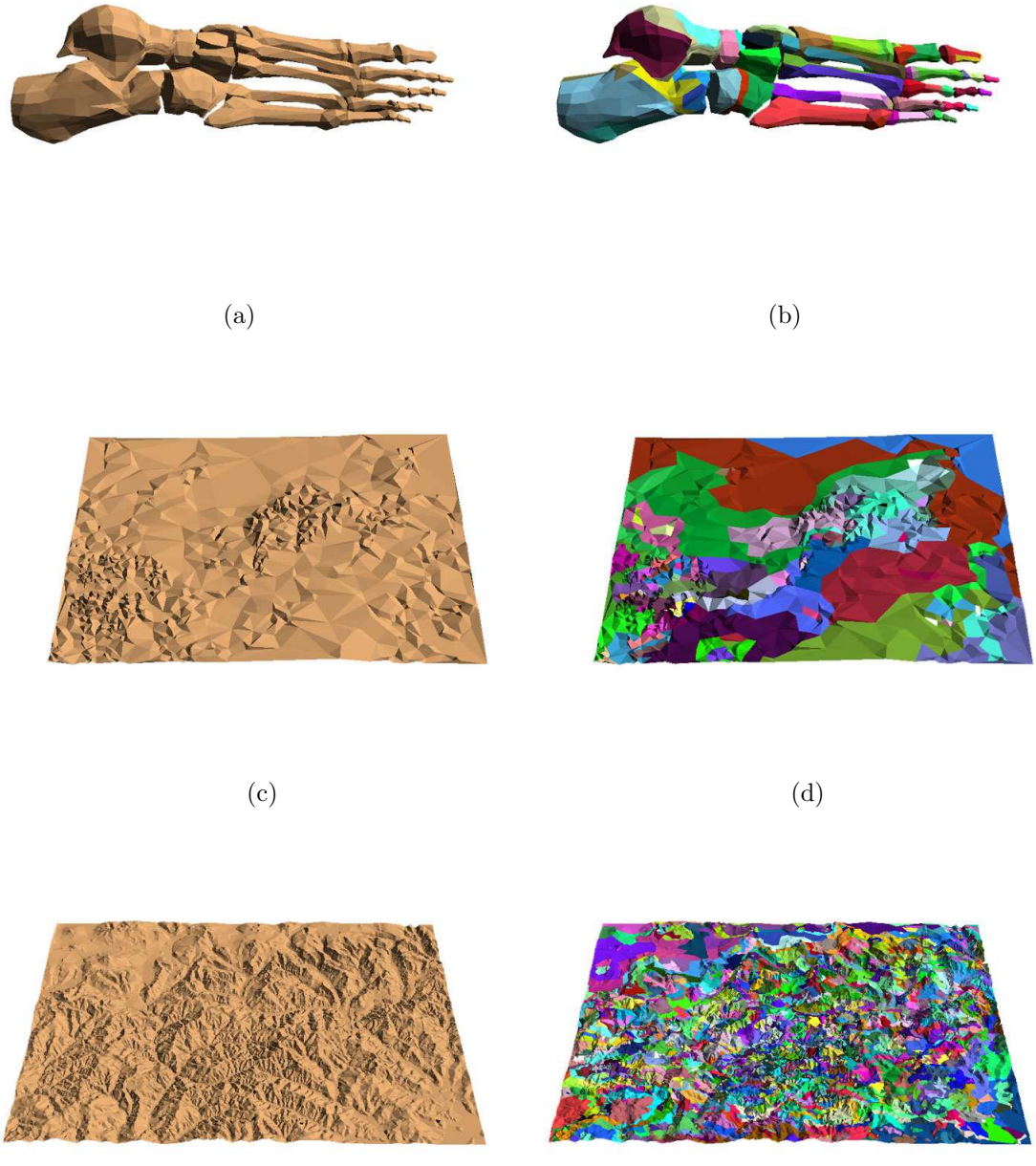


(e) (f)

Figura 4.5: Modelos com suas respectivas *strips*. (a,b) *grand canyon*; (c,d) *jackie*; (e,f) *emorypeak*.

4.2 Discussão dos Resultados

Com base nos experimentos realizados em diversas malhas triangulares, vale ressaltar alguns aspectos relevantes. Mesmo sendo executado em modelos com uma baixa quanti-



(a) (b)

(c) (d)

(e) (f)

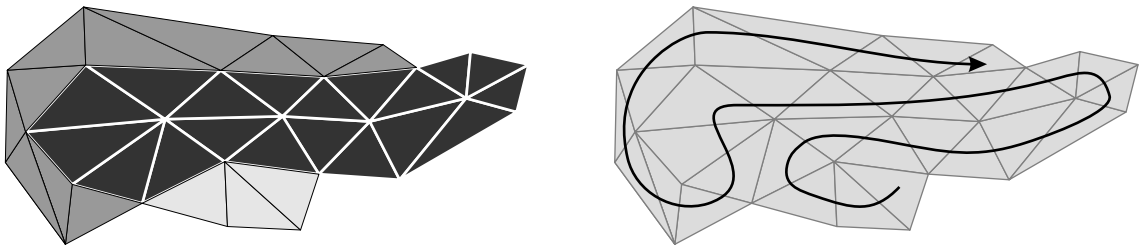
Figura 4.6: Modelos com suas respectivas *strips*. (a,b) *foot*; (b,c) *rl3955*; (d,e) *roseburg*.

dade de triângulos, representando uma amostra pequena de possibilidades de redução, o algoritmo apresentou uma taxa de compactação significativa.

Em amostras com uma quantidade maior de triângulos, a taxa de compactação aumentou consideravelmente, como se pode notar na tabela 4.4. Esta tabela inclui os resultados

obtidos pela aplicação do algoritmo desenvolvido em diferentes resoluções do modelo *jackie*, mostrado na figura 4.1.

Presume-se que este comportamento seja devido à heurística global aplicada pelo algoritmo. Diferente do método desenvolvido, o *SStrip* usa uma heurística local baseada em um algoritmo guloso, no qual, uma vez que um triângulo é adicionado à *strip*, ele não é mais removido. Existem concatenações de *strips* no método *SStrip*, porém elas são aplicadas apenas às extremidades das *strips* se estas puderem formar uma única *strip*. Assim como no método de tunelamento [31], o *SStrip* proíbe a formação de *strips* infinitas. Entretanto essas *strips*, também chamadas de *strips* circulares neste trabalho, tornaram-se indispensáveis para os ganhos obtidos pela aplicação do método desenvolvido. No fim do processo não são geradas *strips* infinitas. O propósito das *strips* infinitas se deve à possibilidade de mudança dos triângulos final e inicial da *strip*. Como descrito no capítulo 3, isto aumenta as possibilidades de redução da *strip* de 1 nodo até o comprimento da *strip* circular. No capítulo 3, foi mostrada a redução de uma malha, mostrada aqui na figura 4.7, pelo uso de uma *strip* circular.



(a) (b)

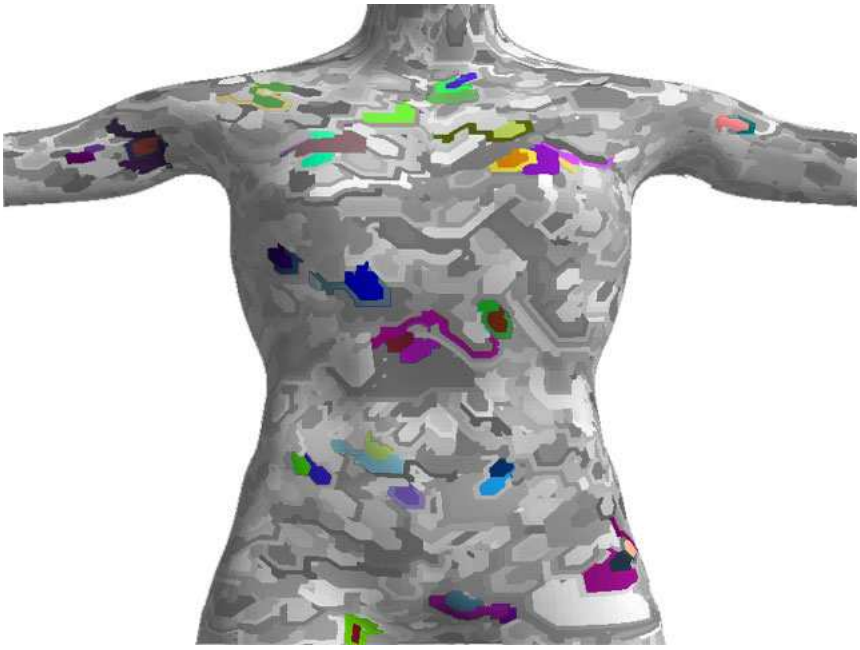
Figura 4.7: Parte de uma malha reduzida a uma única *strip*.

Este tipo de padrão, consistindo em uma *strip* circular envolvida por uma ou mais *strips* não circulares, pode ser observado em vários trechos do modelo *jackie*, usado como entrada para o método desenvolvido (figura 4.8(a)). A redução, nesses casos, é feita trocando-se os triângulos de início e fim da *strip* circular de modo que eles possam ser concatenados às *strips* vizinhas, propiciando a redução.

Entretanto, com o intuito de diminuir a quantidade de nodos analisados pelo algoritmo, somente foi usada a redução caso uma *strip* não circular fosse conectada à outra e não

percorrida a *strip* inteira à procura de padrões de redução como estes. Dessa maneira, pode-se manter controlada a complexidade do algoritmo.

(a)



(b)

Figura 4.8: (a) Modelo *jackie* e (b) padrão que favorece a redução da quantidade de *strips* da malha.

Devido à necessidade de informações adicionais sobre a topologia da malha triangular,

Modelo	Triângulos	Método SStrip	Método Desenvolvido	Redução (%)
jackie010	35594	484	425	12.19
jackie030	106782	1325	1161	12.37
jackie050	177972	2578	2062	20.01
jackie070	249160	5510	3948	28.34
jackie090	320348	8942	6187	30.80
jackie100	355944	11441	7692	32.76

Tabela 4.4: Taxas de compactação em relação ao método *SStrip*, quando aplicado a diferentes resoluções do modelo *jackie*.

o algoritmo desenvolvido consome uma maior quantidade de memória. Ele possui sua própria estrutura de dados, visto que foi projetado para ser aplicado a qualquer conjunto de *strips* a ser otimizado.

Além disso, como o algoritmo requer uma análise das extremidades das *strips* e de apenas seus vizinhos, as informações necessárias em memória para acesso aleatório podem ser significativamente reduzidas.

Como a implementação do algoritmo proposto neste trabalho foi incorporada ao *SStrip*, algumas informações passaram a estar presentes nas duas estruturas, tornando-se redundantes. Houve um acréscimo de tempo no que se refere à extração destas informações sobre os triângulos das malhas, sendo que elas precisam fornecer dados à duas estruturas distintas.

O principal propósito deste trabalho foi desenvolver um método eficiente para compactação de *strips*, suficientemente genérico para ser aplicado a qualquer conjunto de entrada. Embora o consumo dos recursos de memória e tempo computacional não tenham sido o foco do trabalho, o algoritmo pode ser otimizado de modo a melhor utilizá-los.

Capítulo 5

Conclusões e Trabalhos Futuros

A geração eficiente de *strips* triangulares é uma tarefa de grande importância na área da computação gráfica e geometria computacional. Para se transmitir uma grande quantidade de dados, geralmente presentes nos modelos 3D, é necessário um processo eficiente de compactação. Diferente de outros tipos de compactação, como a compressão de imagens, a compactação de modelos 3D tem ainda a necessidade de uma descompactação muito mais rápida que a compressão. Esta descompactação é feita em nível de hardware e, devido a isto, sua implementação deve ser possível de ser realizada nos hardwares existentes atualmente.

Este trabalho descreveu as principais técnicas para codificação de malhas poligonais e propôs um método para geração de *strips* triangulares baseado na manipulação de uma árvore obtida a partir das *strips* geradas por outro algoritmo de compactação, o *SStrip*. As características do algoritmo desenvolvido proporcionam uma significativa melhora na taxa final de compactação.

Com base nos testes realizados em diferentes modelos triangulados, sugere-se, como trabalho futuro, a eliminação de informações redundantes contidas na estrutura para armazenar a topologia da malha. Esta redundância deve-se ao fato de que o algoritmo desenvolvido pode ser aplicado a qualquer conjunto de *strips* a ser otimizado.

Esta melhora consiste na separação do método desenvolvido em duas funcionalidades principais. A primeira destas funcionalidades consiste em um algoritmo exclusivamente de otimização, com a função de reduzir o número de *strips* a serem utilizadas. A segunda

consiste em um método completo de compactação da geometria de malhas poligonais, utilizando uma única estrutura de dados otimizada, de modo a reduzir a quantidade de memória necessária para a execução do algoritmo.

Referências Bibliográficas

[1] Akeley, K., Haeberli, P. e Burns, D. `tomesh.c`: Program on SGI Developer's Toolbox CD, 1990.

[2] Arkin, E., Held, M., Skiena, S. e Mitchell., J. Hamiltonian triangulations for fast rendering. *ACM Transactions on Graphics 15*, 2 (Abril 1996), 141–152.

[3] Bar-Yehuda, R. e Gotsman, C. Time/space tradeoffs for polygon mesh rendering. *ACM Transactions on Graphics 15*, 2 (1996), 141–152.

[4] Belmonte, O., Ribelles, J., Remolar, I. e Chover, M. Searching triangle strips guided by simplification criterion. In *WSCG 2001 Conference Proceedings* (2001), V. Skala, Ed.

[5] Cassidy, R., Gregg, E., Reeves, R. e Turmelle, J. *IGL: The Graphics Library for the i860*, 1991.

[6] Chow, M. Optimized geometry compression for real-time rendering. In *Proceedings of IEEE Visualization'97* (1997), pp. 347–354.

[7] da Silva, M., van Kaick, O. e Pedrini, H. Fast mesh rendering through efficient triangle strip generation. *Journal of WSCG 10*, 1 (Fevereiro 2002), 127–134.

[8] de Floriani, L., Magillo, P. e Puppo, E. Efficient implementation of multi-triangulation. In *Proceedings Visualization'98* (Outubro 1998), pp. 43–50.

[9] Deering, M. Geometry compression. In *SIGGRAPH'95 Conference Proceedings* (Los Angeles, Califórnia, Estados Unidos, 1995), Annual Conference Series, pp. 13–20.

[10] El-Sana, J., Azanli, E. e Varshney, A. Skip strips: Maintaining triangle strips for view-dependent rendering. In *Proceedings of IEEE Visualization* (1999), pp. 131–138.

[11] El-Sana, J. e Varshney, A. Generalized view-dependent simplification. In *Eurographics'99* (Milão, Itália, 1999).

[12] Evans, F. `http://www.cs.sunysb.edu/~stripe/`: Stripe, 1998.

[13] Evans, F., Skiena, S. e Varshney, A. Optimizing triangle strips for fast rendering. In *Proceedings of IEEE Visualization'96* (1996), pp. 319–326.

[14] Funkhouser, T. e Sequin, C. Adaptive display algorithm for interactive frame rate during visualization of complex virtual environments. In *Computer Graphics, SIGGRAPH'93 Proceedings* (Agosto 1993), pp. 247–254.

[15] Garland, M. e Heckbert., P. Surface simplification using quadric error metrics. In *Computer Graphics* (1997), pp. 209–216.

[16] Georgia Institute of Technology. Large Geometric Models Archive. `http://www.cc.gatech.edu/projects/largemodels/`.

[17] Gross, M., Gatti, R. e Staadt, O. Fast multiresolution surface meshing. In *IEEE Visualization '95 Proceedings* (1995), G. M. Nielson and D. Silver,editors, pp. 135–142.

[18] Guéziec, A., Taubin, G., Horn, B. e Lazarus, F. A framework for streaming geometry in VRML. *IEEE Computer Graphics and Applications 19*, 2 (1999), 68–78.

[19] Held, M. Efficient and reliable triangulation of polygons. In *Proc. Comput. Graphics Internat.* (1998), pp. 633–643.

[20] Hoppe, H. View-dependent refinement of progressive meshes. In *Computer Graphics, SIGGRAPH'97 Proceedings* (Los Angeles, Califórnia, Estados Unidos, Agosto 1997), pp. 189–197.

[21] Hoppe, H., Derose, T., Duchamp, T., MCDonald, J. e Stuetzle, W. Surface reconstruction from unorganized points. In *Computer Graphics, SIGGRAPH'92 Conference Proceedings* (Julho 1992), pp. 71–78.

[22] ISO. Information Processing Systems - Computer Graphics - Programmer's Hierarchical Interactive Graphics System (PHIGS). Technical Report ISO/IEC 9592, International Organization of Standardization, 1989.

[23] Lindstrom, P., Koller, D., Ribarsky, W., Hughes, L., Faust, N. e Turner, G. Real-time, continuous level of detail rendering of height fields. In *Computer Graphics, SIGGRAPH'96 Proceedings* (1996), ACM SIGGRAPH, pp. 109–118.

[24] Luebke, D. e Erikson, C. View-dependent simplification of arbitrary polygonal environments. In *SIGGRAPH'97 Conference Proceedings* (Agosto 1997), ACMPress, pp. 198–208.

[25] Microsoft DirectX. Microsoft. `http://www.microsoft.com/windows/directx/`.

[26] Nasa's Planetary Data System. NASA. `http://pds.jpl.nasa.gov/`.

[27] Neider, J., Davis, T. e Woo, M. *OpenGL Programming Guide: The Official Guide to Learning OpenGL*. Addison-Wesley, New Jersey, 1993.

[28] Pugh, W. Skip lists: A probabilistic alternative to balanced trees. In *Workshop on Algorithms and Data Structures* (1989), pp. 437–449.

[29] Shewchuk, J. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In *In Lecture Notes in Computer Science* (1996), vol. 1148, pp. 203–233.

[30] Stanford Computer Graphics Laboratory. The Stanford 3D Scanning Repository. `http://graphics.stanford.edu/data/3Dscanrep/`.

[31] Stewart, A. J. Tunneling for triangle strips in continuous level-of-detail meshes. *Graphics Interface* (Junho 2001), 91–100.

[32] Taubin, G. A signal processing approach to fair surface design. In *Computer Graphics, SIGGRAPH'95 Proceedings* (Agosto 1995), pp. 351–358.

[33] Taubin, G. e Rossignac, J. Geometry compression through topological surgery. *ACM Transactions on Graphics 17*, 2 (1998), 84–115.

[34] Taubin, G., Zhang, T. e Golub, G. Optimal surface smoothing as filter design. In *Proceedings of the European Conference on Computer Vision* (1996), pp. 283–292.

[35] United States Geological Survey. USGS. `http://www.usgs.gov/`.

[36] van Kaick, O., da Silva, M. e Pedrini, H. Efficient generation of triangle strips from triangulated meshes. *Journal of WSCG 12*, 1–3 (Fevereiro 2004), 475–481.

[37] Vanecek, P. Triangle Strips for Fast Rendering. Technical Report No DCSE/TR-224-05, Pilsen, Czech Republic, Abril 2004.

[38] Velho, L., Figueiredo, L. e Gomes, J. Hierarchical generalized triangle strips. *The Visual Computer 15*, 1 (1999), 21–35.

[39] Wernecke, J. *The Inventor Mentor*. Addison-Wesley, New Jersey, 1994.

[40] Xia, J., El-Sana, J. e Varshney, A. Adaptive realtime level-of-detail-based rendering for polygonal models. *IEEE Transactions on Visualization and Computer Graphics* (Junho 1997), 171–183.

[41] Xia, J. C., El-Sana, J. e Varshney, A. Adaptive real-time level-of-detail-based rendering for polygonal models. *IEEE Transactions on Visualization and Computer Graphics 3*, 2 (/1997), 171–183.

[42] Xiang, X., Held, M. e Mitchell, J. Fast and effective stripification of polygonal surface models. In *Proceedings of ACM Symposium on Interactive 3D Graphics* (1999). http://www.ams.sunysb.edu/∼xxiang/strip.html.