Universidade Federal de Alagoas

Instituto de Matemática

Programa de Pós-Graduação em Matemática

José Fábio Boia Porto

Visualização de Malhas com Adaptação de Resolução e Textura Dependentes do Observador

Maceió - Alagoas

2009

José Fábio Boia Porto

Visualização de Malhas com Adaptação de Resolução e Textura Dependentes do Observador

Dissertação de Mestrado em Matemática, na área de Computação Gráfica, submetida em 29 maio de 2009 à Banca Examinadora, designada pelo colegiado do Programa de Pós-Graduação em Matemática da Universidade Federal de Alagoas, como parte dos requisitos à obtenção do grau de mestre em Matemática.

Orientador: Prof. Dr. Adelailson Peixoto

Maceió - Alagoas

2009

Catalogação na fonte
Universidade Federal de Alagoas
Biblioteca Central
Divisão de Tratamento Técnico
Bibliotecária Responsável: Helena Cristina Pimentel do Vale

P853v Porto, José Fábio Boia.
Visualização de malhas com adaptação de resolução e textura dependentes do observador / José Fábio Boia, 2009.
66 f. : il.

Orientador: Adelailson Peixoto da Silva.
Dissertação (mestrado em Matemática) – Universidade Federal de Alagoas. Instituto de Matemática. Maceió, 2009.

Bibliografia: f. 63-66.

1. Malhas adaptativas. 2. Reconhecimento visual de textura. 3. Visualização volumétrica. 4. Processamento de dados em tempo real. 5. Computação gráfica. I. Título.

CDU: 519.6

José Fábio Boia Porto

Visualização de Malhas com Adaptação de Resolução

e

Textura Dependentes do Observador

Dissertação de Mestrado em Matemática, na área de Computação Gráfica, submetida em 29 maio de 2009 à Banca Examinadora, designada pelo colegiado do Programa de Pós-Graduação em Matemática da Universidade Federal de Alagoas, como parte dos requisitos à obtenção do grau de mestre em Matemática.

Prof. Dr. Adelailson Peixoto

UFAL

(Orientador)

Prof. Dr. Dimas Martinez Morera

UFAL

Prof. Dr. Perfilino Eugênio Ferreira Júnior

UFBA

A Deus e Sra das Graças

e à Minha Esposa Lu.

Agradecimentos

Agradecer a todos que ajudaram a construir esta dissertação não é tarefa fácil. O primeiro problema que se coloca para o agradecimento seletivo não é decidir quem incluir, mas decidir quem não mencionar. Então, a meus amigos que, de uma forma ou de outra, contribuíram com sua amizade e com sugestões efetivas para a realização deste trabalho, gostaria de expressar minha profunda gratidão. Se devo ser seletivo, então é melhor começar agradecendo a Deus, que me deu força para seguir nesta caminhada, guiando meus passos dia após dia, *muito obrigado Senhor*!

Agradeço, de forma muito carinhosa, a atuação de minha esposa no período de construção deste trabalho. Sua paciência infinita e sua crença absoluta na capacidade de realização a mim atribuída foram, indubitavelmente, os elementos propulsores desta dissertação. Ela soube compreender, como ninguém, a fase pela qual eu estava passando. Sempre tentou entender minhas dificuldades e minhas ausências, procurando se aproximar de mim através da própria dissertação. Agradeço-lhe, carinhosamente, por tudo isto.

Meus pais, irmãos, cunhados e demais familiares, agradeço pelo apoio, por acreditarem em mim e por compreenderem minha ausência em tantos momentos. Sempre estive com vocês, mesmo não estando tão perto quanto gostaria. Minha seleção, no âmbito acadêmico, deve iniciar com o Prof° Adelailson Peixoto. Agradeço a consideração de ter aceito a orientação de minha dissertação, na esperança de retribuir, com a seriedade de meu trabalho, a confiança em mim depositada. Sigo agradecendo a dois grupos de amigos: os da computação gráfica que colaboraram tirando minhas dúvidas, principalmente em questões referentes a termos técnicos e implementações; e os do mestrado da matemática, que mesmo tendo nos afastado um pouco por diferença de áreas, sempre torceram e confiaram em mim, sendo muito importantes na realização deste trabalho.

Incluo, de forma especial de agradecimento aos colegas e amigos: *Alan , Alex Santana, Allyson Cabral, Arlyson Alves, Carlos Alberto, Douglas Sedrin, Erikson Fonseca, Everson Feitosa, Leonardo Carvalho, Márcia, Michel Alves, Priscila Ramos, Renata Thomaz, Silvinha e Thales Miranda.*

Agradeço ainda, a todos os professores do Programa de Pós-Graduação de Matemática da UFAL, que além de contribuírem com minha formação sempre me respeitaram e me atenderam da melhor força possível. Quero agradecer em especial ao professor Ádan Corcho (Coordenador do Programa).

Foi sorte estas pessoas terem cruzado meu caminho acadêmico nesta etapa de conclusão de mestrado. Suas idéias permearam meu trabalho e permaneceram eternamente em minha memória.

Não poderia deixar de agradecer a **Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - CAPES** pelo apoio financeiro durante esses dois anos, que para mim foi essencial para um bom desenvolvimento do curso. Sei que cometo injustiça, pois não conseguirei lembrar de todos os que me foram de valia nestes dois anos de esforço, mas meu muito obrigado a todos.

Resumo

Este trabalho trata do problema de visualização de malhas triangulares de modo que a geometria e a textura das mesmas possam ser calculadas adaptativamente de acordo com a posição do orientador. Como apenas as informações necessárias de malha tendem a permanecer, as principais aplicações envolvidas estão relacionadas à visualização de malhas em tempo real, as quais têm sido de grande interesse de pesquisa em diversas áreas científicas. Na adaptação da geometria da malha, sua resolução vai sendo localmente alterada através de inserção/remoção de vértices, arestas e faces, de acordo com a posição do observador, de modo que o número de polígonos preserve uma forma visual suave e bem próxima da malha original. Na adaptação da textura, o mapeamento de textura associado à malha vai sendo adaptado às mudanças de sua resolução de modo a reduzir ou eliminar as distorções de textura resultantes do colapso de vértices, arestas e faces.

Abstract

This work has the goal of explore the triangulated mesh visualization problem in order that geometry and texture information could be computed adaptively according to the view position. As only the essential information from the mesh is going to persist, the main applications involved are linked with real-time mesh visualization which concerns a large number of scientific areas. To adapt the mesh geometry, its resolution is being locally changed through the insertion/removal of vertices, edges and faces, view-dependent considering the number of polygons to preserve smooth forms near to the original mesh. In the texture adaptation, the texture map association of the mesh is being adapted to the resolution changes minimizing or eliminating the texture distortion resulting from the collapse of vertices, edges and faces.

Lista de Figuras

Sumário

Capítulo 1

Introdução

Apesar de existirem abordagens alternativas de representação e renderização de superfícies, as malhas poligonais ainda são consideradas como padrão para a computação gráfica. A resolução de um objeto, isto é, a quantidade de informação armazenada por unidade de comprimento, mais adequada para uma malha depende das aplicações envolvidas. Se por um lado, malhas de baixa resolução podem comprometer a qualidade dos resultados nas aplicações, malhas de alta resolução podem dificultar o processamento nas aplicações em tempo real. Mesmo com o desenvolvimento de processadores e o aumento da capacidade das memórias, cresce também, na mesma ordem, a necessidade de processamento e o consumo de memória, impondo a necessidade do desenvolvimento de métodos e algoritmos capazes de reduzir o tempo de processamento.

A visualização de superfícies em tempo real é, por exemplo, de suma importância, principalmente para aplicações interativas. Para a visualização podemos ter como entrada uma malha de *baixa resolução* ou uma malha de *alta resolução*. Em ambos os casos, para obtermos uma boa visualização são necessários trata-

mentos na malha. No primeiro caso, a baixa resolução pode comprometer a qualidade do contorno do objeto, o qual pode apresentar "bicos", por conta do baixo números de triângulos, causando um desconforto visual. Chegamos assim a um dos principais problemas impostos por essa representação.

Já no caso das malhas de alta resolução, apesar de o contorno geralmente se apresentar suave, é difícil tratar de aplicações em tempo real devido ao grande número de vértices e faces a serem processados. Neste caso é necessário simplificar a malha, tendo como desafio manter a qualidade de algumas regiões, como as situadas no contorno do objeto.

O contorno do objeto do plano de projeção depende da posição do observador no momento que o objeto é visualizado. Muitos estudos ([3] e [8]) já foram feitos no tratamento de malhas para visualização em tempo real, mas poucos [1] levam em conta a posição do observador. Nos dois problemas citados acima, a posição do observador pode nos ajudar a tomar decisões para melhorarmos a visualização em tempo real, pois se o obervador muda, então mudam também várias informações do objeto, como o contorno.

Outra informação importante a ser considerada na visualização de objetos são atributos como cor e textura. Na tentativa de obtermos uma boa visualização em tempo real, preservando fielmente as características da malha original é fundamental que o mapeamento de textura associado ao objeto também seja tratado durante a adaptação da malha. Este processo ajuda a manter as características originais da cena.

1.1 Objetivo

O principal objetivo deste trabalho é o tratamento da geometria e conectividade de uma malha através de ajustes locais da sua resolução, durante o processo de visualização. Este ajuste da resolução é feito de acordo com a posição do observador visando manter a qualidade da malha e a visibilidade do seu processamento em tempo real. Além do ajuste da resolução, também tratamos a adaptação da textura dos objetos durante a visualização.

Em malhas de baixa resolução tentamos melhorar a qualidade da malha através de um processo de suavização do contorno. Em malhas de alta resolução discutimos técnicas de simplificação, de modo que apenas informações que não comprometam sua qualidade visual sejam eliminadas.

1.2 Aplicações

Hoje são muitas as áreas que buscam a visualização de objetos em tempo real, para os diversos fins. Isso porque é bem mais barato e prático fazê-las no computador. Tomemos como exemplo uma situação da engenharia de aviação. Ima-gine fazer testes de aerodinâmica nas asas de um avião, usando para cada teste uma nova asa. É óbvio que testes como estes ficam muito mais baratos se feitos no computador.

Estudos de problemas de visualização trazem melhorias, principalmente, para simulações interativas em tempo real que são necessárias em diversas áreas científicas,

tais como: engenharia, medicina, geologia, modelagem geométrica, visão computacional [28] e *etc*.

1.3 Estrutura do Trabalho

Este trabalho está organizado com a seguinte estrutura.

No *capítulo 2* apresentamos conceitos básicos, necessários para uma melhor compreensão do trabalho. No *capítulo 3* descrevemos os métodos de adaptação da geometria. Em seguida, no *capítulo 4* tratamos o desafio de adaptação da textura, quando a malha muda sua resolução. Por fim, apresentamos nos *capítulos 5* e no *capítulo 6*, respectivamente, os resultados obtidos e conclusões e trabalhos futuros.

Capítulo 2

Conceitos Gerais

Neste capítulo apresentamos um resumo dos conceitos e definições necessários à compreensão desta dissertação, tais como: malhas, contorno de superfícies e mapeamento de textura.

2.1 Malhas

Mesmo existindo na computação gráfica diversas abordagens alternativas de representação e renderização de objetos, são mais frequentes aquelas que usam malhas poli-gonais, em geral triangulares.

Podemos definir uma *malha* como um complexo $M = (V, A, F)$ onde V, A e F são, respectivamente, conjuntos de vértices $v_i \in V$, arestas $[v_i, v_j] \in A$ e faces $(v_i, v_j, ..., v_k) \in F$ [23]. Toda malha possui uma *resolução* que pode ser entendida como a quantidade de vértices por unidade de área.

Uma *malha triangular* é descrita por uma triangulação. Segundo [14] uma *triangulação 2D* em $\mathbb{R}^3$ é uma coleção $\mathcal{T} = T_i$ de triângulos em $\mathbb{R}^3$ tal que dados

dois triângulos $T_i,\ T_j \in \mathcal{T}$, se $T_i \cap T_j \neq \{\}$ então

- $T_i \cap T_j$ é um vértice, ou
- $T_i \cap T_j$ é uma aresta.

Algumas aplicações, como as interativas, requerem um processamento rápido da malha, já que a resposta às interações do usuário deve ser imediata. Estas aplicações, em geral, têm como entrada uma malha de baixa resolução. Neste caso, para obtermos uma boa visualização é natural acrescentarmos detalhes até atingirmos um bom nível visual. Esse processo de aumentar a resolução da malha é conhecido como *refinamento*, o qual inclui na malha pontos internos convenientemente distribuídos de modo a manter a topologia.

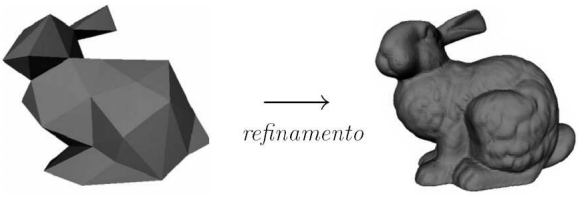


Figura 2.1: Processo de simplificação e refinamento (imagem adaptada de [13])

Malhas de alta resolução também podem ser tomadas como entrada para renderização em tempo real. Neste caso, a grande quantidade de informações transmitida ao hardware gráfico pode causar congestionamento. Assim, pode ser necessário eliminar informações excessivas da malha, através de um processo de *simplificação*, limitando a quantidade de descrições geométricas a serem transmitidas. Nesta etapa, são aplicados esquemas que eliminam vértices, arestas e faces [21]. A figura 2.2 ilustra, genericamente, estes dois processos

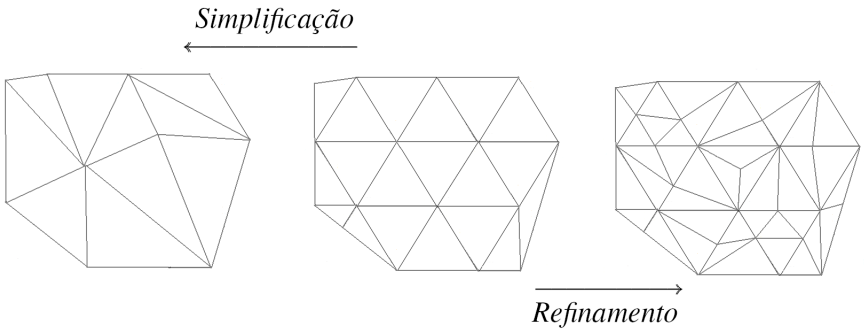


Figura 2.2: Simplificação e refinamento

Existem diversas formas para refinar uma malha. Uma das técnicas mais simples de refinamento insere um ponto no baricentro das faces triangulares e faz uma triangulação local com o novo ponto. Este processo, visto na primeira ilustração da figura 2.3, é de fácil implementação, mas em normalmente não gera um malha com boa qualidade. Em [20] é apresentado um algoritmo de refinamento adaptativo com base em um *erro*, neste caso, o refinamento passa por duas etapas: primeiro, a subdivisão dos triangulos e, em seguida, a adaptação da textura. As técnicas de refinamento apresentadas por [3] têm como entrada uma triangulação de Dalaunay, e retornam uma malha de boa qualidade.

Na figura 2.3 são mostrados esquemas para refinamento de malhas onde os vértices cinza são os vértices adicionados não necessariamente pertencente ao mesmo plano da face original [18].

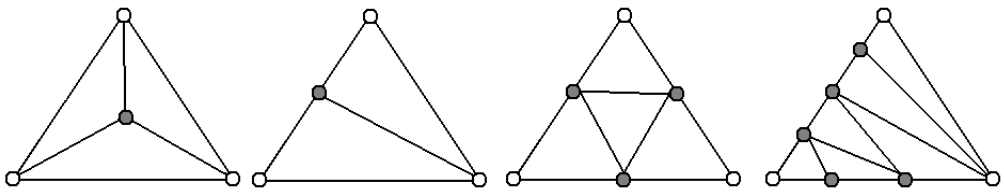


Figura 2.3: Os pontos cinza, não necessariamente no mesmo plano da região inicial, foram incluídos para o refinamento

São muitos os trabalhos publicados na última década que tratam o problema da simplificação das malhas. Em [24] e [25] são apresentados diversos métodos para simplificação de malhas. Em [1] é apresentado um algoritmo em nível de detalhes dependente do observador para malhas triangulares.

A representação de malhas em multiresolução (*figura 2.4*) permite que, tendo uma sequência de n operações de simplificação organizadas hierarquicamente, possamos partir de uma malha fina M^n para uma malha mais simples M^0, reduzindo a complexidade da malha. O caminho inverso também pode ser feito por meio de operações inversas, isto é, dada uma malha grosseira M^0 e i diferentes níveis de detalhes, podemos reconstruir M^i refinando a malha com a introdução de i vértices [22].

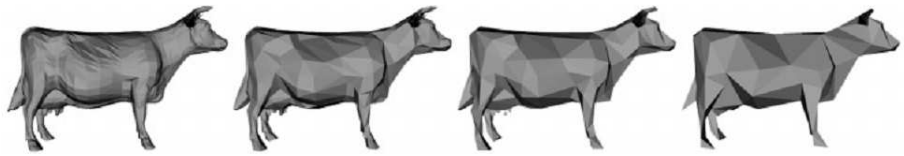


Figura 2.4: Representação de um objeto em multiresoluções [13]

Em [16], uma sequência de operações de *n colapsos de arestas (ecol)* são aplicadas para a simplificação de uma malha M^n para uma M^0, mantendo a mesma topologia, reduzindo, consideravelmente, o número de vértices. Dada a malha em baixa resolução M^0, com i diferentes níveis de detalhes de M^i podemos reconstruir a malha original aplicando o operador $vsplit$ (operador inverso do $ecol$).

2.1.1 Operadores para malhas

Esta seção trata de alguns operadores que tem sido tradicionalmente usados para a simplificação e refinamento de malhas. Cada um deles altera localmente a

resolução da malha.

Colapso de arestas e inserção de vértices

O operador *edge-collapse (ecol)* que faz o colapso de arestas foi proposto para a simplificação de malhas, inicialmente, em [17]. O operador colapsa uma aresta (v_i, v_j) em um vértice v_n (figura 2.5). Isto causa a remoção da aresta (v_i, v_j), assim como a remoção dos triângulos adjacentes a ela. Na figura 2.5 as faces A e B são removidas pela operação $ecol$. Topologicamente, este operador remove da malha original um vértice, três arestas e duas faces.

O operador inverso de um *ecol* é denominado *vertex-split (vsplit)*, que adiciona a aresta (v_i, v_j) no lugar de um vértice (*aumentando, consequentemente, um vértice*) e suas faces adjacentes, juntamente a duas outras arestas, devolvendo as três arestas retiradas pelo *Edge-collapse*. Desse modo, o operador *Edge-collapse* simplifica a malha, enquanto o operador *vsplit* adiciona mais detalhes a malha. A figura 2.5 ilustra os operadores *ecol* e *vsplit*

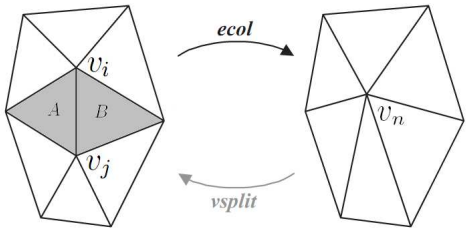


Figura 2.5: Operações *edge-collapse* e *vertex-split*

Existem duas variações do *edge-collapse* (*ver detalhes em [19]*), são elas: *half-edge collapse* e *full-edge collapse*. No *half-edge collapse* o vértice em que a aresta é colapsada é um de seus extremos, isto é, ou $v_n = v_i$ ou $v_n = v_j$. Já no

full-edge collapse o novo vértice v_n pode não pertencer ao conjunto dos vértices antes do colapso. Embora o *ecol* seja de fácil implementação, ele pode apresentar incosistências topológicas e inversão de faces (*figura* 2.6), por isso, quando o implementamos, devemos observar e tratar os diversos casos que possam trazer "danos" a malha.

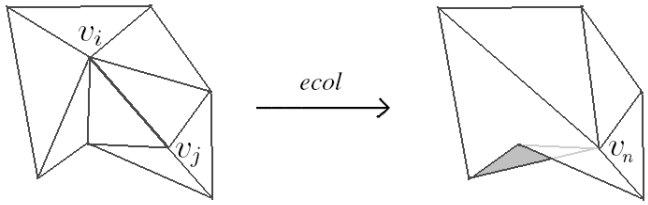


Figura 2.6: Aplicando $ecol$ de v_i a v_j obtemos uma dobradura

Colapso de faces

Um operador *face-collapse (fcol)* simplifica a malha pelo colapso de uma face v_i, v_j, v_k em um vértice v_n. As arestas que definem a vizinhança de v_n são as arestas vizinhas dos vértices v_i, v_j e v_k. O vértice v_n pode ser qualquer um dos vértices (v_i, v_j, v_k) ou um novo vértice calculado a partir de v_i, v_j e v_k. Veja a figura 2.7. Topologicamente este operador remove dois vértices, seis arestas e quatro faces.

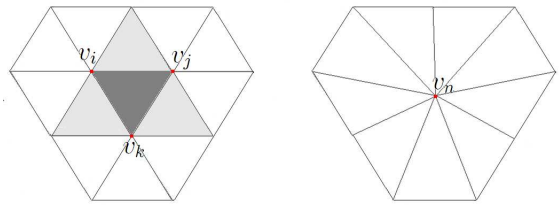


Figura 2.7: Colapso de uma face em um vértice

Remoção de vértices

O operador *removal-vertex (vrem)* remove um vértice e suas arestas e triângulos adjacentes deixando um buraco na malha. Em seguida, retriangulamos a região removida pelo *vrem* usando apenas os vértices do bordo. É fácil observar na figura 2.8 que o operador *vrem* executa uma operação semelhante a um colapso de uma aresta e, por isso, podemos considerar o operador *vrem* como uma generalização do operador *ecol*.

O número de possibilidades de retriangulação da região obtida na remoção do vértice é dado pela fórmula

$$C(i) = \frac{1}{i+1} * \binom{2i}{i},$$

onde $C(i)$ é o número de possibilidades para triangular um polígono convexo com $i + 2$ lados.

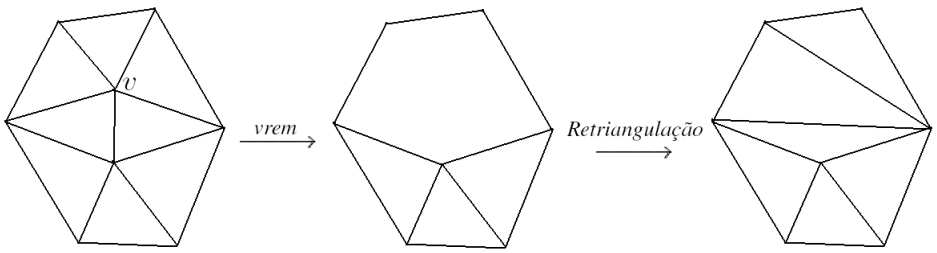


Figura 2.8: remoção do vértice v e retriangulação da região cinza

Mais detalhes desses e outros operadores podem ser consultados em [19].

2.1.2 Estrutura de dados

As aplicações computacionais que utilizam malhas requerem o uso de estruturas de dados que possam armazenar toda a geometria e a conectividade dos objetos representados por elas. De um modo geral, são necessárias estruturas que armazenam listas de vértices, listas de arestas e listas de faces. As diversas formas como estas listas estão relacionadas e estruturadas implicam diretamente na eficiência do processamento das malhas durante as aplicações. Dois exemplos de estruturas clássicas utilizadas para armazenar malhas são a estrutura *winged-edge*[2] e a estrutura *half-edge* [21].

Esta última é uma estrutura muito eficiente para buscar relações de adjacências entre os elementos da malha. Por exemplo, dado um vértices v, encontrar as arestas, as faces e os vértices adjacentes à v, se torna relativamente simples. A busca por essas relações de adjacências é de extrema importância em operações de colapso de arestas, por exemplo, durante o processo de simplificação/refinamento da malha. A seguir descrevemos melhor a estrutura half-edge, uma vez que ela será empregada no método [21], descrito na seção 3.2.

Estrutura de dados *Half-edge*

A estrutura *half-edge* armazena informações de conectividade da malha triangular e cada face é representada implicitamente por um conjunto ordenado de três *half-edges* orientadas. Cada *half-edge* h armazena seu reverso ($h.r$), o seguinte ($h.n$), o anterior ($h.p$) e o vértice ($h.v$) que é o vértice de origem da *half-edge (figura 2.9)*. A malha é armazenada como um $array$ de $3m$, onde m é a quantidade de triângulos da malha. A representação da malha dessa forma nos permite

percorrê-la e obter sua vizinhança com grande eficiência.

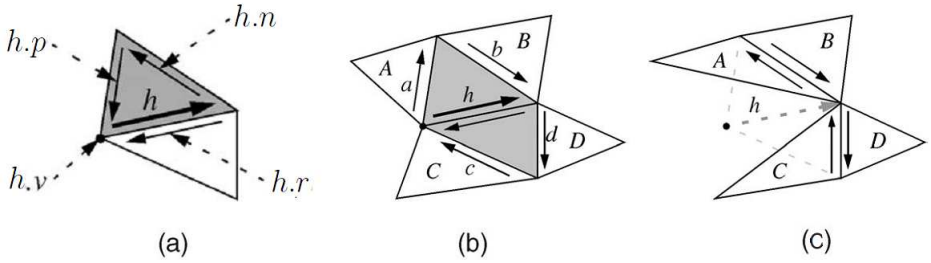


Figura 2.9: Armazenamento da estrutura *half-edge*

Quando um *edge-colapse* é executado devemos atualizar a conectividade da malha. Isto pode ser feito eficientemente usando a estrutura *half-edge*, bastando marcar como inativas as faces adjacentes à aresta colapsada (*as faces inativas não são renderizadas*) e, além disso, atualizar as *half-edge* a e b como inversas, assim como c e d, fazendo com que as faces A e B, assim como C e D, tornem-se vizinhas (*figura 2.9*). Por fim, fazemos $h.v = h.n.v$ para todas as *half-edge* que tem $h.v$ como origem. As atualizações, para cada par de triângulos que tornam-se adjacentes após o colapsamento da aresta, podem ser feitas eficientemente do seguinte modo:

$$h.p.r.r = h.n.r \quad \text{e} \quad h.n.r.r = h.p.r. \tag{2.1}$$

Vamos esclarecer a equação 2.1. Temos, na primeira igualdade, que tomado uma *half-edge* h (h *é a half-edge relacionada a aresta selicionada para o colapso*), acessamos a *half-edge* anterior $h.p$, depois a reversa da reversa $h.p.r.r$ e fazemos isso igual a $h.n.r$ que é a reversa da *half-edge* seguinte a h.

Dada uma *half-edge* h que sofreu um colapso (*figura 2.9*), o operador *vsplit*

restaura a conectividade e inclui os dois triângulos originalmente adjacentes a h. Isso porque, nenhuma informação foi deletada, apenas, aquelas faces adjacentes as *half-edge* colapsadas não foram desenhadas, mas continuam existindo. Tomando como base a figura 2.9, a conectividade das faces A e B podem ser recuperadas usando a *half-edge* h da tabela de entrada nas equações

$$h.p.r.r = h.p \text{ and } h.n.r.r = h.n.$$

De forma similar obtemos a conectividade das faces C e D (*figura 2.9*).

Operações de simplificação e refinamento ficam mais difíceis quando a superfície a ser simplifica apresenta fronteira.

Uma aresta de fronteira h na estrutura de dados half-edge pode ser facilmente detectada quando a *half-edge* $(h.r)$ não existe no conjunto ou se tem um valor inválido.

As operações de refinamento e simplificação na estrutura de dados *half-edge*, apresentadas acima, trabalha bem para malhas orientadas sem bordo. No entanto, malhas com fronteiras simples não se apresentam como um grande problema para a estrutra *half-edge* ou operações de colapsamento de arestas. Discutimos agora como tratar o problema da fronteira.

Podemos detectar facilmente quando uma aresta h é de fronteira, neste caso, $h.r$ não pertence ao conjunto das *half-edge*. Mesmo assim, malhas com fronteiras necessitam de mais cuidados quando a percorremos. Por exemplo, dividimos em duas etapas a visita às arestas incidentes em um vértice v. Começamos com uma *half-edge* h, percorremos no sentido anti-horário, com $t = t.p.r$ até obtermos

t inválido, ou seja, quando o seu inverso $(h.r)$ não existir. Depois, no sentido horário até que $t.r$ seja inválido com $t = t.r.n$.

A atualização das arestas inversas atribuídas por 2.1 devem ser feitas somente quando $h.n$ e $h.p$ não são arestas de fronteira. Assim atualizamos apenas $h.n.r.r = h.p.r$ para os triângulos que contém h.

2.2 Visualização dependendo do observador

Esta seção discute vários critérios que podem ser utilizados na visualização de malhas triangulares considerando a posição do observador.

Existem diversos modelos poligonais complexos que exigem representação com muitos detalhes. Estes são difícies de serem visualizados em tempo real devido ao enorme número de polígonos, excedendo a capacidade dos hardware gráficos.

A representação de malhas triangulares em multiresolução é uma importante ferramenta para reduzir a complexidade da malha em ambientes de representação interativa. Técnicas de visualização baseadas em níveis de detalhes [5] permitem a renderização de um mesmo objeto usando vários tipos de malhas com complexidades variadas.

Sabemos que modelos muito simples podem comprometer a qualidade visual do objeto. Desse modo, a complexidade da malha pode ser ajustada de acordo com a posição relativa do observador. O ideal para a renderização é que a quantidade de triângulos na malha seja reduzida o máximo possível em regiões que não comprometam sua qualidade visual. Encontrar o número ideal de triângulos para a representação da malha é um problema muito difícil e consome tempo demais

para ser resolvido em tempo real.

Muitos métodos de simplificação ou refinamento de malhas e triangulação em multiresolução têm sido desenvolvidos para criar diferentes níveis de detalhes, sequências de malhas em níveis de detalhes com complexidades diferentes e triangulações hierárquicas para a renderização (*Ver detalhes em [11]*) .

2.2.1 Critérios para alterar a resolução da malha

Alguns critérios importantes que podem ser considerados no problema de visualização dependente do observador [21] são:

Volume de visão

A fim de evitar o congestionamento com o envio de uma grande quantidade de dados da memória principal para o hardware gráfico, reduzindo o desempenho de execução do *volume de visão*, as regiões da malha fora do *volume de visão* podem ser mantidas em baixa resolução. Assim, o colapso de arestas pode ser realizado se elas não interceptarem com o volume de visão, ou seja, estiverem fora do campo de visão do observador.

Faces escondidas

Para uma malha triangular grande e complexa, boa parte dos triângulos que estão dentro do campo de visão serão descartados no *pipeline* de renderização na etapa de eliminação das faces escondidas (*back-face culling*). Estes triângulos

desnecessários podem ter um grande custo, aumentando signifitivamente o tempo de computação, mesmo sendo selecionados para descartes durante a etapa de eliminação de faces escondidas. Logo, eles tem que ser transmitidos para a memória gráfica, geometricamente transformados e testados pela seleção do *back-face*. Para prevenir processamentos de vértices desnecessários, podemos manter regiões do *back-facing* da malha numa resolução mais grosseira.

Tela de projeção

Se uma malha poligonal é renderizada sem usar técnicas de *antialiasing*, fica intuitivamente claro que polígonos muito pequenos podem criar apenas artefatos na imagem renderizada que não contribuem para uma imagem suave. Mesmo quando usamos *antialiasing*, depedendo da limitação de resolução da tela, não faz muito sentido renderizar milhares de triângulos. Além disso, a diminuição dos dados transferidos durante o processo de renderização pode ser obtida com a simplificação de triângulos muito pequenos.

Regiões planas

Critérios de simplificação geométrica, como os mencionados acima, não são suficientes para simplificar um número grande de regiões planas na superfície. Uma malha triangular ainda pode ser simplificada, se a diferença nos efeitos de ilumina-ção e de sombreamento não é visualmente significativa. Podemos medir o desvio potencial em um sombreamento difuso para um determinado calapso de aresta pela variação nas normais aos triângulos da região afetada pelo colapso.

Silhueta

A silhueta de um objeto carrega informações importantes para a visualização, por esta razão, sua percepção é muito importante. Distorções ao longo da silhueta são facilmente observadas e podem dificultar a compreensão de alguns objetos. Por isso, as regiões da malha situadas na silhueta do objeto devem ter o máximo de informação possível, ou seja, ter uma alta resolução. Assim, malhas originalmente com baixa resolução dever ser refinadas em sua silhueta (*seção 3.1*), enquanto malhas com alta resolução dever ter sua silhueta preservada durante o processo de refinamento (seção 3.2).

Empregar estes critérios em todos os quadros, por exemplo, pode onerar a aplicação. Neste caso, a representação de superfícies triangulares em tempo real exige uma boa coerência na renderização dos triângulos de um frame para o outro. Por exemplo, se um triângulo foi desenhado em um frame, então é muito provável que ele seja desenhado no próximo frame, isso porque o campo de visão e a posição do observador devem mudar suavemente. Diante disso, é razoável pensar que a complexidade da malha pode ser ajustada de acordo com a posição relativa do objeto, podendo trazer grandes vantagens para a visualização, como a redução do tempo de execução e menor consumo de memória.

2.2.2 Contornos das malhas

Neste trabalho consideramos fundamentalmente a alteração das regiões de contorno, visando otimizar a visualização da malha. Classificamos os elementos de contorno da malha em dois tipos, são eles: a *silhueta* e as *arestas características*.

A *silhueta* é de fundamental importância na visualização, pois o olho humano a tem como o maior aliado na percepção das formas. Na verdade, a silhueta é a primeira caracterísca que o olho identifica quando mira um objeto. A grosso modo, podemos dizer que a silhueta é formada pelos pontos de uma superfície, onde o vetor de visão é paralelo ao plano tangente da superfície. Mais formalmente, temos:

Definição 2.2.1 *Sejam P o ponto sobre uma superfícies S, com normal N_p e E_p a direção de visão do observador ao ponto P. Então o ponto P é dito de silhueta se $E_p \cdot N_p = 0$.*

As *arestas características (AC)* (*figura* 2.10), assim como a silhueta, são facilmente observadas na visualização (*figura* 2.10).

Definição 2.2.2 *Arestas Características são aquelas que aproximam Curvas Características Suaves determinadas pela interseção de duas superfícies suaves [29].*

Definição 2.2.3 *Arestas não-características são todas as arestas da malha que não são características. Ou seja, são as arestas que formam o conjunto $\tilde{AC} = \mathcal{M} - AC$, onde $\mathcal{M}$ é a malha.*

O problema de renderização das regiões de contorno tem sido extensivamente estudado em variados modelos geométricos (como, por exemplo em [8]). Muitos métodos de renderização de malhas em tempo real requerem, como entrada, uma malha fina. Neste caso, para uma renderização mais eficiente, a malha é simplificada progressivamente com dependência na posição do observador, apresentando

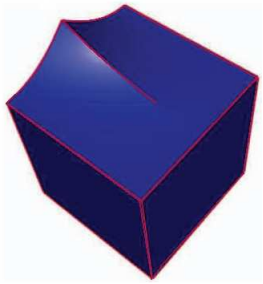


Figura 2.10: Arestas características em vermelho. (*Ilustração de [29]*)

uma aparência suave nas regiões de contorno. Estes métodos incluem técnicas baseadas em múltiplos níveis de detalhes como em [1]. As técnicas de recorte de silhueta [26] também necessitam do auxílio de uma malha fina. Entretando, em muitas aplicações somente malhas em baixa resolução são avaliadas; malhas muito finas não podem ser usadas ou porque são de difícil avaliação ou por conta da grande quantidade de memória necessária para o armazenamento.

O método empregado em [31] toma uma malha triangular de baixa resolução como entrada; ele constrói uma superfície cúbica suave para cada triângulo da malha. Métodos como este usam *patch* globais da superfície na reconstrução (*figura 2.11*), objetivando alcançar uma suavização em toda a superfície renderizada, isto melhora a suavização nas regiões de contorno.

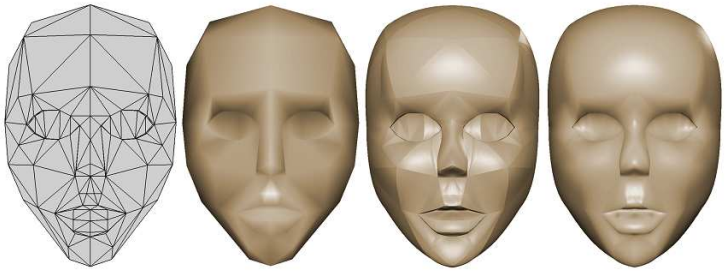


Figura 2.11: Visualização de [31]

Para obter uma renderização rápida e suavização nas regiões de contorno, o método apresentado em [30] faz uma abordagem local e dependente do observador e não envolve a geometria global para a reconstrução. Neste, curvas poligonais da malha são projetadas em uma plano e, a partir daí, é feita uma suavização local.

São muitos os métodos para extrair as *arestas-silhueta* [1] e *AC* de uma malha poligonal. Entretando, existem algumas dificuldades fundamentais para obter uma silhueta suave ou uma curva característica suave a partir das arestas amostradas na malha. Considere a aproximação da esfera na figura 2.12

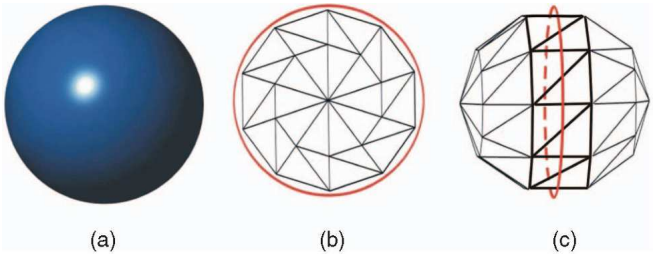


Figura 2.12: (a) Esfera; (b) Amostragem da esfera por uma malha em baixa resolução com sua curva silhueta; (c) Curva silhueta vista em outro ângulo

Suponha que todos os vértices da malha estão na esfera; a figura (b) nos mostra que a curva silhueta não passa sobre os vértices da malha que formam a silhueta poligonal. Na figura 2.12(c) vemos que a *curva-silhueta* passa sobre uma série de triângulos, que estão marcados em negrito.

Esta observação deixa evidente imprecisões quando usamos as *arestas-silhuetas* para a reconstrução da curva silhueta suave. Isso porque a curva silhueta suave não faz, necessariamente, correspondência com a curva silhueta poligonal. Além disso, a curva silhueta assim computada não é visualmente coerente com o movimento do observador.

[1] Uma aresta é dita de silhueta se um de seus dois triângulos adjacentes é visível e o outro é invisível.

2.3 Mapeamento de Textura

Técnicas de mapeamento foram introduzidas em Computação gráfica na tese de doutorado de Ed Catmull 1974. Ele desenvolveu um método, chamado *mapeamento de textura*, para pintar superfícies com muitos detalhes de cor, padrões regulares ou imagens, com base no teorema da Aplicação de Riemann [10], que afirma que qualquer região simplesmente conexa de um plano pode ser mapeada em qualquer outra região simplemente conexa, tal como um disco unitário.

Quando discretizamos uma superfície por uma malha é natural que alguns atributos geométricos dela sejam passados à malha. Esses atributos guardam informações como cor, normal, coordenadas de textura, *etc*.

O mapeamento de textura permite que você associe uma ou mais imagens a polígonos, sendo muito útil para realçar imagens em superfícies com muitos detalhes, podendo dar realismo ou outros efeitos às formas, como por exemplo, efeitos naturais, tais como: água, mármore, madeira, entre outros.

Pintar uma superfície como uma parede de tijolos, fica muito mais prático se usarmos o mapeamento de textura em vez de modelar os tijolos. Além disso, o mapeamento de textura possui um custo computacional baixo em relação à modelagem dos objetos, tornando a renderização de uma cena mais rápida.

Para uma malha dada $\mathcal{M}$, seu mapeamento de textura é obtido pela parametrização de $\mathcal{M}$ no plano de textura. A parametrização de $\mathcal{M}$ é uma aplicação F bijetiva que mapeia cada vértice V de $\mathcal{M}$ em um ponto v no plano de textura. Para uma sequência $\{\mathcal{M}^i\}_{i=0,1,...,n}$ de malhas, obtidas a partir de uma malha $\mathcal{M}$ pelo colapso de arestas, cada $\mathcal{M}^i$ é um subconjunto de $\mathcal{M}$. A grosso modo, podemos dizer que o mapeamento de textura reproduz uma imagem sobre uma superfície

associando um ponto $2D$ do plano de textura a um ponto $3D$ da superfície (*figura* 2.13).

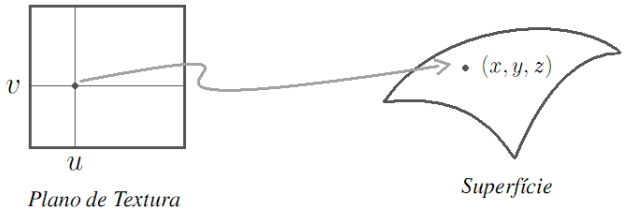


Figura 2.13: Associação de um ponto do plano de textura com um ponto na superfície

Dada uma malha com um mapeamento de textura associado, ao alterarmos sua resolução, mundanças são feitas no mapeamento de textura.

Uma forma de tratar essa questão, embora exija grande consumo de memória, é ter uma sequência de mapeamentos de textura, sendo um para cada mudança na resolução *figura* 2.14. Entretanto, quando aplicamos o mapeamento de textura depois de alterarmos a resolução, uma série de distorções são observadas devido as mudanças geométricas e a interpolação linear empregada pelo mapeamento de textura do hardware gráfico.

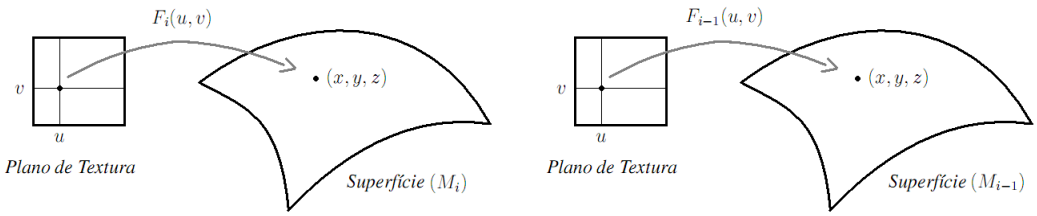


Figura 2.14: Para as *malhas* M_i e M_{i-1} (*com resolução diferentes*) os mapeamentos de textura F_i e F_{i-1}

Para reduzir a distorção na textura causada pela mudança na resolução da malha, várias técnicas foram propostas [16]. As adaptações na textura são capazes de eliminar essa distorção, de modo a torná-la insignificante. Outras técnicas con-

sideram a distorção como um erro métrico [12], implicando que mudanças nas malhas não devem ser feitas quando estas acarretarem em um alto desvio.

Em [4] é apresentado um método que preserva atributos da superfície no mapeamento de textura para cada nível de detalhes. Este método requer um mapeamento preciso para cada nível de detalhes, fazendo com que seja necessário um grande espaço para armazenamento e um longo processamento. Em [**?**] é proposto um mapeamento de textura geral para minimizar a distorção de textura podendo mapear em todos os níveis de detalhes num estágio de pré-processamento. Concluída essa etapa, uma métrica de distorção é usada para guiar a simplificação da malha.

Diante disto, usamos neste trabalho uma parametrização apropriada para o mapeamento de textura na tentativa de minimizar distorções causadas pela mudança na resolução, conforme mostramos no *capítulo 4*.

A figura 2.15 mostra uma malha poligonal plana $M = M^2$ e com textura T. As simplificações M^1 e M^0 com textura T apresentam sérias distorções. Observamos, neste exemplo, que mesmo mantendo a geometria, a interpolação linear do hardware afeta a imagem.

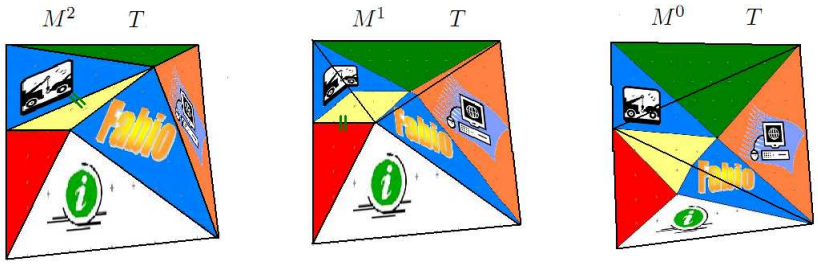


Figura 2.15: M^2 com textura T, M^1 com textura T e M^0 com textura T

[14] cita três métodos básicos para obtermos um mapeamento de textura.

1. Escaneamento de imagens reais.

2. Sintetizar a partir de imagens reais.

3. Definir de forma algorítmica.

O método implementado neste trabalho é baseado em formas algorítmicas ou procedurais. Este método funciona muito bem para qualquer dimensão.

Antes da textura ser aplicada à superfície do objeto, deve-se determinar de que modo ela preencherá a superfície. Existem três modos básicos para gerar as coordenadas de textura: cúbico, cilíndrico e esférico.

Considere a equação paramétrica do cilindro, dada por $C(\theta, z) = (\cos\theta, \text{sen}\theta, z)$. No mapeamento cilíndrico as coordenadas de textura são obtidas associando cada ponto do cilindro (θ, z) a um ponto de textura (u, v), pela equação

$$(u, v) = \left(\frac{\theta}{2\pi}, z\right), \quad u, v \in [0, 1].$$

Agora, considere a equação paramétrica de um pedaço da esfera de raio 1, dada por

$$S(\theta, \phi) = (\cos\theta\text{sen}\phi, \text{sen}\theta\text{sen}\phi, \cos\phi), \;\; 0 \leq \theta \leq \pi/2 \;\text{ e }\; \pi/4 \leq \phi \leq \pi/2.$$

Para o mapeamento esférico as coordenadas de textura (u, v) são dadas pela associação

$$(u, v) = \left(\frac{\theta}{\pi/2}, \frac{\pi/2 - \phi}{\pi/4}\right).$$

Capítulo 3

Adaptação da Geometria

Este capítulo trata de alguns desafios para adaptar a geometria e a conectividade de uma malha triangular de acordo com a posição do observador. Essa adaptação é feita de modo que a malha seja visualizada de forma suave e não sobrecarregue recursos computacionais tanto no armazenamento quanto no processamento da malha. Estes requisitos possibilitam o desenvolvimento de aplicações onde as malhas sejam visualizadas em tempo real. Também discutimos duas estratégias para tratar este problema.

No capítulo 2 discutimos alguns critérios que, dependendo da posição do observador, podem ajudar neste processo de visualização. O principal critério que vamos abordar aqui é que a malha deverá ter uma resolução maior ao longo do contorno (*silhueta e arestas características*), uma vez que o contorno é que permite ao observador ter uma visão e uma compreensão mais global e geral do objeto. Na verdade, o contorno é o que primeiro, o olho humano identifica quando mira um objeto.

Tratamos esse problema em duas direções as quais dependem da resolução da malha. Objetos representados por malhas de *baixa resolução* podem não apresen-

tar um aspecto visual suave, formando, por exemplo, “bicos” nos contornos do objeto. Por outro lado, objetos representados por malhas de *alta resolução* podem apresentar um aspecto visual mais suave, porém podem dificultar as aplicações em tempo real, devido à grande quantidade de triângulos a serem processados.

A seguir discutimos estas duas estratégias de visualização com base na resolução da malha de entrada.

3.1 Visualização de dados com baixa resolução

Normalmente quando representamos um objeto por uma malha de baixa resolução, ficam bem visíveis “bicos” em sua silhueta e arestas caracteristicas, causando um desconforto visual, e isso ocorre devido ao reduzido número de triângulos da malha. Diante disto, nosso desafio é lidar com um objeto de baixa resolução, mas que tenha um aspecto visualmente suave na silhueta e arestas características. Esta primeira estratégia tem como entrada uma malha de baixa resolução em seguida fazemos um refinamento na silhueta e arestas características. Este método é baseado em [29].

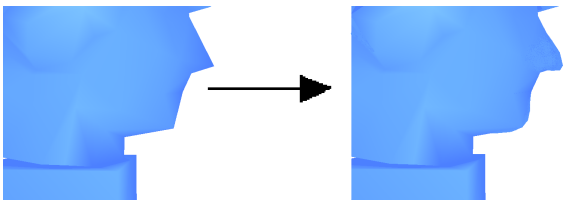


Figura 3.1: Observe que o processo de refinamento fez uma considerável suavização no nariz e queixo

Inicialmente, identificamos as *regiões relevantes (arestas características e sil-*

hueta). Em seguida aplicamos uma suavização nessas regiões Por fim, aplicamos uma reamostragem nessas regiões com o objetivo de suavizar a malha.

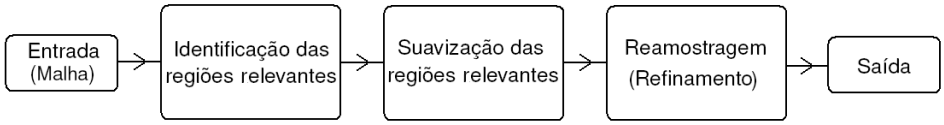


Figura 3.2: Pipeline do método

Tendo como entrada uma malha triângular $\mathcal{M}$ fechada (*sem bordo*) e de baixa resolução, formada pelos triângulos $T_1, T_2, \ldots, T_N$ e vértices $v_1, v_2, \ldots, v_n \in \mathbb{R}$. Uma aresta de $\mathcal{M}$ será denotada por $e = [v_i, v_j]$ onde $[\ ]$ representa o fecho convexo de um conjunto. O vetor normal a um triângulo $T = (v_i, v_j, v_k)$ tem direção do vetor $(v_j - v_i) \times (v_k - v_j)$ e norma 1. Vamos representar $\mathcal{M}$ por dois conjuntos, sendo um de arestas, e outro de faces. Cada um desses conjuntos será dividido em dois subconjuntos distintos. O de arestas em: Arestas Características(AC) e Arestas Não Característica($A\tilde{C}$). E o de faces em: Triângulos Silhueta(TS) e Triângulos Não Silhueta ($T\tilde{S}$), dois a dois disjuntos.

3.1.1 Identificação das regiões relevantes

As regiões relevantes consistem nas arestas características e na silhueta.

Arestas Características

Considere uma aresta $[v_i, v_j]$ cujas faces adjacentes são f_a e f_b, com normais N_a e N_b, respectivamente. Vamos considerar como *aresta característica* toda aquela cujo ângulo entre os vetores normais às faces adjacentes tem ângulo maior que 60^o, ou ainda, quando $\langle N_a, N_b \rangle < 0.5$. Veja a figura 3.3.

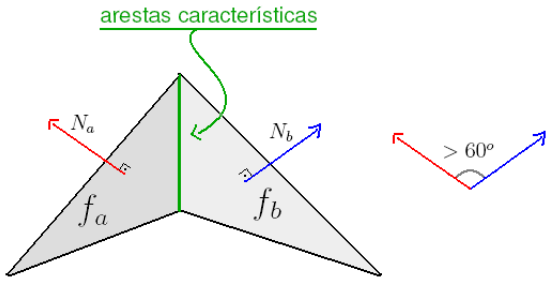


Figura 3.3: Identificação de uma aresta caracrerística

Triângulos silhueta

Os *triangulos silhueta* são aqueles que se apresentam na silhueta do objeto, sendo determinados pelo status de visibilidade de seus vértices.

A visibilidade dos vértices depende do ângulo entre a direção do observador ($\overrightarrow{w}$), e a direção do vetor normal ao vértice v que será denotado por $\overrightarrow{N_v}$. Vamos adotar a notação dos sinais "+" e "-", para representar, respectivamente, quando um vértice é *visível* ou *invisível*.

Por definição, vamos tomar o vertor normal a um vértice v como sendo o vetor unitário no sentido de

$$\overrightarrow{N_v} = \sum_{1=0}^{n} \left(\overrightarrow{N}_{f_i} * \text{área de} f_i \right),$$

onde f_i são as faces que compartilham o vértice v e N_{f_i} seus normais, com $i = 0, 1, \ldots, n$.

Considere E o observador, v um vértice em uma malha qualquer como na figura 3.4. Com isso, o vértice v é dito visível quando o ângulo entre seu normal ($\overrightarrow{N}_v$) e o vetor $\overrightarrow{w} = E - v$ é menor ou igual que 90^o (*observe que $\overrightarrow{w}$ é a direção de visão do observador*). Caso contrário, o vértice v é dito invisível.

Mais precisamente:

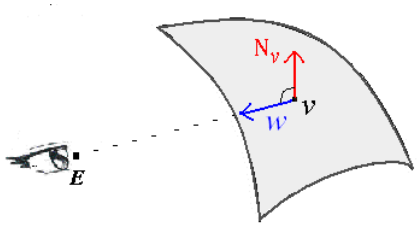


Figura 3.4: Esquema de visualização de um vértice

Definição 3.1.1 *Seja v um vértice de uma malha $\mathcal{M}$ qualquer e w a direção de visão do observador. Se $\langle v, w\rangle \geq 0$, v é dito visível e caso contrário, ou seja, $\langle v, w\rangle < 0$, o vértice v é denotado invisível.*

Definição 3.1.2 *Dada uma malha $\mathcal{M}$ formada por faces triangulares, uma face é dita um Triângulo Silhueta (TS) quando seus vértices tem status de visibilidade não todos iguais, isto é, seus três vértices não tem o mesmo sinal.*

Ou ainda, o vértice v é visível se, e somente se, o ponto E estiver acima do plano determinado pela equação $P(X, v) \equiv N.(X - v) = 0$, isto é, quando $N.(E - v) \geq 0$. A partir disso, podemos determinar o status de visibilidade de todos os vértices v de uma malha qualquer $\mathcal{M}$.

Em particular, damos o nome de *aresta de silhueta* a toda aresta do triângulo silhueta com extremos positivos. Por exemplo, na figura 3.5 a aresta $[V_2, V_3]$ é uma aresta de silhueta. Na verdade, uma das faces adjacentes a aresta de silhueta é visível (*frontfacing*) e a outra invisível (*backfacing*).

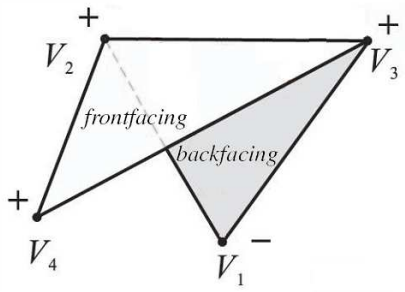


Figura 3.5: Aresta de silhueta

3.1.2 Suavização das regiões relevantes

Arestas características

As arestas características podem ser agrupadas formando linhas poligonais. Estas linhas serão usadas individualmente na suavização.

Depois de identificarmos as arestas características, fazemos a suavização localmente, via interpolação cúbica de cúbica. Para isto, são necessárias as coordenadas dos vértices da linha poligonal, assim como seus vetores tangentes.

Suponha que a linha poligonal formada por arestas características contenha no mínimo três vértices consecutivos $(V_0, V_1, \cdots, V_k), \quad k \geq 2$. Para um vértice interior à linha, isto é, $V_i, \quad i = 1, 2, \cdots, k-1$, a direção do vetor tangente $\hat{T}_i$ em V_i é a direção do vetor tangente ao círculo unitário determinado pelos três vértices consecutivos $V_{i-1}, \quad V_i$ e V_{i+1}. Seja

$$W_0 = V_i - V_{i-1} \quad \text{e} \quad W_1 = V_{i+1} - V_i$$

é fácil ver que

$$\hat{T}_i = |W_1| \cdot W_0 + |W_0| \cdot W_1.$$

O vetor tangente estimado T_i para V_i, usado na interpolação Hermite, tem a

mesma direção de $\hat{T}_i$, mas tem seu comprimento determinado de modo que a curva de Hermite resultante seja uma boa aproximação do arco circular.

Como não podemos calcular da mesma forma os vetores tangentes nos pontos T_0 e T_k, já que estes não possuem, respectivamente, um anterior e um posterior, tomamos os vetores tangentes mais próximos na linha. Isto é, $T_0 = T_1$ e $T_k = T_{k-1}$.

Dados os pontos V_i, V_{i-1} e seus vetores tangentes podemos calcular a interpolação cúbica hermitiana por meio da equação

$$\begin{aligned} S(u) &= (2V_{i-1} - 2V_i + T_{i-1} + T_i)u^3 \\ &- (3V_{i-1} - 3V_i + 2T_{i-1} + T_i)u^2 \\ &+ T_{i-1}u + V_{i-1}, \quad u \in [0,1]. \end{aligned} \tag{3.1}$$

A spline, obtida a partir da equação 3.2 com u variando de $[0,1]$, é usada na reamostragem que suaviza as arestas características. Note que se na equação 3.2 $u = 0$ obtemos $S(u) = V_{i-1}$ e se $u = 1$ obtemos $S(u) = V_i$.

Triângulos silhueta

Por definição, em cada triângulo silhueta existe exatamente um vértice com sinal diferente dos outros dois vértices, isto nos dá duas arestas com sinais diferentes $(+,-)$ nas extremidades, como ilustramos na figura 3.6.

Em cada um desses lados com sinais diferentes vamos criar, por meio de uma interpolação cúbica de hermite, segmentos ligando os vétices com sinais diferentes (*veja a figura 3.7*). A estes segmentos damos o nome de *Pontes de silhueta*.

Vamos tomar o exemplo do triângulo acima para mostrar como calcular as

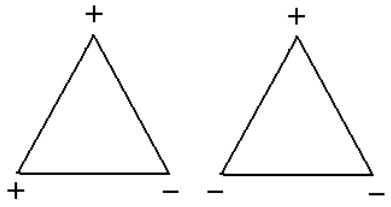


Figura 3.6: Sinais dos triângulos silhueta

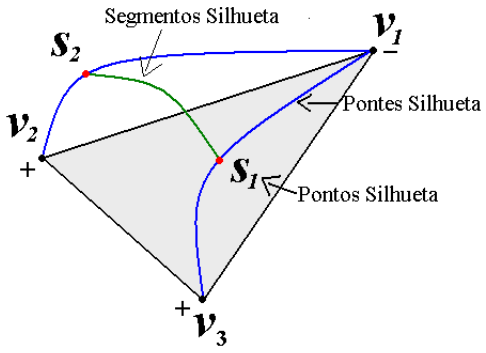


Figura 3.7: Construção das Pontes Silhueta

pontes silhueta. Com base na figura 3.7, as pontes de silhuetas unirão V_1 a V_2 e V_1 a V_3.

Para calcular as pontes silhueta vamos usar a *interpolação cúbica de Hermite*:

$$\begin{aligned} S(u) &= (2V_1 - 2V_2 + T_1 + T_2)u^3 \\ &- (3V_1 - 3V_2 + 2T_1 + T_2)u^2 \\ &+ T_1u + V_1, \quad u \in [0,1], \end{aligned} \tag{3.2}$$

onde T_1 e T_2 são os vetores tangentes aos vértices V_1 e V_2.

Para obtermos as duas pontes de silhueta do triângulo mostrado na figura 3.7, além dos vértices v_1, v_2 e v_3, são necessários:

(i) Vetores tangentes nos vértices (T_1, T_2 e T_3), respectivamente;

(ii) Vetore normais nos vértices (N_1, N_2 e N_3), respectivamente.

Considere os vetores

$$\hat{T_1} = (v_2 - v_1) - [(v_2 - v_1).N_1] .N_1 \tag{3.3}$$

e

$$\hat{T_2} = (v_2 - v_1) - [(v_2 - v_1).N_2] .N_2. \tag{3.4}$$

Para obtermos os vetores T_1 e T_2, basta tomarmos vetores, respectivamente, nas direções de $\hat{T_1}$ e $\hat{T_2}$, mas que tenha comprimentos L_1 e L_2.

$$L_1 = \frac{2\,|v_2 - v_1|}{1 + cos\theta_1} \qquad L_2 = \frac{2|v_2 - v_1|}{1 + cos\theta_2} \tag{3.5}$$

Veja na figura abaixo, para uma melhor compreensão, os vetores $\hat{T_1}$ e $\hat{T_2}$.

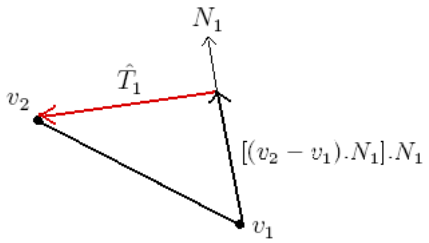


Figura 3.8: Construção do vetor tangente

onde θ_i é o ângulo formado pelos vetores $\overrightarrow{v_1 v_2}$ e $T_i,\ i = 1, 2$.

A escolha do comprimento dos vetores tangentes é feita de modo que a curva obtida pela interpolação cúbica de hermite seja uma boa aproximação do arco circular.

Agora, por um processo semelhante ao que fizemos para calcular as pontes de silhueta, vamos calcular os segmentos de silhueta sobre os *pontos de silhueta*.

Definição 3.1.3 *Um ponto $S(u) = P$ em uma ponte de silhueta é dito ponto de silhueta quando seu vetor normal é perpendicular a direção de visão.*

Seja P um ponto sobre a ponte silhueta obtida a partir dos vértice v_1 e v_2, cujos normais são, respectivamente, N_1 e N_2. Então, definimos o vetor normal sobre a ponte silhueta pela intepolação linear

$$N = (1 - u)N_1 + uN_2.$$

Logo o ponto de silhueta sobre a ponte silhueta é tal que $E.\big((1-u)N_1+uN_2\big) = 0$. Daí, aplicamos o parâmetro u sobre as pontes de silhueta (*figura* 3.7) e obtemos o ponto de silhueta. Calculados, para cada triângulo de silhueta, os dois pontos de silhueta $S(u_1)$ e $S(u_2)$ (*um em cada ponte de silhueta*) seguimos em busca do segmento de silhueta. Este sim, será fundamental para a suavização da malha.

De posse dos pontos de silhueta, para obtermos os *segmentos de silhueta*, basta aplicarmos na equação 3.2 fazendo $V_1 = S(u_1)$ e $V_2 = S(u_2)$. Quando fazemos o parametro u variar de 0 a 1 obtemos o segmento de silhueta. Sobre o segmento de silhueta vamos tomar amostras de pontos para a suavização. Veja a figura 3.10.

3.1.3 Reamostragem (refinamento)

Refinamento nas arestas características

Obtemos anteriormente uma curva que suaviza as arestas características. Agora, tomamos pontos na curva de Hermite e criamos uma estrutura de triângulos localmente. Cada um desses triângulos terá dois pontos na curva de Hermite e o outro em um dos vértices opostos à aresta característica. A figura 3.9 ilustra a reamostragem local nas arestas características.

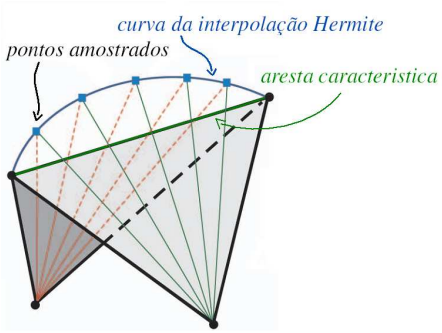


Figura 3.9: Suavização das arestas características

Refinamento na silhueta

Para obtermos uma silhueta suave construímos triângulos menores cujos vértices estão sobre o segmento de silhueta e sobre os triângulos silhueta.

Podemos pensar nesse problema criando triângulos de modo bem semelhante ao da seção anterior, onde no lugar das arestas características colocaríamos as arestas de silhueta cujos triângulos adjacentes são *frontfacing* e *backfacing*. Fazendo isso, não temos nem a necessidade de calcular as pontes de silhueta.

Embora esse modo de suavizar tenha limitações, como as ilustradas na figura 2.12, este é de fácil implementação e baixo custo computacional.

Com base em [29] construímos, também, uma estrutura de triângulos para o refinamento, onde obtemos uma silhueta bem mais próxima da real, diminuindo bastante a diferença entre a silhueta obtida e a real que pode ocorrer no método citado anteriormente. A seguir descrevemos esta estrutura.

No item (a) da figura 3.10 o triângulo silhueta é $V_1V_2V_3$, o segmento de silhueta

$S(t)$ fica por trás do plano que contém a aresta V_2V_3 e o ponto V_4. Neste primeiro caso, tomamos pontos sobre o segmento $S(t)$ e sobre a aresta V_2V_3. Estes pontos são conectados formando triângulos como ilustrado.

No item (b) da figura 3.10 o triângulo de silhueta é $V_2V_3V_4$ e o segmento de silhueta $S(t)$ está a frente do plano contendo o segmento V_2V_3 e o ponto V_4. Neste caso, tomamos os pontos amostrados sobre o segmento de silhueta e ligamos ao vértice V_4.

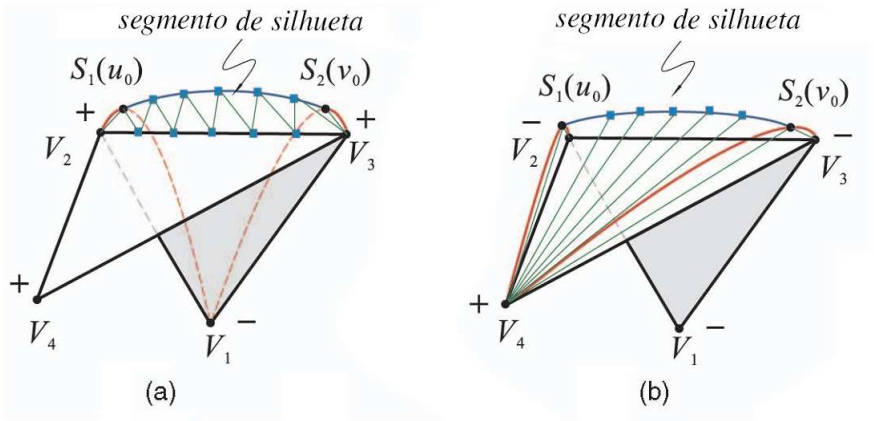


Figura 3.10: Conectividade para a suavização da silhueta

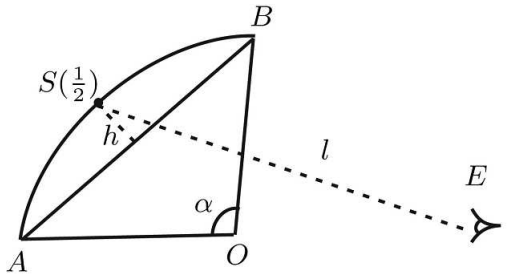
Para a implementação, [29] trata separadamente os casos onde uma aresta característica é um dos lados do triângulo silhueta. Podemos separar estes casos em dois:

- a aresta característica é um dos lados cujos extremos têm sinais diferentes, ou seja, visibilidades diferentes;
- a aresta característica tem o mesmo status de visibilidade em seus extremos.

Em nosso trabalho, quando temos uma aresta característica em um triângulo de silhueta aplicamos o refinamento apenas no triângulo silhueta. A aresta característica não é refinada.

O número de pontos amostrados depende da curvatuva e do comprimento do segmento de silhueta projetado na imagem. Como o segmento de silhueta é uma aproximação de uma arco de círculo, obtemos uma estimativa do número de pontos como segue.

Sejam A e B os pontos extremos de um segmento de silhueta $S(t)\ \ t \in [0,1]$. Seja h a altura do segmento de silhueta no plano de projeção, aproximado pela distância do ponto médio $S(1/2)$ do segmento de silhueta do ponto médio do segmento AB. Seja $r = \text{sen}(\alpha/2) = \sqrt{(1-\cos\alpha)/2}$, onde α é o ângulo central do arco AB e assim o $\cos\alpha$ pode ser aproximado pelo produto interno dos dois vetores normais unitários nos pontos A e B (*figura* 3.1.3). Seja e a tolerância permitida, em pixels, então, a quantidade n_s de pontos amostrados é dado por



$$n_s = \left[\frac{1}{2} \cdot \sqrt{h.r/e}\right]$$

A interpretação para n_s é que se a quantidade de pontos amostrados é no mínimo n_s, então a linha poligonal por eles formada é uma boa aproximação a curva de silhueta suave com um erro e.

3.2 Visualização de malhas de alta resolução

Este método, desenvolvido em [21], procura simplificar as regiões de uma malha fina que menos afetam a qualidade visual da mesma. A escolha das regiões a serem simplificadas é feita de acordo com alguns critérios, conforme a seção 2.2.1, os quais são analisados do ponto de vista do observador. O método emprega a estrutura *half-edge* para armazenar a malha, conforme descrita na seção 2.1.2, permitindo assim a realização eficiente de operações de colapsação de arestas, vértices e faces durante a simplificação.

O algoritmo recebe como entrada uma malha de alta resolução. Em seguida são identificadas as regiões que devem ser simplificadas em função do ponto de vista do observador. Uma vez definidas as regiões, são aplicadas operações para simplificar a malha. A figura 3.11 mostra as principais etapas do método.

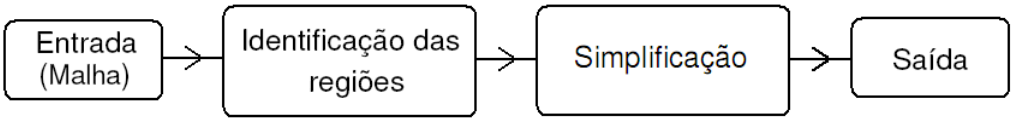


Figura 3.11: Pipeline do método de visualização de objetos de alta resolução

3.2.1 Identificação das regiões

Neste etapa são identificadas as regiões da malha a serem simplificadas, de acordo com os critérios da seção 2.1.2. Na realidade são identificadas as arestas da malha situadas nestas regiões. Como a malha está armazenada em uma estrutura *half-edge*, são identificadas as *half-edges* e *vértices* situados nas regiões a serem simplificadas. Estas *half-edges* e vértices são armazenados em duas estruturas

hierárquicas auxiliares: uma *árvore binária de vértices*, cujos nós representam os vértices das arestas que devem ser colapsadas, e uma outra *árvore binária de half-edges*, cujos nós representam as *half-edges* das arestas a serem colapsadas (*figura*3.12).

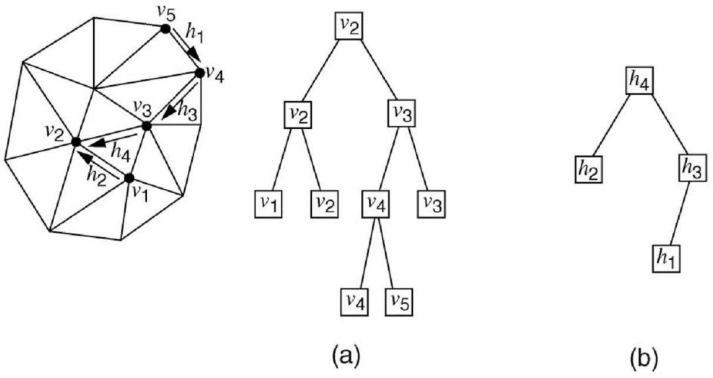


Figura 3.12: Hierarquia binária H do colapso de arestas. (a) Hierarquia de vértices; (b) Hierarquia H de *half-edges*. A ordem do colapso se inicia nas folhas e termina na raiz

Além de armazenar as regiões que devem ser simplificadas de acordo com o ponto de vista do observador, estas duas estruturas (*hierarquia de vértices e hierarquia de half-edges*) são importantes principalmente para atualizar, de forma eficiente, as regiões a serem simplicadas/refinadas a cada vez que o observador muda de posição. Isto porque, em vez de percorrer toda estrutura *half-edge* que representa a malha, é necessário apenas percorrer as informações a partir das estruturas hierárquicas durante esta atualização.

Capítulo 4

Adaptação de Textura

Neste capítulo descrevemos um método para adaptar a textura de uma malha cuja resolução é alterada de acordo com o ponto de vista do observador [6]. Veja figura 4.1.

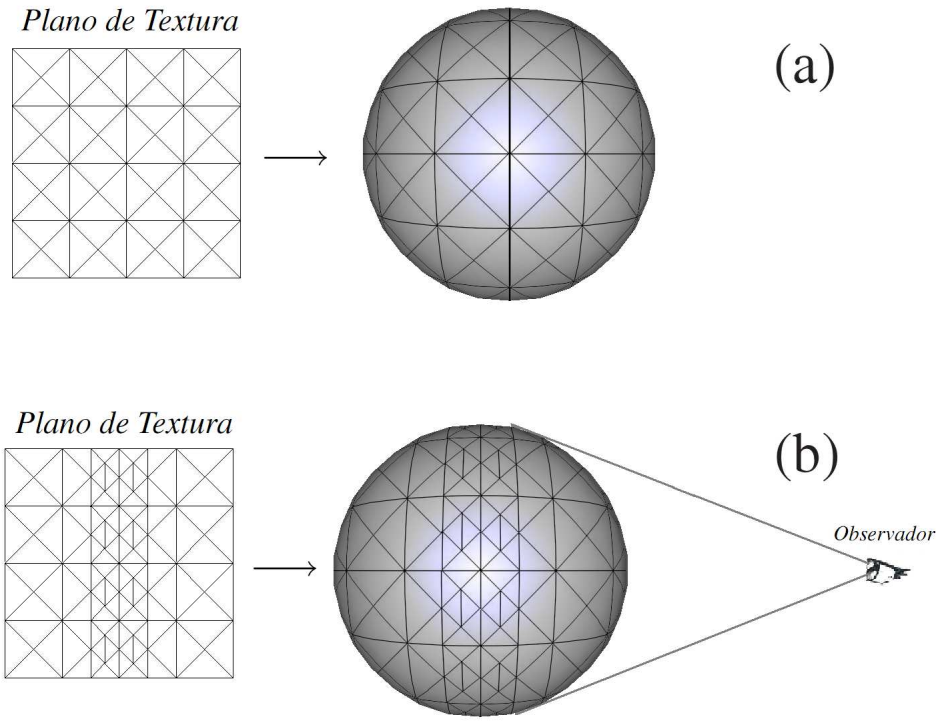


Figura 4.1: (a) Mapeamento de textura da esfera. (b) Refinamento na malha dependente do observador e seu refinamento correspondente na textura

A idéia do problema de adaptação da textura, tendo como entrada uma malha $\mathcal{M}$ e um mapeamento de textura T associado, é encontrar as novas coordenadas de textura para cada nova malha $\mathcal{M}^i$ $(i = 1, 2, \ldots, n)$ obtida a partir de $\mathcal{M}$ por mudanças na resolução.

Quando mudamos a resolução de $\mathcal{M}$, obtendo uma nova malha $\overline{\mathcal{M}}$, podemos aplicar de diversas formas o mapeamento de textura em $\overline{\mathcal{M}}$. Uma delas é ter um novo mapeamento de textura para cada alteração na malha, mas isso requer mais trabalho artístico e um maior consumo de memória.

Um método prático e com menor consumo de memória para aplicarmos o mapeamento de textura à $\overline{\mathcal{M}}$, é tomarmos as coordenadas de textura dos novos vértices com base no mapeamento de textura original. Podemos fazer isso a partir de uma interpolação linear em cada nova face. Este método, mesmo sendo muito simples, geralmente apresenta muitas distorções na textura quando aplicada à superfície.

Em [6] é apresentado um método que visa minimizar a distorção de textura causada no colapso de arestas.

A motivação de usarmos este método é acreditarmos que, sendo aplicado em conjunto com o método de suavização de contornos do *Capítulo 3*, ele possa permitir a visualização de malhas de alta resolução em tempo real, conservando características geométricas e outros atributos dos objetos.

4.1 Processamento da textura

Tendo como entrada uma malha de *alta resolução*, em alguns casos é necessário colapsarmos arestas para obtermos uma malha em *baixa resolução* que possa ser renderizada em tempo real.

Nossa meta é, após alguns colapsos de arestas feitos em uma malha $\mathcal{M}$ de alta resolução com seu mapeamento de textura, eliminar a distorção da textura causada pelo colapso. Na verdade fazemos um tratamento local na textura de forma que ao passarmos a textura para a superfície essa distorção torne-se insignificante do ponto de vista perceptual.

Vamos considerar para uma sequência de malhas progressivas, um mapeamento de textura T^i associado com cada modelo reduzido de $\mathcal{M}^i$ pelo $ecol$, isto é,

$$\begin{array}{ccccccccc} \mathcal{M} = \mathcal{M}^n & \overset{ecol_{n-1}}{\rightarrow} & \ldots & \overset{ecol_1}{\rightarrow} & \mathcal{M}^1 & \overset{ecol_0}{\rightarrow} & \mathcal{M}^0 \\ T = T^n & & & & T^1 & & T^0 \end{array}$$

Desejamos que a adaptação de textura de T^i para T^{i-1}, quando mudamos a malha de M^i para M^{i-1}, seja *invertível*. Consequentemente, temos inversamente, a partir de M^0

$$\begin{array}{ccccccccc} \mathcal{M} = \mathcal{M}^n & & & & \mathcal{M}^1 & & \mathcal{M}^0 \\ T = T^n & \overset{vsplit_{n-1}}{\leftarrow} & \ldots & \overset{vsplit_1}{\leftarrow} & T^1 & \overset{vsplit_0}{\leftarrow} & T^0 \end{array}$$

A adaptação da textura e sua inversa envolvem as mesmas operações, não importa se vamos de $\mathcal{M} = \mathcal{M}^n$ para $\mathcal{M}^0$ ou o contrário.

Para um colapso de arestas $ecol_{i-1}$ (*usando o half edge collapse*) que reduz M^i para M^{i-1}, a adaptação da textura é local para R_i, onde R_i é a região formada pelas faces que sofrem alterações geométricas em virtude do colapso (*figura 4.2*).

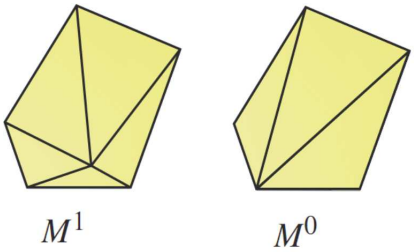


Figura 4.2: Região R_i formada pelas faces que mudam com o colapso

A região R_i (*figura4.3*) pode ser particionada no mesmo número de células para T^i e T^{i-1}, e dentro de cada uma das células a textura é interpolada linearmente. Uma vez que os pares correspodentes são encontrados, a adaptação de textura pode ser passada para um *hardware gráfico*.

4.2 Células correspondentes de dois níveis diferentes

Para *simplificar* malhas usando colapso de arestas, vamos considerar que os conjuntos de pontos de $\mathcal{M}^i$ é um subconjunto de $\mathcal{M}$.

Quando sobrepomos as regiões R_i antes e depois do colapso de arestas $ecol_{i-1}$, as arestas sobrepostas de T^i e T^{i-1} podem se intersectar, particionando a região R_i em células, como na figura 4.3, onde as intersecções ocorrem em a e b, respectivamente, particionando a região R_i em nove células. Nesta seção usamos letras maiúsculas para representar os pontos sobre a superfície e letras minúsculas para representar pontos sobre o plano de textura.

Entretando, as arestas que se intersectam no plano da textura, em geral, não são

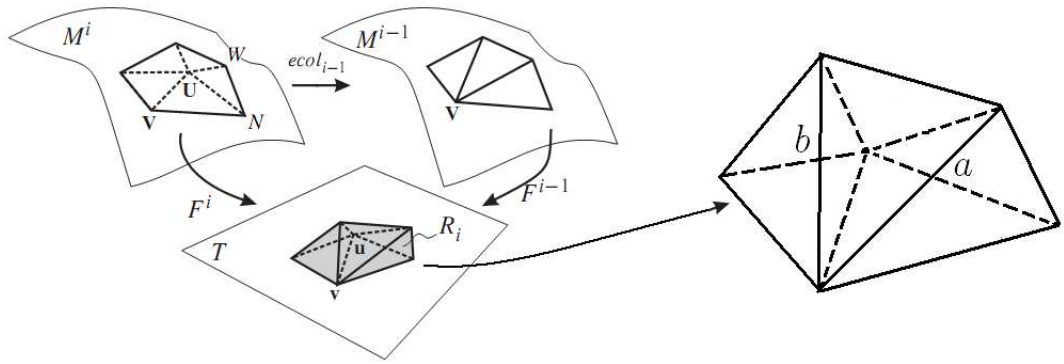


Figura 4.3: Intersecção das regiões no plano de textura antes e depois do colapso de UV

coplanares no espaço. Porém, para cada par de arestas que se intersectam no plano podemos calcular os pontos correspondentes no espaço, um em cada aresta, que denotamos por A_i e A_{i-1}.

Tomando, como exemplo, a malha da figura 4.3, computamos o par de pontos mais próximos A_i e A_{i-1}, onde A_i está em UN de M^i e A_{i-1} em VW de M_{i-1}. Os pontos A_i e A_{i-1} são obtidos, calculando-se a menor distância de dois pontos, sendo um em cada aresta. A solução do sistema a seguir são os parâmetros que usamos na interpolação para obtermos os pontos que minimizam a distância.

$$\begin{bmatrix} \vec{x}\cdot\vec{x} & -\vec{y}\cdot\vec{x} \\ \vec{x}\cdot\vec{y} & -\vec{y}\cdot\vec{y} \end{bmatrix} \begin{bmatrix} \alpha_i \\ \alpha_{i-1} \end{bmatrix} = \begin{bmatrix} (V-U)\cdot\vec{x} \\ (V-U)\cdot\vec{y} \end{bmatrix},$$

onde, $\vec{x} = N - U$ e $\vec{y} = W - V$.

A solução do sistema acima são os parâmteros α_i e α_{i-1}, $\alpha_i, \alpha_{i-1} > 0$. As coordenadas de textura de A_i, denotada por a_i, e A_{i-1}, denotada por a_{i-1}, são calculadas pelas seguintes equações.

$$\begin{cases} a_i = u + \alpha(n-u), \\ a_{i-1} = v + \alpha_{i-1}(w-v). \end{cases}$$

Note que as coordenadas de a_i são obtidas por uma interpolação linear entre U e n, e as coordenadas de a_{i-1} são obtidas pela interpolação linear de v e w.

Pelas equações acima é fácil ver que, em geral, a_i e a_{i-1} são diferentes.

Quando colapsamos U em V as coordenadas de textura de v em T não mudam, assim como as demais coordenadas da região. Então só nos resta o trabalho de calcular as coordenadas de u_{i-1} em T^{i-1}. Para isso, dividimos o plano de textura, antes e depois do colapso, em nove regiões. Daí, identificadas as células correspondentes, fazemos o mapeamento das células de uma região na outra. A imagem gerada após a operação de colapso será a textura T_{i-1} que corrige a distorção em T_i.

Veja na figura 4.4 a imagem de textura original passada a malha simplificada e, no segundo caso, com o ajuste na textura. Observe que no primeiro caso fica evidente uma distorção, já no segundo, quando mudamos a textura, não há mais distorção aparente.

Inicialmente encontramos o ponto U_{i-1} em M_{i-1} que está próximo de U e identificamos o triângulo T_k contendo U_{i-1} (*que é a projeção de U na malha depois do colapso*). Então computamos as coordenadas baricêntricas de U_{i-1} em relação as coordenadas dos vértices do triângulo T_k. Agora, aplicamos os coeficientes das coordenadas baricêntricas de U_{i-1} em u_i e assim obtemos as coordenadas de u_{i-1}, usadas na correção distorção da textura.

Depois de obtermos $a_i,\ a_{i-1},\ b_i,\ b_{i-1}$ e u_{i-1}, movemos a para a_i, b para b_i e assim obtemos a divisão da região de textura em nove células. De modo análogo, movemos a para a_{i-1}, b para b_{i-1} e u para u_{i-1}, particionando a nova região de

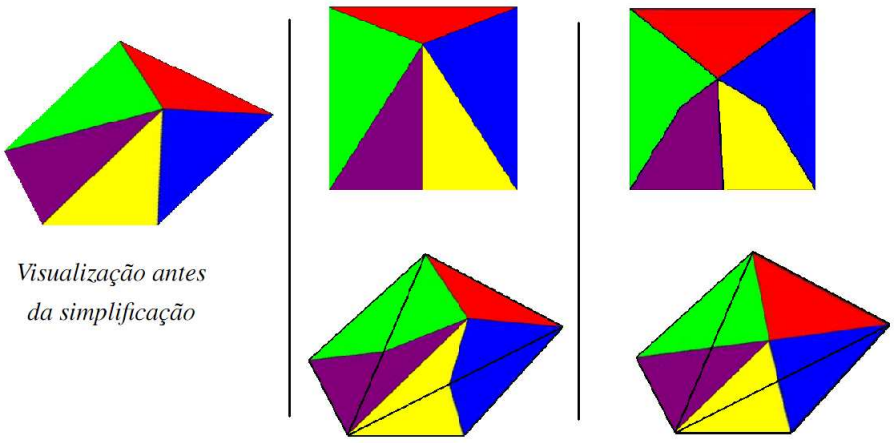


Figura 4.4: A imagem da esquerda foi renderizada com a textura original antes do colapso. Ao centro estão textura original acima, e malha simplificada, já no figura do canto direito foi feito um tratamento na textura para compensar a distorção (*ver malha no canto inferior esquerdo*).

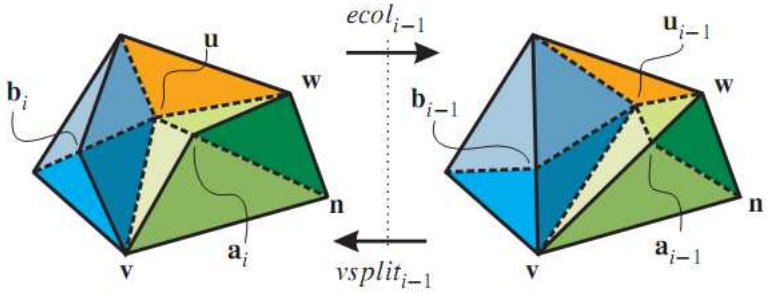


Figura 4.5: Distorção na textura

textura (*agora distorcida* para serem aplicadas em M^{i-1} (*figura 4.5*)).

Quando aplicamos a textura distorcida sobre a região simplificada devemos obter uma imagem próxima da original, sem distorções aparentes da textura na superfície, como visto na figura 4.4.

4.3 Adaptação da textura

Depois de feitas as correspondências para todas as células na região R^i, a adaptação de textura de T^i para T^{i-1} é executada para cada par de células correspondentes.

Para adaptarmos a textura da célula $\Delta a_i b_i c_i$ de T^i para sua célula correspondentes $\Delta a_{i-1} b_{i-i} c_{i-1}$ de T^{i-1}, tomamos a fonte de textura original e a última malha obtida e, daí aplicamos o mapeamento de textura. Fazemos de modo similar quando tomamos o caminho inverso da adaptação de T^{i-1} para T_i, onde $\Delta a_{i-1} b_{i-1} c_{i-1}$ é a textura fonte (*original*) e $\Delta a_i b_i c_i$ é a malha obtida após o processo.

Capítulo 5

Resultados

Este capítulo apresenta alguns resultados obtidos com os métodos de visualização descritos ao longo da dissertação, assim como o tratamento de textura quando são feitas mudanças na resolução das malha.

5.1 Triângulos de silhueta

No item (a) da figura 5.1 mostramos a extração dos triângulos de silhueta em malhas de baixa resolução, onde localizamos 214 triângulos de silhueta em 5.1. Já no item (b) da mesma figura, mostramos os triângulos que fazem a suavização na silhueta.

5.2 Resultado de *Silhueta*

Na figura 5.2 temos as seguintes observações. No item (a) mostramos um objeto rende-rizado com 15.014 triângulos; no item (b) temos o mesmo objeto renderizado com apenas 500 triângulos; no item (c) temos a silhueta construída a

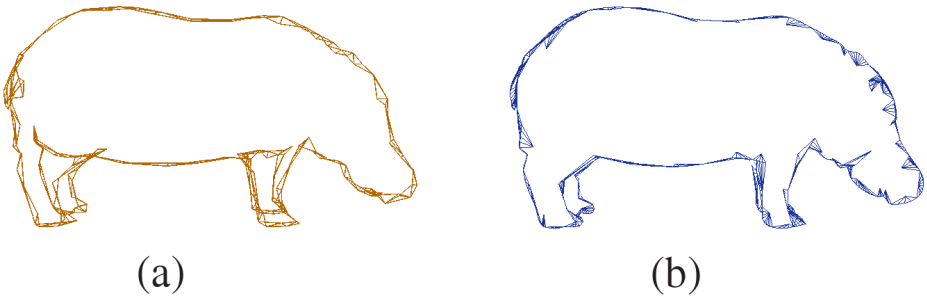


Figura 5.1: Extração de triângulos de silhueta e refinamento na silhueta

partir dos segmentos de silhueta descritos na seção 3.1.2; e no item (d) temos a silhueta construída pela interpolação direta nas arestas de silhueta.

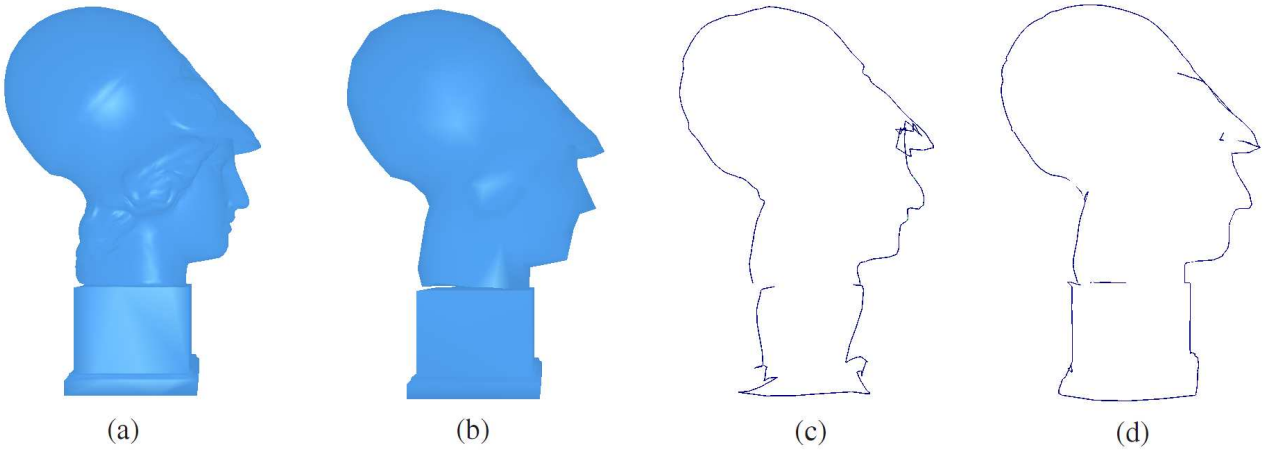


Figura 5.2: Suavização da silhueta em um objeto renderizado com 500 triângulos

Na figura 5.3 temos as seguintes observações. No item (a) mostramos um objeto renderizado com 46.000 triângulos; no item (b) temos o mesmo objeto renderizado com apenas 1000 triângulos; no item (c) temos a silhueta construída a partir dos segmentos de silhueta descritos na seção 3.1.2; e no item (d) temos a silhueta construída pela interpolação direta nas arestas de silhueta.

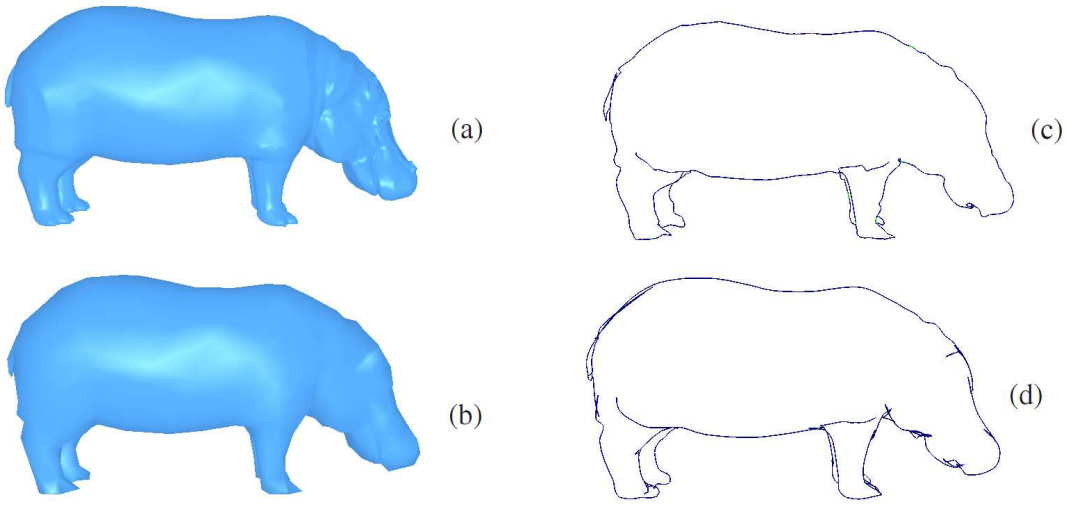


Figura 5.3: Reconstrução da silhueta a partir de um objeto renderizado com 1000 triângulos

5.3 Visualização em baixa resolução

Na figura 5.4 mostramos a suavização nos contornos dos objetos mostrados, respectivamente, nas figuras 5.2(b) e 5.3(b). Em (a) adicionamos 584 novos triângulos no contorno do objeto, com isso o objeto fica com 1084 triângulos. Já em (b) adicionamos 1312 triângulos no contorno, com isso, saímos de 1000 triângulos (*quantidade de triângulos do objeto da figua 5.2(b)*) para 2312 triângulos. Queremos observar que originalmente nas figuras 5.2(a) e 5.3(a), temos esses objetos renderizados, respectivamente, com 15.014 e 46.00 triângulos.

5.4 Mapeamento de textura

Implementamos, também, o artigo [6], onde obtivemos bons resultados com exemplos simples. Devido a falta de exemplos adequados, não tivemos oportunidade de testar malhas mais complexas.

Na figura 5.6 ilustramos uma textura mapeada em uma região simples de uma

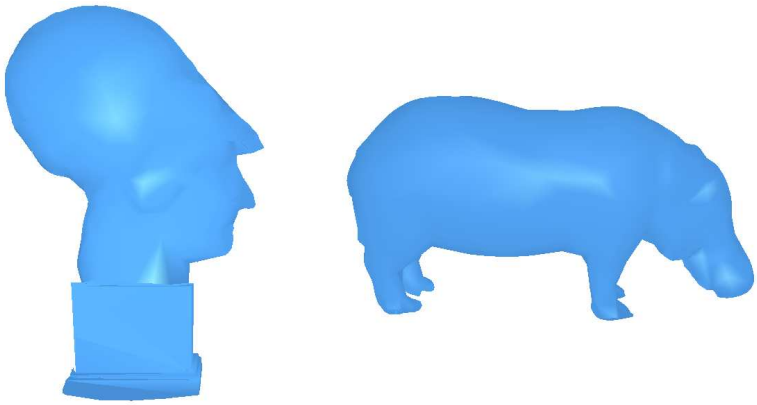


Figura 5.4: Objetos de baixa resolução com suavização nos contornos

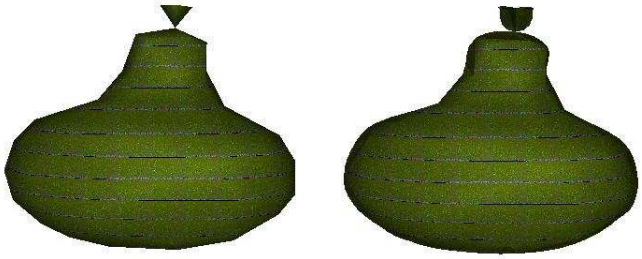


Figura 5.5: Objeto antes e depois da aplicação do processo de suavização

superfície. Depois do colapso de uma aresta obtemos a região com apenas três faces. Quando passamos à textura observamos uma distorção (*veja figura* 5.7). Por fim, fazemos um tratamento na textura na tentativa de compensar a distorção causada pelo colapso da aresta antes de aplicarmos a malha. *Veja a figura* 5.8.

Podemos observar na figura 5.8 que a distorção, aparentemente, desapareceu .

Mostramos nas figura 5.9 e 5.10 mais dois resultados de tratamento da textura, quando uma aresta é colapsada.

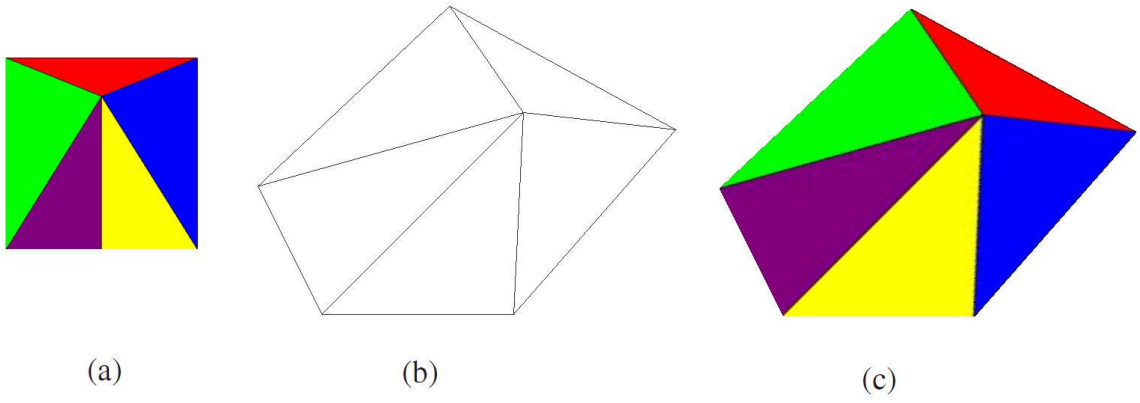


Figura 5.6: (a) Textura; (b)Wireframe; (c) Mapeamento da textura em uma região simples;

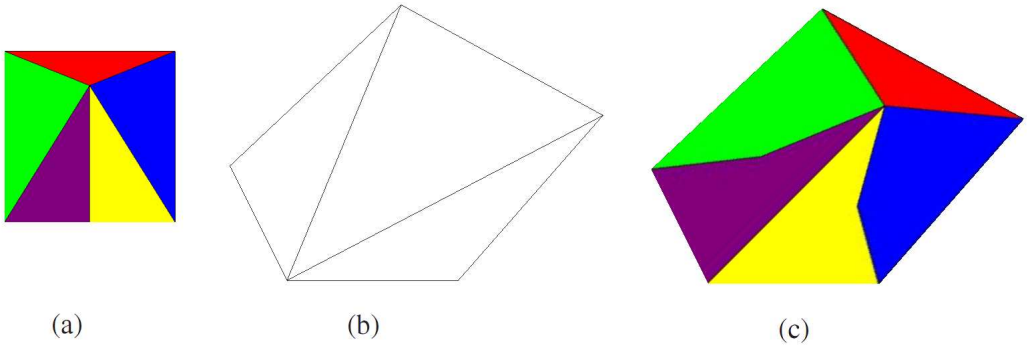


Figura 5.7: Colapso de uma aresta (*nova região com três faces*) e textura original

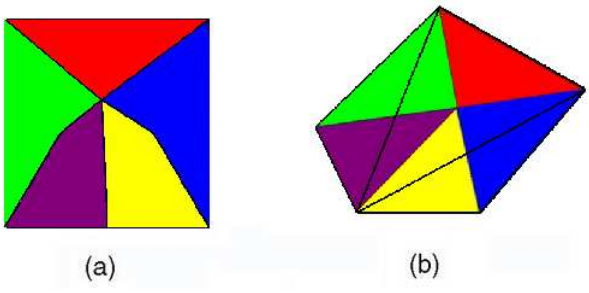


Figura 5.8: Aplicação da textura deformada à região com três faces (figura 5.7 (b))

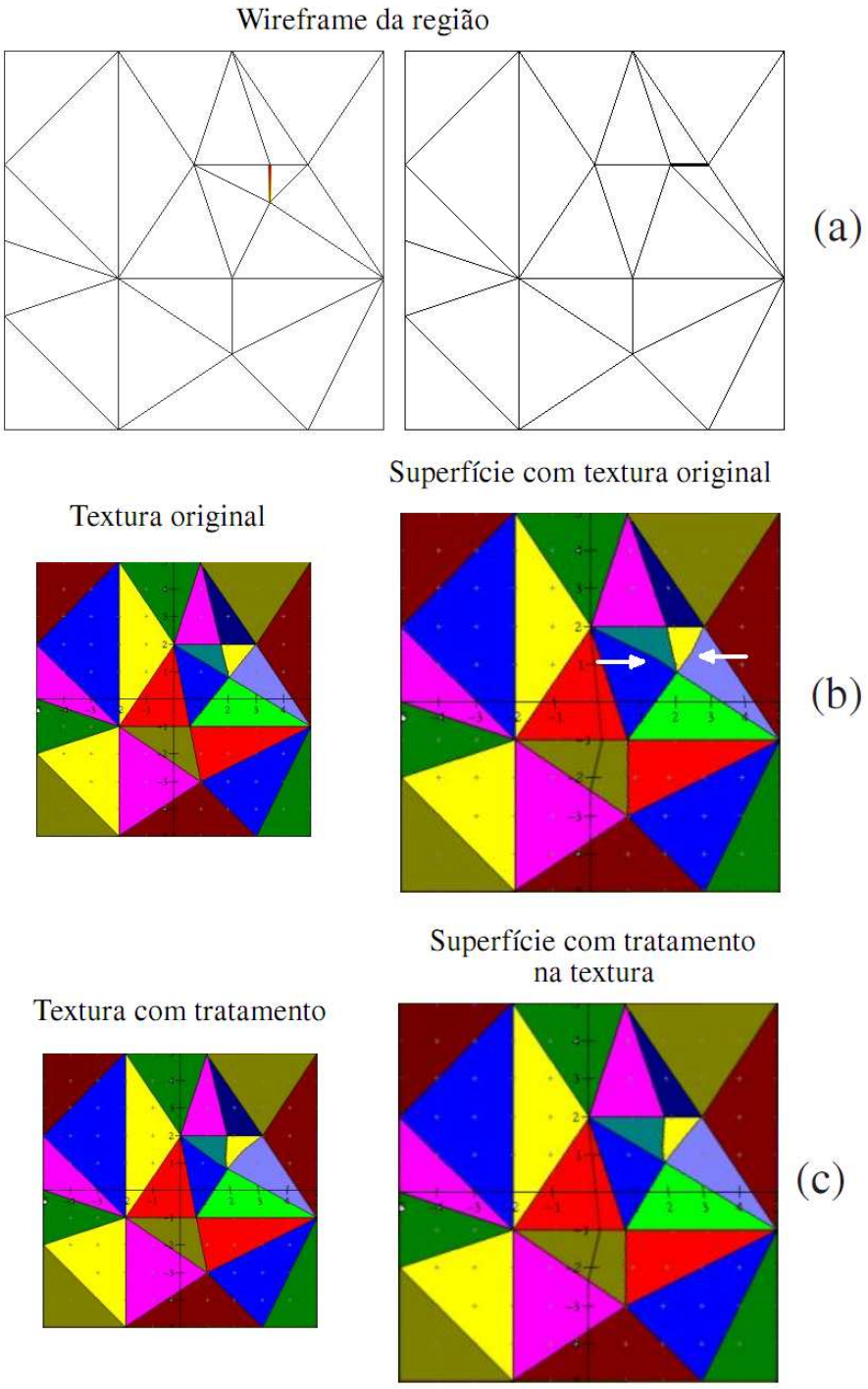


Figura 5.9: (a) Wireframe das regiões antes e depois do colapso de uma aresta; (b) Textura original passada à região depois do colapso. As setas brancas apontam deformações na superfície; (c) Textura com tratamento passada à região depois do colapso.

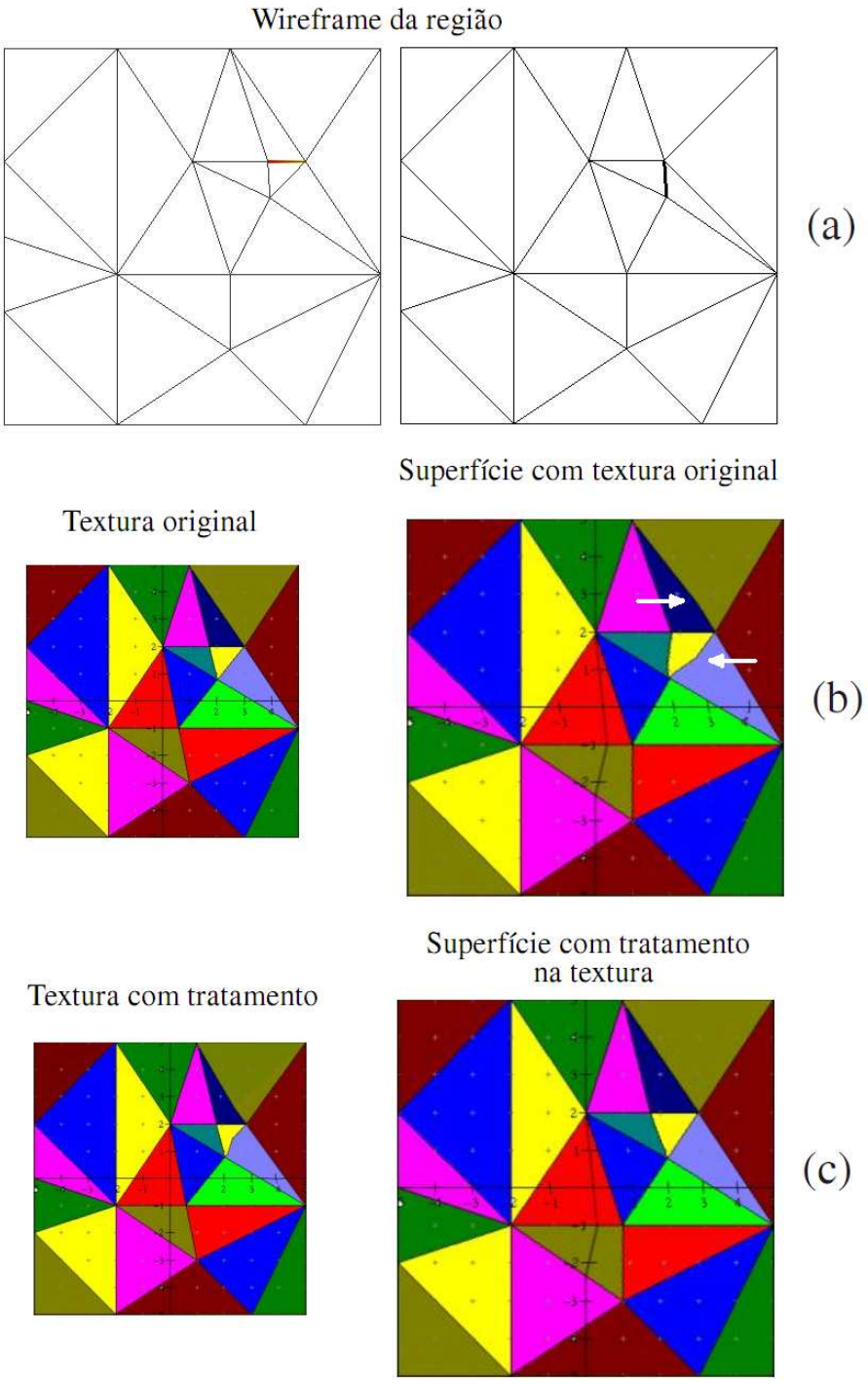


Figura 5.10: (a) Wireframe das regiões antes e depois do colapso de uma aresta; (b) Textura original passada à região depois do colapso. As setas brancas apontam deformações; (c) Textura com tratamento passada à região depois do colapso.

Capítulo 6

Conclusões e Trabalhos Futuros

Este capítulo apresenta as conclusões deste trabalho e suas direções futuras

6.1 Conclusões

Neste trabalho discutimos o problema de visualização de malhas cujas resolução e textura são adaptadas ao ponto de vista do obervador. Implementamos um método para a adaptação da geometria de malhas de baixa resolução [29] e discutimos outro método para adaptação da geometria de malhas de alta resolução [21]. Implementamos ainda um método para adaptação da textura durante o processo de simplificação da malha [6].

Conforme vimos no capítulo anterior a suavização da silhueta em malhas de alta resolução [29] melhora muito o aspecto visual, suavizando os "bicos" causados pela baixa resolução obtida com a simplificação. Para essa suavização usamos uma *interpolação cúbica de Hermite*. Sobre o método em [29], não programamos na GPU, como indicado no artigo. No entanto, toda a estrutura das pontes de silhueta, pontos de silhueta, segmentos de silhueta, foram implementados com todo

rigor. Notamos que a não *perturbação* da malha pode causar mais problemas do que o esperado, pois na extração da silhueta muitas curvas ficaram por trás do objeto (*figura 6.1*).

Em [21], temos uma malha de alta resolução como entrada. Seu método faz a simplificação de regiões com base em um *erro heurístico*, de modo que não prejudiquem a visualização do objeto (*como, por exemplo, regiões escondidas*) e nem mudem sua topologia.

Em conjunto com estes métodos, discutimos um trabalho de adaptação de textura para a simplificação de malhas [6], do qual fizemos alguns testes de implementação. A adaptação de textura é feita na tentativa de compensar a distorção causada pelo colapso de arestas. Pelos resultados mostrados no capítulo anterior [6] e (*figura* 5.8) vimos que o tratamento da textura remove significativamente as distorções na textura causadas pelo colapso de arestas.

6.2 Trabalhos Futuros

Como trabalho futuro, acreditamos, com base nos resultados obtidos isoladamente, que a junção destes métodos se mostram adequados para obtermos uma visualização de malhas em tempo real e de qualidade. Pois poderíamos mudar a resolução da malha obtendo um equilíbrio entre resolução e visualização e ainda poderíamos fazer com que a textura acompanhe essa mudança, sem distorções aparentes.

Como ponto de partida para os trabalhos futuros, pensamos no desenvolvimento de métodos para detectar e decidir quais regiões da malha devem ser simplificadas, quais devem ser refinadas e quais devem ser mantidas. Junto a [29] queremos analisar condições para indentificar regiões que não devem ser suavizadas, mesmo estando na silhueta. Por exemplo, na figura 5.2 (c) a silhueta na base da *Athena* sofre uma suavização, onde o ideal seria que o pedestal mantivesse suas arestas retas.

Pretendemos ainda implementar a estrutura de aglomeração hierárquica na *esfera de Gauss* e a perturbação local para o caso de quando a curva de interpolação cúbica ficar por trás da superfície, que corrige problemas como os apresentados no toro figura 6.1.

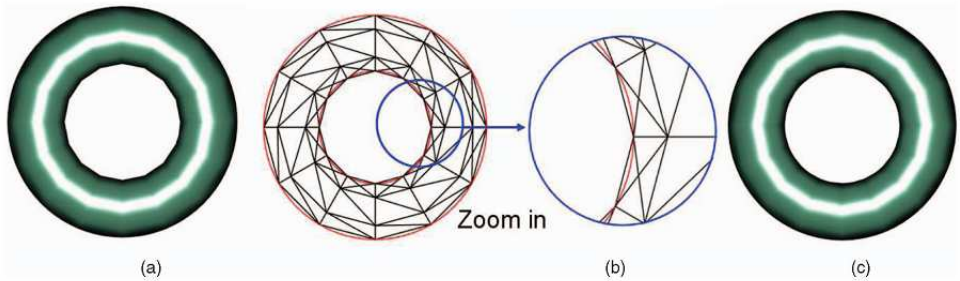


Figura 6.1: (a) Antes da perturbação; (b) Malha antes da perturbação; (c) Depois da perturbação (*Visualização de* [29])

Além disso, pensamos em:

- desenvolver o tratamento da textura, como por exemplo, em regiões com valência diferente de cinco;
- implementar uma *malha progressiva* para ajudar tanto no mapeamento, quanto na resolução no momento em que a malha é alterada;

- incluir um erro para medir o *"tamanho"* da distorção causado na textura pelo colapso de uma aresta particular.

Ainda no tratamento da textura, pensamos em incluir, semelhante ao feito em [6], um *mapeamentos de índices* para que não seja necessário que a alteração da textura seja feita a cada colapso. Este deverá armazena em cada face as coordenadas de textura, ao longo do refinamento, referentes ao mapeamento de textura original $\mathcal{T}$.

Portanto é natural idealizarmos, com a composição destes métodos, uma visualização mais suave em tempo real e de grande qualidade.

Referências Bibliográficas

[1] AZUMA, Daniel *et al. View-Dependent Refinement of Multiresolution Meshes with Subdivision Connectivity*, Proc. Second Int Conf.Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa (AFRIGRAPH 03), pp. 69-78, 2003.

[2] BAUMGART, Bruce G. *Winged-Edge Polyhedron Representation for Computer Vision.* Disponível em: ¡http://www.baumgart.org/winged-edge/winged-edge.html¿. Acesso em: 10 jan. 2009.

[3] CHEW, L.P. *Guaranteed-quality triangular meshes*. Technical Report TR 89-983. Department of Computer - Cornell University, 1989.

[4] CIGNONI P. *et al. Preserving Attribute Values on Simpli?edMeshes by Re-sampling Detail Textures*.The Visual Computer 15, 10 (1999), 519539.

[5] CLARK, James H. *Hierarchical Geometric Models for Visible Surface Algorithms,* Comm. ACM, vol. 19, n^o. 10, pp. 547-554, 1976.

[6] CHEN, Chih-Chun; CHUANG, Jung-Hong. *Texture Adaptation for Progressive Meshes*. EUROGRAPHICS 2006 / E. Gröller and L. Szirmay-Kalos Volume 25 (2006), Number 3.

[7] COHEN, Marcelo; MANSSOUUR, Isabel Harb. OpenGl - *Uma aboradagem Prática e Objetiva*. Novatec Editora. São Paulo, SP - Brasil, 2006.

[8] DYKEN, Christopher; REIMERS, Martin. *Real-Time Linear Silhouette Enhancement*. Proc. Math. Methods for Curves and Surfaces, pp. 135-143, July 2004.

[9] ESPERANÇA, Claudio; CAVALCANTI, P. Roma. *Introdução a computação gráfica*.

[10] FLOATER, Michael S.; HORMAN, Kai. *Surface Parameterization: a Tutorial and Survey. Computer Science Department Oslo University, Norway*.

[11] FLORIANI, Leila; PUPPO, Enrico. *Hierarchical Triangulation for Multiresolution Surface Description,* ACMTrans. Graphics, vol. 14, no. 4, pp. 363-411, 1995.

[12] GARLAND; M. HECKBERT; P. S. *Simplifying Surfaces with Color and Texture Using Quadric Error Metrics*. In Proceedings of IEEE Visualization - 98 (Oct.1998), p. 263269.

[13] GARLAND, Michael; HECKBERT, Paul. *Surface Simplification Using Quadric Error Metrics.* Proc. SIGGRAPH 97, pp. 209-216, 1997.

[14] GOMES, Jonas; VELHO, Luiz. *Fundamentos da Computação Grágica*. Instituto Nacional de Matemática Pura e Aplicada, 2005.

[15] HERTZMANN, Aaron e ZORIN, Denis,*Illustrating Smooth Surfaces*. Proc. ACM SIGGRAPH 00, pp. 517-526, 2000.

[16] HOPPE, Hugues. *Progressive Meshes*. In Proc. SIGGRAPH 96, Rushmeier. pp. 99108 - 1996.

[17] HOPPE, Hugues; *et al. Mesh Optimization*. Proceeding of SIGGRAPH 93. pp. 19-26. 1993.

[18] KÄHLER, Kolja; HABER, Jorg; SEIDEL, Hans-Peter. *Dynamic Refinement of Deformable Triangle Meshes for Rendering*.

KI KIM, H. e WOHN K. *MultiresolutionModel Generation with Geometry and Texture*. In Seventh International Conference on Virtual Systems and Multimedia (VSMM01) (Oct 2001), IEEE, pp. 780789.

[19] LUEBKE, David; REDDY, Martin; *et al. Level of Detail for 3D graphics*. Editora Morgan Kaufmann. San Francisco(EUA), 2003.

[20] MOLINARI, M *et al. Adaptive mesh refinement techniques for electrical impedance tomograph*. Institute of Physics (91 - 96), 2001.

[21] PAJAROLA, Renato; DECORO, Christopher. *Efficient Implementation of Real-Time View-Dependent Multiresolution Meshing*. Ieee Transactions on Visualization And Computer Graphics, VOL. 10, NO. 3, MAY/JUNE 2004

[22] PAJAROLA, Renato. *Fastmesh: Efficient View-Dependent Meshing,* Proc. Pacific Graphics 2001, pp. 22-30, 2001.

[23] PEIXOTO, Adelailson. *Extração de Malhas Adaptativas em Multi-resolução a partir de Volumes, usando Simplificação e Refinamento*. Tese (Doutorado). Pontifícia Universidade do Rio de Janeiro, Departamento de Informática. Rio de Janeiro, PUC-Rio, 2002.

[24] P.S. Heckbert and M. Garland, *Survey of Polygonal Surface Simplification Algorithms,* SIGGRAPH 97 Course Notes 25, 1997.

[25] P. Cignoni, C. Montani, and R. Scopigno, *A Comparison of Mesh Simplification Algorithms.* Computers Graphics, vol. 22, no. 1, pp.37-54, 1998.

[26] SANDER, P.V. *et al*, *Silhouette Clipping*. Proc. ACM SIGGRAPH 00, pp. 327-334, 2000.

[27] SANDER, P. V. *et al. Texture Mapping Progressive Meshes*. In Proc. SIGGRAPH 2001 (New York, NY, USA, 2001), ACM Press, pp. 409416.

[28] SOUZA, Rafael Landim *et al. Visualização Anatômica de Estruturas Vasculares a partir de Imagens de Tomografia Computadorizada.* Disponível em: ¡http://www.sbis.org.br/cbis9/arquivos/487.pdf¿. Acesso em: 10 jan. 2009.

[29] WANG, Lu; *et al. Silhouette Smoothing for Real-Time Rendering of Mesh Surfaces*. IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS, VOL. 14, NO. 3, MAY/JUNE 2008.

[30] WANG, B. *et all. Silhouette Smoothing by Boundary Curve Interpolation*. Proc. Eighth Int Conf. Computer-Aided-Design and

[31] Vlachos A. *et al. Curved PN Triangles*. Proc. Symp. Interactive 3D Graphics (I3D 01), pp. 159-166, 2001.