Geração Adaptativa de Malha Baseada em Erro de Curvatura

Daniel Lima Sousa

Dissertação apresentada ao
Mestrado em Ciência da Computação
Universidade Federal do Ceará

Orientador: Joaquim Bento Cavalcante Neto
Co-orientador: Creto Augusto Vidal

Fortaleza, Setembro de 2004

Dedico este trabalho à minha maravilhosa família: meus pais, Adalberto Carvalho de Sousa e Maria do Rozário Lima Sousa, meus irmãos, Adriana, Danilson e Danilo, e minha avó, Joana de Carvalho Lima.

Agradecimentos

Em primeiro lugar, agradeço a Deus pela oportunidade a mim ofertada e por ter traçado esse caminho tão iluminado.

A minha família pelo apoio, carinho e incentivo e em especial a meus pais pela educação dos quais recebi.

Aos meus orientadores, os professores Joaquim Bento e Creto Vidal, pelo apoio, incentivo, amizade e confiança, sem eles não seria possível a realização deste trabalho. Ao professor Romildo, pelo apoio, incentivo e pela suas idéias que contribuíram para o êxito do trabalho.

A todos os meus amigos que nunca desistiram de minha amizade, mesmo tendo que me afastar, por algum tempo, por motivos de força maior.

A todas as pessoas que contribuíram de forma direta ou indireta para a realização deste trabalho.

Resumo

Esse trabalho descreve uma estratégia de adaptação para geração de malhas, utilizando um critério geométrico. Esse critério é baseado nos erros entre as curvaturas (analítica e discreta), calculadas para as regiões da malha. A curvatura analítica é representada pela formulação matemática usada para modelar o domínio e a curvatura discreta é uma aproximação dessa curvatura, que depende da malha utilizada. É importante ressaltar que a estratégia funciona para qualquer tipo de geometria, já que o processo adaptativo é feito no espaço paramétrico e, portanto, a estratégia trata tipos de curvas e superfícies de uma forma genérica. A estratégia foi implementada seguindo o paradigma de orientação a objetos, onde esse tratamento genérico é facilmente considerado, permitindo inclusive expansões para inclusão de novos tipos de curvas e superfícies, de maneira muito rápida e eficiente.

Abstract

This work describes a strategy for adaptive mesh generation, using a geometric criterion. This criterion is based on the error between the curvatures (analytic and discrete), calculated for the regions of the mesh. The analytic curvature is represented by the mathematical formulation used to model the domain and the discrete curvature is an approximation of this curvature, which depends on the mesh used. It is important to mention that the presented strategy works for any type of geometry, since the adaptive process is done in the parametric space and, for this reason, the strategy is able to deal with all kinds of curves and surfaces in a generic way. The strategy was implemented based on an object oriented paradigm, where the generic treatment was easily considered, allowing expansions for inclusion of new types of curves and surfaces, in a faster and efficient way.

Índice

Lista de Figuras

Lista de Tabelas

Lista de Símbolos

P(u) – representa um ponto no espaço tridimensional para uma curva em função do parâmetro u, com $0 \leq u \leq 1$.

$\mathbf{P_i}$(u) – representa o ponto inicial de uma curva de Hermite no espaço tridimensional em função do parâmetro u.

$\mathbf{P_j}$(u) – representa o ponto final de uma curva de Hermite no espaço tridimensional em função do parâmetro u.

$\mathbf{DP_i}$(u) – representa a derivada do ponto inicial de uma curva de Hermite no espaço tridimensional em função do parâmetro u.

$\mathbf{DP_j}$(u) – representa a derivada do ponto final de uma curva de Hermite no espaço tridimensional em função do parâmetro u.

P(u,w) – representa um ponto no espaço tridimensional para um *patch* em função dos parâmetros u e w, com $0 \leq u \leq 1$ e $0 \leq w \leq 1$.

$\mathbf{P_u}$(u,w) – representa a derivada na direção u de um ponto **P**(u,w) para um *patch*.

$\mathbf{P_w}$(u,w) – representa a derivada na direção w de um ponto **P**(u,w) para um *patch*.

$\mathbf{P_{u\,w}}$(u,w) – representa a derivada parcial na direção u e w de um ponto **P**(u,w) para um *patch*. Cada $\mathbf{P_{u\,w}}$(u,w) é um vetor twist.

$\boldsymbol{v}$ – vértice de um *patch.*

$\boldsymbol{k_a}$ – curvatura analítica.

$\boldsymbol{k_d}$ – curvatura discreta.

$\boldsymbol{Q_u}$ – derivada parcial em relação ao parâmetro u, ou seja, $Q_u = \partial Q/\partial u$.

$\boldsymbol{A_{adjcente}}$ – área de cada vértice *xi* utiliza para o cálculo das curvaturas discretas.

***Abs*(x)** – função que retorna o valor absoluto do real x.

η - erro calculado no modelo.

$\eta_{\text{vértice}}$ – erro calculado no vértice.

Capítulo 1

Introdução

Nesse capítulo, analisa-se a importância de estratégias adaptativas para geração de malha e discute-se porque essas estratégias são relevantes no contexto desse trabalho. Além disso, são discutidos os objetivos do trabalho e é mostrada a sua organização geral.

1.1 Motivação e trabalhos relacionados

Geração de malha é uma técnica de representação discreta de um domínio geométrico qualquer, em forma de elementos mais simples. Esse assunto vem ganhando destaque desde a década de 70, quando foram propostos algoritmos para geração de malhas baseados em técnicas de geometria computacional. Diagrama de Voronoi [(Berg et al., 1987), (O'Rourke, 1998)], triangulações de Delaunay [(Chew, 1989), (Lee & Hobbs, 1999), (Joe, 1986), (Lo, 1989)][1] e outras técnicas formaram a base para a implementação de algoritmos para geração de malhas.

O tempo de processamento para a geração de uma malha depende, entre outros fatores, do tamanho da discretização desejada, ou seja, do número de elementos desejados nas diferentes regiões do modelo.

A qualidade da malha gerada é outro aspecto importante a ser considerado e pode ser medida pela quantidade e forma dos elementos que a compõem. Assim, muitas vezes é desejável que cada elemento se aproxime de uma forma específica. Em malhas triangulares bidimensionais, por exemplo, é comum ser desejável que cada elemento se aproxime de um triângulo eqüilátero.

Essa qualidade da malha é fundamental para várias aplicações. Em análise pelo método dos elementos finitos, por exemplo, a qualidade da malha tem uma influência fundamental na qualidade dos resultados ou até mesmo na convergência da solução. Em computação gráfica, por sua vez, a qualidade da malha tem influência direta no resultado visual dos modelos. Na Figura 1.1, por exemplo, mostra-se o efeito da qualidade da malha

[1] Existem mais trabalhos sobre esses assuntos do que os mencionados, mas todos seguem conceitos similares.

na renderização da metade de uma face. Pode-se notar a diferença na qualidade da representação do modelo em questão, principalmente em regiões de alta curvatura, como o nariz e a boca, considerando-se a malha à esquerda (menos refinada) e à direita (mais refinada). A Figura 1.2 traz um *zoom* dessas regiões, realçando essa diferença.

Figura 1.1: Face modelada.

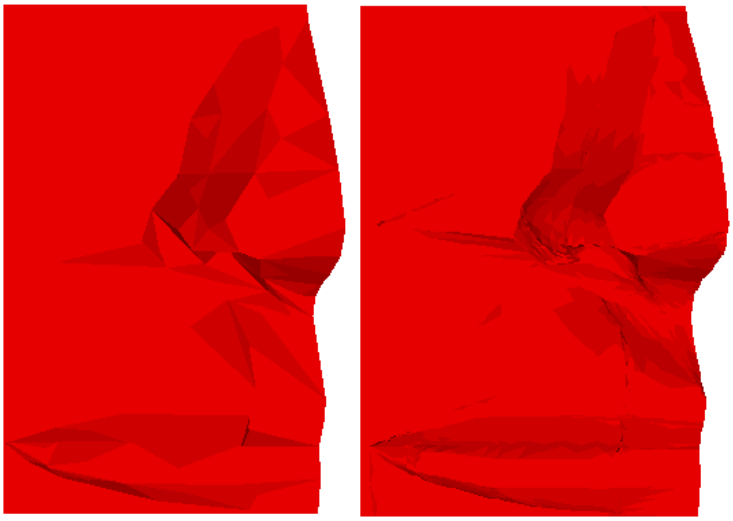

Figura 1.2: Detalhes da face (nariz e boca).

Assim como a qualidade da malha, medidas de avaliação dessa qualidade são também fundamentais. Existem várias medidas para avaliar a qualidade dos elementos de uma malha. Somente para citar uma dessas medidas, que é uma das mais usadas, existe o chamado *aspect ratio*, que pode ser calculado de várias formas [(Modi, 1997), (Miranda et

al., 1999)], e que em geral envolve, de alguma maneira, a razão entre os círculos inscritos e circunscritos a um triângulo, em duas dimensões.

Apesar da geração de malhas de boa qualidade ser importante, gerar malhas de boa qualidade nem sempre é uma tarefa fácil, pois depende da estratégia em si, bem como da experiência do usuário ou de alguma "inteligência" existente no sistema gerador de malhas para definir que discretização é necessária para a malha, ou seja, quantos elementos são necessários para as diferentes regiões do modelo. Além disso, são necessários mecanismos para garantir que a malha gerada seja de boa qualidade para essas regiões.

Dentro desse contexto, um tema que vem sendo discutido com grande ênfase é a adaptabilidade das malhas. Como o nome sugere, técnicas adaptativas são aquelas que tentam melhorar a malha de acordo com algum critério adotado, modificando-a se necessário. Essas modificações podem ser feitas de várias formas. Por exemplo, elementos da malha podem ter seus tamanhos reduzidos ou aumentados; elementos podem ser apagados e novos elementos gerados nos seus lugares. Em geral, em uma estratégia adaptativa, uma malha inicial deve ser considerada e então aperfeiçoada até que se obtenha um parâmetro que esteja dentro de um critério adotado. Esse critério de adaptação é um ponto crucial para o processo, pois a cada passo deve-se saber se a malha atual já se encontra na forma que atende as exigências do critério adotado. É importante, portanto, que a forma de comparação de tal critério não seja caro computacionalmente.

A vantagem dessa estratégia adaptativa é que, dentre outros aspectos, o usuário ou o sistema não precisam se preocupar com a malha que está sendo gerada, pois ela será somente a malha inicial, para um processo de otimização da mesma, até atingir-se uma malha final de qualidade garantida.

Dada a importância crescente de métodos adaptativos para geração de malhas, tem crescido muito a pesquisa nessa área. Desde o início da década de 70, a adaptação para malhas vem ganhando destaque na literatura, sendo que os primeiros trabalhos foram desenvolvidos utilizando a adaptação da malha para o método dos elementos finitos. Foi no final da década de 70 que Babuska e Rheimboldt (1979) conceituaram a adaptação automática da malha baseada em estimadores de erros. Nessa ocasião, a computação gráfica passou a ser utilizada para visualização de malhas, permitindo ao analista minimizar erros existentes nos modelos.

Existe uma infinidade de trabalhos na literatura que utilizam uma estimativa de erro, baseada em uma análise numérica de elementos finitos, como critério para adaptação da malha inicial [(Cavalcante-Neto et al., 1998), (Bank et al., 1997), (Bond et al., 1996), (Cheng & Topping, 1998), (Katragadda & Grosse, 1996), (Kiahs et al., 1995), (Lee & Hobbs, 1999), (Cavalcante-Neto, 1994)]. Os estimadores buscam a malha ótima tentando igualar o valor do erro nos diversos elementos. Sistemas adaptativos que se baseiam na análise de elementos finitos procuram reduzir o erro de discretização do modelo.

O uso da análise de elementos finitos, como critério de adaptação para malha, supõe que uma solução numérica do modelo matemático seja obtida, já que a estimativa de erro é baseada na resposta da análise (tensões), o que nem sempre faz sentido ou é conveniente e necessário. Em várias aplicações onde computação gráfica é usada, como, por exemplo, aplicações de modelagem e renderização, o que se deseja é somente uma estratégia que gere malhas de boa qualidade, sem recorrer a análises numéricas. Além disso, em muitos casos, o custo computacional dessas análises numéricas é alto. É necessário, então, utilizar outro critério de adaptação que consiga otimizar a malha e que tenha uma boa convergência, além de não ser caro computacionalmente. Uma idéia é usar um critério geométrico e não de elementos finitos para esses problemas.

Existem trabalhos na literatura que tratam de processos adaptativos dessa forma, ou seja, que não sejam baseados em análises por elementos finitos. Alguns desses trabalhos tentam eliminar "ruídos" na malha (proveniente de escaneamento não preciso, por exemplo), suavizar e melhorar a malha, baseando-se em cálculos da curvatura nas regiões dessa malha, como, por exemplo, o trabalho apresentado por Meyer et al. (2002). Outros trabalhos recentes já levam em consideração as curvaturas quando da geração da malha, como, por exemplo, o trabalho apresentado por Miranda & Martha (2002).

As estratégias adaptativas podem ser classificadas, em geral, como *a priori* e a *a posteriori*. O que se procura, em estratégias *a priori*, é se cercar de cuidados para que a geração da malha já seja de boa qualidade, o que é válido e importante, mas nem sempre possível. Por isso, muitas vezes opta-se por uma estratégia *a posteriori*, onde se parte de uma malha inicial já gerada e tenta-se melhorar a sua qualidade, como é o caso da estratégia adaptativa proposta nesse trabalho.

1.2 Objetivos do trabalho

No contexto de adaptação de malhas, para problemas não baseados em análise por elementos finitos, é necessária uma estratégia que considera apenas critérios geométricos. Nesse trabalho, é proposta uma estratégia que leva em consideração, como critério de adaptação das malhas, erros nas curvaturas (analítica e discreta), calculadas para os triângulos da malha. A curvatura analítica é representada pela formulação matemática usada para modelar o domínio e a curvatura discreta é uma aproximação dessa curvatura, que depende da malha utilizada. Baseado nisso, os principais objetivos desse trabalho são:

- Propor uma estratégia de auto-adaptação para malhas utilizando um critério geométrico, que possa ser aplicado em problemas de computação gráfica ou em outros problemas quaisquer, onde a adaptação é baseada em geometria.
- Discutir a estratégia utilizada na auto-adaptação, bem como sua implementação.
- Apresentar resultados da aplicação da estratégia proposta, através de exemplos ilustrativos, para que se possa avaliar e perceber as vantagens da técnica utilizada.

Deve ficar claro que não é objetivo desse trabalho discutir a acurácia dos operadores de curvatura discreta que foram implementados. Esses operadores são baseados no trabalho de Meyer et al. (2002), e não foram modificados ou adaptados. O enfoque principal desse trabalho é a estratégia proposta de redefinição da malha bem com o critério geométrico utilizado em tal estratégia.

Para o processo de adaptação, um modelo inicial é considerado. Este modelo é formado pela descrição geométrica dos vários retalhos (*patches*) que compõem o modelo, assim como suas malhas iniciais. A malha inicial do modelo, assim como sua geometria, servirá de entrada para o processo adaptativo, que vai gerar a malha final, de melhor qualidade.

1.3 Organização do trabalho

Esse trabalho está organizado em 7 capítulos. No Capítulo 2, é apresentada a estratégia de auto-adaptação utilizada. Nesse capítulo, uma visão geral da metodologia adotada é explicada, mostrando-se todas as fases que envolvem o processo adaptativo.

No Capítulo 3, são apresentados os conceitos, desenvolvimento matemático e aspectos de implementação para o cálculo das curvaturas. Essas curvaturas serão usadas para o cálculo das estimativas de erro. Este capítulo mostra o papel da análise de curvatura dentro do processo de auto-adaptação.

No Capítulo 4, explica-se como é feita a primeira etapa da auto-adaptação, que é o refinamento das curvas de fronteiras. Essa discretização das curvas de bordo é feita independentemente da discretização do domínio do modelo.

No Capítulo 5, é detalhada a geração da malha no interior do domínio, processo este que se subdivide em dois: geração de malha no interior do domínio através da uma árvore quaternária (*quadtree*) e geração da malha de contorno.

O Capítulo 6 é dedicado exclusivamente a exemplos, mostrando a eficácia da técnica usada neste trabalho. Finalmente, no Capítulo 7, são apresentadas as conclusões do trabalho e algumas sugestões para trabalhos futuros.

Capítulo 2

Estratégia de auto-adaptação

Nesse capítulo, descreve-se uma visão geral da estratégia de auto-adaptação proposta nesse trabalho. Nos capítulos seguintes, detalhes específicos de cada fase da estratégia são fornecidos. Para o processo de auto-adaptação proposto, é fornecido, como já mencionado, um modelo inicial que é composto pela descrição geométrica dos vários *patches* que compõem o modelo, assim como de suas malhas iniciais. A malha inicial do modelo e sua geometria servirão de entrada para o primeiro passo do processo auto-adaptativo.

A estratégia de auto-adaptação adotada envolve duas etapas: a análise de erro da curvatura e a adaptação da malha. Inicialmente, é feita uma análise de erro baseada na curvatura considerando a malha inicial do modelo, ficando para a segunda etapa o processo de adaptação da malha em questão. Uma nova malha é gerada e um ciclo, contendo essas duas etapas, se repete até que um critério de erro global pré-definido seja atendido. Um aspecto importante a ser mencionado é que, por se tratar de um processo adaptativo, é necessário que a cada passo se faça uma nova análise de erro na curvatura para a adaptação da malha do passo corrente, levando em consideração a malha do passo anterior. Dentro de cada passo, a auto-adaptação é regida pelo refinamento adaptativo das curvas e das malhas para os *patches* do modelo. A Figura 2.1 ilustra o processo adaptativo proposto.

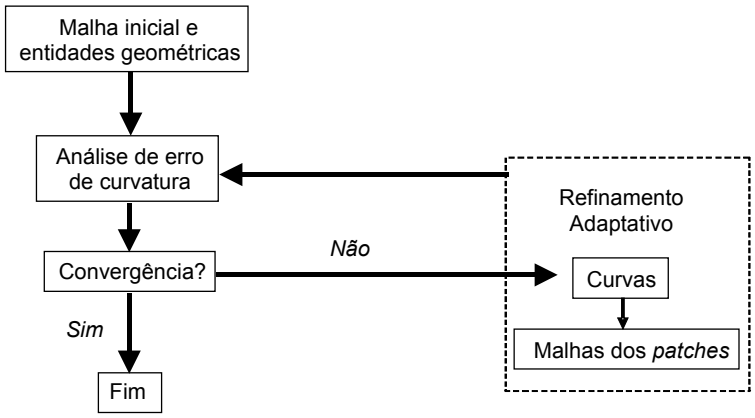

Figura 2.1: Estratégia auto-adaptativa proposta nesse trabalho.

2.1 Geração do modelo inicial

A estratégia de auto-adaptação proposta parte de uma malha inicial para o modelo. Além disso, a estratégia precisa de uma descrição geométrica de cada uma das curvas que compõem o contorno de seus *patches*, bem como das informações geométricas de cada *patch*. As descrições relativas às curvas e aos *patches* podem ser feitas através de um formato pré-estabelecido (ver apêndice B), embora a estratégia funcione para qualquer tipo de formato, desde que as informações geométricas desejadas sejam fornecidas corretamente. A descrição da geometria do modelo é feita considerando-se os *patches* que compõem o modelo e a definição da geometria de suas curvas. Essa geometria é usada para a geração de uma malha inicial para o modelo. É importante ressaltar, entretanto, que a estratégia também funciona para uma malha inicial fornecida.

2.1.1 Descrição da geometria

A geometria do modelo é composta por vários *patches* e estes por suas curvas. Portanto, para cada *patch*, devem ser fornecidas as curvas que o delimitam. Uma determinada curva, por sua vez, pode participar de mais de um *patch* no modelo. Por esse motivo, a descrição da geometria deve ser feita de uma forma hierárquica, ou seja, primeiro deve-se definir as curvas do modelo e em seguida, deve-se definir os *patches*, identificando as suas curvas.

Uma curva é descrita especificando-se suas informações geométricas. Por exemplo, no caso de uma curva de Hermite, são fornecidas as coordenadas tridimensionais de seus pontos extremos e as coordenadas das derivadas em cada ponto. Já a descrição dos *patches* é feita através da identificação das curvas que o compõem.

É importante ressaltar, entretanto, que a estratégia funciona para qualquer tipo de curvas e *patches*, desde que eles possam ser parametrizados de [0 a 1], já que ela trata curvas e *patches* de uma forma genérica. Portanto, a estratégia funciona igualmente tanto para curvas de Bezier e de Hermite, por exemplo, como para um arco e uma reta, entre outras. O mesmo pode-se afirmar com relação aos tipos de *patches*, ou seja, a estratégia trata *patches* de forma genérica, sejam eles *patches* de Bezier, Hermite ou qualquer outro.

A Figura 2.2 ilustra a formação de um *patch* identificado como *PATCH* 1, criado utilizando-se quatro curvas identificadas como CURVA 1, CURVA 2, CURVA 3 e CURVA 4..

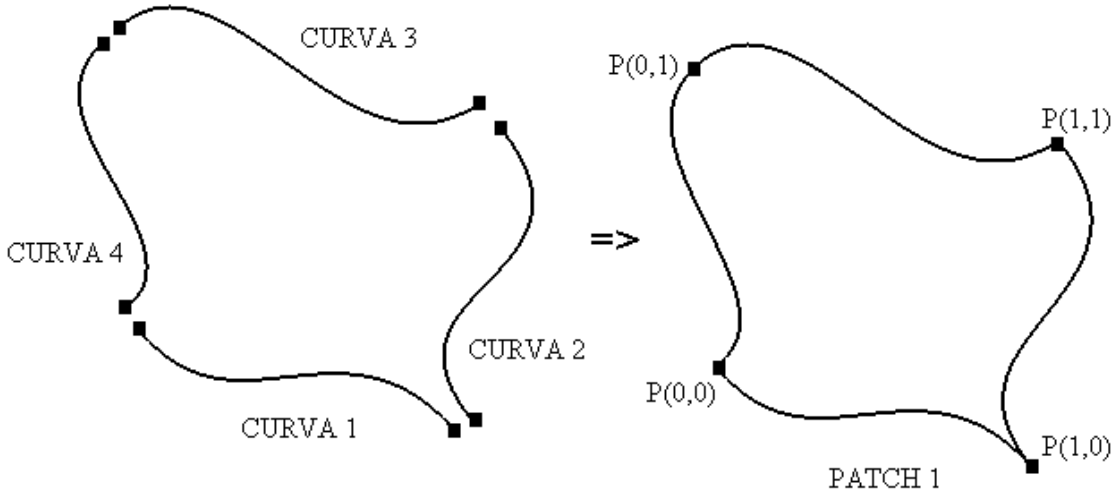

Figura 2.2: Formação de um *patch*.

2.1.2 Geração da malha inicial

A malha inicial do modelo é a junção das malhas iniciais dos *patches* que compõem esse modelo. Uma vez criado um *patch*, é gerada uma malha inicial para esse *patch*, que será uma primeira aproximação para a malha desejada. Essa malha é gerada baseada na discretização das curvas que compõe o *patch*, sendo o número de triângulos gerados para o *patch* baseados nessa discretização. A discretização das curvas, por sua vez, é representada pelo número de segmentos de cada curva e pode ser fornecida pelo usuário, mas nesse trabalho foi adotada uma discretização inicial *default*, como uma aproximação inicial. A Figura 2.3 mostra um exemplo de um *patch* composto pelas curvas CURVA 1, CURVA 2, CURVA 3 e CURVA 4, todas elas com quantidade de segmentos igual a três, produzindo um *patch* com 9 regiões. Esta subdivisão é dada no espaço paramétrico de cada curva e cada *patch* e mapeada para o espaço tridimensional.

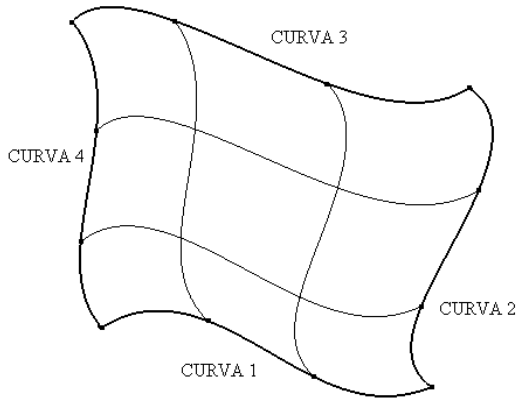

Figura 2.3: *Patch* discretizado com suas curvas.

A malha inicial do *patch* é criada da seguinte forma: para cada região do *patch* discretizado, acha-se o ponto médio no espaço tridimensional e com esse ponto e os quatro vértices do canto da área serão formados quatro triângulos correspondentes àquela área. Para se achar esse ponto médio no espaço tridimensional, inicialmente encontra-se o ponto médio no espaço paramétrico, que é dado pela média das coordenadas paramétricas dos vértices de canto de cada área, e esse ponto médio é então mapeado para o espaço tridimensional. A Figura 2.4 mostra a malha inicial criada utilizando essa idéia para o *patch* da Figura 2.3. Convém ressaltar que a forma como a primeira malha é gerada não é importante para o sistema auto-adaptativo, podendo ela ser formada de uma outra maneira ou mesmo fornecida inicialmente.

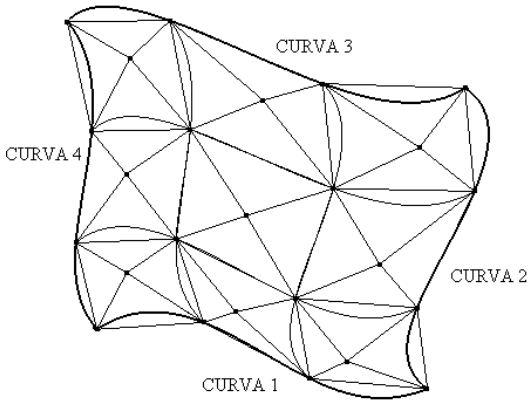

Figura 2.4: Malha inicial para o *patch*.

2.2 Análise de erro baseada no cálculo das curvaturas

Em um sistema auto-adaptativo, o processo de adaptação é regido, em geral, por um critério de erro, que vai guiar esse processo. Por exemplo, o trabalho desenvolvido por Cavalcante-Neto (1994) faz uso, assim como diversos outros trabalhos da literatura [(Banichuk et al., 1995), (Bank & Smith, 1997), (Borouchaki et al., 1996), (George, 1999), (Hinton et al., 1991), (Holzer, 1999), (Katragadda & Grosse 1996)], da análise numérica de elementos finitos para definir o critério de adaptação, onde tal critério se baseia numa estimativa de erro dos elementos. Esse critério, entretanto, em geral não se adequa à computação gráfica, onde o se que precisa é uma boa representação do domínio em questão, mas através de critérios geométricos e não de elementos finitos.

O critério geométrico utilizado neste trabalho é baseado na curvatura de cada vértice que compõe a malha em questão. O processo de auto-adaptação precisa de uma comparação no seu ciclo para determinar se a qualidade da malha já é adequada segundo um fator determinado, ou se o processo irá continuar. No presente trabalho, a comparação do critério de adaptação é feita baseando-se nas curvaturas analíticas e discretas da malha. A idéia do processo é fazer com que a diferença entre essas duas curvaturas fique de acordo com um erro global pré-definido. Esse erro nas curvaturas irá determinar o novo tamanho dos triângulos da malha, para a modificação da malha em questão, se necessário.

O Capítulo 3 traz os detalhes da formulação matemática e da implementação das curvaturas analíticas e discretas para o processo auto-adaptativo proposto.

2.3 Discretização do contorno do domínio

Na estratégia proposta, para a geração auto-adaptativa da nova malha do domínio do modelo, é necessário que se tenha o contorno dos *patches* desse domínio formado por curvas de bordo já discretizadas. Portanto, a primeira fase do processo proposto é a discretização dessas curvas, proveniente do refinamento ou desrefinamento das mesmas. Isto é feito com base nas propriedades geométricas e nos tamanhos característicos (novos tamanhos) dos vértices de borda (adjacentes às curvas), determinados a partir da estimativa de erro calculada na etapa de análise.

Esta discretização das curvas de bordo é feita independentemente da discretização do domínio do modelo e é uma das vantagens do processo proposto, pois isto vai resultar em uma discretização mais regular no contorno e consistente com as características geométricas de suas curvas. A discretização a *priori* também é importante para compatibilizar as malhas dos diversos *patches*. Isto não seria necessariamente obrigatório para aplicações em computação gráfica, mas é conveniente para evitar "buracos" entre os retalhos. Essa estratégia de discretização das curvas de borda independente da discretização do domínio foi originalmente proposta por Cavalcante-Neto (1994), mas a discretização não era realizada através de um critério geométrico como neste trabalho.

O algoritmo usado para refinar o contorno é uma versão unidimensional (em cada curva) do algoritmo utilizado para refinar o domínio, baseado na técnica de *quadtree* [(Baehmann et al., 1987), (Samet, 1984), (Cavalcante-Neto et al. 1993)]. O refinamento de cada curva utiliza uma técnica de enumeração espacial recursiva, baseada em uma estrutura de dados de árvore binária (*binary tree*), que será descrita em detalhes no Capítulo 4. Esse processo de refinamento é feito para todas as curvas do contorno e vai definir a nova discretização para a geração da nova malha do domínio. O refinamento de cada curva, assim como de cada *patch*, é baseada no critério de erro na curvatura. O refinamento de uma curva leva em consideração as suas superfícies adjacentes. Essa estratégia permite que a discretização de uma determinada curva reflita as informações provenientes dessas superfícies. A Figura 2.5 mostra o resultado do refinamento de bordo para o primeiro passo do processo de auto-adaptação para o *patch* da Figura 2.3.

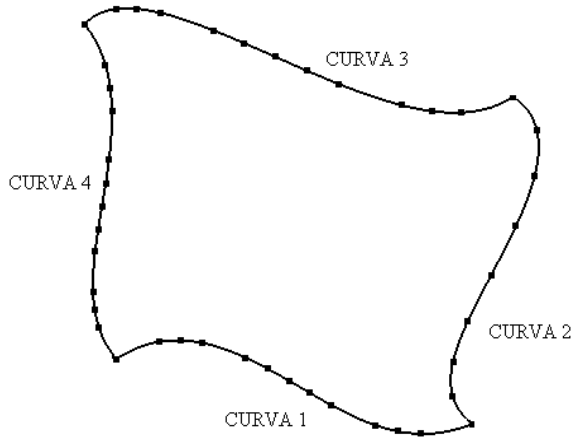

Figura 2.5: Nova discretização das curvas do *patch*.

2.4 Geração da malha no domínio

Com as curvas do domínio já com a nova discretização, baseada no erro de curvatura existente no modelo, a próxima fase é a da geração da nova malha no domínio. O algoritmo usado para refinar o domínio é baseado em uma *quadtree* (Samet, 1984). A *quadtree* foi aplicada ao sistema auto-adaptativo proposto de forma que a malha no interior de um *patch* fique de acordo com uma discretização calculada para suas curvas. Isso é feito da seguinte maneira: gera-se a malha no interior do domínio pelo uso da *quadtree*, deixando uma faixa próxima das curvas para ser discretizada por uma técnica de triangulação por Delaunay [(Joe, 1986), (Chew, 1989), (Lo, 1989)]. Este procedimento, com uma discretização das curvas feita a *priori*, é importante, pois assim pode-se combinar malhas geradas em *patches* adjacentes. A geração da malha no domínio é descrita detalhadamente no Capítulo 5.

O tamanho da célula da *quadtree* é controlado por dois fatores: o tamanho dos segmentos das curvas discretizadas do *patch* e o tamanho característico (novo tamanho) dos triângulos no interior do domínio ditado pela análise de erro de curvatura. Observa-se que esse processo não só refina regiões onde se espera uma discretização mais expressiva, como desrefina regiões que não necessitam de tanto refinamento, se assim necessário. A Figura 2.6 mostra a nova malha para a discretização, de acordo com a Figura 2.5, onde se pode observar refinamento e desrefinamento, além de se notar a boa transição na malha, que é uma característica do algoritmo de *quadtree*, pois ele realmente "sente" a discretização do contorno, já que a criação da malha é fortemente influenciada por ela.

Essa nova malha, com os cálculos das curvaturas consistentemente refeitos, é re-avaliada pela etapa de análise e o ciclo continua até que o critério global desejado seja atendido e se obtenha uma malha final otimizada.

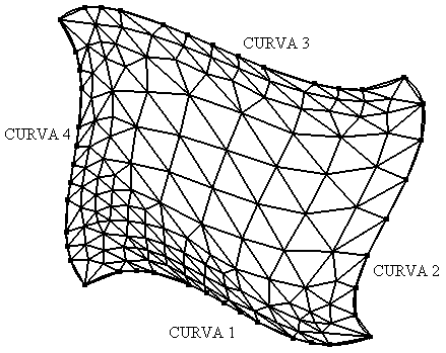

Figura 2.6: Nova malha gerada para o *patch*.

Capítulo 3

Análise de erro baseado em curvatura

A primeira etapa do processo adaptativo proposto nesse trabalho é uma análise de erro baseado em curvatura, que é essencial para determinar as modificações que a malha sofrerá, se necessárias. Para essa análise, é fundamental o cálculo das curvaturas, que são divididas em duas: curvaturas analítica e discreta. A análise do erro entre esses dois tipos de curvaturas é que vai determinar em que parte a malha será refinada ou desrefinada, ou ainda se a malha já se encontra otimizada segundo critérios pré-determinados. Esse capítulo descreve os procedimentos para cálculo das curvaturas analítica e discreta, bem como a análise de erro entre elas. O erro nas curvaturas é que vai guiar o processo adaptativo.

3.1 Cálculo das curvaturas analíticas

Para qualquer vértice *v* sobre um *patch*, a curva gerada pela interseção do *patch* com um plano contendo o vetor normal ao *patch* em *v* e o *patch* tem uma curvatura analítica k_a em *v* (Rogers & Adams, 1990) (Figura 3.1(a)). Existe uma única direção onde, a interseção do plano com o *patch*, produz uma curvatura mínima e uma outra única direção que produz uma curvatura máxima (Figura 3.1(b)), que são chamadas de curvaturas principais.

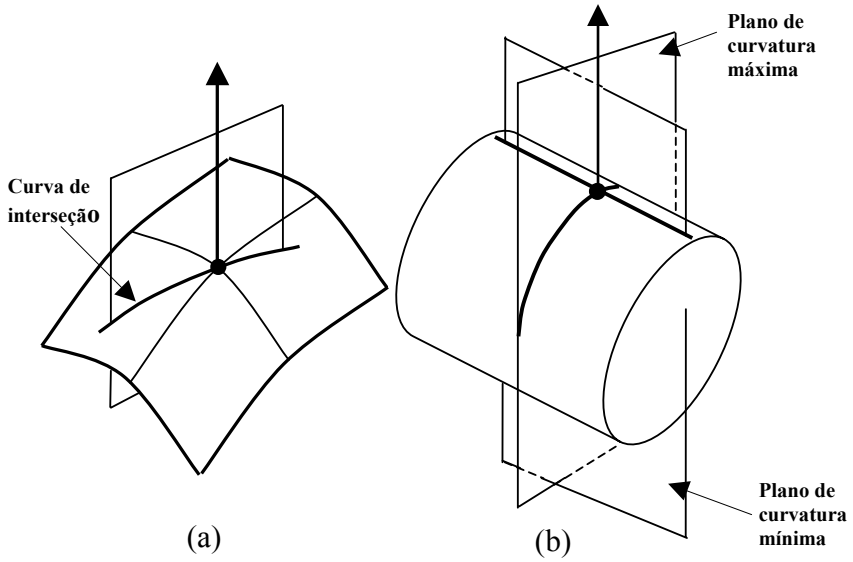

Figura 3.1: Curvatura em um vértice *v*.

Essa curvatura k_a é calculada analiticamente utilizando-se a descrição matemática do *patch* em questão e, dessa forma, obtém-se a curvatura analítica de cada vértice do *patch*. Para o nosso propósito, utilizam-se dois tipos de combinação das curvaturas principais: a curvatura Gaussiana e a curvatura Média, que representam a curvatura ideal estimada para o modelo.

3.1.1 Formulação matemática

Como mencionado, o grande interesse sobre curvaturas analíticas, para o propósito desse trabalho, é com uma combinação das duas curvaturas principais k_{min} e k_{max}, que são as chamadas curvaturas Média (H) e Gaussiana (K), e que são dadas pelas fórmulas (Rogers & Adams, 1990):

$$H = (k_{min} + k_{max}) / 2 \qquad (\mathbf{3.1})$$

e

$$K = k_{min} * \ k_{max} \ . \qquad (\mathbf{3.2})$$

Dill (1981) mostra que, para superfícies bi-paramétricas, a curvatura Média e a curvatura Gaussiana são calculadas pelas seguintes fórmulas:

$$H = (A \mid Q_w \mid^2 - 2B Q_u \cdot Q_w + C \mid Q_u \mid^2) \ / 2 \mid Q_u \mathbf{x} Q_w \mid^3 , \qquad (\mathbf{3.3})$$

e

$$K = (AC - B^2) / \mid Q_u \mathbf{x} Q_w \mid^4 \qquad (\mathbf{3.4})$$

onde

$$(A \ B \ C) = [Q_u \mathbf{x} Q_w] \cdot [Q_{uu} \ \ Q_{uw} \ Q_{ww}] \qquad (\mathbf{3.5})$$

e, Q_u é a derivada parcial em relação ao parâmetro u, ou seja, $Q_u = \partial Q / \partial u$, seguindo a mesma notação para as derivadas parciais Q_w , Q_{uu} , Q_{uw} e Q_{ww}.

Como mostra a Tabela 3.1 (Rogers & Adams, 1990), a curvatura Gaussiana serve para caracterizar a forma local da superfície: elíptica, hiperbólica, cilíndrica e cônica. Isso é mais um motivo da preferência em utilizar a curvatura Gaussiana em primeira instância

nesse trabalho, pois se conhecendo o valor da curvatura (Gaussiana) em um determinado vértice de uma região de um *patch*, pode-se inferir a forma da superfície na região visualizada. Isso irá ajudar na compreensão dos resultados (refinamento/desrefinamento).

k_{min} , k_{max}	K	**Forma**
Mesmo sinal	> 0	Elíptica
Sinais contrários	< 0	Hiperbólica (ponto de sela)
Uma ou ambas zero	0	Cilíndrica/Cônica/Plano

Tabela 3.1: Tipos de superfícies.

3.1.2 Aspectos de implementação

O cálculo das curvaturas analíticas segue o paradigma de programação orientada a objetos. Isso permite, portanto, que se implemente o cálculo dessas curvaturas facilmente para qualquer tipo de curva, sejam elas curvas de Hermite, Bezier ou qualquer outra, em relação ao *patch* ao qual uma determinada curva se relaciona.

Na classe específica de *patch* **CLPCurva** são implementadas os métodos para os cálculos da curvatura analítica. Essa classe se relaciona com os tipos de curvas possíveis, como por exemplo, **CLPHermite** ou **CLPBezier**, para curvas de Hermite e Bezier, respectivamente. O método público **CalculoDeCurvaturas** é o responsável pelo cálculo das curvaturas analíticas, para cada vértice de cada triângulo do *patch*. Esse método utiliza métodos privados da classe, chamados **CurvGaussianaAnalitica** e **CurvMediaAnalitica**, onde será calculada cada curvatura analítica específica.

Os métodos C**urvGaussianaAnalitica** e **CurvMediaAnalitica** recebem dois argumentos que são as coordenadas paramétricas ($0 \leq u \leq 1$, $0 \leq w \leq 1$) de um vértice em questão e retornam um valor real que representa a curvatura analítica. Esses métodos são responsáveis por implementar as Equações (**3.3**) e (**3.4**), respectivamente, e para isso os dois métodos usam a implementação da Equação (**3.5**).

3.2 Cálculo das curvaturas discretas

Diferentemente da curvatura analítica, a curvatura discreta não é analisada fazendo uso de qualquer formulação matemática analítica sobre o *patch*, mas se baseia nos triângulos adjacentes de cada vértice e na forma desses triângulos.

As curvaturas discretas consideradas nesse trabalho também são a Gaussiana e a Média. O cálculo das curvaturas discretas em cada vértice da malha é feito baseado nos operadores de curvatura apresentados por Meyer et al. (2002). Não é objetivo desse trabalho discutir a eficácia desses operadores, ressaltando-se apenas que os mesmos foram incorporados nesse trabalho por serem de implementação relativamente fácil. Mas, como se verá nos casos estudados, mostrados no Capítulo 6, esses operadores deram resultados bastante satisfatórios, para a estratégia de adaptação proposta.

3.2.1 Formulação matemática

Para o cálculo das curvaturas Média e Gaussiana discreta, é necessário calcular uma área que se baseia nos triângulos adjacentes a um vértice x_i qualquer da malha. Essa área é calculada segundo o procedimento apresentado em Meyer et al. (2002) e mostrado na Figura 3.1.

$A_{adjacente} = 0$;
Para (cada triângulo T adjacente ao vértice x_i) **faça**
{
    Se (*T não é obtusângulo*) **então**
        // Adiciona a área de Voronoi para o vértice x_i
        $A_{adjacente} = A_{adjacente} + A_{Voronoi}(x_i)$
    Senão
        // Ou adiciona área(T)/2 ou área(T)/4
        Se (o ângulo de T em x_i é obtuso) **então**
            $A_{adjacente} = A_{adjacente}$ + área(T)/2
        Senão
            $A_{adjacente} = A_{adjacente}$ + área(T)/4
}
Retorne $A_{adjacente;}$

Figura 3.1: Procedimento para cálculo da área de triângulo.

A área de *Voronoi* ($A_{Voronoi}(x_i)$) é calculada segundo uma série de considerações. Seja um triângulo não obtusângulo com vértices x_i, x_j e x_k e com circumcentro o, como mostrado na Figura 3.2 (b). Usando as propriedades dos bissetores perpendiculares, tem-se que a + b + c = π/2, e então, a = π/2 - $\angle x_k$ e c = π/2 - $\angle x_j$ e como, por hipótese, o triângulo é não obtusângulo, pode-se concluir que a área de *Voronoi* é igual a (1/8)*($|x_i x_j|^2 \cot\angle x_k$ + $|x_i x_k|^2 \cot\angle x_j$) (Meyer et al., 2002).

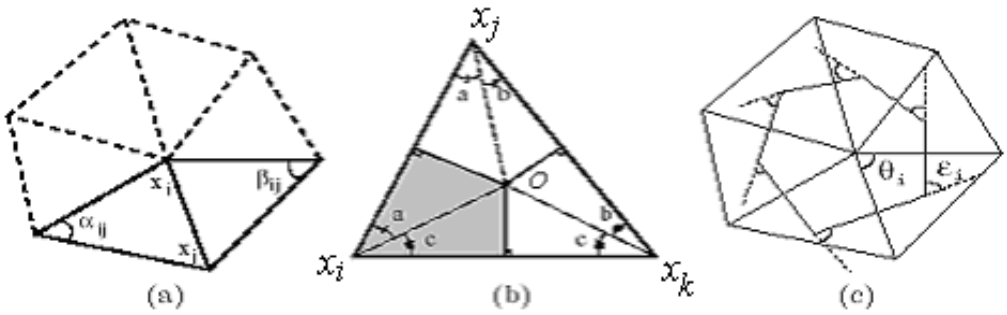

Figura 3.2: (a) triângulos adjacentes a um vértice *xi* na malha com seus ângulos opostos α_{ij} e β_{ij} para uma aresta $x_i x_j$; (b) região de *Voronoi* sobre um triângulo não obtusângulo; (c) ângulos de uma região de *Voronoi*.

Para o cálculo de curvatura Média discreta define-se um operador de curvatura média através da seguinte expressão matemática (Meyer et al. 2002):

$$\mathbf{K}(x_i) = (1/2A_{\text{adjacente}})^* \sum_{x_j \in N_{(i)}} (\cot \alpha_{ij} + \cot \beta_{ij})\,(x_i - x_j), \qquad \textbf{(3.6)}$$

onde $N_{(i)}$ é o conjunto de vértices dos triângulos adjacentes a x_i e que forma uma aresta $x_i x_j$ qualquer.

Através da expressão anterior pode-se facilmente computar o valor da curvatura Média discreta:

$$K_H = |\mathbf{K}(x_i)| / 2. \qquad \textbf{(3.7)}$$

A curvatura Gaussiana discreta é calculada através da seguinte expressão:

$$K_G(x_i) = (2\pi - \Sigma\theta) / A_{adjacente} , \quad (\mathbf{3.8})$$

onde Σθ é o somatório de todos o ângulo dos triângulos adjacente a x_i (ver Figura 3.2 c).

3.2.2 Aspectos de implementação

O cálculo das curvaturas discretas, assim com o das curvaturas analíticas, também segue o paradigma de programação orientada a objetos, permitindo que sejam calculadas igualmente para quaisquer tipos de curvas desejadas.

Os métodos **CurvGaussianaDiscreta** e **CurvMediaDiscreta**, que são privados da classe **CLPCurva**, recebem como argumento **CLPonto**, que representa o vértice no qual se quer calcular a curvatura, e uma malha local que corresponde a todos os triângulos adjacentes àquele vértice. Uma estrutura de dados que forneça eficientemente essa informação de adjacência de triângulos a vértices é fundamental nesse processo.

Nesse trabalho, foi implementada uma classe **CLCurvatura,** onde são definidas funções de cálculos para a curvatura discreta em cada vértice x_i da malha, passado para o construtor da classe no momento de instanciar um objeto deste tipo. Além do vértice no qual se quer calcular a curvatura (x_i), o construtor da classe **CLCurvatura** recebe também um vetor dinâmico cujos elementos são os triângulos adjacentes ao vértice em questão. Todas as equações para cálculo de curvatura discreta Média e Gaussiana são implementadas dentro dos métodos privados da classe **CLCurvatura.**

3.3 Estimativa de erro baseado nas curvaturas

O processo adaptativo consiste basicamente no refinamento das bordas dos *patches*, representadas pelas curvas que a compõem, assim como no refinamento do domínio desses *patches*. Esse refinamento é feito baseado no erro entre as curvaturas (analítica e discreta) nos vértices dos triângulos adjacentes às curvas que definem as bordas, bem como nos vértices dos triângulos que compõem o domínio. Os detalhes desses refinamentos serão discutidos em capítulos posteriores. Através desse erro, calcula-se um novo tamanho *h* para a região do vértice em questão e, dependendo da discrepância entre a curvatura analítica e

discreta nesse vértice, a curva ou o domínio será refinado, desrefinado ou não sofrerá alteração. A estimativa de erro, portanto, trata-se de calcular essa discrepância e estimar qual é o tamanho necessário para que esse erro seja diminuído. Como o erro é calculado para cada vértice, pode-se verificar onde o refinamento se faz necessário, tanto para as curvas quanto para o domínio.

A Tabela 3.2 mostra os casos possíveis, adotados nesse trabalho, e a determinação dos novos tamanhos, para cada caso. Nessa tabela, K_a representa a curvatura analítica enquanto que K_d, a discreta.

$K_a \cong K_d$ ($K_a/K_d \rightarrow 1$)		$K_a >> K_d$		$K_a << K_d$	
$K_a \rightarrow 0$	$K_a \not\rightarrow 0$	$K_a \rightarrow 0$	$K_a \not\rightarrow 0$	$K_a \rightarrow 0$	$K_a \not\rightarrow 0$
$h_{novo} = h_{velho} * fator$	$h_{novo} = h_{velho}$	$h_{novo} = h_{velho} * fator$	$h_{novo} = h_{velho} / fator$	$h_{novo} = h_{velho} * fator$	$h_{novo} = h_{velho} / fator$
Desrefinar	Parar	Desrefinar	Refinar	Desrefinar	Refinar

Tabela 3.2: Casos de erro e estimativa de novos tamanhos.

No primeiro caso, onde K_a é aproximadamente igual a K_d ($K_a \cong K_d$), não será preciso refinar a região onde o vértice se encontra, pois a curvatura estimada real (K_d) é aproximadamente igual à curvatura estimada ideal (K_a). Isso é representado pela opção onde $K_a \not\rightarrow 0$, onde se tem que o novo tamanho h (h_{novo}) é igual ao tamanho atual (h_{velho}). Entretanto, caso K_a seja próximo de zero (como em um plano, por exemplo, representado por $K_a \rightarrow 0$), pode-se desrefinar a região. Isso indica que o novo tamanho h (h_{novo}) deve ser maior que o tamanho atual (h_{velho}), levando-se em consideração um fator de desrefinamento ($h_{novo} = h_{velho} * fator$).

No segundo e terceiro casos ($K_a >> K_d$ ou $K_a << K_d$), há uma discrepância entre as curvaturas e então, dependendo da curvatura analítica (que é a considerada ideal), poderá ocorre um refinamento ou um desrefinamento. Nos dois casos, quando K_a for próximo de zero ($K_a \rightarrow 0$), pode-se desrefinar a região e quando essa curvatura analítica não for próxima de zero será preciso refiná-la.

Em resumo, quando há uma discrepância entre as curvaturas analíticas e discretas é porque o tamanho h precisa sofrer alteração, pois as estimativas entre as curvaturas reais e

a ideais ainda estão muito desiguais. A curvatura K_a próximo de K_d indica que o tamanho h já se encontra adequado de acordo com os critérios pré-estabelecidos, mas se K_a for próximo de zero (como no caso plano, por exemplo) ainda é caso de desrefinamento.

A comparação das discrepâncias entre as curvaturas analítica e discreta, isto é, o fato de dizer que K_a é próximo de K_d ($K_a/K_d \rightarrow 1$) ou são muito diferentes ($K_a >> K_d$ ou $K_a << K_d$), leva em consideração uma precisão, para evitar erros de análise. Com relação aos fatores de refinamento e desrefinamento, o que se deseja, na verdade, são fatores que indiquem se determinada região deve ser refinada, desrefinada ou mantida. Portanto, nesse trabalho, usou-se fatores empíricos, que simplesmente captam essa tendência, isto é, dão uma direção de convergência para o processo. Quanto maior os valores desses fatores, mais rápida será a convergência. Para esse trabalho, foram considerados fatores de refinamento e desrefinamento de 2^{Passo}, onde *Passo* representa o número do passo corrente do processo adaptativo (*Passo* = 1,2,3...). Isso ainda necessita de testes mais exaustivos, mas funcionou muito bem para todos os casos testados. A idéia é que, se um novo passo foi necessário, é porque o erro ainda está alto, e então os triângulos devem ser refinados ou desrefinados a uma taxa mais alta, para acelerar a convergência e evitar um número demasiado de passos.

A Figura 3.3 mostra o algoritmo para cálculo do erro e estimativa dos novos tamanhos. O algoritmo recebe como parâmetro o vértice em questão, o tamanho h chamado de h_{velho} e retorna o tamanho h_{novo}, considerando **Precisão**, **FatorRefina** e **FatorDesrefina** como constantes.

Os primeiros comandos de atribuições do algoritmo são referentes às curvaturas analítica e discreta (K_a e K_d). Dois pontos são importantes ressaltar com relação ao cálculo dessas curvaturas. O primeiro é que a curvatura utilizada é sempre a Gaussiana, mas caso essa curvatura seja zero, passa-se a considerar a curvatura Média para efeito de cálculos. O outro ponto importante é que a curvatura analítica, calculada em qualquer vértice, é sempre com relação ao *patch* em questão, e não com relação à curvatura da curva ao qual ele pertence, se for um vértice de borda. Esse problema não existe se o vértice é de domínio, ou seja, se ele está no interior do *patch*. Escolheu-se trabalhar com a curvatura Gaussiana devido a sua melhor precisão para os exemplos testados. Mas a estratégia proposta funciona para qualquer curvatura usada, pois o que se precisa realmente é algum critério consistente que guie o processo adaptativo, embora possa convergir para o resultado satisfatório de maneira mais lenta.

Algoritmo: Calcula h_{novo}.
Entrada: v (vértice dado) e h_{velho}.
Saída: h_{novo}.
K_a = Curvatura_analítica no vértice;
K_d = Curvatura_discreta no vértice;

// 1^0 Caso: a razão k_a/k_d é aproximadamente igual a 1.

Se (((($K_d - K_a$) / K_d) <= **Precisão**) && ((($K_d - K_a$) / K_d) >= **(-1)*Precisão**)) **Então**

Inicio

Se (K_a <= **Precisão**) **Então** // Se K_a tende a 0: desrefina

h_{novo} = (FatorDesrefina)* h_{velho};

Senão // K_a não tende a 0: já está bom

$h_{novo} = h_{velho}$;

Fim

// 2^0 Caso: ($K_a >> K_d$)

Senão Se ($K_a > K_d$) **Então**

Inicío

Se (K_a <= **Precisão**) **Então** // Se K_a tende a 0: desrefina

h_{novo} = (FatorDesrefina)*h_{velho};

Senão // K_a não tende a 0: refina

$h_{novo} = h_{velho}$ /(FatorRefina);

Fim

// 3^0 Caso: ($K_a << K_d$)

Senão Se ($K_a < K_d$) **Então**

Inicío

Se (K_a <= **Precisão**) **Então** // K_a tende a 0: desrefina

h_{novo} = (FatorDesrefina)* h_{velho};

Senão // k_a não tende a 0: refina

$h_{novo} = h_{velho}$ /(FatorRefina);

Fim.

Retorne h_{novo};

Figura 3.3: Algoritmo para casos de erro e estimativa de novos tamanhos.

É importante notar que nas atribuições das curvaturas analíticas e discretas (os dois primeiros cálculos) um teste é realizado para saber se a curvatura analítica Gaussiana é zero. Se isso acontecer, passa-se a considerar a curvatura Média. Isso é necessário pelo fato de que uma curvatura Gaussiana nula não implica sempre em uma região localmente plana (como no vértice da Figura 3.1(b)). Na Figura 3.4, por exemplo, a curvatura Gaussiana no ponto indicado é nula, já que uma das curvaturas principais (direção B-B) é nula. No entanto, na direção A-A e em qualquer outra direção intermediária a curvatura é diferente de zero.

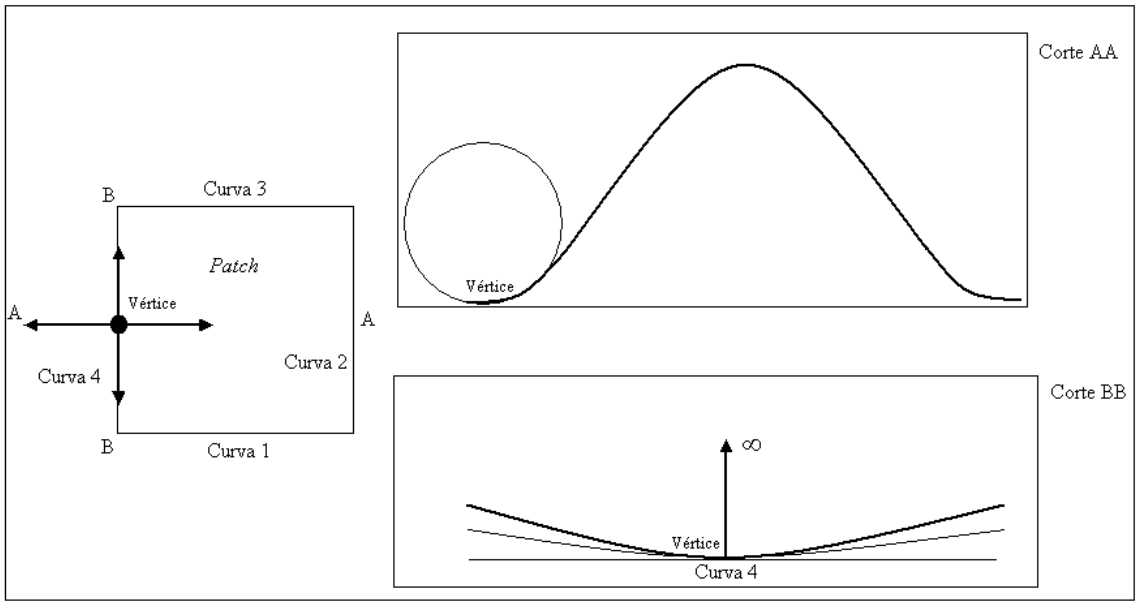

Figura 3.4: Exemplo onde a curvatura Gaussiana é nula e a Média é não nula.

3.4 Estimativa do erro global

Como se trata de um processo iterativo, é necessário um critério de parada para definir quando a malha do passo corrente é satisfatória. É necessário, portanto, ter uma estimativa do erro global, para definir esse critério. O cálculo desse erro será explicado a seguir. Para cada vértice pertencente à malha, calcula-se, na etapa de análise de curvatura, o erro de curvatura correspondente ao vértice. Esse erro é o erro relativo entre as curvaturas, ou seja, é a diferença entre a curvatura analítica e a discreta em valor absoluto dividido pela maior curvatura, discreta ou analítica (Fórmula 3.9). Esse erro, evidentemente, resultará num valor entre zero, que corresponderia a um erro de 0% (caso não exista erro algum) e um,

correspondendo a um erro de 100% (caso exista a maior discrepância possível entre as curvaturas). Ao final do processo de análise de curvatura, é feito um somatório, com todos os valores dos erros relativos dos vértices pertencentes ao modelo e esse somatório é dividido pelo número total de vértices que compõem esse modelo, resultando no cálculo de erro global η (Fórmula 3.10). A divisão pela maior curvatura se deve ao fato dos operadores de curvatura apresentarem alguns problemas, especialmente na borda. Por exemplo, pode-se ter uma curvatura analítica de 200 e uma curvatura discreta de 10, e vice-versa. Se não fosse usada a maior curvatura no denominador, o erro não ficaria sempre entre 0 e 1, como desejado, e a análise seria prejudicada. As fórmulas dos erros são então:

$$\eta_{\text{vértice}} = \text{abs}(K_a - K_d)/K_{maior} \qquad (\mathbf{3.9})$$

e

$$\eta = \Sigma\, \eta v_{\text{vértice}} /(\text{número de vértices}). \qquad (\mathbf{3.10})$$

Como para aplicações desse tipo não se tem ainda uma idéia exata de quanto seria um erro razoável limite para uma aplicação, optou-se, nesse trabalho, apenas por mostrar que os erros realmente caem, à medida que a malha é melhorada, indicando um processo de convergência. Estudos mais aprofundados para definição desse erro limite seriam importantes, pois com esse limite pode-se considerar que, se o erro global η for igual ou menor do que o limite, a malha em questão é satisfatória e o processo pode parar. Entretanto, é importante mencionar que não é objetivo desse trabalho estudar esses critérios de parada. O que se deseja realmente é mostrar a eficácia da estratégia proposta, como se pode verificar pelos exemplos analisados no Capítulo 6.

Capítulo 4

Refinamento das curvas de fronteiras

Dentro da estratégia de auto-adaptação proposta nesse trabalho, descrita no Capítulo 2, este capítulo descreve a estratégia de refinamento das curvas de cada *patch* do domínio. O refinamento das fronteiras dos *patches* é feito independentemente da discretização do domínio do modelo, sendo implementado dentro de uma filosofia de orientação a objetos, o que permite tratar todos os tipos de curvas de uma forma genérica.

4.1 Refinamento das curvas de fronteiras dos *patches*

A estratégia usada para refinar o contorno é uma versão unidimensional (em cada curva) do procedimento utilizado para refinar o domínio, baseado na *quadtree*, que será apresentado no capítulo seguinte. O refinamento de cada curva dos *patches* utiliza uma técnica de enumeração espacial recursiva, baseada em uma estrutura de dados de árvore binária (*binary tree*). A idéia é gerar uma discretização regular nas curvas da fronteira em função do novo tamanho dos triângulos da malha adjacentes aos vértices dessas curvas. Esses novos tamanhos são calculados baseados na estimativa de erro das curvaturas nesses vértices em questão. É importante mencionar que, se uma curva possuir mais de uma superfície adjacente, as curvaturas analítica e discreta de cada vértice da curva, que irão orientar o refinamento, levam em consideração essas superfícies. A curvatura discreta considera todos os triângulos adjacentes (ao vértice) que pertencem às superfícies, podendo esses triângulos estar em superfícies diferentes, e a curvatura analítica considera todas as curvaturas analíticas provenientes das superfícies adjacentes, ao qual o vértice pertence, para representar o refinamento mais adequado, quando necessário, para uma dada curva. O refinamento de cada curva é dividido em etapas, conforme descrito nas subseções a seguir.

4.1.1 Inicialização da árvore binária

A primeira fase do refinamento consiste em inicializar a árvore binária que irá orientar o refinamento da curva. Essa árvore é inicializada com as coordenadas da célula pai da

árvore, parametrizada ao longo da curva entre zero (coordenada mínima) e um (coordenada máxima). Isso é feito para permitir um procedimento de refinamento genérico para qualquer tipo de curva existente no modelo, seja ela linha reta, arco de círculo, curva de *Bézier*, curva de *Hermite*, etc. A profundidade da árvore nesse ponto é inicializada como zero, já que se trata inicialmente de um único nível.

4.1.2 Refinamento da árvore binária

Inicializada a árvore, parte-se para o seu refinamento. Cada curva possui, na sua estrutura interna, o conjunto de seus vértices adjacentes, que foi gerado pelo passo anterior de refinamento ou, no caso de malha inicial, que foi gerado pelo algoritmo que constrói a primeira malha. Além disso, a curva também possuirá o conjunto de vértices que serão inseridos ao final desta etapa de refinamento da curva. Isso é importante porque esses dois conjuntos devem co-existir, até que se tenha gerado totalmente a nova malha e, portanto, isso deve ser gerenciado de forma adequada. Esses conjuntos de vértices da curva estão ordenados segundo a parametrização de cada vértice em relação à curva ao qual ele pertence.

Percorre-se então todos os vértices da curva, iniciando-se pelo primeiro vértice (com parâmetro zero), de forma que cada dois vértices representem um segmento da curva, refinando a árvore segundo o procedimento descrito a seguir. Para cada segmento da curva, calcula-se o seu tamanho, que será o tamanho chamado de h_{velho} e que representa o tamanho real do passo atual do qual se deseja refinar ou desrefinar. De posse desse tamanho, calcula-se um novo tamanho chamado de h_{novo} baseado na média das curvaturas dos vértices extremos do segmento e no h_{velho}, e em seguida parametriza-se esse tamanho h_{novo} em relação ao tamanho real da curva. Calcula-se o vértice médio do segmento, no espaço paramétrico, e com esse vértice médio é realizada uma busca para saber em que célula ele se localiza na árvore (a célula pai é a própria curva parametrizada de zero a um). Se o tamanho da célula é maior que o tamanho h_{novo}, então subdivide-se a célula em duas, incrementando-se um nível de profundidade na árvore, e assim sucessivamente até ser menor. Passa-se para o próximo segmento da curva e repete-se o processo. Tanto o tamanho h_{novo} como o vértice médio de cada segmento devem estar no espaço paramétrico

porque a curva é tratada no seu espaço paramétrico e isso permite, inclusive, que se considere vários tipos diferentes de curvas através desse mesmo procedimento.

Após percorrer todos os segmentos da curva, tem-se a árvore refinada segundo o erro da curvatura dos vértices adjacentes à curva. Cada célula folha da árvore (célula que não possui filhos) vai gerar um lado de um triângulo na nova discretização da curva. A Figura 4.1 mostra um exemplo de refinamento de curva e na Figura 4.2 tem-se a sua árvore correspondente.

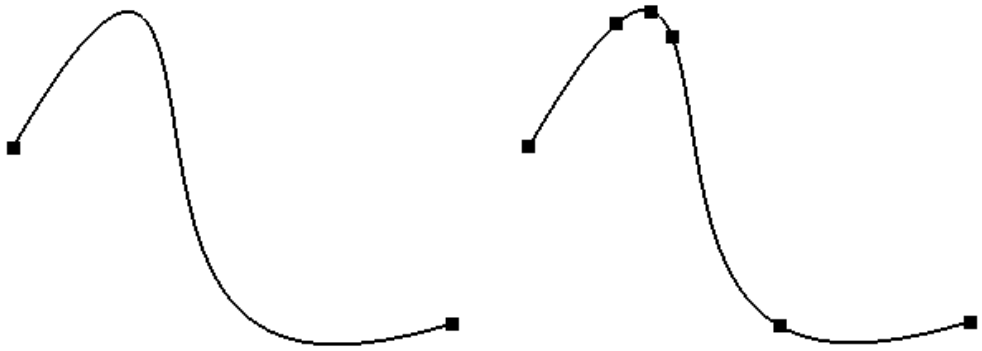

Figura 4.1: Exemplo de refinamento de uma curva.

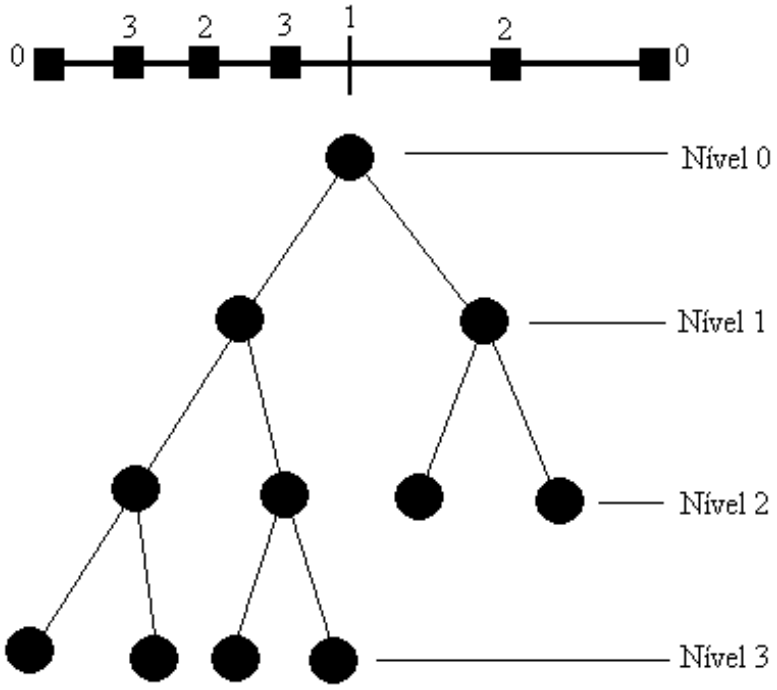

Figura 4.2: Bitree para o exemplo da Figura 4.1.

4.1.3 Atualização da discretização da curva

Após a árvore estar definida para a curva considerada, inclui-se a nova discretização na estrutura da curva. A nova discretização é a definida pelas coordenadas extremas das células terminais (folhas) da árvore. Para evitar que a cada coordenada (máxima ou mínima) de uma célula se faça uma busca na lista de nós da curva para verificar se aquela coordenada já foi inserida, faz-se o seguinte procedimento: obtém-se a coordenada paramétrica mínima (que é zero) da célula pai da árvore, acha-se a coordenada Cartesiana correspondente em relação à curva considerada, e inclui-se essa coordenada na estrutura da curva; percorre-se então toda a árvore, transformando-se para coordenadas Cartesianas todas as coordenadas máximas das células folhas e incluindo-se na lista de nós da curva, tendo-se ao final, de forma ordenada, toda a nova discretização dessa curva.

O motivo de se considerar somente a coordenada máxima de uma célula folha se deve ao fato de que sempre uma coordenada mínima de uma célula folha é igual a uma coordenada máxima da célula folha vizinha. Deste modo evita-se repetições de pontos na discretização da curva. Isto também justifica a inclusão da coordenada mínima da célula pai da árvore no primeiro passo; caso contrário o primeiro ponto da curva (correspondente à coordenada paramétrica zero) não seria incluído na discretização.

A Figura 4.3 mostra um exemplo de refinamento de fronteira, onde a figura à esquerda representa o modelo original e a figura à direita representa a discretização de borda para o primeiro passo. Como se pode observar nessa figura, locais onde a curvatura é mais alta, como no canto inferior direito, foram mais discretizados.

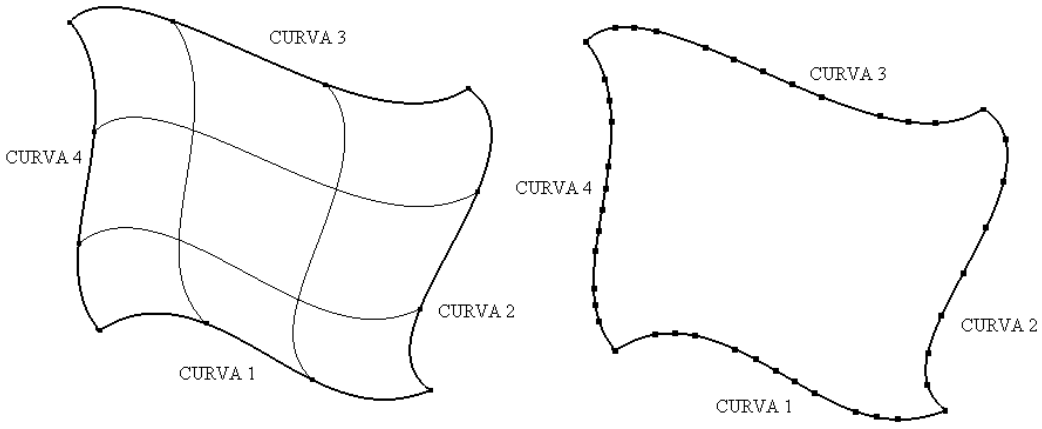

Figura 4.3: Exemplo de refinamento de fronteira.

Após ser completado o refinamento da curva, atualiza-se o conjunto de vértices adjacentes à essa curva e exclui-se o conjunto de vértices relativo ao passo anterior. Essa atualização do conjunto de vértices é importante para que o refinamento no interior do *patch* seja coerente.

4.2 Aspectos de implementação

O tratamento do refinamento nas fronteiras necessita de uma organização que propicie um procedimento genérico para quaisquer tipos de curvas. Dessa forma, novos tipos de curvas podem ser introduzidas com facilidade. O paradigma de orientação a objetos é ideal para esse tratamento e acarreta uma implementação natural da estratégia. A estrutura de classes que trata as curvas bem suas relações são descritos nas subseções seguintes.

4.2.1 Classe curva

Para a implementação de cada tipo de curva específica, seguindo o paradigma orientado a objetos, foi implementada uma classe genérica onde todas as curvas (arco, Bezier, Hermite, entre outras) herdam as características inerentes à entidade Curva. Essa classe genérica, chamada **CLCurva** possui atributos e métodos que são comuns a todas as curvas específicas. Por exemplo, toda curva (não importando o seu tipo) contém um conjunto de vértices adjacentes, um conjunto de vértices rediscretizados e um conjunto de triângulos adjacentes a ela, assim como toda curva precisa ser capaz de realizar o cálculo de um novo tamanho (h_{novo}) dentro de um segmento adjacente a ela. Portanto, os atributos **VerticesAdj**, **VerticesRed** e **TriangulosAdj** pertecem a classe genérica (também chamada de super classe), assim como o método **CalculaHnovo**. A curva genérica nunca é instanciada, pois ela foi implementada para servir de "molde" para os tipos de curvas específicas implementadas.

Uma vez implementada a curva genérica descrita acima, é necessária a implementação das curvas específicas. Cada curva específica contém seus atributos e métodos específicos, além de herdarem todos atributos e métodos da classe genérica. Por exemplo, a curva de Hermite possui atributos próprios: seus vértices extremos e suas derivadas.

Além de atributos específicos, cada tipo de curva possui, também, seus métodos específicos, como por exemplos métodos implementados para desenhar cada tipo de curva e métodos que retornam valores dos atributos da classe. Esses últimos métodos foram implementados para permitir um encapsulamento dos atributos da classe, pois isso facilita a manutenção da própria classe bem como das classes relacionadas a ela. Além dos métodos citados acima, é de extrema importância, para o processo de refinamento da curva, que cada tipo de curva implemente dois métodos fundamentais, os quais são responsáveis pela conversão de coordenadas Cartesianas para coordenadas paramétricas e vice-versa. Essa conversão é importante porque a árvore binária criada trabalha com a curva no espaço paramétrico. Esses métodos que fazem essa conversão são comentados na próxima seção.

4.2.2 Métodos genéricos

Para o processo de refinamento de cada curva, foi implementada uma classe chamada de **CLBitree**. Essa classe recebe, no momento da criação da árvore, através de seu construtor, uma curva com sua discretização e retorna a mesma curva com sua nova rediscretização. A classe **CLBitree** é genérica no sentido de receber como parâmetro (em seu construtor) qualquer tipo de curva, ou seja, o processo de rediscretização da curva, utilizando uma árvore binária, independe do tipo de curva, podendo esta ser um arco, uma curva Bezier, uma curva Hermite, etc. A classe **CLBitree** possui atributos próprios como **primeiro** (que representa o primeiro parâmetro de cada segmento da curva), **seguinte** (que representa o parâmetro seguinte de cada segmento da curva), **profundidade** (que representa a profundidade do nó na árvore) e o atributo **filhos** que guarda o endereço de memória dos filhos do nó em questão. Além desses atributos, a classe **CLBitree** tem seus métodos próprios. Dentre esses métodos os três principais são: **PercorreBitree**, **DivideBitree** e **RediscretizaCurva.**

O método **PercorreBitree**, como o próprio nome sugere, percorre toda a árvore, para cada segmento da curva, achando a localização do vértice médio de cada segmento. Após achar o nó onde o vértice médio se encontra, é feita uma subdivisão da célula do nó achado, subdivisão essa que é realizada pelo método **DivideBitree**. Neste método, são criados dois novos nós e atribuídos como filhos do nó em questão, ou seja, o atributo **filhos** do nó localizado conterá um endereço de memória onde estarão as informações dos filhos

criados. Após esse processo se repetir para cada segmento da curva, é realizada a rediscretização da mesma com o método **RediscretizaCurva**. Nesse método, percorre-se toda a árvore e cada nó folha formará um vértice da curva passada como parâmetro para a classe **CLBitree**.

Como mencionado no final da seção anterior, cada tipo de curva implementa métodos os quais são responsáveis pela conversão de coordenadas Cartesianas para coordenadas paramétricas e vice-versa. Essa conversão é muito importante porque a árvore trabalha com valores parametrizados para os vértices, ou seja, trabalha com a curva parametrizada. Para os atributos **primeiro** e **seguinte** da classe **CLBitree**, são atribuídos os valores parametrizados dos vértices que compõem o segmento da curva e para isso a classe específica da curva contém um método chamado de **DeXYZparaT**. Esse método recebe um vértice em coordenadas Cartesianas e retorna o seu valor parametrizado dentro da curva. O valor retornado por **DeXYZparaT** é o que será considerado na criação dos nós da árvore. No método **RediscretizaCurva** da classe **CLBitree** é realizado o processo inverso, ou seja, tem-se os nós folhas (que formarão os novos vértices da curva) com seus respectivos valores parametrizados (**primeiro** e **seguinte**) e é necessário obter as coordenadas Cartesianas dos vértices. Para esse caso, cada classe específica de curva implementa um método chamado de **DeTparaXYZ.**

Capítulo 5

Geração adaptativa da malha no domínio

Esse capítulo descreve o método para geração de malhas no domínio do modelo, dentro do processo auto-adaptativo. O algoritmo procura aliar as vantagens das técnicas de *quadtree* e da triangulação de Delaunay por contração do contorno, partindo de uma discretização de bordo fornecida. A idéia básica é gerar os triângulos no interior do domínio pala técnica de *quadtree* (Baehmann et al., 1987), deixando uma faixa próxima à fronteira do domínio para ser gerada pela técnica da triangulação por contração do contorno (Shaw & Pitchen, 1978). Essa última técnica é ainda modificada para usar informações da estrutura *quadtree* já construída de modo a evitar, na formação de um triângulo, que um vértice longe da aresta base seja verificado, desta forma acelerando o método. O algoritmo proposto se baseia em um trabalho desenvolvido por Cavalcante-Neto et al. (1993), com a diferença que é considerada a informação do erro nas curvaturas para a geração da *quadtree*, o que é necessário em um processo adaptativo, e que não é usado no trabalho mencionado, que trata apenas de geração automática de malha em domínios arbitrários. Essa informação de erro foi também considerada em outros trabalhos que usam o algoritmo mencionado [(Cavalcante-Neto et al., 1993), (Cavalcante-Neto et al., 1998), (Paulino et al., 1999)], com a diferença que nesses trabalhos as informações de erro provém de análises por elementos finitos, enquanto que no presente trabalho essa informação advém do erro nas curvaturas.

O algoritmo proposto gera a malha no domínio partindo de uma discretização do contorno, representado pelas curvas de fronteira, fornecida. Essa discretização foi gerada, nesse trabalho, na fase de auto-refinamento das fronteiras, descrita no capítulo anterior. Para a geração de malha no domínio, uma observação é importante. Como o algoritmo de *quadtree* é executado no plano, é necessário um mapeamento de cada vértice do espaço tridimensional para o espaço paramétrico de cada *patch*. Além disso, ao final da geração da malha, é necessário um novo mapeamento do espaço paramétrico de cada *patch* para o espaço tridimensional, para considerar todos os novos triângulos gerados. Isso permite que se trate qualquer tipo de *patches* de uma forma genérica, sejam eles *patches* de Hermite, Bezier, entre outros. O refinamento de cada *patch* é dividido em etapas, conforme descrito nas seções a seguir.

5.1 Geração da malha no interior do domínio através da *quadtree*

Essa seção define como a malha é criada no interior do domínio do *patch* através do uso da *quadtree*, deixando uma região entre essa malha interna e o contorno do *patch* para ser gerada usando uma técnica de triangulação de Delaunay. O uso da *quadtree* garante que a densidade interior das células da árvore seja sensível à discretização fornecida do contorno e também garante uma boa transição entre regiões com diferentes graus de refinamento da malha a ser gerada. A geração da malha é dividida em várias fases, descritas a seguir.

5.1.1 Criação da árvore inicial

O processo é iniciado pela criação da *quadtree* baseada na discretização do contorno dos *patches* que é fornecida. Primeiro, as coordenadas máximas e mínimas do contorno são usadas para gerar uma caixa envolvente do modelo no espaço paramétrico (para cada *patch*), que será o nó pai da *quadtree*. Como cada vértice do *patch*, no espaço tridimensional, é mapeado para o espaço paramétrico, as coordenadas mínimas serão sempre (0,0) e as coordenadas máximas serão sempre (1,1). Percorre-se, então, todos os segmentos da discretização das curvas do contorno do *patch* em questão, um por vez, para criar a árvore usando-se um procedimento que se repete para cada segmento. Inicialmente, o comprimento e o ponto médio do segmento corrente é calculado, e uma busca é feita no estado atual da árvore para saber em que célula este ponto está. Após isso, verifica-se então se o tamanho desta célula é maior que uma percentagem do comprimento do segmento e nesse caso subdivide-se essa célula, continuando o processo até o tamanho da célula ser menor que essa percentagem. Passa-se então a um novo segmento e repete-se o processo. Com relação a essa percentagem, é recomendado um fator entre 0,7 e 1,4 (Potyondy, 1993) e nesse trabalho usou-se 1,0 (100%), pois testes com vários tipos de contorno foram feitos e de um modo geral o uso desse valor foi o que deu melhores resultados.

Com relação às células da árvore, vale dizer que elas podem ser de quatro tipos: células exteriores, que são células completamente fora do domínio; células vértices, que são células que contém no seu interior algum vértice do contorno do domínio; células do contorno, que são células atravessadas por algum segmento desse contorno; e células

interiores, que estão completamente dentro do domínio. Essa classificação será importante nas próximas fases. Considerando a discretização da Figura 5.1, a Figura 5.2 ilustra a criação da árvore inicial.

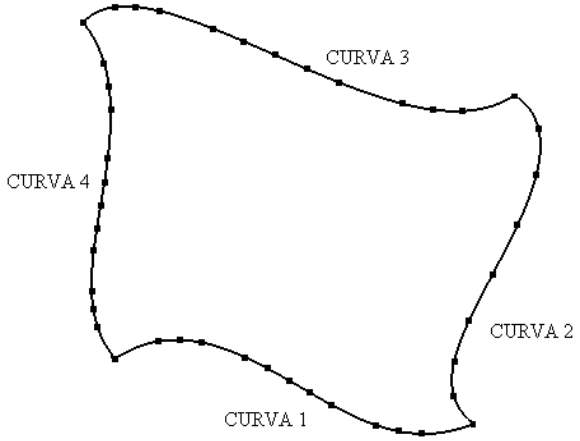

Figura 5.1: *Patch* para exemplificar a criação da *quadtree.*

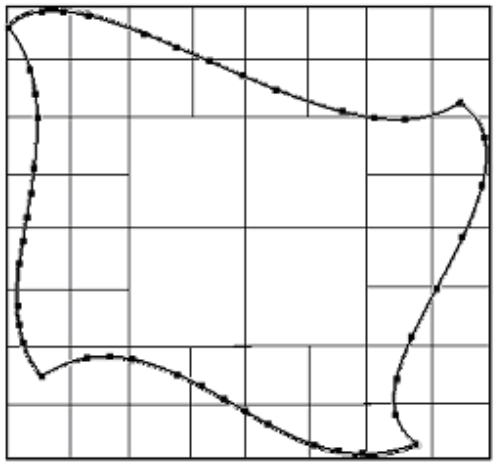

Figura 5.2: Criação da árvore inicial.

5.1.2 Ajustes devidos ao erro nas curvaturas dos elementos

Criada a árvore inicial, a compatibilidade com o erro de curvatura de cada triângulo é imposta no interior do *patch*. Isto é feito segundo um procedimento que se repete para cada elemento e que é descrito a seguir. Para cada elemento da malha original fornecida, calcula-se o seu ponto médio e encontra-se a célula da *quadtree* que contém este ponto. Compara-se então o tamanho da célula com o novo tamanho do elemento, ditado pela análise de erro. Se esse tamanho é maior que o novo tamanho do elemento, subdivide-se a célula e continua-se sucessivamente até seu tamanho ser menor. Passa-se então ao próximo elemento e repete-se o processo, até ter-se ajustado a árvore para todos os triângulos. A Figura 5.3 mostra a árvore depois de feito esses ajustes devido ao erro de curvatura.

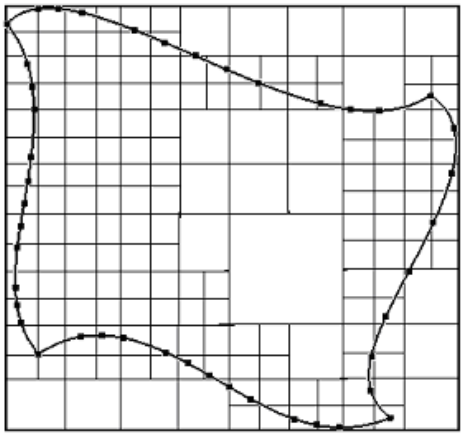

Figura 5.3: Árvore após ajustes devidos ao erro nos triângulos.

O novo tamanho de cada elemento, usado para ser comparado com o tamanho da célula da *quadtree* e chamado tamanho característico do elemento, é calculado baseado nas curvaturas para os vértices desse elemento. Esse cálculo segue os seguintes passos:

- Calcula-se o tamanho atual do elemento através de **sqrt(área(elemento)/áreaTotal)**, onde a função sqrt é a raiz quadrada e áreaTotal é a área total de todos os triângulos do *patch* em questão. Isso é feito para parametrizar o tamanho do elemento, pois o processo trabalha com tamanhos parametrizados para cada elemento em relação à área do *patch* considerado. Esse tamanho será o h_{velho} para o elemento em questão. Isso também ajuda a evitar distorções entre os espaços paramétrico e Cartesiano, que pode ocorrer por se usar uma caixa envolvente de (0,0) a (1,1) para as várias superfícies.

- Calcula-se uma estimativa das curvaturas discreta e analítica para o elemento. Essa estimativa é dada pela média dessas curvaturas nos três vértices do triângulo.
- Com base no h_{velho} do elemento e nas curvaturas, calcula-se o novo tamanho do elemento, que será o h_{novo}, com base na análise de erro das curvaturas, conforme mostrado no Capítulo 3, onde se mede a discrepância entre essas curvaturas.

5.1.3 Ajustes para um nível de diferença entre células adjacentes

Após as fases anteriormente descritas, ajustes devem ser feitos para garantir somente um nível de diferença de profundidade na árvore entre células adjacentes. Esse tipo de *quadtree*, onde as células adjacentes são do mesmo tamanho ou no máximo com um nível de diferença de profundidade entre elas, é chamada de *quadtree* restrita (Von Herzen & Barr, 1987). Isto é necessário, pois, posteriormente, serão criados triângulos em células interiores da *quadtree* através de padrões e isto exige que na árvore exista somente um nível de diferença entre células adjacentes. Estes padrões são usados para gerar triângulos, dada a configuração de uma célula e de suas células adjacentes. Isto será abordado com mais detalhes na seqüência. Além disso, um único nível de diferença assegura uma boa qualidade de transição entre os triângulos na malha a ser gerada. A Figura 5.4 mostra a árvore depois de assegurado um único nível de diferença entre células adjacentes.

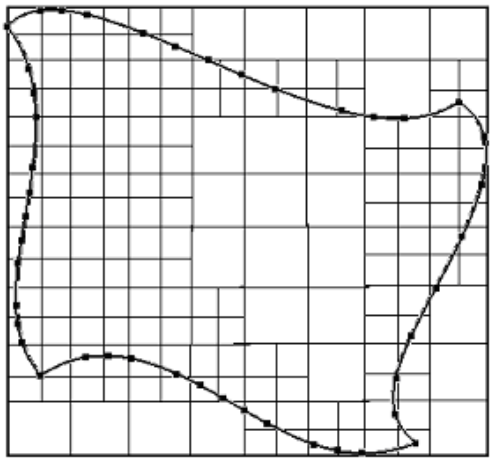

Figura 5.4: Árvore após fase de um único nível de diferença.

5.1.4 Eliminação de células perto do contorno

Após essas fases de ajustes, todas as células da árvore, que forem interiores ao domínio, irão formar triângulos através de padrões e os outros triângulos serão gerados através de triangulação de Delaunay por contração de contorno. Precisa-se, então, classificar bem o que são células interiores, pois células que estiverem muito perto do contorno poderão ocasionar triângulos de forma ruim quando da triangulação da faixa do contorno. A solução é re-classificar células interiores que estejam a uma distância de um segmento do contorno menor que uma percentagem do seu comprimento. No caso desse algoritmo, adotou-se um fator de 0,2 (20%).

5.1.5 Geração de malha em células interiores por padrões

Após esses ajustes feitos, a árvore está pronta para concluir a primeira parte do algoritmo, que é a geração de malha no interior do domínio, criando os triângulos nas células interiores através de padrões. Como já foi dito anteriormente, estes padrões são usados para gerar triângulos dada a configuração de uma célula e sua adjacência. Existem alguns tipos de padrões para elementos triangulares e outros para elementos preferencialmente quadrilaterais, como se pode ver na Figura 5.5 (Baehmann et al., 1987). Nota-se daí a importância de se garantir apenas um único nível de diferença entre células adjacentes na árvore, como foi descrito em fase anterior.

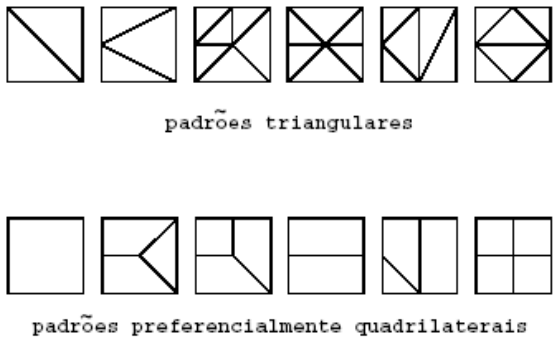

Figura 5.5: Padrões para elementos triangulares e quadrilaterais.

Com o uso desses padrões, gera-se então a primeira parte da malha, que normalmente ocupa a maior área do domínio. Uma pequena faixa, entre as células interiores da *quadtree*

e o contorno do domínio sobra para que a malha seja gerada através de uma triangulação de Delaunay por contração de contorno.

5.2 Geração de malha no contorno do domínio

Esta seção descreve como a malha é criada na região entre a *quadtree* interna e o contorno. Isto é feito através da técnica de triangulação de Delaunay por uma contração de contorno [(Shaw & Pitchen, 1978), (Vianna, 1992), (Potyondy, 1993)]. Isto completa o método de geração de malha no domínio.

5.2.1 Inicialização da lista de arestas ativas do contorno

O processo se inicia com a formação de uma lista de arestas ativas do contorno, originalmente contendo todos os segmentos do contorno que são fornecidos. Escolhe-se uma aresta dessa lista, que será a aresta base para formação de um novo triângulo.

5.2.2 Escolha de vértice interior para formação de um elemento da malha

De posse da aresta base, deve-se achar um vértice do interior ou do contorno corrente que forme com a aresta o melhor triângulo. Seguindo o critério de Delaunay, o melhor triângulo é aquele que tem o maior ângulo formado pelo vértice candidato com os vértices da aresta base, como se pode ver pela Figura 5.6.

A escolha do melhor triângulo em um algoritmo de contração do contorno genérico é feita testando-se a aresta base contra todos os outros vértices, o que torna esse algoritmo de complexidade quadrática. No algoritmo proposto, a seleção dos possíveis candidatos a vértice do triângulo é feita de uma maneira mais otimizada. A cada aresta base é formada uma lista de células da árvore adjacentes às células que contêm o vértice inicial e final dessa aresta base. Essas células são armazenadas se forem células interiores que, portanto, possuem vértices interiores possíveis candidatos, ou se forem células vértices, que são as células que possuem vértices do contorno original, pois também podem vir a formar um melhor triângulo. Dessa forma, o teste do maior ângulo é feito para uma lista reduzida

de vértices, evitando testes exaustivos contra vértices que estejam bem longes da aresta base.

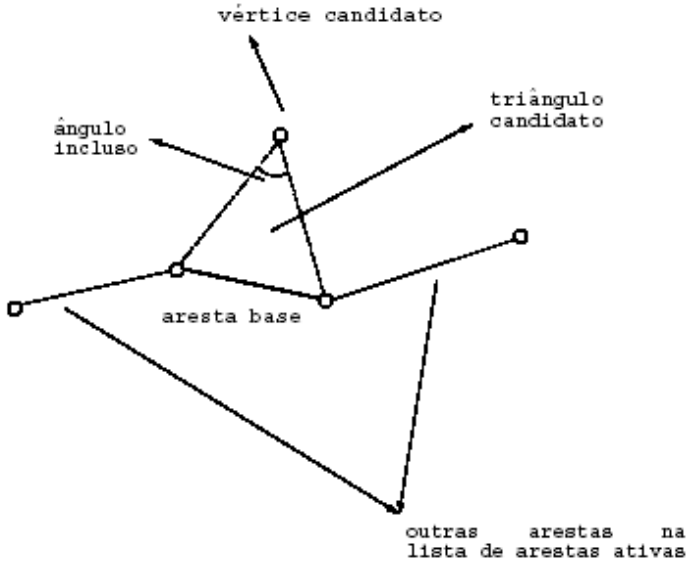

Figura 5.6: Definição do melhor triângulo pela técnica de Delaunay.

Além disso, a seleção dos vértices candidatos à formação do melhor triângulo é feita de forma bastante eficiente, pois explora a busca em uma árvore quaternária, já construída anteriormente. É possível que, na criação da lista de células adjacentes, não exista nenhuma célula da árvore que seja interior ou vértice (somente células do contorno), ou seja, não se ache nenhum vértice possível candidato. Esses casos são raros e só acontecem quando o contorno original tem uma discretização irregular e mal comportada para uma malha. Nesses casos, extende-se a pesquisa na direção de cada adjacência (que são oito para cada célula: pelos lados e cantos) até que, em cada direção, se tenha uma célula com um vértice possível candidato. Vale ressaltar que cada vértice testado só poderá ser escolhido caso esteja obviamente dentro do domínio a ser gerada a malha e também se a aresta do elemento que ele irá formar não interceptar nenhuma aresta já existente. Para garantir esses aspectos, testes geométricos são feitos usando-se técnicas de geometria computacional.

5.2.3 Atualização da estrutura de dados

Quando se chega a essa fase, um novo elemento triangular foi formado. Esse elemento é adicionado à lista de triângulos já existentes e é feita uma atualização da lista de arestas

ativas para fazer a contração do contorno propriamente dita. A técnica de contração de contorno tradicional é tal que, formado um elemento, a aresta base é retirada da lista de arestas ativas e cada uma das arestas formadas é inserida na lista, se for uma nova aresta, ou retirada dela, se já existir. Isto faz com que o contorno se contraia até completar todo o domínio, isto é, até gerar a malha em toda a região.

No algoritmo desse trabalho, o processo de retirar a aresta base e inserir ou não as novas é o mesmo da contração de contorno tradicional, mas com duas modificações. Primeiro, a contração não cobre todo o domínio, pois já existem triângulos no interior do modelo gerado por *quadtree*: a contração pára na *quadtree,* representando um número menor de atualizações na lista de arestas. Segundo, para evitar uma procura na lista de arestas ativas a cada novo elemento formado, é criada uma estrutura de dados auxiliar para otimizar esse processo. Cada nó dessa estrutura é referenciado pelo identificador de um vértice da malha e contém uma lista com as arestas que esse vértice formou (cada aresta é identificada por sua posição na lista de arestas ativas). Assim, ao se formar uma aresta nova, para verificar se ela já existe na lista de arestas ativas basta procurar na estrutura auxiliar de um dos vértices dessa aresta, sem precisar fazer uma busca completa cada vez na lista. Isto acelera a busca, já que a lista de arestas ativas em certos momentos da contração pode ser bastante grande enquanto que, na estrutura auxiliar, cada vértice forma um número reduzido de arestas.

5.2.4 Finalização da contração do contorno

Se após a atualização, a lista de arestas ativas ficar vazia, significa que a malha foi gerada, pois todas as arestas já foram utilizadas como base. Senão, seleciona-se uma nova aresta e repete-se o processo.

5.2.5 Suavização da malha

Finalmente, após ser gerada toda a malha, executa-se um processo de suavização. A suavização é simplesmente uma média, para cada vértice, considerando as coordenadas dos vértices pertencentes aos triângulos adjacentes ao vértice em questão. Essa média é repetida quatro ou cinco vezes para dar um resultado final satisfatório e afeta somente os vértices

interiores, já que não se pode alterar as coordenadas dos vértices originais do contorno. A Figura 5.7 mostra a malha final gerada, após todos esses passos.

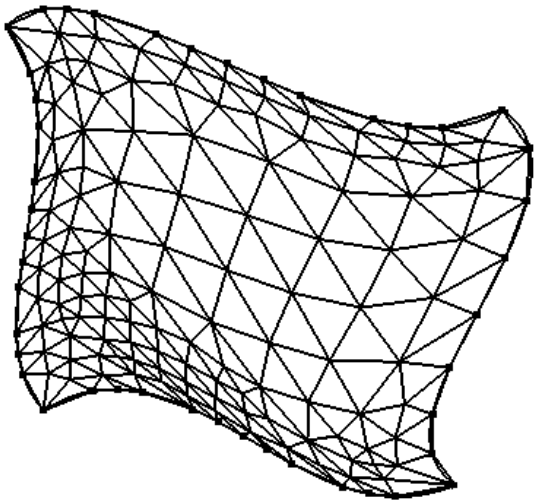

Figura 5.7: Malha final gerada.

Capítulo 6

Exemplos

Esse capítulo é dedicado aos exemplos e resultados, com o objetivo de demonstrar a eficácia da estratégia adaptativa proposta nesse trabalho. Todos os exemplos foram gerados a partir de um sistema gráfico desenvolvido, baseado na estratégia proposta.

Uma observação é importante com relação à modelagem dos *patches*. Para os exemplos cujos modelos se constituem de somente um *patch*, foram criados e considerados *patches* auxiliares que envolvam o *patch* que será modelado. Sem a criação desses *patches* auxiliares envolventes, não se pode ter um resultado razoável para os cálculos das curvaturas discretas na borda, utilizando-se os operadores de curvatura apresentados por Meyer et al. (2002). Isso acontece porque, para cada vértice, precisa-se, para o cálculo de curvatura, da descrição de todos os triângulos adjacentes aos vértices, triângulos esses que perfazem um ciclo com centro no vértice onde se deseja calcular a curvatura discreta. Apesar de não ser objetivo desse trabalho o estudo do cálculo dessas curvaturas discretas, não se considera uma limitação o uso desses *patches* envolventes, já que eles não são considerados como parte do modelo em questão, sendo somente usados para um cálculo mais preciso das curvaturas discretas. Além disso, mesmo com o uso desses *patches*, os resultados foram bastante satisfatórios, o que comprova que a estratégia é eficiente, mesmo que o cálculo dos erros nas curvaturas possa ser ainda melhorado. Um estudo mais aprofundado para melhorar esses operadores de curvatura seria importante, mas isso foge ao escopo desse trabalho.

Para os exemplos que constituem sólidos, não é necessária a criação de qualquer *patch* extra, pois, para cada *patch*, tem-se quatro *patches* vizinhos, podendo os operadores de curvatura discreta na borda serem calculados normalmente, utilizando-se o método adotado neste trabalho e descrito no Capítulo 3.

Uma outra observação é também importante com relação à modelagem dos *patches*. Para efeito de validação da estratégia proposta e considerando que muitos dos problemas que se desejava analisar para esse trabalho podem ser modelados e representados por formulação de Hermite, esse tipo de *patch* foi usado nesse trabalho. A formulação completa dos *patches* de Hermite e todas as informações geométricas podem ser

encontradas no apêndice A. Entretanto, como a estratégia proposta funciona no espaço paramétrico, tanto para o tratamento das curvas, como também dos *patches*, o procedimento de auto-adaptação funciona para quaisquer tipos de curvas e *patches*.

Com relação ao critério de parada, foi utilizada a estimativa de erro global descrita no final do Capítulo 3. Os exemplos mostram que os erros caem, à medida que a malha é melhorada, indicando um processo de convergência.

6.1 Exemplo 1: *Patch* com curvatura alta e fronteira plana (Bolha alta)

Para ilustrar o processo adaptativo de um modelo com alta curvatura no domínio e fronteira plana, usou-se o exemplo de uma bolha cujo modelo está representado pela Figura 6.1**.** Considera-se a representação da geometria do modelo, incluindo a borda e o domínio, além dos *patches* auxiliares, de acordo com a Figura 6.2.

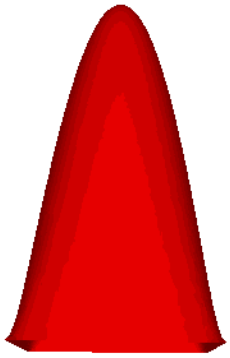

Figura 6.1: Modelo para a bolha.

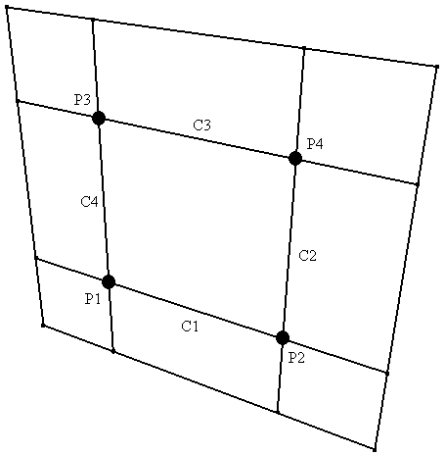

Figura 6.2: Geometria do modelo para o exemplo da bolha.

A Figura 6.2 representa dezesseis vértices, vinte e quatro curvas e nove *patches*. O *patch* central é o que será utilizado para o modelo e todos os outros são *patches* auxiliares, que foram criados para os cálculos de curvatura discreta. O *patch* central (rotulado como *Patch* 1) é formado pelas curvas C1, C2, C3 e C4, onde cada curva está abaixo definida:

- Curva C1, formada pelos vértices P1 e P2,
- Curva C2, formada pelos vértices P2 e P4,
- Curva C3, formada pelos vértices P4 e P3,
- Curva C4, formada pelos vértices P3 e P1,

e, para o *patch*, são definidos os *twists* especificados pelas componentes $t_x = 0$, $t_y = 0$ e $t_z = -30$, ou seja, quatro vetores de módulo 30 no sentido contrário ao eixo z (entrando no plano XY). Esses *twists* são necessários por se tratar de *patches* de Hermite, conforme define a especificação mostrada nos apêndice A e B.

A malha inicial para o modelo é mostrada na Figura 6.3. Como se pode observar, a malha inicial é uma pirâmide, ou seja, a pior representação que a bolha pode ter. Isso foi feito intencionalmente, para mostrar que, mesmo partindo de uma malha de péssima representação, a estratégia consegue chegar em uma malha bastante satisfatória. A Figura 6.4 mostra o resultado do primeiro passo do algoritmo, onde nota-se que mais triângulos foram inseridos no topo da pirâmide, onde o erro de curvatura é maior. Isso acontece também no próximo passo, mostrado na Figura 6.5.

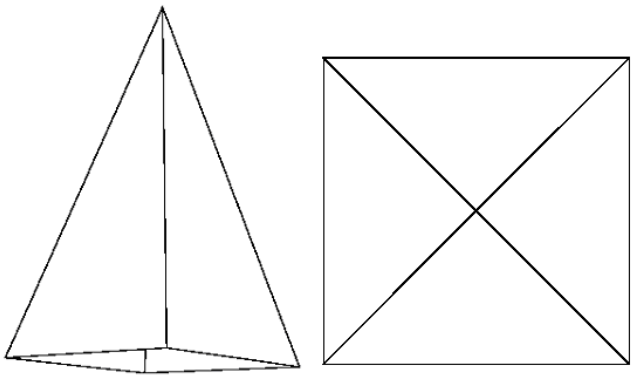

Figura 6.3: Pirâmide representando a malha inicial da bolha.
(Número de vértices: 5; Número de triângulos: 4; $\eta = 0{,}98$ (98%)).

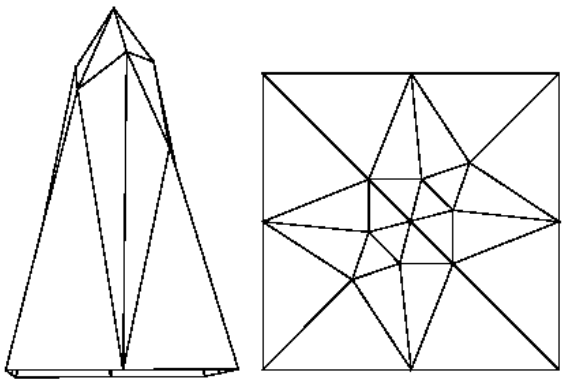

Figura 6.4: Primeiro passo para o bolha.
(Número de vértices: 17; Número de triângulos: 24; $\eta = 0{,}65$ (65%)).

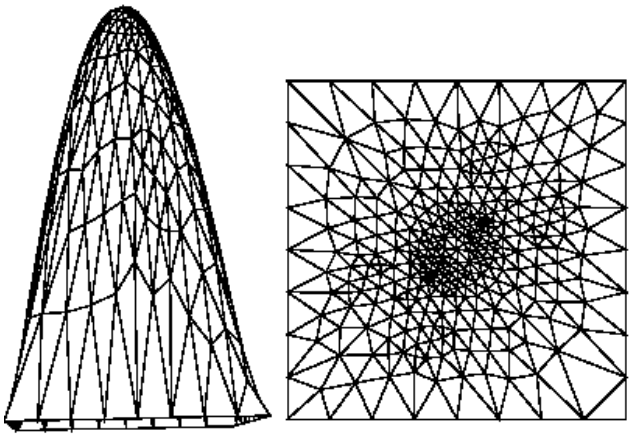

Figura 6.5: Segundo e último passo para o bolha.
(Número de vértices: 317; Número de triângulos: 600; $\eta = 0{,}24$ (24%)).

Nesse exemplo da bolha, percebe-se um maior refinamento no topo da bolha, onde a curvatura é maior. A borda também é refinada, pois existe curvatura nela, já que a curvatura de comparação (a analítica) é a do *patch* e não a da curva, que no caso é uma reta. O erro global caiu de 98% no primeiro passo, para 24% no último. Como se pode perceber nesse exemplo, foram precisos apenas dois passos de adaptação para se reduzir drasticamente o erro, indicando uma convergência rápida e uma eficácia da estratégia proposta. Saiu-se de uma pirâmide, com pouquíssimos triângulos, para uma bolha bem definida, baseada na curvatura desejada.

6.2 Exemplo 2: *Patch* com curvatura baixa e fronteira plana (Bolha baixa)

O exemplo a seguir representa um *patch* com seus vetores *twist* com valores baixos, enquanto que no exemplo anterior (bolha alta), os vetores *twist* se encontram com valores dos módulos elevados, para ficar caracterizada uma bolha com uma alta curvatura. A descrição da geometria do modelo é a mesma do exemplo anterior.

As Figuras 6.7 a 6.9 mostram as malhas para todos os passos efetuados. O erro global caiu de 90% no primeiro passo, para 15% no último. Como se pode perceber nesse exemplo, foram precisos apenas dois passos de adaptação para que o erro caísse para 15%, indicando uma convergência ainda mais rápida e uma eficácia da estratégia proposta. Da mesma forma que o exemplo da bolha alta, saiu-se de uma pirâmide, com pouquíssimos triângulos, para uma bolha bem definida, baseada na curvatura desejada.

Figura 6.6: Modelo para a bolha (curvatura baixa).

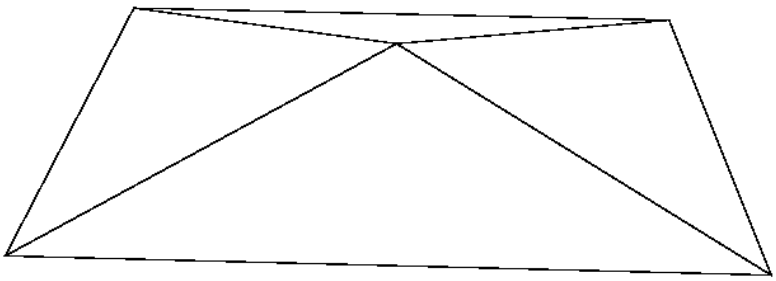

Figura 6.7: *Patch* com vetores *twist* baixo, malha inicial.
(Número de vértices: 5; Número de triângulos: 4; $\eta = 0{,}90$ (90%)).

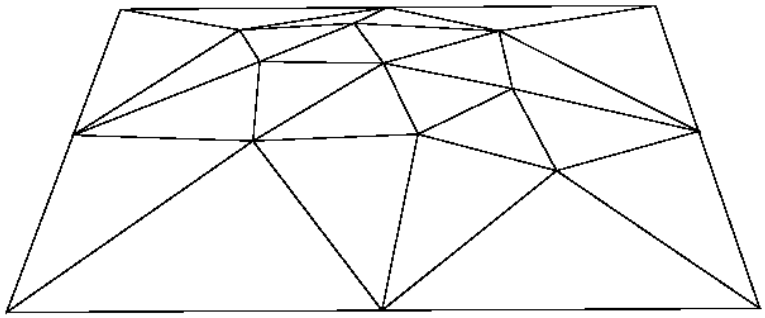

Figura 6.8: *Patch* com vetores *twist* baixo, primeiro passo.
(Número de vértices: 17; Número de triângulos: 24; $\eta = 0{,}37$ (37%)).

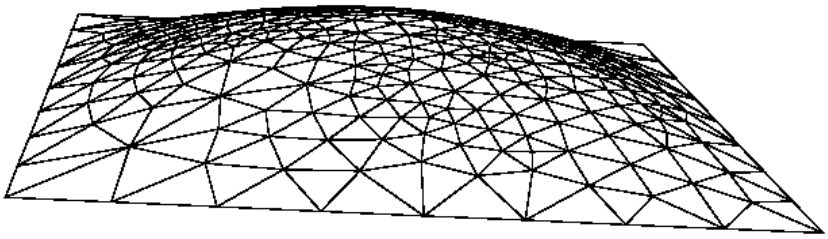

Figura 6.9: *Patch* com vetores *twist* baixo, segundo e último passo.
(Número de vértices: 305; Número de triângulos: 576; $\eta = 0{,}15$ (15%)).

6.3 Exemplo 3: *Patch* com curvatura alta e fronteira não-plana (Sela)

Para ilustrar o processo adaptivo de um modelo com alta curvatura e fronteira não-plana, usou-se a adaptação de uma sela, cujo modelo está representado pela Figura 6.10**.** Considera-se a representação da geometria do modelo, incluindo a borda e o domínio, de acordo com a Figura 6.11. O *patch* central (rotulado como *Patch* 1) é formado pelas curvas C1, C2, C3 e C4, onde cada curva está abaixo definida:

- Curva C1, formada pelos vértices P1 e P2,
- Curva C2, formada pelos vértices P2 e P3,
- Curva C3, formada pelos vértices P3 e P4,
- Curva C4, formada pelos vértices P1 e P4,

e, para o *patch*, são definidos os *twists* especificados pelas componetes $t_x = 0$, $t_y = 0$ e $t_z = 0$, ou seja, seus vetores *twists* têm módulos iguais a zero porque sua fronteira é plana (no sentido de não haver elevação).

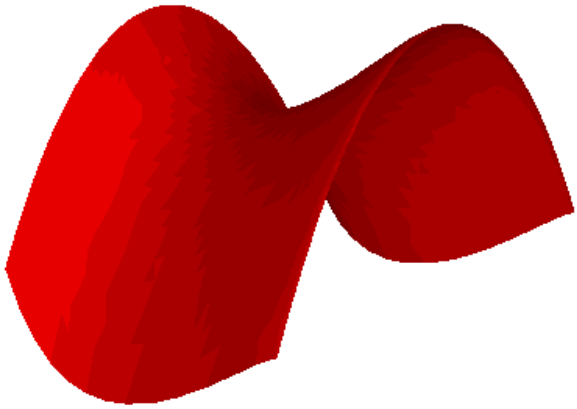

Figura 6.10: Modelo para a sela.

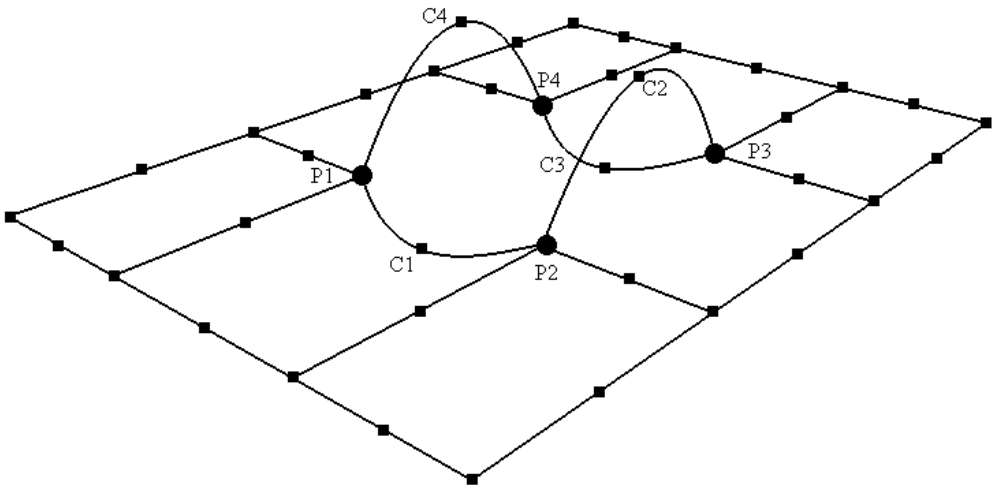

Figura 6.11: Geometria do modelo para o exemplo da sela.

As Figuras 6.12 a 6.14 mostram as malhas para todos os passos efetuados. O erro global caiu de 87% no primeiro passo, para 28% no último. Como se pode perceber nesse exemplo, foi preciso apenas dois passos de adaptação para que o erro caísse, considerando-se que cada curva tem dois segmentos, e a malha inicial tem dezesseis triângulos, como ilustrado na Figura 6.12.

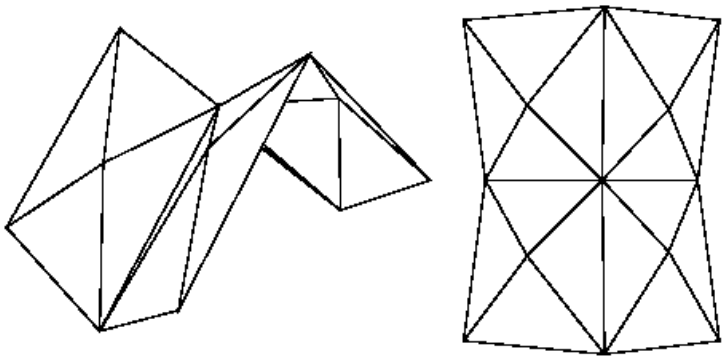

Figura 6.12: Sela, malha inicial respeitando a geometria da Figura 6.11.
(Número de vértices: 13; Número de triângulos: 16; $\eta = 0{,}87$ (87%)).

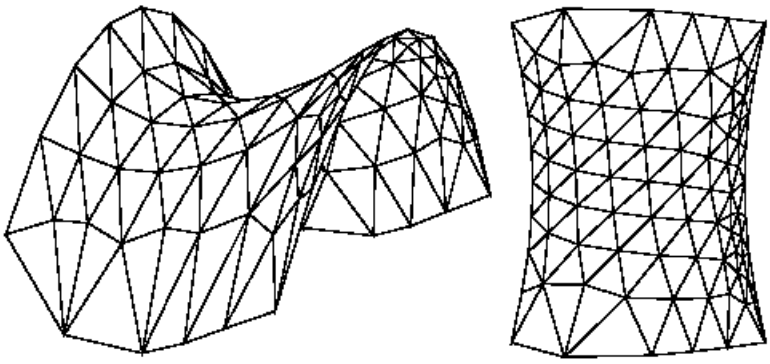

Figura 6.13: Sela, primeiro passo para malha inicial (Figura 6.12).
(Número de vértices: 89; Número de triângulos: 150; $\eta = 0{,}40$ (40%)).

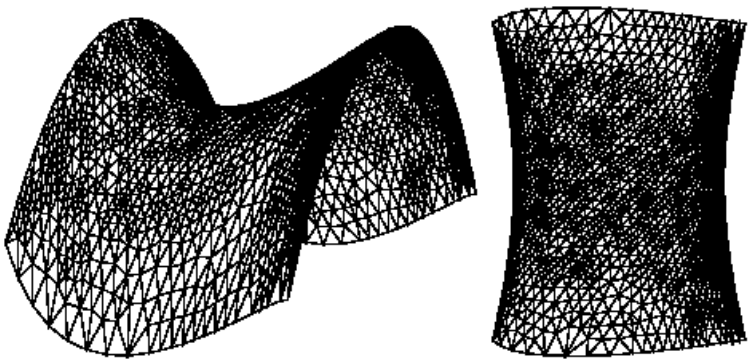

Figura 6.14: Sela, segundo passo.
(Número de vértices: 1783; Número de triângulos: 3426; $\eta = 0{,}28$ (28%)).

Como se pode observar nesse exemplo da sela, a geometria da borda do modelo pode ser divida por qualquer número de segmentos maior ou igual a um. Dependendo da quantidade de segmentos da geometria, o processo de convergência (refinamento e desrefinamento) pode ser acelerado ou retardado de um passo para o outro. Apesar disso, a malha final será atingida e será de configuração similar e de mesma magnitude do tamanho dos triângulos. A Figura 6.15 mostra a malha inicial para sela do modelo da Figura 6.11, mas aqui cada curva de borda só tem um segmento. Comparando as Figuras 6.15 e a 6.12, que são as representações da malhas iniciais, pode-se perceber que a malha da Figura 6.12 é melhor que a da Figura 6.15, pois aquela tem dezesseis triângulos e esta tem apenas quatro, e, por essa razão, a malha do próximo passo da Figura 6.12 (ver Figura 6.13) se encontra melhor do que a malha do próximo passo da Figura 6.15 (ver Figura 6.16). Entretanto, após alguns passos, o modelo convergirá também para o resultado esperado.

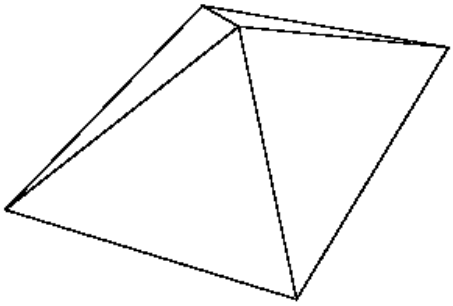

Figura 6.15: Sela, malha inicial respeitando a geometria da Figura 6.11 com apenas um segmento para cada curva.
(Número de vértices: 5; Número de triângulos: 4; $\eta = 0{,}72$ (72%)).

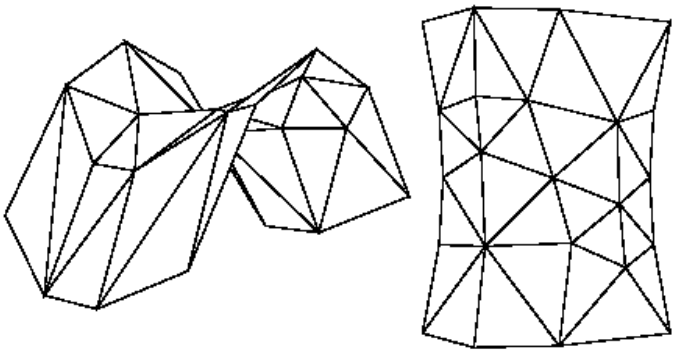

Figura 6.16: Sela, primeiro passo.
(Número de vértices: 23;Número de triângulos: 30; $\eta = 0{,}61$ (61%)).

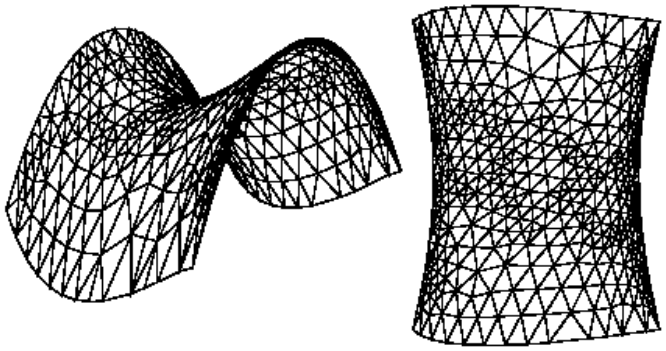

Figura 6.17: Sela, segundo e último passo.
(Número de vértices: 429; Número de triângulos: 782; $\eta = 0{,}31(31\%)$) .

6.4 Exemplo 4: Dois *patches* com curvatura alta (Pneu)

Para ilustrar o processo adaptivo de um modelo com mais de um *patch*, considerou-se um refinamento de um pneu, cujo modelo está representado pela Figura 6.18. Considera-se a representação da geometria do modelo de acordo com a Figura 6.19. Tem-se quatro vértices, seis curvas e dois *patches* organizados da seguinte forma:

- Curva C1, formada pelos vértices P1 e P2,
- Curva C2, formada pelos vértices P1 e P2,
- Curva C3, formada pelos vértices P3 e P4,
- Curva C4, formada pelos vértices P3 e P4,
- Curva C5, formada pelos vértices P1 e P3,
- Curva C6, formada pelos vértices P2 e P4,
- *Patch* 1, formado pelas curvas C1, C6, C3 e C5,
- *Patch* 2, formado pelas curvas C2, C6, C4 e C5.

Para os *Patches* 1 e 2 foram considerados vetores *twist* de módulo 10, perpendicular ao plano YZ. Para o *Patch* 1, seus vetores *twists* estão no sentido contrário ao eixo X, e para *Patch* 2, seus vetores *twists* estão no mesmo sentido do eixo X.

Vale ressaltar que o modelo do pneu da Figura 6.18 tem uma abertura interior, ou seja, o modelo constitui de um sólido oco com aberturas nas suas partes internas (representando realmente um pneu de um automóvel).

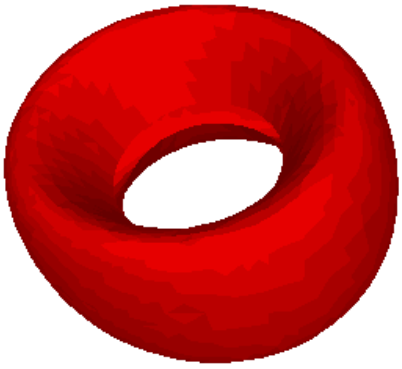

Figura 6.18: Modelo para o pneu.

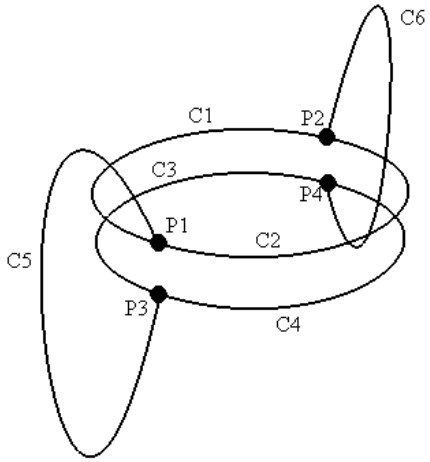

Figura 6.19: Geometria do modelo para o exemplo do pneu.

A Figura 6.20 mostra a primeira malha para o exemplo, considerando a quantidade de segmentos das curvas iguais a um. Foi escolhido esse número para ilustrar a capacidade de refinamento do algoritmo. As próximas figuras mostram os próximos passos do processo, até a formação do pneu. O erro global caiu de 52% no primeiro passo, para 18% no último. Como se pode perceber nesse exemplo, foram precisos também apenas dois passos de adaptação para reproduzir o pneu e para que o erro diminuísse bastante. Da mesma forma que nos outros exemplos, saiu-se de uma malha inicial, com pouquíssimos triângulos, para uma malha final bem definida, baseada na curvatura desejada. Olhando-se a malha inicial é inclusive difícil de se dizer que se trata de um pneu. Isso demonstra que a preocupação com a malha a ser gerada, para uma curvatura desejada, passa a não ser importante, pois para qualquer malha fornecida, mesmo que de qualidade bem baixa, a estratégia proposta nesse trabalho adapta a malha para uma representação bastante satisfatória.

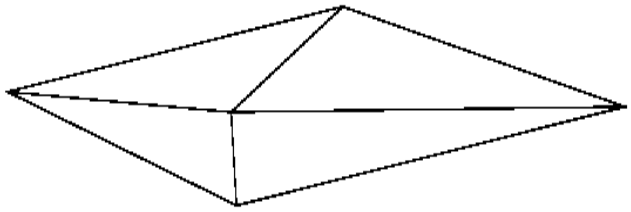

Figura 6.20: Malha inicial para formação do pneu.
(Número de vértices: 6; Número de triângulos: 8; $\eta = 0{,}52$ (52%)).

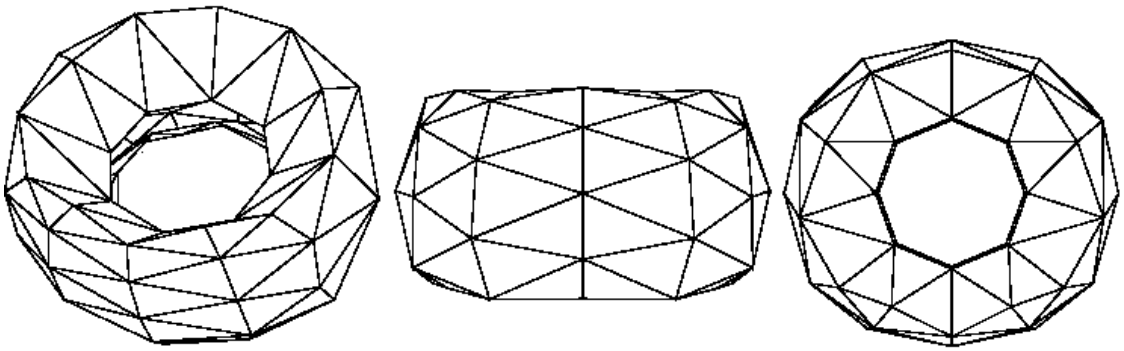

Figura 6.21: Primeiro passo na formação do pneu.
(Número de vértices: 72; Número de triângulos: 128; $\eta = 0{,}33$ (33%)).

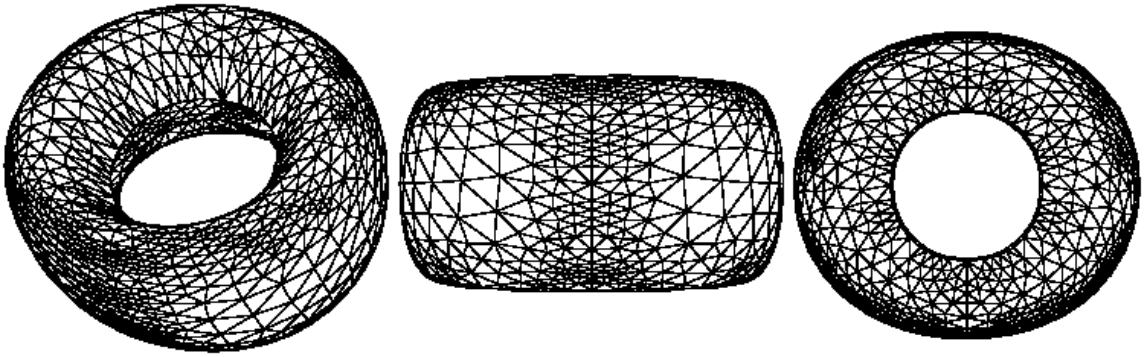

Figura 6.22: Segundo passo na formação do pneu;
(Número de vértices: 1546; Número de triângulos: 2996; $\eta = 0{,}18$ (18%)).

6.5 Exemplo 5: *Patch* Plano (Cubo)

Nos exemplos até aqui mostrados, procurou-se verificar a capacidade de refinamento da estratégia proposta, para a obtenção de uma malha otimizada. Agora, deseja-se mostrar que a estratégia é capaz, também, de desrefinar regiões que estejam com uma quantidade demasiada de triângulos, ou seja, muito refinados. Isso é muito importante em várias situações, como, por exemplo, no caso da testa da face, exemplificada no Capítulo 1. Esse aspecto, entretanto, não é muito encontrado na literatura, já que a maioria dos trabalhos adaptativos se preocupa com o refinamento das malhas, mas não com o seu desrefinamento.

Nesse exemplo, é ilustrado um cubo contendo seis *patches* (um para cada face). A malha inicial é bem refinada e espera-se que sejam desrefinados todos os *patches*, pois não existe curvatura para os vértices do cubo. A Figura 6.23 mostra uma imagem sólida do modelo e a Figura 6.24 mostra a definição da geometria para o modelo em questão.

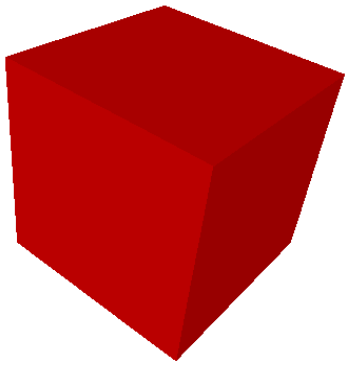

Figura 6.23: Modelo sólido para o cubo.

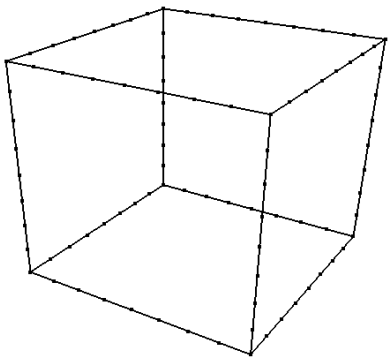

Figura 6.24: Geometria do modelo para o exemplo do cubo.

Nesse modelo não foi necessária a criação de *patches* auxiliares, porque, para cada *patch*, existem quatro outros *patches* vizinhos. As Figuras 6.25 a 6.28 mostram as malhas para todos os passos efetuados. O passo final resultou na malha da Figura 6.28, que é a ideal para esse exemplo, pois como cada *patch* é um plano, apenas dois triângulos são necessários para representar cada face exatamente.

Com relação ao cálculo do erro, entretanto, houve uma distorção nesse cálculo para todos os passos, pois o erro foi sempre 100%. A explicação para essa distorção está relacionada aos problemas nos operadores de curvatura, como mencionado anteriormente. Considere-se, por exemplo, a malha do segundo passo, mostrado na Figura 6.27. As curvaturas analíticas para todos os vértices do modelo são zero, o que é esperado, mas as curvaturas discretas somente são zero para os vértices interiores de cada *patch*. Nos vértices da fronteira, que são os oito cantos do cubo, existe curvatura discreta (nesse caso em torno de 7,14 para a curvatura Gaussiana e 3,94 para a curvatura Média), o que acarreta um erro nesses vértices de 100%, de acordo com a Fórmula 6.1, e, conseqüentemente, em um erro global de 100% para toda a malha conforme a Fórmula 6.2. Como cada um dos oito cantos do cubo têm três superfícies adjacentes, e o calculo das curvaturas discretas leva em consideração todos os triângulos adjacentes a um dado vértice, estando eles em qualquer superfície, isso provavelmente acarreta problemas nesse caso, pelo fato das superfícies fazerem um ângulo de 90 graus entre elas.

Apesar da distorção no cálculo do erro, pode-se perceber que, nesse exemplo, houve um desrefinamento, como desejado, sendo precisos apenas três passos de adaptação. A malha saiu de muitos triângulos (na malha inicial), para apenas dois triângulos por *patch* na malha final, e isso, nesse caso, representa uma malha "ótima", já que, como se trata de um plano, esse é o número mínimo de triângulos que se pode ter para o problema. O fator de refinamento/desrefinamento de 2^n, onde n representa o passo atual do processo de geração da malha adaptativa, também contribuiu para aumentar a convergência. É importante ressaltar que, embora a distorção no cálculo do erro provavelmente se deva a problemas nos operadores de curvatura, que precisam ser mais bem estudados, mesmo com os problemas nesses operadores, a estratégia foi capaz de adaptar a malha de forma muitos satisfatória, resultando na melhor malha possível para o problema em questão.

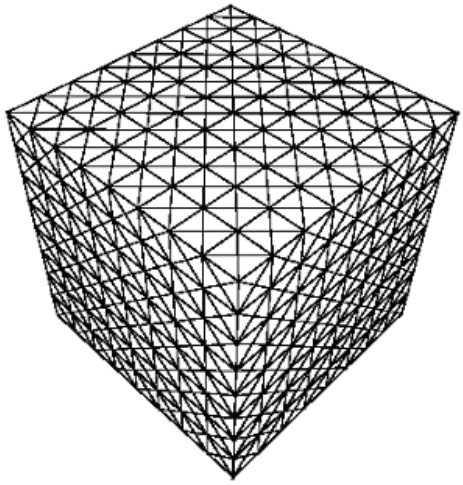

Figura 6.25: Cubo, malha inicial.
(Número de vértices: 590; Número de triângulos: 1176; $\eta = 1{,}0$ (100%)).

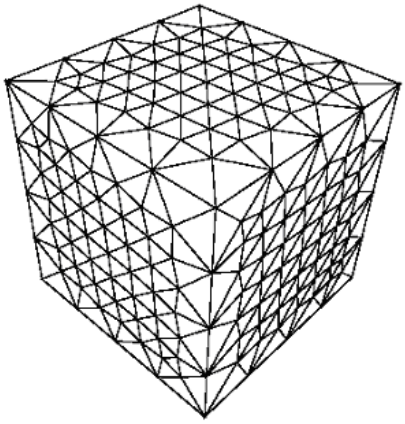

Figura 6.26: Cubo, primeiro passo do desrefinamento.
(Número de vértices: 338; Número de triângulos: 672; $\eta = 1{,}0$ (100%)).

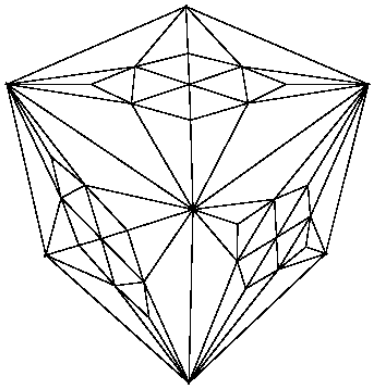

Figura 6.27: Cubo, segundo passo do desrefinamento.
(Número de vértices: 62; Número de triângulos: 120; $\eta = 1{,}0$ (100%)).

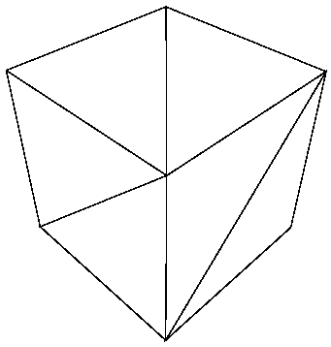

Figura 6.28: Cubo, terceiro passo do desrefinamento.
(Número de vértices: 8; Número de triângulos: 12; η = 1,0 (100%)).

6.6 Outros exemplos

Essa seção ilustra outros exemplos gerados utilizando a estratégia adaptativa. Nos dois exemplos que seguem, a primeira figura representa o modelo e em seguida a representação da geometria de acordo com o modelo. A malha inicial e as malha dos próximos passos são mostrados nas figuras seguintes. Pode-se notar em todos os exemplos que a estratégia gera mais triângulos nas regiões de curvatura mais alta, e que o erro global cai, à medida que a malha é adaptada.

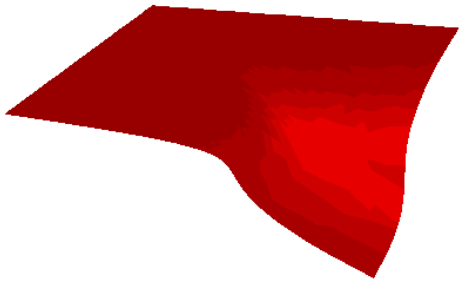

Figura 6.29: Modelo 1.

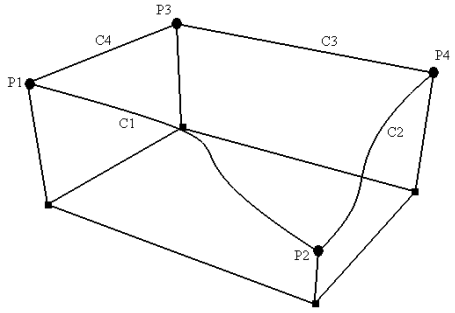

Figura 6.30: Geometria do modelo para a Figura 6.29.

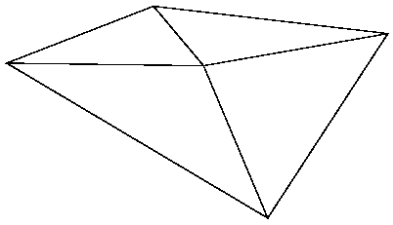

Figura 6.31: Malha inicial.
(Número de vértices: 5; Número de triângulos: 4; $\eta = 0{,}92$ (92%)).

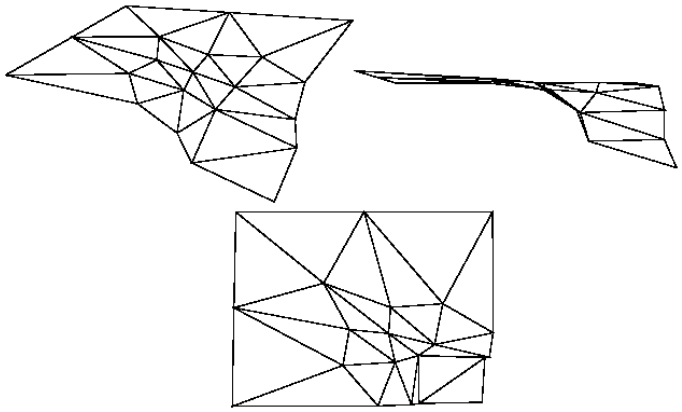

Figura 6.32: Primeiro passo.
(Número de vértices: 21; Número de triângulos: 28; $\eta = 0{,}62$ (62%)).

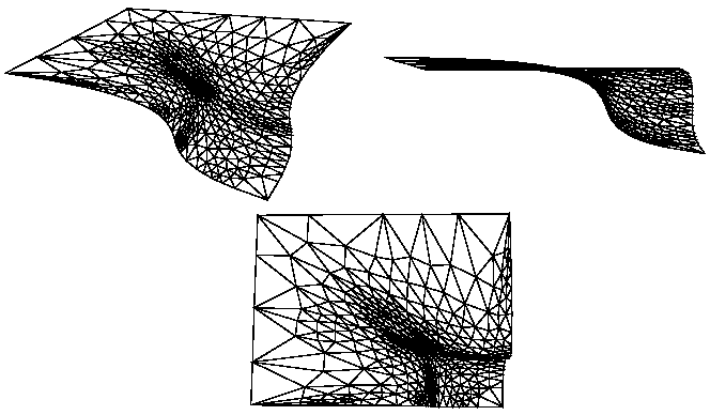

Figura 6.33: Segundo passo.
(Número de vértices: 443; Número de triângulos: 822; $\eta = 0{,}34$ (34%)).

Figura 6.34: Modelo 2.

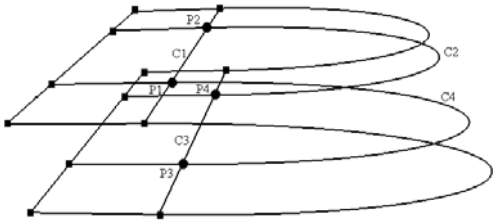

Figura 6.35: Geometria do modelo para a Figura 6.34.

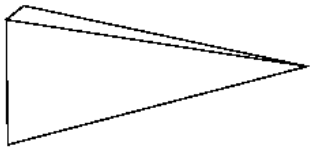

Figura 6.36: Malha inicial.
(Número de vértices: 5; Número de triângulos: 4; $\eta = 0{,}90$ (90%)).

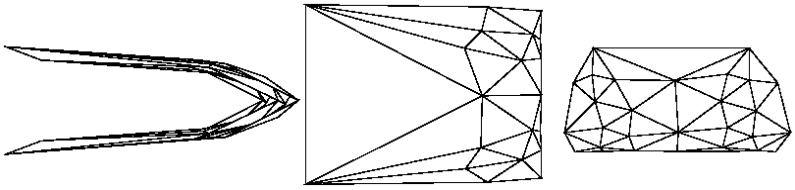

Figura 6.37: Primeiro passo.
(Número de vértices: 33; Número de triângulos: 54; $\eta = 0{,}55$ (55%)).

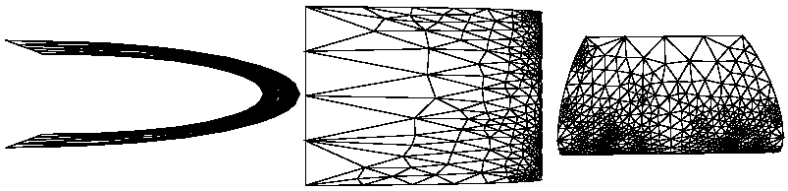

Figura 6.38: Segundo passo;
(Número de vértices: 687; Número de triângulos: 1312; $\eta = 0{,}23$ (23%)).

Capítulo 7

Conclusão

Esse trabalho descreve uma estratégia de auto-adaptação para malhas, utilizando um critério geométrico. Esse critério é usado na adaptação das malhas e é baseado nos erros entre as curvaturas (analítica e discreta), calculadas para regiões da malha. A curvatura analítica é representada pela formulação matemática usada para modelar o domínio e a curvatura discreta é uma aproximação dessa curvatura, que depende da malha utilizada. É importante ressaltar que a estratégia funciona para qualquer tipo de curvas e *patches*, que definem o modelo, desde que eles possam ser parametrizados de [0 a 1], já que ela trata curvas e *patches* de uma forma genérica. A estratégia foi implementada seguindo o paradigma da orientação a objetos, onde esse tratamento genérico é facilmente considerado, permitindo inclusive futuras expansões para inclusão de novas curvas e *patches* de maneira muito rápida e eficiente.

7.1 Principais contribuições

A maioria dos trabalhos na literatura utiliza uma estimativa de erro, baseada em uma análise numérica de elementos finitos, como critério de adaptação para as malhas. Os estimadores buscam a malha ótima tentando igualar o valor do erro nos diversos elementos. Estratégias adaptativas que se baseiam na análise de elementos finitos procuram reduzir o erro de discretização do modelo. O uso da análise de elementos finitos como critério de adaptação para malha nem sempre é possível ou conveniente e necessário, como, por exemplo, em aplicações de computação gráfica, pois a estimativa de erro é baseada em tensões, que não tem muito significado nessas aplicações. Além disso, em muitos casos, o seu custo computacional é alto.

A principal contribuição desse trabalho é uma estratégia de adaptação para malhas utilizando um critério geométrico, baseado no cálculo das curvaturas. A estratégia modifica a discretização das curvas de fronteira, independentemente da modificação discretização do

domínio, que é feita após esse tratamento das curvas. Isso permite que se tenha uma adaptação mais eficiente e uniforme. A estratégia proposta forneceu resultados bastante satisfatórios, obtendo-se malhas otimizadas para os modelos estudados.

Um outro aspecto importante, abordado nesse trabalho, é o critério proposto a ser utilizado na estratégia de adaptação, para se decidir quando refinar, desrefinar ou manter a discretização corrente. Esse critério é baseado na discrepância entre as curvaturas analíticas e discretas para a malha. Apesar de não ser objetivo desse trabalho o estudo da acurácia dos operadores de curvatura, usado para o cálculo das curvaturas discretas, e desses operadores apresentarem alguns problemas para *patches* que não possuam quatro *patches* vizinhos, como mostrado em alguns exemplos, o critério adotado apresentou resultados bastante satisfatórios, mesmo com os problemas em alguns casos por causa desses operadores.

Finalmente, foi desenvolvido um sistema que gera as malhas adaptativas, utilizando a estratégia proposta. Esse sistema gera a malha inicial, considerada uma aproximação inicial, refina as curvas de bordo e gera a nova malha no domínio. O processo é repetido até se alcançar a malha final otimizada. O sistema pode também ler uma malha inicial fornecida, desde que esteja no formato padrão descrito no apêndice B.

7.2 Sugestões para trabalhos futuros

O critério adaptativo, que define como modificar a malha, baseado na discrepância entre as curvaturas analíticas e discretas, apresentou resultados bastante satisfatórios. Entretanto, um estudo mais aprofundado desse critério seria interessante. Em alguns exemplos, o critério gera algumas distorções, embora corrigidas pela estratégia de adaptação proposta, que se mostrou bastante eficaz. Obviamente, um critério ainda melhor gerará resultados ainda mais otimizados.

Um estudo mais aprofundado dos operadores de curvatura discreta também é importante. Esses operadores têm problemas nas bordas de *patches* que não possuem vizinhos. Esse problema foi sanado, em parte, pela adoção de *patches* auxiliares envolventes. E, mais uma vez, a estratégia de adaptação corrigiu as distorções apresentadas. A estratégia de adaptação seria ainda mais eficiente se esses operadores fossem melhorados.

Com relação ao uso dos *patches* auxiliares, eles não foram criados em modelos sólidos, pois cada *patch* tinha os quatros *patches* vizinhos. Entretanto, pelos problemas dos operadores de curvatura, alguns resultados não foram muito satisfatórios, embora corrigidos pela estratégia proposta, principalmente quando na borda de um sólido tem-se uma "aresta viva", o que é comum. Uma sugestão é a criação de *patches* auxiliares para cada superfície, em todos os casos, mesmo em sólidos, somente para o cálculo das curvaturas. Esses *patches* não fariam parte dos sólidos e após o cálculo das curvaturas seriam descartados. Isso permitiria inclusive o tratamento de modelos *não-manifolds* de maneira adequada.

Com relação à geração da malha no domínio, o uso dos padrões triangulares para criar os triângulos internos e o critério de maior ângulo para criar os triângulos perto do contorno, pode gerar distorções nos triângulos gerados. Isso pode acontecer pelo fato dos padrões estarem definidos no espaço paramétrico e dos ângulos serem calculados também nesse espaço paramétrico, o que pode acarretar essas distorções quando esses triângulos forem mapeados para o espaço tridimensional. Uma idéia para evitar esses problemas seria usar uma técnica para geração adaptativa da malha que já levasse em conta as curvaturas no domínio, como por exemplo a técnica mostrada no trabalho de Miranda & Martha (2002).

Um outro aspecto importante a ser considerado é o critério de parada para o processo adaptativo. Nesse trabalho, foi sugerido um processo de avaliação de erro global para a malha, baseado no erro entre as curvaturas. Não se quantificou quando se pararia o processo, pelo fato de não se ter uma melhor noção do que representa realmente esse erro, sendo, portanto, difícil estabelecer um critério de parada. O que se mostrou, que era o objetivo desse trabalho, é que o erro cai à medida que as malhas são adaptadas, indicando que o processo converge. Entretanto, um critério de parada, mais apropriado, melhoraria o processo adaptativo.

Finalmente, sugere-se a adoção da estratégia desse trabalho para aplicações diversas. Por exemplo, em realidade virtual, onde avatares são modelados, a estratégia pode ser utilizada para gerar malhas otimizadas para esses avatares. O que se vê freqüentemente são avatares com malhas muito refinadas em regiões planas, como a testa, por exemplo, e não suficientemente refinadas em regiões de alta curvatura, como o nariz e a boca. Um outro exemplo de aplicação da estratégia proposta seria em simulações de deslocamento de objetos, onde a cada unidade de tempo a malha do objeto deve ser modificada, quando necessário e segundo alguns critérios.

Apêndice A

Formulação matemática para curvas e *patch* Hermite[2]

A expressão matemática que calcula cada ponto no espaço tridimensional de cada curva Hermite, dentro de seu espaço paramétrico ($0 \leq u \leq 1$), e que permite a sua construção é:

$$\mathbf{P}(u) = \mathbf{P_i} * \mathbf{F_1}(u) + \mathbf{P_j} * \mathbf{F_2}(u) + \mathbf{DP_i} * \mathbf{F_3}(u) + \mathbf{DP_j} * \mathbf{F_4}(u), \qquad \textbf{(A.1)}$$

onde $\mathbf{F_1}(u)$, $\mathbf{F_2}(u)$, $\mathbf{F_3}(u)$ e $\mathbf{F_4}(u)$ são as *blending functions* e são dadas por:

$$\mathbf{F_1}(u) = (2u^3 - 3u^2 + 1), \qquad \textbf{(A.2)}$$

$$\mathbf{F_2}(u) = (- 2u^3 + 3u^2), \qquad \textbf{(A.3)}$$

$$\mathbf{F_3}(u) = (u^3 - 2u^2 + u), \qquad \textbf{(A.4)}$$

$$\mathbf{F_4}(u) = (u^3 - u^2), \qquad \textbf{(A.5)}$$

$\mathbf{P_i}$ e $\mathbf{P_j}$ são os pontos extremos da curva, e, além disso, $\mathbf{DP_i}$ e $\mathbf{DP_j}$ representam as derivadas nesses pontos, respectivamente. As figuras abaixo ilustram exemplos desse tipo de curva com seus atributos.

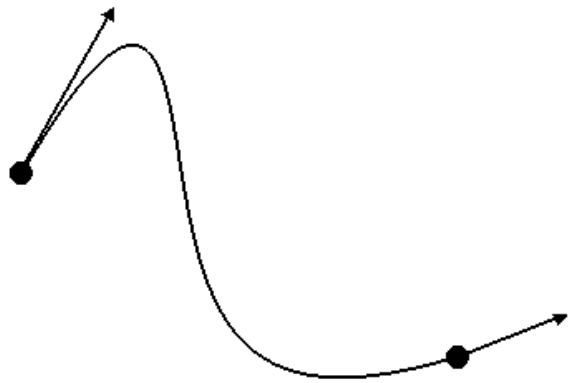

Figura A.1: Curva Hermite.
$\mathbf{P_i}$ = (-0.7 , 0.1 , 0); $\mathbf{P_j}$ = (0.7 , -0.1 , 0); $\mathbf{DP_i}$ = (3 , 5 , 0) e $\mathbf{DP_j}$ = (4 , 3 , 0).

[2] Na verdade a formulação é genérica para qualquer *patch* de superfícies bicúbicas (com curvas cúbicas). O termo *patch* Hermite nesse trabalho se refere a *patches* de superfícies bicúbicas limitados por curvas Hermite.

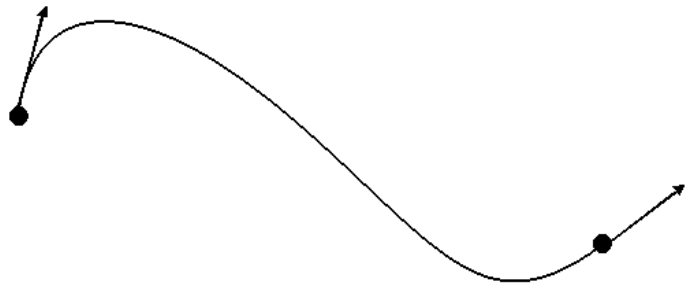

Figura A.2: Curva Hermite.
$\mathbf{P_i}$ = (-0.5, 0.5, 0.5); $\mathbf{P_j}$ = (0.8, -0.5, -1.5); $\mathbf{DP_i}$ = (3,5,2) e $\mathbf{DP_j}$ = (3,5,2).

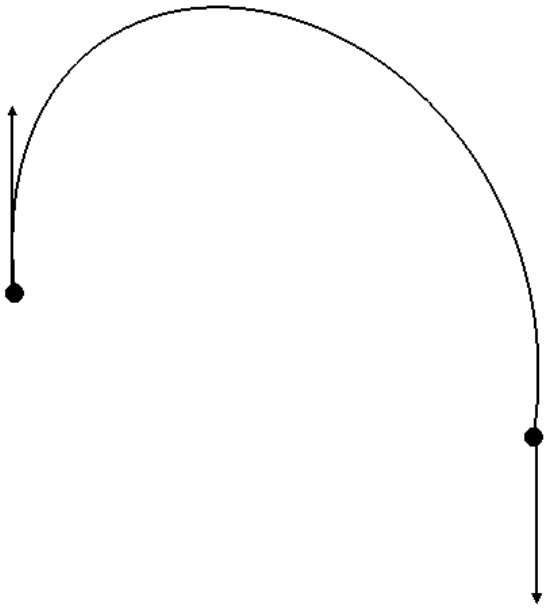

Figura A.3: Curva Hermite.
$\mathbf{P_i}$ = (-1 , 0 , 0); $\mathbf{P_j}$ = (1 , 0 , 0); $\mathbf{DP_i}$ = (0 , 5 , 0) e $\mathbf{DP_j}$ = (0 , -5 , 0).

A expressão matemática que calcula cada ponto no espaço tridimensional de cada *patch* Hermite, dentro de seu espaço paramétrico ($0 \leq u \leq 1$ e $0 \leq w \leq 1$), e que permite a sua construção é:

$$
\begin{aligned}
\mathbf{Q}(u,w) = \\
&\mathbf{F_1}(w) * [\, \mathbf{F_1}(u)*\mathbf{P}(0,0) + \mathbf{F_2}(u)*\mathbf{P}(1,0) + \mathbf{F_3}(u)*\mathbf{P_u}(0,0) + \mathbf{F_4}(u)*\mathbf{P_u}(1,0)] + \\
&\mathbf{F_2}(w) * [\, \mathbf{F_1}(u)*\mathbf{P}(0,1) + \mathbf{F_2}(u)*\mathbf{P}(1,1) + \mathbf{F_3}(u)*\mathbf{P_u}(0,1) + \mathbf{F_4}(u)*\mathbf{P_u}(1,1)] + \\
&\mathbf{F_3}(w) * [\, \mathbf{F_1}(u)*\mathbf{P_w}(0,0) + \mathbf{F_2}(u)*\mathbf{P_w}(1,0) + \mathbf{F_3}(u)*\mathbf{P_{u\,w}}(0,0) + \mathbf{F_4}(u)*\mathbf{P_{u\,w}}(1,0)] + \\
&\mathbf{F_4}(w) * [\, \mathbf{F_1}(u)*\mathbf{P_w}(0,1) + \mathbf{F_2}(u)*\mathbf{P_w}(1,1) + \mathbf{F_3}(u)*\mathbf{P_{u\,w}}(0,1) + \mathbf{F_4}(u)*\mathbf{P_{u\,w}}(1,1)] \qquad \textbf{(A.6)}
\end{aligned}
$$

onde $\mathbf{F_1}(u)$, $\mathbf{F_2}(u)$, $\mathbf{F_3}(u)$ e $\mathbf{F_4}(u)$ são as *blending functions* já mencionadas, e $\mathbf{P}(0,0)$, $\mathbf{P}(1,0)$, $\mathbf{P}(0,1)$ e $\mathbf{P}(1,1)$ representam os quatro vértices que formam o *patch*. $\mathbf{P_{uw}}(0,0)$, $\mathbf{P_{uw}}(1,0)$, $\mathbf{P_{uw}}(0,1)$ e $\mathbf{P_{u\,w}}(1,1)$ representam os vetores *twist* (derivadas parciais) dos quatros vértices.

No arquivo de entrada de dados (ver Apêndice B), para a definição da geometria de cada *patch* serão fornecidas as quatro curvas que irá compor esse *patch* e os seus quatros vetores *twist* $\mathbf{P_{uw}}(0,0)$, $\mathbf{P_{uw}}(1,0)$, $\mathbf{P_{uw}}(0,1)$ e $\mathbf{P_{u\,w}}(1,1)$.

As figuras a seguir ilustram exemplos desse tipo de *patch* com seus atributos:

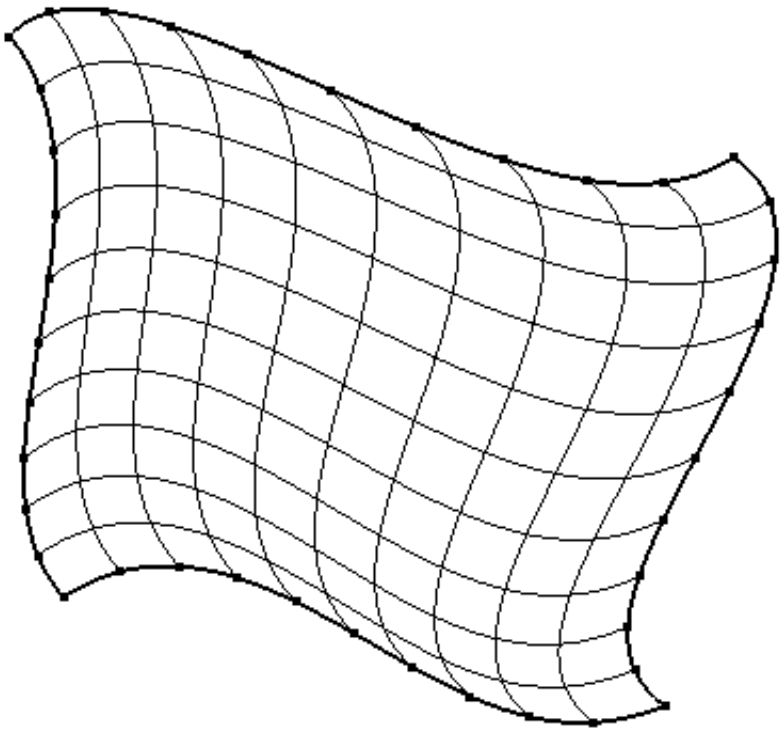

Figura A.4: Patch Hermite.
$\mathbf{P}(0,0)=(-0.8,-0.8,0)$; $\mathbf{P}(1,0)=(0.8,-0.5,0)$; $\mathbf{P}(0,1)=(-0.8,0.8,0)$; $\mathbf{P}(1,1)=(0.8,0.8,0)$;
$\mathbf{P_u}(0,0)=(2,2,0)$; $\mathbf{P_u}(1,0)=(1.5,1.5,0)$; $\mathbf{P_u}(0,1)=(-1,0.5,0)$; $\mathbf{P_u}(1,1)=(-1,-1,0)$;
$\mathbf{P_w}(0,0)=(-1,-1,0)$; $\mathbf{P_w}(1,0)=(1,-1,0)$; $\mathbf{P_w}(0,1)=(1,-1,0)$; $\mathbf{P_w}(1,1)=(1,-1,0)$;
$\mathbf{P_{uw}}(0,0)=(0\ ,\ 0\ ,\ 0)$; $\mathbf{P_{uw}}(1,0)=(0\ ,\ 0\ ,\ 0)$; $\mathbf{P_{uw}}(0,1)=(0\ ,\ 0\ ,\ 0)$; $\mathbf{P_{uw}}(1,1)=(0\ ,\ 0\ ,\ 0)$.

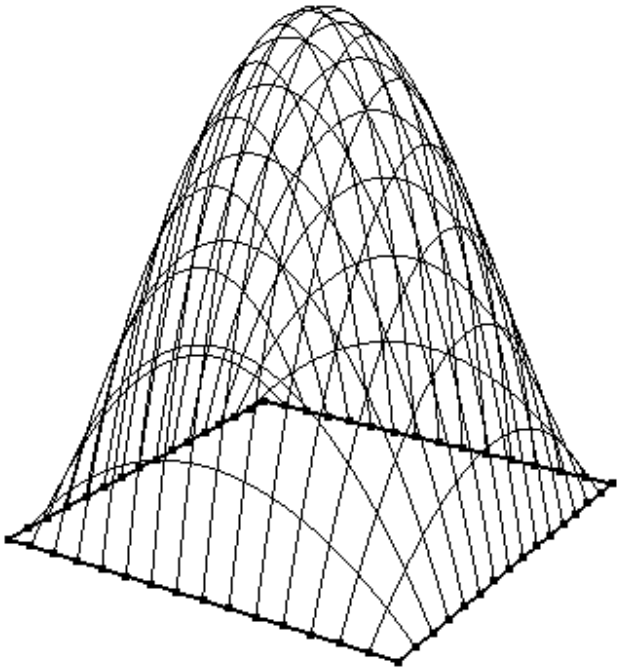

Figura A.5: Patch Hermite.
P(0,0)=(-0.5,0,0.5); **P**(1,0)=(0.5,0,0.5); **P**(0,1)=(-0.5,0,-0.5); **P**(1,1)=(0.5,0,-0.5);

$\mathbf{P_u}(0,0)=(1,0,0)$; $\mathbf{P_u}(1,0)=(1,0,0)$; $\mathbf{P_u}(0,1)=(1,0,0)$; $\mathbf{P_u}(1,1)=(1,0,0)$;

$\mathbf{P_w}(0,0)=(0,-1,0)$; $\mathbf{P_w}(1,0)=(0,-1,0)$; $\mathbf{P_w}(0,1)=(0,-1,0)$, $\mathbf{P_w}(1,1)=(0,-1,0)$;

$\mathbf{P_{uw}}(0,0)=(0,20,0)$; $\mathbf{P_{uw}}(1,0)=(0,-20,0)$; $\mathbf{P_{uw}}(0,1)=(0,-20,0)$; $\mathbf{P_{uw}}(1,1)=(0,20,0)$.

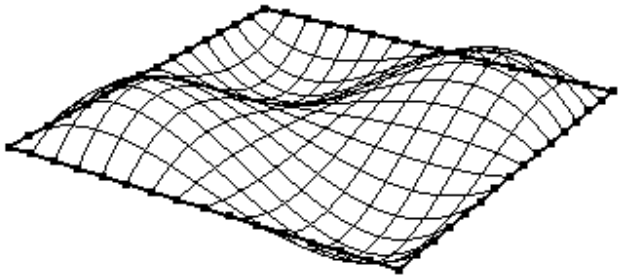

Figura A.6: Patch Hermite.
P(0,0)=(-0.5,0,0.5); **P**(1,0)=(0.5,0,0.5); **P**(0,1)=(-0.5,0,-0.5) , **P**(1,1)=(0.5,0,-0.5);

$\mathbf{P_u}(0,0)=(1,0,0)$; $\mathbf{P_u}(1,0)=(1,0,0)$; $\mathbf{P_u}(0,1)=(1,0,0)$; $\mathbf{P_u}(1,1)=(1,0,0)$;

$\mathbf{P_w}(0,0)=(0,-1,0)$; $\mathbf{P_w}(1,0)=(0,-1,0)$; $\mathbf{P_w}(0,1)=(0,-1,0)$; $\mathbf{P_w}(1,1)=(0,-1,0)$;

$\mathbf{P_{uw}}(0,0)=(0,15,0)$; $\mathbf{P_{uw}}(1,0)=(0,15,0)$; $\mathbf{P_{uw}}(0,1)=(0,15,0)$; $\mathbf{P_{uw}}(1,1)=(0,15,0)$.

Apêndice B

Formato de descrição da geometria da fronteira

Para a criação de curvas e *patches* no sistema desenvolvido nesse trabalho, foi necessário definir um padrão de arquivo, pois através dele é possível importar, alterar e salvar modificações na geometria especificada. O padrão segue as seguintes definições:

- Inicialmente, definem-se todos os vértices utilizados pelo modelo. Cada vértice deve conter um identificador seguido do caractere ":" (dois pontos) e suas coordenadas Cartesianas separadas por vírgula. O escopo de definição dos vértices está entre as palavras reservadas PONTOS e FIM_DE_PONTOS;
- As tangentes poderiam ser definidas dentro do escopo dos vértices, mas optou-se por um escopo à parte com o objetivo de aumentar a compreensão e a legibilidade do formato. A definição das tangentes é análoga aos vértices;
- A seguir definem-se as curvas. Primeiro especifica-se o tipo de curvas que vem a seguir, pela palavra reservada CURVAS_TIPO (por exemplo, CURVAS_HERMITE). Cada tipo de curva deve ter uma identificação e seus próprios atributos inerentes ao tipo em questão. Por exemplo, para a curva do tipo Hermite, são fornecidos os seus vértices extremos e suas tangentes, todos em linhas separadas. A palavra reservada FIM_DA_CURVA_(Identificação) indica o final de uma curva específica. Após as definições de todas as curvas do modelo, a palavra reservada FIM_CURVAS_TIPO (por exemplo, FIM_CURVAS_HERMITE) finaliza essa definição;
- Finalmente, têm-se as definições dos *patches*. A palavra reservada PATCHES_TIPO (por exemplo, PATCHES_HERMITE) indica a definição dos *patches* do tipo especificado. Cada *patch* deve conter uma identificação e sua definição. Para o *patch* Hermite, após a identificação, deve-se ter (após o símbolo ":") as identificações das quatro curvas que o compõem. A seguir, deve-se especificar os atributos inerentes a cada *patch* (para o *patch* Hermite são os quatro vetores *twists* dos quatros vértices do canto). A palavras reservada FIM_DE_PATCH_(Identificação) finaliza o *patch* sendo definido. Finalmente, a palavra reservada FIM_PATCHES_TIPO (por exemplo, FIM_PATCHES_HERMITE) finaliza a definição de todos os *patches* desse tipo.

O arquivo segue então o seguinte padrão:

```
// Cabeçalho
==========> MoDIELador <==========
 **** O MODELADOR DESENVOLVIDO POR DANIEL (DIEL) ****
 **** VERSÃO 1.0 ****
NUM_SEG N // Definição da quantidade de segmentos de cada curva
// Definição de todos os vértices utilizados
PONTOS
P1  : x , y , z
 . . .
Pn  :  x ,  y , z
FIM_DE_PONTOS
// Definição de todas as tangentes utilizadas
TANGENTES
T1 :    x ,  y , z
 .  .  .
Tn :    x ,  y , z
FIM_DE_TANGENTES
// Definição de curvas, indicando o seu tipo no inicio e definindo um rótulo para a mesma.
CURVAS_HERMITE
C1
Pi  : P1
Pj  : P2
DPi : T1
DPj : T1
FIM_DA_CURVA_C1
 .  .  .
Cn
Pi  : P1
Pj  : P3
DPi : T2
DPj : T2
FIM_DA_CURVA_Cn
FIM_CURVAS_HERMITE
// Definição de patches, indicando o seu tipo no início e definindo um rótulo para o mesmo.
PATCHS_HERMITE
PATCH_1 : C1 C2 C3 C4
VETORES_TWIST
Puw00 : x , y , z
Puw01 : x , y , z
Puw10 : x , y , z
Puw11 : x , y , z
FIM_DE_PATCH_PATCH_1
 .  .  .
PATCH_n : C1 C2 C3 C4
VETORES_TWIST
Puw00 : x , y , z
Puw01 : x , y , z
Puw10 : x , y , z
Puw11 : x , y , z
FIM_DE_PATCH_PATCH_n
FIM_DE_PATCHS_HERMITES
FIM_DO_ARQUIVO_C:\Exemplo.txt
```

O Exemplo abaixo descreve a geometria do modelo usada nos capítulos 2, 3, 4 e 5:

```
==========> MoDIELador <==========
 **** O MODELADOR DESENVOLVIDO POR DANIEL (DIEL) ****
NUM_SEG 3
PONTOS
P1  : -0.8 , -0.8 , 0
P2  : 0.8 ,  -0.5 , 0
P3  : -0.8 ,  0.8 , 0
P4  :  0.8 ,  0.8 , 0
FIM_DE_PONTOS
TANGENTES
T1 :    2 ,  2 , 0
T2 :    1.5 ,  1.5 , 0
T3 :    -1 ,  0.5 , 0
T4 :    -1 ,  -1 , 0
T5 :    -1 ,  -1 , 0
T6 :    1 ,  -1 , 0
FIM_DE_TANGENTES
CURVAS_HERMITE
C1
Pi  : P1
Pj  : P2
DPi : T1
DPj : T2
FIM_DA_CURVA_C1
C2
Pi  : P2
Pj  : P4
DPi : T3
DPj : T3
FIM_DA_CURVA_C2
C3
Pi  : P4
Pj  : P3
DPi : T5
DPj : T5
FIM_DA_CURVA_C3
C4
Pi  : P3
Pj  : P1
DPi : T6
DPj : T6
FIM_DA_CURVA_C4
FIM_CURVAS_HERMITE
PATCHS_HERMITE
PATCH_1 : C1 C2 C3 C4
VETORES_TWIST
Puw00 : 0 , 0 , 1
Puw01 : 0 , 0 , -1
Puw10 : 0 , 0 , 1
Puw11 : 0 , 0 , 1
FIM_DE_PATCH_PATCH_1
FIM_DE_PATCHS_HERMITES
FIM_DO_ARQUIVO_C:\usuarios\Diel\ExemploDaDissertacao.txt
```

Referências bibliográficas

Babuska, I. & Rheimboldt, W. C. (1979). *Adaptive Approaches and Realiability Estimation in Finite Element Analysis.* Comp. Meth. In Applied Mec. And Engn., vol. 18, pp. 519-540.

Baehmann, P. L.; Wittchen, S. L. & Shephard, M. S. (1987). *Robust Geometrically Based, Automatic Two-Dimensional Mesh Generation.* Int. J. Num. Meth. Engng., vol. 24, pp. 1043-1078.

Banichuk, N.V.; Barthold, F. J.; Falk, A. & Stein, E. (1995). *Mesh refinement for shape optimization.* Structural Optimization, Springer-Verlag, Vol 9, pp.46-51.

Bank, R. E. & Smith, R. K. (1997). *Mesh Smoothing Using A Posteriori Error Estimates.* SIAM J. Numerical Analysis, Elsevier, Vol 34, pp.979-997.

Berg, M.; Kreveld, M.; Overmars, M. & Schwarzkopf, O. (1997). *Computational Geometry: Algorithms and Applications*. Springer-Verlag.

Bond, T. J.; Li, L. Y.; Bettess, P.; Bull, J. W. & Applegarth, I. (1996). *Adaptive Mesh Refinement for Shells with Modified Ahmad Elements.* Computers and Structures, Pergammon, Vol 61, Num 6, pp.1135-1141.

Borouchaki, H.; Castro-Diaz, M.J.; George, P.L.; Hecht, F. & Mohammadi, B. (1996). *Anisotropic adaptive mesh generation in two dimensions for CFD.* 5th International Conference On Numerical Grid Generation in Computational Field Simulations, Mississppi State University, Vol 3, pp.197-206.

Cavalcante-Neto, J. B.; Carvalho, M. T. M. & Martha, L. F. (1993). *Combinação das Técnicas de Quadtree e Delaunay para Geração Automática de Malhas de Elementos Finitos.* In: SIBGRAPI'93 (VI Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens), Recife, Pernambuco, Brasil. Anais do VI Simpósio Brasileiro de computação gráfica e processamento de imagens. Recife: UFPE/SBC, 1993. v. 1, pp. 285-291.

Cavalcante-Neto, J. B.; Carvalho, M. T. M. & Martha, L. F. (1993). *Geração Auto-adaptativa de Malhas de Elementos Finitos Utilizando uma Técnica de Enumeração Espacial Recursiva.* In: CILAMCE'1993 (XVI Congresso Ibero-Latino Americano sobre Métodos Computacionais para Engenharia), São Paulo, Brasil. Anais do XVI Congresso Ibero Latino Americano sobre Métodos Computacionais para Engenharia. São Paulo: IPT-USP/AMC, 1993. v. 2, p. 1285-1294.

Cavalcante-Neto, J. B. (1994). *Simulação Auto-adaptativa Baseada em Enumeração Espacial Recursiva de Modelos Bidimensionais de Elementos Finitos*. Dissertação de Mestrado, PUC-Rio.

Cavalcante-Neto, J. B.; Martha, L. F.; Menezes, I. F. M. & Paulino, G. H. (1998). *A Methodology for Self-Adaptive Finite Element Analysis Usando an Object Oriented Approach*. In: WCCM'1998 (Forth World Congress on Computational Mechanics), Buenos Aires, Agentina. Preceedings of the Forth World Congress on Computational Mechanics, Computational Mechanics: New Trends and Applications. Buenos Aires: IACM, 1998. v. 2, p. 1-20.

Cheng, B. & Topping, B. H. V. (1998). *Improved adaptive quadrilateral mesh generation using fission elements*. Advances in Engineering Software, Elsevier, Vol 29, Num 7, pp.733-744.

Chew, L. P. (1989). *Constrained Delaunay Triangulation.* Algorithmica, vol. 4, pp. 97-108.

Dill, J. C. (1981). *An Application of Color Graphics to the Display of Surface Curvature.* Comp. Graph., Vol. 15, pag. 153-161. SIGGRAPH 81.

George, P. L. (1999). *Tet Meshing: Construction, Optimization, And Adaptation.* Proceedings, 8th International Meshing Roundtable, South Lake Tahoe, CA, U.S.A., pp.133-141.

Hinton, E.; Rao, N.V.R. & Ozakca, M. (1991). *Mesh Generation With Adaptive Finite Element Analysis*. Advances in Engineering Software, Elsevier, Vol 13, Num 5, pp.238-262.

Holzer, S. M (1999). *Mesh Generation for hp-Type Finite Element Analysis of Plates.* 2nd Symposium on Trends in Unstructured Mesh Generation, University of Colorado, Boulder.

Joe, B. (1986). *Delaunay Triangular Meshes in Contex Polygons.* SIAM J. Sci. Comput., vol. 7, pp. 514-539.

Katragadda, P. & Grosse, I.R. (1996). *A Posteriori Error Estimation and Adaptive Mesh Refinement for Combined Thermal-Stress Finite Element Analysis.* Computers and Structures, Pergammon, Vol 59, Num 6, pp.1149-1163.

Kiahs, J.; Shoaf, S.; Russell, R. & Ward, P.. (1995). *Finite Element Analysis Using Adaptive Mesh Generation.* SDRC report, SDRC.

Lee, C. K. & Hobbs, R. E. (1999). *Automatic adaptive finite element mesh generation over arbitrary two-dimensional domain using advancing front technique*. Computers and Structures, Pergammon, Vol 71, pp.9-34.

Lo, S. H. (1989). *Delaunay Triangulation of Non-Convex Planar Domains.* International Journal for Numerical Methods in Engineering, vol. 28, pp. 2695-2707.

Meyer, M.; Desbrun, M.; Schröder, P. & Barr, A. H. (2002). *Discrete Differential-Geometry Operators for Triangulated 2-Manifolds.* In Visualization and Mathematics III, H.-C. Hege and K. Polthier eds., pp. 35–57.

Miranda, A. C. O.; Cavalcante-Neto, J. B. & Martha, L. F. (1999) *An Algorithm for Two-Dimensional Mesh Generation for Arbitrary Regions with Cracks.* In: SIBGRAPI'99 (XII Brazilian Symposium on Computer Graphics, Image Processing and Computer Vision), Campinas, São Paulo, Brasil. Proceedings of XII Brazilian Symposium on Computer Graphics, Image Processing and Computer Vision. Unicamp/SBC, IEEE Computer Society Order Number PRO0481, ISBN 0-7695-0481-7, 1999. v. 1, pp. 29-38.

Miranda, A. C. O. & Martha, L. F. (2002). *Mesh Generation on High-Curvature Surfaces Based on a Background Quadtree Structure.* In: 11th International Meshing Roundtable, 2002, Ithaca, NY, EUA. Proceedings of 11th International Meshing Roundtable. , 2002. p.333 – 341.

Modi, A. (1997). *Unstructured mesh generation on planes and surfaces using graded triangulation*. Department of Aerospace Engineering. Indian Institute of Technology, Bombay.

O'Rourke, J. (1998) *Computational Geometry in C*. Cambridge University Press. Segunda edição.

Paulino, G. H.; Menezes, I. F. M.; Cavalcante-Neto, J. B.; Martha, L. F.; (1999). *A Methodology for Adaptive Finite Element Analysis: Towards an Integrated Computacional Environment.* Computational Mechanics, Springer-Verlag HeideIberg, v. 23, n 5/6, p. 361-388.

Potyondy, D. O. (1993). *A Software Framework for Simulating Curvilinear Crack Growth in Pressurized Thin Shells.* Ph. D. Thesis, School of Civil Engineering, Cornell University.

Rogers, D. F. & Adams, J. A. (1990). *Mathematical elements for computer graphics.* 2nd ed., pag. 420-421.

Samet, H. (1984). *The Quadtree e Related Hierarchical Data Structures.* ACM Computer Surveys, vol. 16, no. 2.

Shaw, R. D. & Pitchen, R. G. (1978). *Modifications to the Suhara-Fukuda Method of Network Generation.* Int. J. Num. Meth. Engng., vol. 12, pp. 93-99.

Vianna, A. C. (1992). *Modelagem Geométrica completa para modelos bidimensionais de elementos finitos.* Dissertação de Mestrado, Departamento de Engenharia Civil, PUC-Rio.

Von, H. B. & Barr, A. H. (1987). *Accurate Triangulations of Deformed, Intersecting Surfaces.* Computer Graphics, vol. 21, pp. 103-110.