PONTIFÍCIA UNIVERSIDADE CATÓLICA DO PARANÁ

MÁRLON DE OLIVEIRA VAZ

GERAÇÃO DE MALHAS DE ELEMENTOS FINITOS TRIANGULARES EM DOMÍNIOS PLANOS USANDO O MÉTODO DO AVANÇO DA FRONTEIRA

MESTRADO EM
ENGENHARIA MECÂNICA
PUCPR

CURITIBA
2003

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO PARANÁ

MÁRLON DE OLIVEIRA VAZ

GERAÇÃO DE MALHAS DE ELEMENTOS FINITOS TRIANGULARES EM DOMÍNIOS PLANOS USANDO O MÉTODO DO AVANÇO DA FRONTEIRA

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Engenharia Mecânica, Curso de PósGraduação em Engenharia Mecânica, Departamento de Ciências Exatas e de Tecnologia, Pontifícia Universidade Católica do Paraná

Orientador: Prof. Dr. João Elias Abdalla Filho

CURITIBA
2003

Vaz, Márlon de Oliveira
V393g Geração de malhas de elementos triangulares em domínios
2003 planos usando o método do avanço da fronteira / Márlon de Oliveira Vaz ; orientador, João Elias Abdalla Filho. -- 2003.
xiv, 154 f. : il.; 30 cm

Dissertação (mestrado) -- Pontifícia Universidade Católica do Paraná, Curitiba, 2003
Inclui bibliografia

1. Geração numérica de malhas (Análise numérica). 2. Método dos elementos finitos. I. Abdalla Filho, João Elias. 11. Pontifícia Universidade Católica do Paraná. Programa de Pós-Graduação em Engenharia Mecânica. III. Título.

CDD 21. ed. - 515.353
515.62

TERMO DE APROVAÇÃO

MARLON DE OLIVEIRA VAZ

"Geração de Malhas de Elementos Triangulares em Domínios Planos usando o Método do Avanço da Fronteira"

Dissertação aprovada como requisito parcial para obtenção do grau de Mestre no Curso de Mestrado em Engenharia Mecânica, Programa de Pós-Graduação em Engenharia Mecânica, do Centro de Ciências Exatas e de Tecnologia da Pontifícia Universidade Católica do Paraná, ela seguinte Banca examinadora:

Presidente: **Prof. Dr. João Elias Abdalla Filho*(Orientador)***
Curso de Engenharia Mecânica (PUCPR)

Prof. Dr. Eduardo Alberto Fancello
Departamento de Engenharia Mecânica (UFSC)

Prof. Dr. Sérgio Scheer
Programa de Pós-Graduação em Métodos Numéricos em Engenharia - PPGMNE (UFPR)

Prof. Dr. Renato Barbieri
Curso de Engenharia Mecânica (PUCPR)

Curitiba, 20 de novembro de 2003.

À minha amada e inesquecível esposa Danny,
pelo seu amor e apoio incondicional.
E ao meu amado filho Arthur por
ajudar a enxergar melhor o mundo e as pessoas.

AGRADECIMENTOS

Ao Prof. João Elias Abadalla Filho meu orientador, por acreditar no meu trabalho e no meu potencial, e durante todo este período nunca deixou de mostrar a direção necessária para completar esta tarefa. Mesmo nos momentos difíceis, sempre apoiou as minhas decisões.

A Jane secretária do mestrado, que sempre esta disposta a resolver os nossos problemas e nos auxiliar.

Aos professores Roberto Dalledone, Nilson Barbieri, Renato Barbieri e Luiz Mauro, por aceitar e respeitar um aluno no mestrado sem as características de um engenheiro.

Aos novos amigos que fiz no mestrado e que da mesma forma me aceitaram e me ajudaram de durante o curso.

Aos meus pais José e Iraci, bem como minha irmã Carla e seu marido Marcos por estarem me apoiando desde o começo.

Obrigado a todos, sem vocês não seria possível chegar até aqui, e muito menos produzir este material.

Resumo

Este trabalho tem como objetivo a construção de uma ferramenta para geração de malhas de elementos finitos em domínios poligonais planos utilizando elementos triangulares. O método adotado para a geração de malhas é o Método do Avanço da Fronteira por ser mais simples de implementar e por garantir a construção de malhas de boa qualidade. A escolha deste método é feita após um estudo compreensivo sobre vários métodos para geração de malhas.

O gerador de malha desenvolvido gera elementos triangulares a partir da fronteira do domínio. Essa geração considera os ângulos entre os segmentos utilizados para definir um triângulo, segmentos estes que são definidos pelos nós do contorno. Desta forma, definem-se ângulos que possibilitam a criação de um, dois ou três elementos triangulares no domínio. Após a inclusão de todos os elementos, utiliza-se um algoritmo para fechar a fronteira aberta, criando-se, assim, uma nova fronteira. Este procedimento é repetido até que, próximo do centro do domínio, não há como adicionar mais elementos, deixando-se uma área vazia. Então, completa-se a geração da malha através de um procedimento de costura.

A malha gerada pode ser uniforme ou gradual. Na malha uniforme, os elementos têm as mesmas dimensões, enquanto que na malha gradual há variação de dimensões a cada novo nó inserido. Para melhorar a configuração final da malha gerada, utiliza-se o procedimento de suavização de Laplace.

Palavras-chave: Geração de malha, Avanço da Fronteira, Malha Gradual, Malha Uniforme, Método dos Elementos Finitos.

Abstract

The objective of this work is to build a computational tool for finite element mesh generation in polygonal domains employing triangular elements. The method adopted for mesh generation is the Advancing Front Method for being simpler to implement and for guaranteeing good quality meshes. The choice of the method was made after a comprehensive study of various mesh generation methods.

The mesh generator developed generates triangular elements at the domain´s front. This generation takes into account the angles between the straight line segments used to define a triangle which are defined by the nodes inserted along the front. Therefore, angles are created that allow for the generation of one, or two, or three triangles inside the domain. After the generation of all elements, an algorithm is used to close the open front, thus generating a new front. This procedure is repeated until an open area is left at the center region of the domain because it is no longer possible to create triangles to fill it. To end the mesh generation, a sewing procedure is then started to close the open area.

The mesh may be uniform or gradual. In the uniform mesh, the elements are of the same size while their sizes vary with the insertion of new nodes in the gradual mesh. Finally, to improve the mesh configuration, a Laplacian smoothing procedure is employed.

Keywords: Mesh generation, Advancing front, Gradual mesh, Uniform mesh, Finite Element Method.

Sumário

Lista de Figuras

Lista de Símbolos

α - Alfa
Ω - Omega
º - Graus
% - Percentual

Lista de Tabelas

Capítulo 1

1 Introdução

1.1 Considerações Iniciais

Problemas da elasticidade envolvem a determinação de tensões e deformações em sólidos. Esses problemas são descritos através de sistemas de equações diferenciais submetidos a certas condições de contorno, ou, alternativamente, através de equações integrais ou funcionais submetidos às mesmas condições de contorno. Um funcional é uma expressão integral que contém implicitamente as equações diferenciais que descrevem o problema. Equações diferenciais declaram o problema na sua *forma forte*, enquanto funcionais declaram o problema na sua *forma fraca*. A forma forte impõe condições que devem ser satisfeitas em cada ponto material, enquanto a forma fraca impõe condições que devem ser satisfeitas somente de modo aproximado.

Problemas simples quando formulados na forma forte podem ser resolvidos analiticamente com relativa facilidade. Por outro lado, soluções analíticas de problemas complexos são de difícil obtenção. Portanto, esses problemas devem ser formulados na forma fraca e resolvidos de maneira aproximada. Entre os métodos de solução, citam-se os analíticos como o método dos resíduos ponderados e os de perturbação, assim como os métodos numérico-computacionais, dentre os quais o método dos elementos finitos (MEF) é o mais difundido.

O método dos elementos finitos descreve o problema através da subdivisão de seu domínio em regiões de geometria simples com simples descrição matemática. Essas regiões são denominadas elementos e possuem nós em seus vértices e, algumas vezes, em pontos intermediários de seus lados. A representação

do domínio através de um conjunto de elementos é denominada malha. Matematicamente, a malha é representada por um sistema de equações algébricas. A aproximação do problema é aprimorada na medida em que a malha é refinada, o quê acarreta no aumento do seu número de elementos e de nós.

A representação do domínio através de uma malha e os seus subseqüentes refinos são largamente facilitados se um gerador de malhas estiver disponível. Entre as várias técnicas de geração de malhas existentes, cita-se o Método do Avanço da Fronteira, os métodos de triangulação de Delaunay e de Voronoï, e os métodos Quadtree, Octree, Circle Packing, Bubble Meshing e Sphere Packing. Cada um destes métodos possui características próprias, adequando-se melhor a certos tipos de problemas. Estes métodos serão descritos neste trabalho.

Finalmente, a solução de problemas via o MEF também é bastante facilitada por técnicas da computação gráfica, que permitem a inserção e visualização de dados do problema de forma gráfica, assim como a geração e o refino de malhas. Através da visualização, o analista pode inferir se o modelo é adequado para a solução do problema em questão e também pode detectar erros na entrada de dados. A visualização da malha gerada também permite ao analista definir se o grau e distribuição do refino permitem uma representação adequada do fenômeno físico.

1.2 Objetivos

O objetivo principal desta dissertação é desenvolver um gerador de malhas de elementos finitos triangulares para domínios poligonais planos. Secundariamente, objetiva-se aprofundar conhecimentos sobre geração de malhas aplicada ao método dos elementos finitos e iniciar a construção de uma ferramenta doméstica que possibilite a implementação dos recursos desejados, servindo de subsídio para

pesquisas em que se emprega o método dos elementos finitos. Esta dissertação marca o início de uma linha de pesquisa em geração de malhas deste Programa de Pós-Graduação em Engenharia Mecânica (PPGEM).

1.3 Motivação

Este trabalho foi motivado pela necessidade de se desenvolver um software de pesquisa doméstico para geração de malhas no intuito de se poder implementar recursos na medida dos avanços do conhecimento na área. Motiva também este trabalho o interesse na pesquisa em análise de erros nas soluções do método dos elementos finitos, que deve utilizar como ferramenta auxiliar um gerador de malhas o mais robusto possível. Finalmente, o conteúdo deste trabalho contempla a formação do autor em Ciência da Computação, e seu desejo em iniciar o desenvolvimento de um software que possa tornar-se comercial.

1.4 Limitações do Trabalho Desenvolvido

Este trabalho limita-se a desenvolver um gerador de malhas para domínios planos empregando o método do avanço da fronteira com elementos triangulares. Quatro domínios simples são empregados no desenvolvimento e teste do programa. As rotinas criadas visaram gerar malhas regulares e graduais sobre esses quatro domínios, de modo que o programa não suporta capacidades mais sofisticadas como gerar malhas em domínios com orifícios, reentrâncias e arestas curvas, por exemplo.

1.5 Etapas da Dissertação

Este trabalho é dividido em 6 capítulos, onde o capítulo 1 apresenta uma introdução, define os objetivos do trabalho e suas motivações.

O capítulo 2 faz a revisão bibliográfica, ou seja, descreve os artigos, teses, e livros utilizados para formar a base teórica necessária à construção dos algoritmos descritos. Além disso, o capítulo apresenta os tipos de algoritmos de geração de malhas existentes na literatura e suas características para a geração de uma malha.

O capítulo 3 apresenta o método do Avanço da Fronteira com a utilização de uma técnica para gerar elementos graduais em relação aos vértices principais.

O capítulo 4 descreve a técnica de suavização Laplaciana, que é utilizada para melhorar a configuração final de uma malha.

O capítulo 5 apresenta os resultados obtidos com o gerador de malha desenvolvido.

O capítulo 6 apresenta as conclusões, bem como alguns possíveis próximos passos para melhorar o trabalho realizado nesta dissertação.

O Anexo A descreve as funções utilizadas no algoritmo de geração de malha triangular.

O Anexo B contém o código fonte das principais rotinas do gerador de malha proposto neste trabalho.

Capítulo 2

2 Geradores de Malhas

2.1 Introdução

Neste capítulo, apresenta-se a visão de alguns autores e suas aplicações para as soluções de problemas de geração de malhas utilizando-se o método dos elementos finitos. Procura-se mostrar a evolução do tópico ao longo do tempo, bem como, indicar as principais diretrizes para garantir uma malha de boa qualidade. Além disso, faz-se a descrição dos principais métodos para geração de malhas. Estes métodos e seus algoritmos são fundamentais para construção dos algoritmos apresentados no capítulo 3.

A geração de malha é conhecida como a fase de pré-processamento para o método dos elementos finitos (MEF). Nesta fase, é feito a discretização de um determinado domínio em subdomínios, possibilitando assim, solucionar os problemas da física-matemática através de métodos numérico-computacionais. Com isso, a geração de malha tornou-se um campo de pesquisa bastante ativo e em constante evolução. Isto pode ser visto pelo trabalho de (BERN & PLASMANN, 1999), onde é feito um apanhado geral sobre os tipos de malhas existentes, os tipos de elementos mais utilizados, bem como a sua relação com os métodos numérico-computacionais que provêem soluções aproximadas para fenômenos físicos. Já no livro de (GEORGE, 1991), este descreve de forma geral os tipos de malhas existentes para domínios em 2D e 3D, os quais, ainda são utilizados por pesquisadores e usuários do método dos elementos finitos (MEF).

2.2 Tipos de Malhas

Existem duas classes fundamentais de malhas, as estruturadas e as não estruturadas, conforme exposto por (GEORGE, 1991) e (LISEIKIN, 1999). Na malha estruturada todos os elementos adicionados têm o mesmo tamanho e os mesmos formatos geométricos, produzindo assim, uma boa "conectividade" entre os elementos. A "conectividade" é identificada pela união de dois elementos pelo mesmo nó, ou seja, é o elemento na qual cada vértice, exceto nas bordas da malha, tem uma vizinhança local isomórfica. As malhas não-estruturadas possuem elementos com tamanhos diferentes e estruturas diferentes e a sua conectividade variam de ponto a ponto, ocorrendo assim, formas e tamanhos diferentes de elementos dentro da malha. Ou seja, é aquela na qual os seus vértices podem apresentar vizinhos locais arbitrariamente variados. Em geral, as malhas não estruturadas correspondem a triangulações com vizinhança local não isomórfica.

Conforme definido por (Reis, Magalhães & Ruthner, 2003), as malhas estruturadas geralmente oferecem uma maior simplicidade e uma maior facilidade de acesso aos dados em relação às malhas não estruturadas. No entanto, sua grande desvantagem está falta de flexibilidade em se ajustar a domínios de formas complexas. Já as malhas não estruturadas ajustam-se bem a estes domínios.

2.3 Propriedades

Para a construção de uma malha, as propriedades de natureza geométrica e física devem ser levadas em consideração. Neste caso, conforme explicitado por (GEORGE, 1991), para as propriedades geométricas, o que deve ser levado em consideração é a transição entre os elementos. Na transição, a diferença de tamanho entre elementos adjacentes não pode ser grande, ou seja, a variação de

tamanho deve ser gradual. Ao término da inserção de todos os elementos, é necessário calcular o gradiente de cada elemento e nas regiões onde ocorrem gradientes elevados, a quantidade de elementos deve ser maior, ou seja, elementos menores, justamente para garantir resultados mais precisos. Portanto, deve haver uma redução gradual no tamanho dos elementos de uma malha no sentido de uma região de pequenos gradientes para uma região de grandes gradientes. Estas disposições são válidas para quaisquer geometrias de elementos. No caso particular de malhas de elementos triangulares, deve-se evitar a existência de ângulos obtusos.

Para as propriedades de natureza física, os elementos inseridos devem representar as condições físicas necessárias em relação à característica física do domínio, como no caso de domínios pequenos, ou domínios com propriedades isotrópicas/anisotrópicas, ou domínios com formas específicas. Onde, em cada elemento deve estar caracterizada a propriedade física em relação ao problema representado.

2.4 Tipos de Métodos Empregados

Para que seja possível produzir uma malha, são definidos dois tipos de algoritmos de varredura ou de construção, o Bottom-UP e o Top-Down, que retratam, respectivamente, a análise do domínio a partir das partes até o todo, e do todo até as suas partes (GEORGE, 1991). Ou seja, o domínio pode ser dividido em vários blocos e a união destes blocos retrata o domínio (Bottom-UP), ou o inverso, como é o caso do Top-Down. Com a identificação da forma de obtenção de uma malha, pode-se então, reconhecer como as malhas se comportam em cada método empregado, isto, de maneira geral.

2.4.1 Quadtree

O *quadtree* é utilizado para identificar e mapear um domínio bidimensional através de quadriláteros. O método empregado por (BERN & PLASMANN, 1999) para discretização de um domínio qualquer utilizando o quadtree, parte da inserção de um quadrilátero que retenha em seu interior todo o domínio em questão. A partir de então, é feita a divisão deste quadrilátero em outros quatro, onde estes quatro quadrados são congruentes. A partir de então, é feita a divisão de cada um dos quadrados de forma recursiva, conforme explicitado acima, até que todo o domínio possa ser representado por pequenos quadriláteros. A partir deste ponto, os quadriláteros são então transformados em elementos triangulares, gerando assim, uma malha triangular não-estruturada (ver Figura 1).

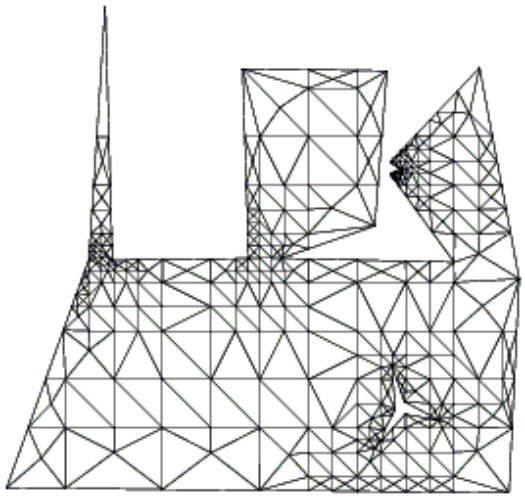

Figura 1 - Quadtree - (BERN & PLASMANN, 1999).

Além de descrever o método *quadtree*, (BERN & PLASMANN, 1999) apresentada uma visão geral sobre geração de malhas, os tipos de malhas (estruturadas, não-estruturadas e híbridas), bem como os métodos numéricos utilizados nas soluções dos fenômenos físicos (método das diferenças finitas,

método dos elementos finitos ou método dos volumes finitos). Descreve ainda, métodos de solução dos sistemas de equações (métodos de fatoração direta, métodos interativos, métodos multigrid e de decomposição do domínio) e a escolha da geometria do elemento depende do tipo de problema a ser modelado. Cita-se, como exemplo: problemas de fluxo de fluido, onde os elementos quadriláteros são mais indicados que os triangulares.

Uma outra forma de ver o *quadtree* é analisando o método exposto por (GEORGE, 1991), onde este método inicia pela marcação de pontos sobre o contorno do domínio. Em seguida é feita à subdivisão do domínio em quadrados congruentes conforme exposto anteriormente no texto acima. A diferença deste método esta na definição do local a ser melhorado, ou seja, a divisão de um quadrado em outros quatro vai depender da existência de mais de um nó no quadrado definido sobre contorno do domínio. Esta divisão recursiva persistirá até que todos os quadrados tenham somente um nó interno, e que este nó, seja do contorno do domínio. A Figura 2 mostra o método empregado por (GEORGE, 1991), onde inicialmente é desenhado um quadrado que contém o domínio. A partir de então, o quadrado é dividido em outros quatro, dividindo o domínio em quatro regiões. Se em uma região foi identificado mais de um nó, esta é subdividida novamente, criando assim, quatro novas sub-regiões e este processo é seguido até que existe somente um nó por região.

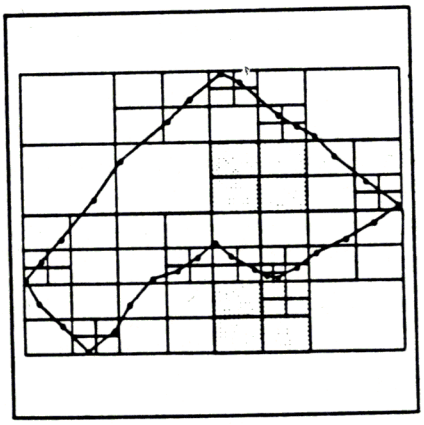

Figura 2 - Quadrtee – (GEORGE, 1991)

2.4.2 Octree

O *octree* é uma técnica de geração de malhas tridimensionais semelhante ao *quadtree*, sendo que sua representação é feita utilizando elementos hexaédricos. No final da divisão do domínio, é possível então, inserir elementos tetraédricos, os quais são utilizados para gerar uma malha tetraédrica não-estruturada, conforme descrito por (BERN & PLASMANN, 1999) e também por (GEORGE, 1991). O *quadtree* pode também ser utilizado como malha de fundo, esta malha ser utilizada como parâmetro para a geração de elementos tetraédricos, em conjunto com outras técnicas de geração de malha, conforme (CAVALCANTE NETO at. al., 2000) e (MARTHA at. al., 2001).

Semelhante ao método utilizado por (GEORGE, 1991), o método de geração de malhas de (CAVALCANTE NETO at. al., 2000) utiliza o método *octree*, porém, para esta solução, optou-se por representar primeiramente um domínio em 2D como uma malha de fundo (ver Figura 3) e posteriormente ligar os planos produzindo assim uma malha em 3D.

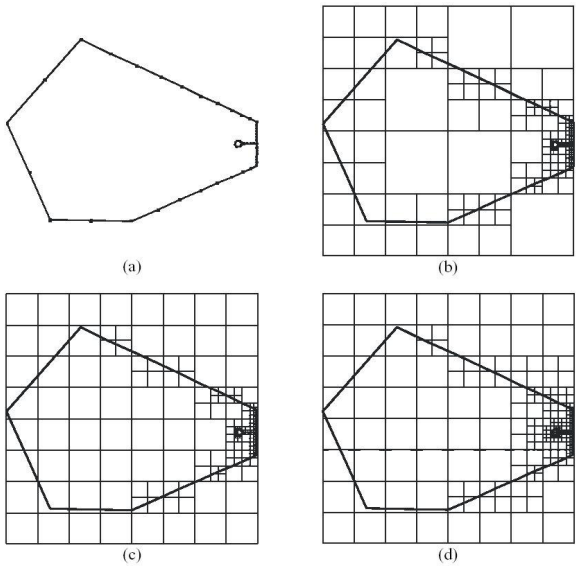

Figura 3 - Quadtree - (CAVALCANTE NETO at. al., 2000)

Neste caso, (CAVALCANTE NETO at. al., 2000) descrevem uma forma para a geração de uma malha tetraédrica, onde são utilizados dois métodos, o avanço da fronteira e o *octree*. Para que a malha possa ser gerada, é necessário ter um domínio com todos os nós sobre o contorno original definidos. A partir deste domínio, o método do *octree* é então iniciado, e gera-se primeiramente uma malha utilizando-se o método *quadtree,* a qual serve como orientação local (malha de fundo) para a definição do tamanho dos elementos tetraédricos a serem gerados pelo método do avanço da fronteira. Para completar a malha desejada é necessário passar por algumas fases, essas fases são: para o *octree*, o início baseia-se na malha de contorno, tendo um refino para forçar o tamanho máximo de célula e um refino para garantir a mínima disparidade de tamanho entre as células adjacentes. Para o avanço da fronteira, tem-se a geração de elementos baseada em geometria, a geração de elementos baseada em topologia e a geração de elementos baseada em procedimento de “volta-passo” com remoção de elemento. Após gerar a malha tetraédrica, então se inicia um algoritmo para melhorar da qualidade da malha. Este

algoritmo faz a suavização Laplaciana com testes de validação e avaliação de qualidade dos elementos inseridos, a qual serve para melhorar a qualidade de cada elemento localmente, podendo-se encontrar outros elementos tetraédricos que possibilitem uma qualidade maior. Além disso, também ativa o procedimento "volta-passo" para remoção de um elemento local, caso seja necessário.

No artigo apresentado por (MARTHA at. al., 2001), o qual é uma continuação do trabalho apresentado por (CAVALCANTE NETO at. al., 2000), além de ser um programa para análise pelo método dos elementos finitos, ele descreve o mesmo algoritmo para geração da malha utilizando o método do avanço da fronteira e o método *octree*, o método de suavização e incluindo uma forma de garantir uma malha tetraédrica de boa qualidade usando uma técnica de retorno da malha (*backtracking*).

O algoritmo segue os seguintes passos para construir a malha de elementos finitos: Primeiro, a geração de uma malha *quadtree* de fundo, que será utilizada para garantir a criação dos elementos tetraédricos (*octree*). Segundo, a geração de uma malha utilizando o método do avanço da fronteira, para garantir a melhor qualidade dos elementos na sua primeira inserção. Com base nisto, é utilizado um vetor que irá armazenar os nós ativos e um outro vetor para armazenar os nós inativos. O programa termina quando não existirem nós inativos para fechar a fronteira. Na terceira e última etapa, são utilizados algoritmos de suavização para melhorar a qualidade dos elementos tetraédricos na malha. O método *backtracking* também é empregado nesta etapa para avaliar a qualidade de elementos e excluir aqueles considerados ruins. O elemento ruim é caracterizado por seu formato, ou seja, se a forma do elemento não for próxima de um triângulo eqüilátero, e os elementos de boa qualidade substituem os elementos excluídos (ver Figura 4).

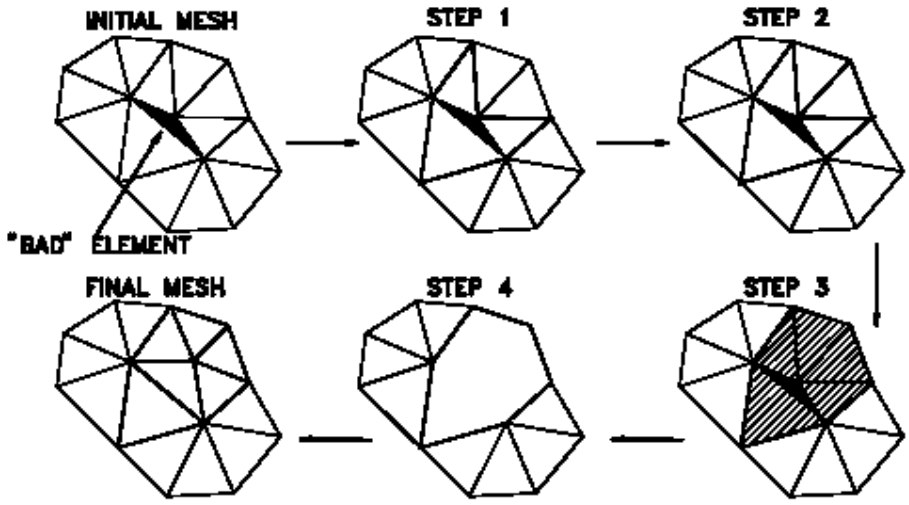

Figura 4 - Melhoria da Malha - (MARTHA at. al., 2001).

2.4.3 Circle Packing

O método do *Circle Packing* está sendo largamente difundido como uma solução para gerar uma malha inicial de boa qualidade, como pode ser visto em (LI at. al., 2000), (BERN & EPPSTEIN, 2000) e (BERN at.al.,1995). O método inicia-se pela criação de círculos dentro do domínio geométrico definido. Depois de gerados todos os círculos dentro do domínio, geram-se os elementos triangulares ou quadriláteros sobre os mesmos, criando-se assim uma malha. Para domínios irregulares, círculos de tamanhos diferentes serão gerados, resultando em uma malha inicial com possível variação gradual no tamanho dos elementos. Assim, em teoria, o circle packing é capaz de gerar uma malha quase-ótima em uma primeira aproximação. Um dos métodos auxiliares que possibilitam uma malha de boa qualidade é o GVD (*Generalized Voronï Diagram*), o qual, auxilia na identificação de possíveis novos nós. O GVD é um método para encontrar o ponto central de um vazio no domínio, e esse método leva em consideração todos os eixos do domínio em contato com este vazio identificado, e partindo de todos estes vértices, o GVD cria uma linha média entre os vértices. A intersecção dessas linhas possibilita a

criação de novos pontos no “vazio”, possibilitando assim, gerar novos nós dentro dos vazios da malha como mostra a Figura 7. O vazio é caracterizado por uma região que não foi preenchida com elementos circulares.

O método do *Circle Packing* proposto por (BERN at. al, 1995) foi criado para produzir uma malha em um domínio poligonal plano. Esta técnica consiste de inserir elementos circulares a partir de um determinado vértice. Para a inserção deste circulo, é necessário saber qual é o tipo de vértice (côncavo ou convexo), no caso de vértice convexo, é inserido um único círculo como mostra a Figura 5. Este deve ser tangente aos eixos do vértice em questão.

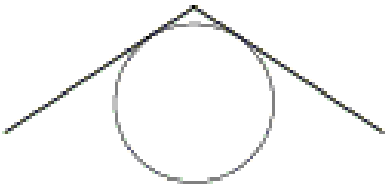

Figura 5 - Círculo em vértice convexo - (BERN at. al, 1995)

O raio do círculo irá depender da existência ou não de outro círculo adjacente, e é verificado se este círculo corta outros eixos do domínio dado. Caso nenhum dos dois fatores anteriores ocorra, então é possível adicionar um círculo. Agora, quando for identificado um vértice convexo, deverão ser adicionados dois círculos. Estes círculos devem possuir o mesmo raio, além de se interceptarem, neste caso, é necessário que os eixos do vértice côncavo tangenciem os círculos, com é visto na Figura 6.

Figura 6 - Círculos em vértice côncavo - (BERN at. al, 1995).

Após a inserção do círculo no primeiro vértice, a inserção continua no sentido horário para os demais vértices do domínio. Ao concluir a inserção dos círculos, sobram vazios que deverão ser preenchidos utilizando-se o método GVD, o que pode ser visto na Figura 7.

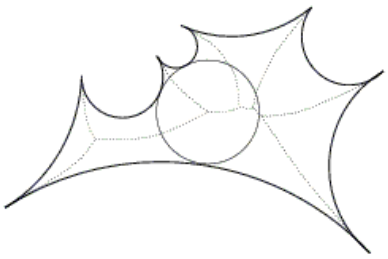

Figura 7 - Inclusão de círculo através do GVD - (BERN at. al, 1995).

Após a definição dos eixos com o método GVD, é possível adicionar novos círculos a partir dos pontos de intersecção destes, completando-se assim, a malha com elementos circulares. Como o domínio foi definido por círculos, é necessário gerar elementos triangulares ou quadriláteros. Para gerar estes elementos, deve-se unir todos os pontos de intersecção com os centros dos círculos, bem como, ligar os vértices ao centro do círculo, conforme demonstrado na Figura 8.

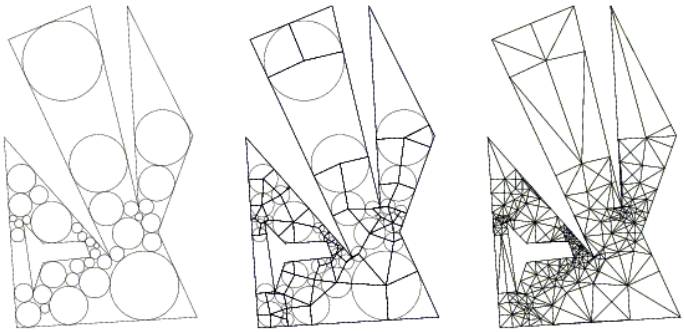

Figura 8 - Geração de Elementos Triangulares (BERN at. al, 1995).

Acima foi visto a geração de malha utilizando o Circle Packing com elementos triangulares. Seguindo os mesmos conceitos definidos por (BERN at. al, 1995), (BERN & EPPSTEIN, 2000) buscaram produzir uma malha de quadriláteros com o método do *Circle Packing*. A grande diferença neste método em relação ao aplicado anteriormente, é que neste, os círculos ultrapassam as fronteiras do domínio desejado, conforme a Figura 9.

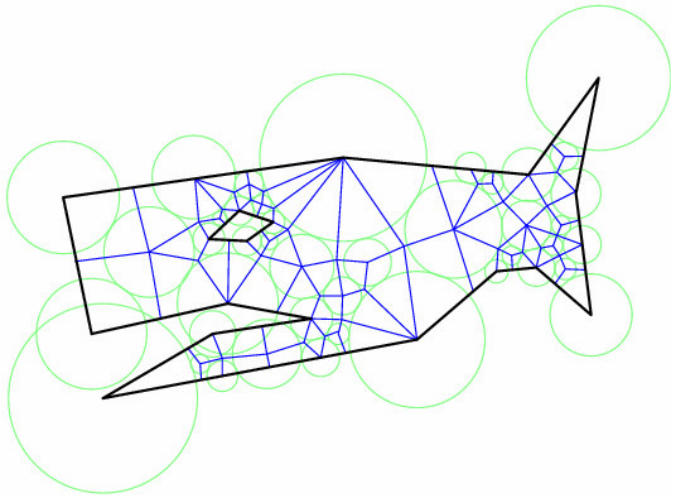

Figura 9 - Circle Packing com quadriláteros - (BERN & EPPSTEIN, 2000).

Por tratar-se de uma técnica geométrica, o circle packing preenche o domínio com círculos possibilitando a criação de aberturas (gaps), as quais são preenchidas com a inserção dos círculos. Para isto, é necessário que a abertura contenha três ou quatro círculos tangentes. Este algoritmo é utilizado no preenchimento de uma

região poligonal com um número qualquer de vértices, tendo como resultado uma triangulação com O(n) novos pontos, no qual nenhum triângulo tem ângulos obtusos, onde n é a quantidade de pontos por elementos. Então, a malha é construída unindo-se os centros dos círculos (novos vértices), os pontos de tangência (entre os círculos e círculos e linhas retas) e as aberturas. Em conjunto com circle packing, são apresentados três outros métodos que para criação de malhas de quadriláteros. O método do diagrama Voronoï geodésico, utilizado para encontrar os pontos médios dentro do domínio, e que gera um quadrilátero entre os centros dos círculos e o centro das aberturas encontradas. O método de triangulação de Voronoï também pode ser utilizado para gerar uma malha de quadriláteros. Este método constrói malhas com quadriláteros a partir de dois ângulos retos, os quais são formados a partir do centro do círculo. Outro método é o método *kite*, onde são gerados quadriláteros convexos com um eixo simétrico ao longo da diagonal. Isto possibilita fazer conexões em vértices côncavos, convexos, círculos tangentes sobre um contorno, círculos tangentes a outros círculos, de forma a garantir a criação de um quadrilátero (ver Figura 9).

2.4.4 Bubble Meshing ou Sphere Packing

O método *Bubble Meshing* utiliza-se de bolhas ou esferas para preencher um domínio e a diferença em relação ao circle packing está no modo de inserção das bolhas, pois neste caso se usa um método de controle populacional. O controle populacional é utilizado para verificar a possibilidade de inserir mais círculos dentro do domínio. Além do controle populacional, é necessário definir se uma determinada bolha pode ser inserida, utilizando-se para isto uma técnica denominada atração e repulsão. Ou seja, utilizando-se da formulação aplicada em vibrações, pode-se chegar em um estado apropriado para cada bolha ou esfera. Este estado refere-se

ao tamanho da bolha utilizada, bem como o tamanho e a posição das bolhas adjacentes. Esta relação entre atração e repulsão pode ser vista na Figura 10, onde o efeito mola é dado pelo centro dos círculos. Esse efeito mola representa a alocação do círculo no domínio. O mesmo efeito é propagado para toda a malha, fazendo assim, os ajustes necessários ou identificando a necessidade de retirada desse elemento circular ou de outro.

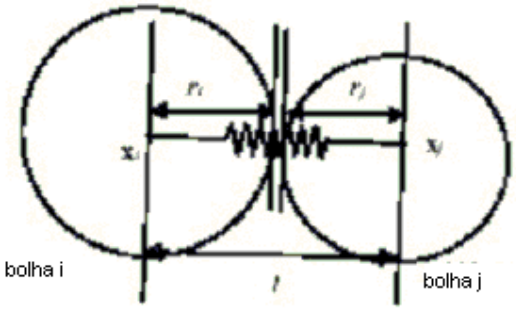

Figura 10 - Atração e Repulsão de Bolhas - (YAMAKAWA & SHIMADA, 2001).

Os resultados obtidos por (FIGUEIREDO & DE CARVALHO, 1996) demonstram a eficácia do método *bubble meshing* em domínios 3D, pois ele investiga o método *bubble meshing*, o qual é descrito como uma técnica para geração automática de malhas triangulares em domínios de geometria complexa. O algoritmo é muito lento em relação aos demais, por que o *bubble meshing* trabalha com a atração e repulsão das partículas com vários passos até atingir o equilíbrio. A atração e repulsão das partículas levam em consideração a utilização da equação de movimento conforme exposto por (YAMAKAWA & SHIMADA, 2001). O artigo propõe o desenvolvimento de um algoritmo que reduza o tempo de construção da malha, sendo, para isto, necessário acelerar a interação das partículas.

Já (YAMAKAWA & SHIMADA, 2001) descrevem um método para gerar malhas para domínios finos em 2D, por exemplo, em um cano de água. Para estes

tipos de domínio, métodos convencionais, como o uso dos geradores de malhas automáticos como os métodos quadriláteros e avanço da fronteira não produzem uma boa malha. Para este caso, o método *Quad-Layer* garante uma malha de boa qualidade a partir da definição de um "eixo médio" em relação ao domínio (ver Figura 11). A divisão deste eixo médio em relação ao domínio possibilita criar elementos quadrilaterais. Para isto, são inseridas esferas no domínio em questão, onde o centro da esfera é eixo médio. Assim o eixo médio gerado a partir das bolhas garante a inserção de um elemento quadrático. O *Quad-Layer* utiliza-se do método chamado *bubble packing*, o qual é um método computacional baseado no aspecto físico do problema.

O *bubble packing* é um algoritmo que utiliza células esféricas ou bolhas, e após a inserção de uma bolha sobre o domínio, é necessário encontrar sua estabilidade, que é determinada utilizando uma simulação dinâmica. Para isto, um esquema de integração numérica padrão é utilizado, como exemplo, pode-se citar o método de Euler ou método de Runge-Kutta de quarta ordem. As forças atuantes sobre as bolhas são aproximadas pelo modelo de massa-amortecimento. A rigidez k é a rigidez da mola, considerando-se que as molas não lineares entre bolhas adjacentes (ver Figura 10). Neste modelo, cada bolha tem sua posição e velocidade, bem como, os coeficientes de massa e amortecimento. Para saber se as bolhas estão se atraindo, ou se repelindo, ou se não interagem, ou se estão em estado estável, a razão entre o tamanho corrente da bolha e o tamanho da mola entre as bolhas é avaliada. Se a razão for igual a um, então as bolhas estão estáveis ou não interagem e se for menor que um então elas se repelem, agora se estiverem entre 1 e 1,5 elas se atraem.

Para controlar a quantidade de bolhas utilizadas, é necessário fazer um controle populacional. Este controle é dado pela razão entre o tamanho do domínio e o diâmetro da bolha. E para garantir a qualidade dos elementos é utilizada a suavização Laplaciana.

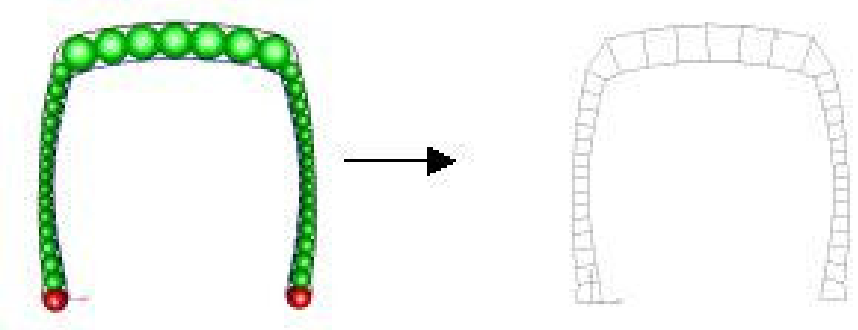

Figura 11 - *Bubble meshing* em domínios estreitos - (YAMAKAWA & SHIMADA, 2001).

2.4.5 Bitting

Variações do *bubble meshing* ou *sphere packing* podem ser utilizados para garantir uma boa malha final. O método *Bitting* consiste em inserir bolhas, levando-se em consideração o ângulo formado pela inserção da bolha e também pelo tamanho da bolha. Em conjunto, utiliza-se o método do avanço da fronteira, o qual garante uma ordem de inserção das bolhas. Os passos utilizados pelo *bitting* são representados na Figura 12. A parte (*a)* mostra o domínio original, a partir desse são gerados círculos sobre os vértices do domínio como visto na parte (*b)*. Da parte *(c)* até a parte *(e)*, pode ser visto o preenchimento com círculos sobre o contorno do domínio. Na parte *(e)* é definido o novo contorno do domínio e esse novo contorno leva em consideração as intersecções geradas pelos círculos. Na parte *(f)* uma nova seqüência de círculos é inscrita a partir da nova fronteira definida.

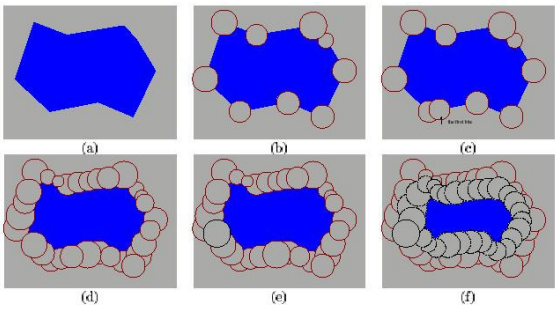

Figura 12 – *Bitting* – (LI at. al., 2000).

O método *biting* (LI at. al., 2000), utiliza-se do método do avanço da fronteira e do método *circle packing* para criar uma primeira fronteira utilizando-se de círculos, sobrepostos ou não (ver Figura 12). Partindo do contorno do domínio, o qual é a primeira fronteira, são adicionados círculos. Esse círculo tem o contorno do domínio como seu centro. Para inserir um novo círculo, deve-se levar em consideração o restante do segmento de reta fornecido, verificando assim, a necessidade de mais de um círculo sobre o contorno em relação às regras acima definidas. Ou seja, controlar o espaço para inserção de um novo círculo e também a qualidade do mesmo em relação ao domínio. Os pontos de intersecção da fronteira com o círculo definiram a nova fronteira, e a cada novo avanço da fronteira, a fronteira anterior é retirada do domínio até a não existir mais fronteira a ser criada.

2.4.6 Delaunay

O método de Delaunay conforme (GEORGE, 1991) consiste em criar elementos triangulares a partir de um conjunto de pontos no espaço, e então é gerado um círculo através dos vértices do triângulo. Sendo que a triangulação somente será valida se não for encontrado vértice de outro triângulo dentro do círculo inscrito.

O caso apresentado pela Figura 13 é uma variação do método de Delaunay, onde o círculo irá fornecer um novo ponto, o qual servirá de base para a criação de

novos elementos triangulares, e pode ser visto na Figura 13 (b). Porém neste exemplo, os nós ABC produzem um triângulo que não é o desejado, neste caso, o desejado é um triângulo eqüilátero ou próximo de ser um triângulo eqüilátero. Sendo assim, um círculo é traçado a partir dos nós A, B e C. Com o círculo desenhado é possível identificar o seu centro, ou seja, o nó D, a partir de então, as linhas que unem os nós AB e AC são removidas e o nó D passa a ser mais um nó do domínio, possibilitando criar novos elementos triangulares. A partir do nó central são traçadas novas linhas representadas pela Figura 13 (b), e o centro do círculo é agora conhecido como nó F. Com o nó F são traçadas as linhas FA, FB, FC, FD e FE, criado novos elementos triangulares melhores que o inscrito anteriormente.

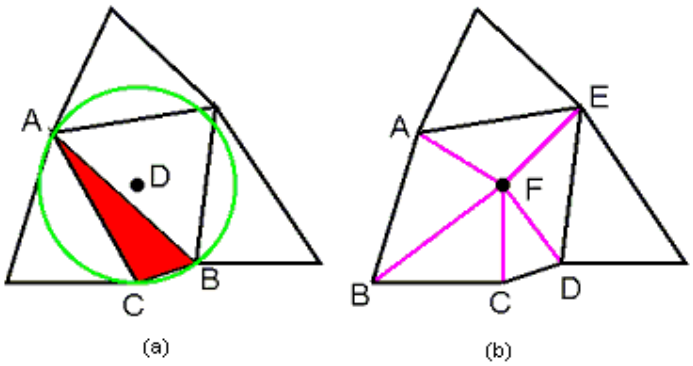

Figura 13 - Método de Delaunay - (SHEWCHUK, 1996).

Em conjunto com o método de Delaunay outros métodos podem ser utilizados, neste caso (SHEWCHUK, 1996) apresenta um programa de geração de malhas utilizando os métodos de triangulação de Delaunay, Constrained Delaunay e diagramas de Voronoï. Para a triangulação de Delaunay são apresentados e utilizados os algoritmos: “dividir e conquistar” (divide-and-conquer), e “*plane-sweep*”. Com base na construção destes algoritmos, é feita uma tabela que demonstra o tempo de geração da malha para uma determinada quantidade de elementos. O algoritmo “dividir e conquistar” define elementos fantasmas para gerar um elemento

triangular válido. Estes elementos fantasmas podem ser um segmento, um triângulo, ou dois eixos colineares. Se o elemento triangular fantasma gerado for válido, então ele passa a ser um triângulo válido. Contudo em alguns casos, pode ocorrer a remoção de um elemento triangular válido para garantir a qualidade da malha gerada. No caso de eixos confinados (*constrained*), estes eixos não podem ser removidos durante o processo de melhoria da qualidade malha, e também não podem ser alterados durante a inserção de um novo vértice.

O algoritmo de refinamento de Ruppert é utilizado na implementação. Este algoritmo é dividido em quatro passos. No primeiro passo, encontram-se todos os vértices no domínio para que se possa fazer a triangulação. No segundo passo, utiliza-se o método de confinamento triangular de Delaunay, que possibilita criar elementos triangulares sem ultrapassar o domínio. No terceiro passo, são retirados os triângulos das concavidades e furos. No quarto passo, são melhorados os elementos triangulares conforme o tamanho do ângulo interno. Para isto, nesse último passo, são utilizados dois recursos para melhorar a malha. Esta fase é a parte principal do algoritmo, o qual é governado por duas regras. A primeira divide um determinado segmento recursivamente até que os triângulos gerados em conjunto com os pontos médios possam garantir elementos triangulares adequados. O segundo, ao encontrar um triângulo ruim, adiciona uma circunferência a partir de vértices do triângulo ruim. Neste caso, a circunferência deve passar pelos vértices desse triângulo, deslocando o centro da circunferência para outra posição que pode estar sobre triângulos adjacentes. O centro da circunferência passa então a ser o ponto de conexão entre os vértices desses triângulos. O ângulo interno do triangulo não pode ser inferior a 20.7º. No caso de sua ocorrência, um triângulo ruim será criado e, conseqüentemente, será necessário utilizar um método de refinamento.

O método de Delaunay também é visto como uma forma de melhoria da qualidade do elemento. Esta qualidade é garantida pelo rearranjo do elemento, como pode ser visto em (GEORGE, 1991) e (BERN & PLASSMANN, 1999).

2.4.7 Voronoï

O método de Voronoï é o dual do método de Delaunay, possibilitando assim, diversas variações, uma delas tem como base à geração de elementos circulares nos vazios do domínio, possibilitando identificar nós que poderão fazer parte do domínio na geração de um elemento triangular ou quadrático.

No trabalho de (BERN at. al., 1995), descreve-se o método do GVD (*Generalized Vornoï Diagram*), que é utilizado para encontrar um novo nó. Este novo nó é gerado a partir da criação de bissetrizes de um domínio não completo, o que pode ser visto na Figura 6 e também na Figura 14. A Figura 14 mostra um vazio gerado pela inserção de elementos circulares. Neste vazio é necessário inserir novos elementos, e para isto são identificados os eixos médios do vazio. A intersecção entre esses eixos possibilita definir um possível nó a ser utilizado e quando isto ocorre, um elemento é inserido.

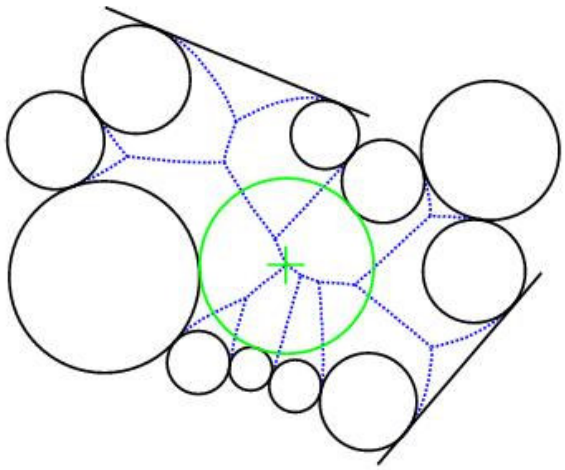

Figura 14 – GVD - (BERN at. al., 1995).

Focando ainda o trabalho de (BERN at. al., 1995) este tem como característica principal gerar triângulos não-obtusos em domínios poligonais, de modo a se ter uma malha adequada para solução de problemas via elementos finitos. A razão é que uma malha de elementos com ângulos não-obtusos garante convergência mais rápida. A expectativa apresentada é que o algoritmo não gere triângulos com ângulos maiores que $5\pi/6$ (150^o). Este algoritmo tem dois estágios, sendo que o primeiro é a inserção de círculos dentro de um domínio 2D. O segundo estágio é a triangulação a partir dos *Steiner points* (DREYER & OVERTON, 1998). Sendo que, neste caso, o *Steiner point* é a escolha de um determinado ponto de intersecção entre todos os pontos de intersecção definidos, onde este ponto escolhido possibilita a inserção de um novo elemento triangular. Após a inserção dos círculos dentro do domínio, alguns vazios serão identificados. Estes vazios seguem o método do GVD, conforme explicado anteriormente, para garantir que o círculo adicionado no vazio esteja no centro e/ou tangencie os círculos já adicionados (ver Figura 14). Sendo que no trabalho de (MCALLISTER, at. al, 1993) o GVD é descrito como sendo o diagrama de Voronoï.

Em relação aos *Steiner points*, (DREYER & OVERTON, 1998), descrevem uma forma de encontrá-lo, ou seja, encontrar um ponto possível de conexão entre um conjunto de pontos dados. Este trabalho produziu duas heurísticas que possibilitam encontrar possíveis pontos de conexão para quaisquer três ou mais pontos dados. Sendo assim, ocorre uma adição de um ponto auxiliar com o propósito de minimizar o tamanho total de pontos do conjunto. Neste caso todos os pontos encontrados têm um aspecto importante. O passo crucial nas duas heurísticas apresentadas é encontrar localmente as posições ótimas para os *Steiner points*, dado uma topologia fixa com seus eixos definidos. Na primeira heurística, o

Steiner point é encontrado em relação a outros três pontos, gerando uma árvore de pontos onde o ângulo formado entre eles deve ser menor que 120º (ver Figura 15). Na segunda heurística, são coletados três pontos e encontrado o melhor ponto entre eles, se for o melhor ponto, este é armazenado fazendo parte do conjunto inicial de pontos. Caso contrário, é feito outras tentativas até encontrar o melhor ponto.

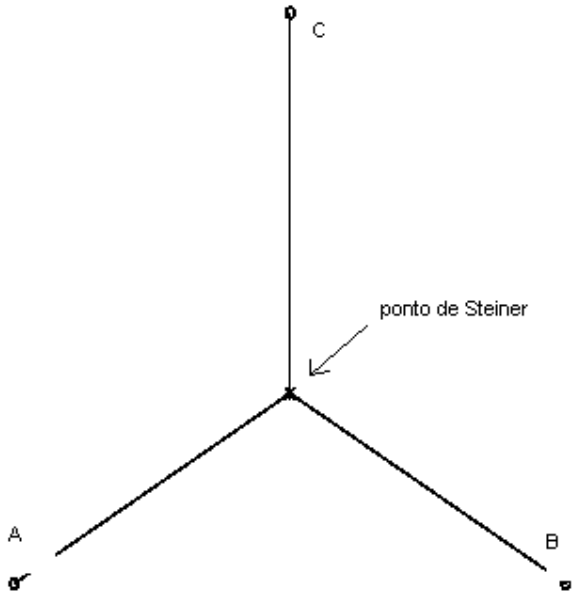

Figura 15 - Ponto de Steiner - (DREYER & OVERTON, 1998)

2.4.8 Quadrática

Para a geração de malhas com quadriláteros e para garantir uma boa malha (Zhu et. al., 1991) utilizou inicialmente triângulos para formá-la, pois, na união de dois triângulos tem-se um quadrilátero. A vantagem de ter uma malha triangular inicial, é que os triângulos ou elementos triangulares são mais fáceis para representar um domínio, pois podem ser colocados vários elementos com diferentes formas. Uma das formas de inserção de elementos é a definição do tamanho do elemento pelo usuário, o que pode ser inadequada para alguns tipos de

considerações físicas, principalmente para problemas da dinâmica de fluidos, eletrodinâmica, acústica, entre outros.

Para geração da malha quadrilátera, é utilizado o método do avanço da fronteira. Contudo, o primeiro passo é definir os pontos sobre o contorno do domínio, onde a partir destes pontos, serão construídos os elementos triangulares. No caso de existir domínio interno é criada uma linha de corte que junta o domínio externo com o domínio interno, gerando assim, um único domínio. Para a geração de uma malha quadrilátera é necessário inserir elementos triangulares, onde dois triângulos produzem um quadrilátero. Porém, é possível ocorrer elementos distorcidos na malha, uma vez que, na criação do elemento pode ocasionar a sua deformidade. Por isto, são utilizados métodos de suavização ou o método conhecido como suavização Laplaciana, o qual auxilia na melhoria da qualidade da malha final. A suavização consiste de algumas formas para melhorar uma malha. Entre elas, existem as modificações internas do domínio, modificações do contorno do domínio e avaliação da qualidade do elemento.

2.4.9 Outros Métodos Auxiliares

Outros métodos utilizados para auxiliar na geração de malhas são o *Chordal Axis Transformation* de (YAMAKAWA & SHIMADA, 2001), o *Straight Skeleton* de (FELKEL & OBDRZÁLEK, 1998) e o *MAT* (*Medial Axis Transform*) de (QUADROS, at.al., 2000) também conhecido como *Skeleton* ou Eixo Simétrico (*Symmetric axis*). Todos esses métodos têm em comum a definição de linhas médias no domínio dado, ou seja, a partir do domínio definido são traçadas bissetrizes em relação ao contorno do domínio. No caso do MAT são gerados círculos a partir dos pontos de intersecção entre os eixos encontrados, conforme mostrado na Figura 16.

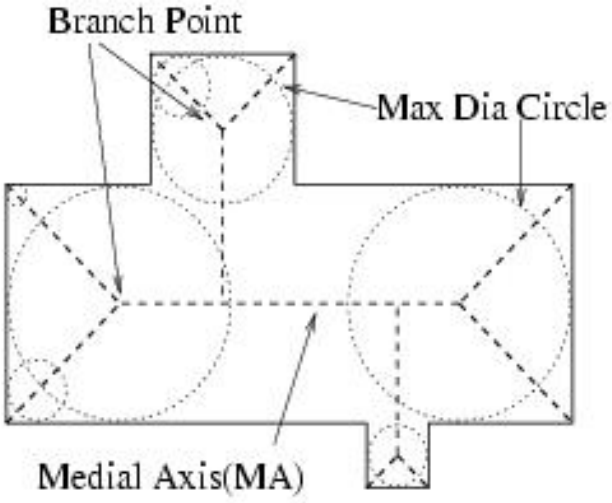

Figura 16 – MAT - (QUADROS, at.al., 2000).

O *MAT* foi descrito por (QUADROS at. al., 2000) e é utilizado para geração de malhas de quadriláteros. Para gerar a malha são utilizadas duas técnicas: a decomposição e o avanço da fronteira. A decomposição subdivide o domínio em formas mais simples e também utiliza modelos para gerar quadrilátero. O avanço da fronteira é utilizado para projetar os nós e gerar elementos triangulares, e com a junção de dois triângulos forma um quadrilátero. O *MAT* é também conhecido como *Skeleton* ou eixo simétrico, e trabalha para construir camadas de trilhas que possibilitem gerar um quadrilátero. Estas trilhas iniciam-se pela inserção de pontos sobre o domínio. Estes pontos partem dos vértices do domínio, e no caso de vértices convexos, este é subdividido em segmentos de mesmo tamanho. Ao traçar uma bissetriz em relação ao um determinado vértice, os pontos sobre o contorno deverão se conectar a esta bissetriz. Para isto, é traçada uma linha perpendicular ao contorno a partir do ponto até a bissetriz ou *MA* (*Medial Axis*), gerando assim, quadriláteros. No caso dos vértices convexos, são gerados dois triângulos e então convertidos para um quadrilátero. A melhoria da malha pode ser resolvida, utilizando-se de uma variação gradual entre dois pontos dados.

O *Straight Skeleton* é utilizado para gerar nós que possibilitem a criação de elementos triangulares, conforme mostrado na Figura 16. A definição dos pontos dentro do domínio inicia pela definição das bissetrizes dos ângulos formados entre os eixos do contorno. Com estes pontos, encontram-se as intersecções entre os eixos das bissetrizes definidas. O ponto gerado pela intersecção será um possível ponto para a criação de um elemento triangular, e os nós sobre o contorno são marcados como utilizados. Após encontrar os pontos de intersecção um novo ponto é calculado, ou seja, o ponto médio entre todos os pontos definidos. Este ponto médio possibilitara a criação de elementos triangulares entre o ponto médio e os nós do contorno.

O método *Skeleton* foi implementado por (FELKEL & OBDRZÁLEK, 1998), o qual é uma estrutura utilizada para descrever uma topologia básica característica de domínios poligonais convexos e côncavos em 2D. A bisecção (*Angular Bisector Network - ABN*) dos vértices é empregada para iniciar o mapeamento do domínio, seja ele côncavo ou convexo. Assim, após encontrar-se a bisecção de todos os vértices, são identificados os pontos de interseção entre as bissetrizes definidas. Com a união destes pontos, tem-se o domínio identificado e mapeado não gerando uma malha de elementos, e sim um domínio com seus pontos médios identificados. A única diferença entre os tipos de polígonos utilizados, é que no polígono côncavo, deve-se testar tanto o vértice que originou a bissetriz quanto o vértice oposto a esta bissetriz, com a finalidade de identificar o ponto médio entre esses pontos (ver Figura 17).

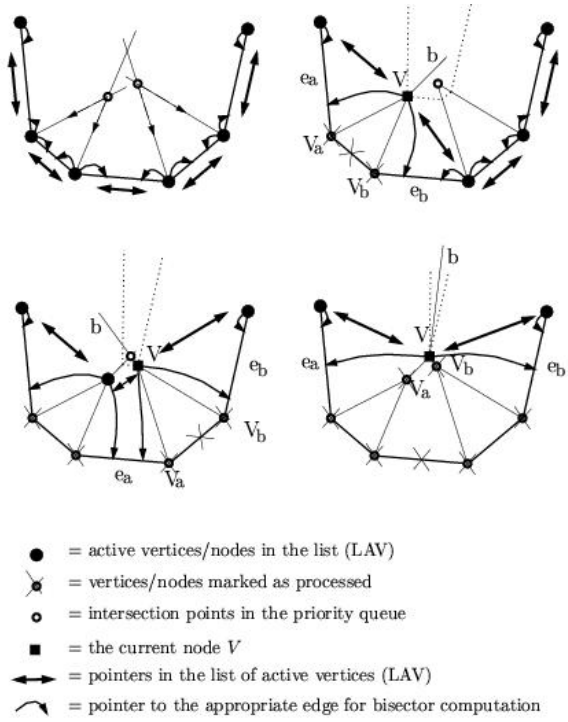

Figura 17 - Straight Skeleton - (FELKEL & OBDRZÁLEK, 1998).

2.4.10 Avanço da Fronteira

Um dos métodos descritos mais utilizados para a geração de malhas é o método do avanço da fronteira, o qual é um dos métodos mais simples. No caso do método do avanço da fronteira conforme (GEORGE, 1991), existem dois tipos de fronteiras. A fronteira por inflação, onde a fronteira consiste de todo o contorno do domínio, e o mesmo vai sendo preenchido por elementos, definindo-se novas fronteiras e este processo é repetido até que não exista mais área ou volume a ser fechado. O outro tipo de fronteira é a fronteira por propagação linear, onde a fronteira pode partir de várias regiões do contorno. Por exemplo, a fronteira pode iniciar de lados opostos do domínio até se encontrarem. De posse desta informação, é possível preencher o domínio com elementos triangulares. O mais importante nesta geração é o tamanho do ângulo gerado entre dois segmentos do contorno. Este ângulo dará o posicionamento do triângulo a ser inscrito no domínio. No caso

de se ter um ângulo menor que $\pi/2$ (90º), pode ser criado um triângulo simples. Se o ângulo estiver entre $\pi/2$ e $2\pi/3$ (90º e 120º), então, são criados dois triângulos e, se o ângulo for maior que $2\pi/3$ (120º), então, é criado um triângulo, e o segmento é retido. Ao reter-se um segmento, o mesmo não é mais utilizado para geração de um novo triângulo. Outro fator que deve ser levado em consideração na inserção dos elementos triangulares é a altura do triângulo inscrito. Esta altura deve possibilitar a criação de um triângulo eqüilátero, garantindo assim, a qualidade da malha e os resultados a serem obtidos pelo MEF. Este procedimento continua até que todo o domínio esteja preenchido com elementos triangulares.

O método utilizado por (BARBIERI, 2000) é o do avanço da fronteira seguindo assim, os conceitos de (GEORGE, 1991), o qual, descreve a forma de geração de uma malha com elementos triangulares. Para a geração dos elementos no domínio, é utilizada a inserção de uma linha auxiliar paralela ao contorno do domínio. Sobre esta linha é que serão definidos os novos nós. Para isto, a distância da linha do contorno do domínio deve satisfazer a altura de um triângulo eqüilátero inscrito. Após serem definidos todos os pontos sobre esta linha, esta passa a ser um novo contorno e assim, iniciando novamente o algoritmo até o total preenchimento do domínio. Com todos os nós definidos, são então criados os elementos triangulares. Criando a malha de elementos finitos triangulares. Para garantir uma melhor qualidade da malha, é feita a suavização Laplaciana. Este método é utilizado para reordenar os nós possibilitando os elementos terem a melhor forma possível, ou seja, triângulos eqüiláteros.

O ARANHA construído por (FANCELLO, 1990) é um gerador de malhas com elementos triangulares utilizando o método do avanço da fronteira. Nesta ferramenta os elementos podem ser identificados com 3 ou 6 nós. A quantidade de nós produz

uma qualidade melhor dos resultados a serem obtidos através das funções de interpolação. A partir do contorno do domínio geométrico definido com linhas e/ou curvas, iniciam-se a inserção dos elementos triangulares. Os nós sobre o contorno são definidos a partir de uma integral de linha, gerando assim a base da triangulação. Com esta base definida, ou seja, com os nós encontrados sobre o contorno é, então, feito à inserção de um elemento triangular. Este elemento triangular deve ter a característica de um triangulo eqüilátero. Para isto, é necessário encontrar o ponto C do triangulo ABC, onde AB é o segmento de reta definido sobre o contorno do domínio. Com o ponto C definido, um círculo a partir deste ponto é inscrito com um tamanho calculado. Este círculo serve para orientar se o ponto C é o ponto ideal e com o ponto ideal selecionado, o triangulo então é traçado dentro do domínio. Com isto, o segmento AB é marcado para não ser utilizado novamente. Os pontos que definem a nova fronteira são armazenados em uma estrutura do tipo lista. O fechamento da malha se dá quando não houver mais fronteiras a serem definidas. Para completar, uma suavização é feita, ou seja, o centro do polígono gerado pela interseção dos triângulos é reposicionado de forma a garantir uma melhor qualidade de todos os triângulos inscritos.

Na continuação do seu trabalho (FANCELLO, 1993) leva em consideração o uso da geração de malha para mecânica da fratura. A ferramenta utilizada é também o ARANHA, porém com mudanças na estrutura de armazenamento dos nós. Neste caso, o tipo de estrutura agora utilizado é o tipo árvore, ou árvore quaternária. Isto quer dizer, que cada ramo ou galho pode possuir quatro folhas. E cada folha passa a ser um ramo, podendo assim, também possuir quatro folhas. A utilização deste tipo de estrutura foi definida para facilitar a busca dos elementos, pois, em cada folha, é vinculada uma outra estrutura do tipo lista. Esta estrutura contém os dados dos nós

e sua localização geométrica. Um ponto forte deste trabalho em relação à geração de malhas é a definição das classes de objetos. Estas classes possibilitam uma eficiência maior em relação aos métodos tradicionais, pois nestes, as informações são armazenadas na forma de uma lista não encadeada, e assim necessitando de um número maior de iterações para encontrar um determinado elemento.

Em decorrência da grande utilização do método avanço da fronteira, (SCHÖBERL, 1997) descreveu neste artigo, um gerador de malha utilizando o método do avanço da fronteira. Por se tratar de um método muito popular, o avanço da fronteira pode ser aplicado em planos ou superfícies, bem como para a geração de malhas em 3D. Como toda ferramenta gráfica, esta necessita de uma forma para produzir os modelos geométricos, sendo, então, utilizado o *Construtive Solid* Geometry (CSG). Este método permite modelar e definir sólidos complexos por operações lógicas aplicadas a primitivas geométricas, onde os sólidos são representados a partir de uma árvore binária.

Com a definição do domínio com o uso do *CSG*, e antes de gerar a malha, é necessário encontrar os pontos especiais, que são pontos de interseção de segmentos (linhas ou curvas) no caso de geometrias em 2D, e interseção de superfícies no caso 3D. Para encontrar os pontos de interseção sobre o contorno do domínio, é necessário utilizar o algoritmo da bisecção. Este algoritmo irá dividir o domínio em quadriláteros. Com isto, o domínio inicialmente terá quatro quadriláteros. A partir de então, tem-se um domínio subdividido. Verifica-se então, em qual dos quadrantes existe ponto de contato, ou seja, a intersecção de planos. Com esta informação, os quadrantes que possuem os pontos de intersecção, são então novamente divididos. Esta divisão irá até que o ponto de intersecção esteja dentro de um pequeno cubo. O método utilizado vai depender do tipo de geometria, 2D ou

3D. Com isto, o algoritmo da bisecção divide o domínio em quadrados ou quadriláteros de mesmo tamanho. Após esta divisão, aos elementos que não encontrarem o domínio, será atribuído o valor 0 (zero). Aos elementos que interagirem com uma parte domínio, será atribuído o valor S, e aos elementos que estiver totalmente dentro do domínio, será atribuído o valor 1. Onde S é um novo modelo *CSG*, pequeno em relação ao domínio principal. Para melhorar a qualidade inicial do contorno, é utilizado o método de Newton (BATHE, 1996) em todos os elementos S. Neste ponto, são feitas divisões sucessivas com cubos. Isto serve para garantir a qualidade da malha nesta região.

Todos os pontos de intersecção encontrados devem ser verificados para saber a sua posição. Para isto, os pontos de interseção são classificados pelos autovalores do primeiro ponto menos o segundo ponto, ou seja, A - B. Assim, se os autovalores tiverem o mesmo sinal e o determinante destes autovalores for positivo, a intersecção ocorre somente em um ponto. Agora se os dois autovalores têm sinal oposto, isto é, se o determinante for negativo, então a intersecção dos planos ocorre neste ponto. Com isto, serão definidos os tipos dos pontos, podendo ser incondicionais ou condicionais. Os pontos de interseção encontrados durante este processo serão pontos incondicionais e são utilizados para gerar a malha. Os pontos condicionais são utilizados para garantir a existência de um eixo para a geração da malha. Ou seja, os pontos condicionais criam um eixo que possibilita encontrar os pontos para a geração da malha. Se os pontos incondicionais não puderem caracterizar uma malha, então os pontos condicionais serão utilizados. Para gerar a malha a partir destes pontos, é necessário um algoritmo global para preencher o domínio. Por isto, algumas regras foram definidas para garantir a geração de uma malha de boa qualidade. Os principais pontos deste algoritmo são: a transformação

para um sistema de coordenadas locais, aplicação das regras, geração de novos pontos localmente, transformação destes pontos para as coordenadas globais; tendo assim uma nova fronteira. No caso de ocorrer um “dead lock” ou parada repentina, é produzido um malha imperfeita. Após o fechamento da malha, é necessário utilizar algoritmos para melhoria da malha.

O método utilizado por (REES, 1997) é o avanço da fronteira com elementos triangulares e quadriláteros, produzindo uma malha híbrida. A malha híbrida é caracterizada pela inserção de diferentes tipos de elementos dentro do domínio. A necessidade de produzir uma malha híbrida vem do fato de que, por exemplo, em determinados casos, um quadrilátero não podendo ser inserido, insere-se então um elemento triangular. Para garantir a inserção dos nós no domínio é utilizada a função espacial. Esta função é simplesmente um campo escalar do comprimento da borda do elemento, que deve cobrir o domínio inteiro com uma malha.

O artigo também descreve algoritmos que possibilitam gerar malhas somente com quadriláteros ou somente com triângulos. Para a construção de quadriláteros, são definidos critérios particulares, alguns deles também utilizados para a geração de elementos triangulares. Porém, isto pode produzir uma malha fraca que necessite de um refino e também de uma suavização. Para fazer esta melhoria na malha, duas formas foram identificadas. A primeira é pela redistribuição da ligação dos nós dentro do elemento. A segunda forma é pela transformação de um elemento triangular em um quadrilátero.

O método de pavimentação exposto por (LOBER at. al., 1995) em conjunto com o método do avanço da fronteira e com técnicas de elementos finitos, produz uma malha de quadriláteros de boa qualidade. A técnica de pavimentação consiste na divisão do domínio em vários subdomínios seguida da geração da malha em

cada subdomínio. As malhas destes subdomínios são geradas a partir de diversas fronteiras até que todo o espaço esteja preenchido (ou não existam mais fronteiras abertas) e são geradas em processos paralelos. Sendo assim, deve-se evitar a sobreposição de malhas em pontos de vizinhança e a detecção da intersecção nas fronteiras opostas, isto no momento em que os processos em paralelo se encontram. Quando utilizada a técnica de paralelização, irão existir diversas áreas ou subdomínios, os quais, são tratados ou gerados individualmente. A malha gerada, bem como os dados de controle de cada um dos subdomínios são armazenados individualmente.

Na geração da malha, existem dois tipos de estruturas, uma serial e uma paralela. A serial é utilizada para definir os subdomínios iniciais. A paralela, leva em consideração o domínio anterior definido pela estrutura serial e tenta melhorar a sua subdivisão de forma a produzir subdomínios melhores. A partir desta estrutura, são então produzidas todas as melhorias necessárias para garantir uma malha de boa qualidade. Para existir esta melhoria, são utilizados dois estágios, o de preenchimento da malha e o de limpeza da malha. O preenchimento garante por procedimentos numéricos uma boa qualidade da malha. A limpeza da malha garante a retirada ou a recolocação dos nós indesejáveis.

Uma nova técnica proposta por (CHENG & TOPPING, 1998) para geração de malha consiste em utilizar o método avanço da fronteira com elementos triangulares para gerar quadriláteros em 2D. Na análise de tensões e deformações em relação aos elementos quadriláteros e triangulares, os quadriláteros fornecem resultados mais exatos. Isto para elementos lineares/bilineares com mesmo número de graus de liberdade.

Para gerar uma malha de quadriláteros através de triângulos, é necessário associar dois elementos triangulares. A qualidade da malha neste caso vai depender dos métodos de suavização e modificação. Para garantir a qualidade da malha e do elemento gerados, são aplicados conceitos da análise de erros de discretização, ou seja, o erro da norma de energia, local e global. O primeiro passo, no entanto, é a construção da malha inicial. Insere-se um elemento triangular dentro do domínio que deverá ter uma altura de $\delta * \operatorname{sen} 60^{o}$ a partir do ponto médio do segmento *AB* dado sobre o contorno, onde δ é um dos lados do triângulo a ser formado pelos pontos ABCo. Onde Co é ponto desejado para definir um triângulo eqüilátero. De um dos segmentos ativos deste triângulo, é criado um novo triângulo com a mesma altura do triângulo anterior, produzindo-se assim um quadrilátero. Este procedimento gera uma malha de quadriláteros com boa qualidade.

A "*fission*" ou divisão de quadriláteros garante uma malha com elementos quadriláteros bem formados. Esta divisão leva em consideração a quantidade de elementos que serão colocados na fronteira. Se na ocasião de adicionar o próximo triângulo, uma nova fronteira não estiver disponível, então um novo triângulo pode ser inserido. Ao se criar um elemento triangular, é necessário adicionar um outro triangulo em um dos lados para permitir a geração do quadrilátero. Neste caso, se existir uma fronteira no lado ativo do triangulo, e esta tiver somente elementos marcados como inativos, então pode ser adicionado um novo triangulo. No caso de existir elementos marcados como ativos, então é necessário buscar a fronteira que tenha um número impar de elementos ativos, possibilitando assim, a criação do outro triângulo nesta fronteira. A qualidade do elemento deve levar em consideração a forma do elemento, neste caso, os ângulos internos do triangulo deverão estar

entre 45º e 135º. Se o ângulo não estiver no domínio entre 30º e 150º, este elemento não será satisfatório.

O método de suavização aplicado neste artigo utiliza o método de suavização de Laplace de acordo com a estratégia do método de suavização Isoparamétrico apresentado por Leonard. A suavização de Leonard consiste de um esquema baseado sobre a transformação isoparamétrica de um elemento quadrático local. Além desta suavização proposta, uma nova suavização é definida. Essa suavização leva em consideração os nós internos, onde existem três de elementos adjacentes. Isto garante uma melhor qualidade da malha.

O ponto P é o centro em relação aos três elementos, ou seja, é o vértice que faz parte dos três elementos. Para que um novo ponto P1 seja definido, é necessário que os ângulos máximos sejam inferiores a 100º, pois, só pode encontrar um novo ponto P, onde este esteja no centro do triângulo. Se o ângulo formado entre os três eixos que definem os três elementos seja maior que 100º, então os segmentos que interceptam o ponto P são retidos e um novo segmento é construído. Se o ângulo for maior que 120º não é possível encontra um novo ponto P1.

Agora, se for encontrado ângulo maior que 165º, então o elemento que contem este ângulo, deverá ser dividido de forma a possibilitar a criação de outros elementos dentro dele. Estes novos elementos devem garantir a existência de ângulos iguais ou menores que 120º no ponto P1 (ver Figura 18).

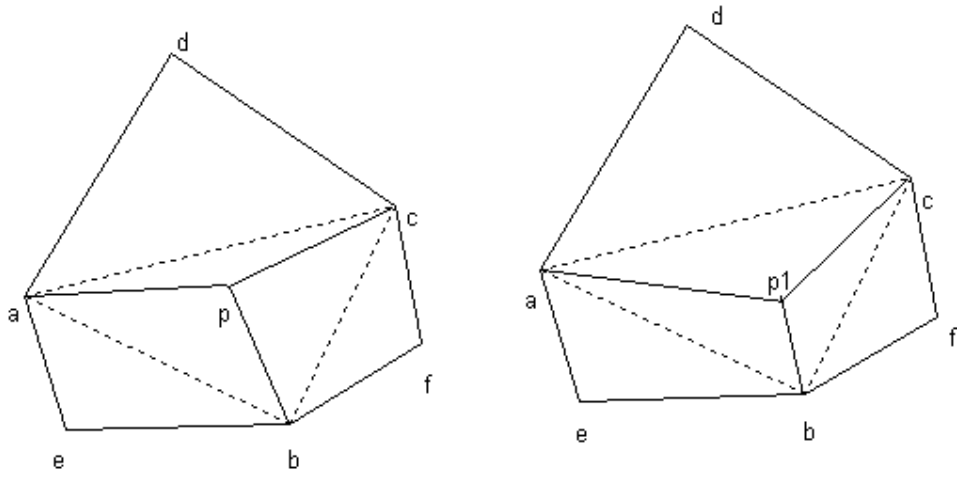

Figura 18 - Reposicionamento nó P para nó P1 - (CHENG & TOPPING, 1998)

O trabalho realizado por (TRISTANO at. al., 1998) apresenta a geração de uma malha de elementos finitos triangulares no espaço paramétrico de uma superfície, utilizando um mapa métrico de Riemannian. Para que esta malha seja gerada, é necessário identificar o domínio, construir uma malha de fundo, fazer a orientação dos segmentos da fronteira, definir a fronteira e processar a fronteira. Sendo assim, é utilizado o método do avanço da fronteira, que define uma fronteira inicial. Com esta fronteira delimitada, são adicionados os triângulos no interior do domínio.

Para garantir que na inserção destes triângulos seja gerada uma boa malha, é necessário calcular as distâncias entre os pontos do triângulo, ou seja, a distância entre os nós, ou então, determinar os tamanhos das arestas. Ao término do laço que preenche todo o domínio, é necessário um refino. Um ponto importante na construção dos elementos triangulares é controlar os seus tamanhos, bem como a sua posição e disposição em relação a outras fronteiras ou outros elementos. Para que um triângulo seja aceito, deve-se testar a sua área no espaço paramétrico, caso seja zero, então não é possível inserir o triângulo. O teste do nó interno identifica se o nó escolhido esta dentro do domínio desejado, ou seja, não ultrapassa outro

triângulo. E por último é feito o teste da intersecção da fronteira. Neste caso, o triângulo é testado em relação ao espaço paramétrico para verificar possíveis intersecções com as frentes definidas. No caso de não se conseguir inserir um triângulo, o método da força bruta é então acionado para inserir um novo triângulo levando-se em conta todo o domínio.

2.5 Conclusão

Este capítulo proporcionou um amplo conhecimento sobre a geração de malha, e métodos necessários para desenvolver a malha. Como por exemplo, os pontos de Steiner e Generalize Voronoï Diagram (GVD). Assim, foram apresentados vários métodos, e para alguns deles suas variações.

Capitulo 3

3 Avanço da Fronteira

3.1 Introdução

O método do avanço da fronteira (*advancing front*) é uma das melhores e mais simples técnica para geração de malhas. O método parte do contorno original do domínio, onde são inseridos elementos triangulares até o preenchimento total do domínio, porém, a cada instante de tempo uma nova fronteira é definida gerando assim, um novo contorno (GEORGE, 1991). A grande vantagem deste método, é que ele pode ser aplicado em conjunto com diversos outros métodos, como pode ser visto em (REES, 1997), (CAVALCANTE NETO at. al., 2000), (LI at. al., 2000), (LOBER at. al, 1995) (FANCELLO at. al., 1990º), (FANCELLO, 1993), (BERN & EPPSTEIN, 2000), (BERN & PLASMANN, 1999) e (TRISTANO at. al.,1998). Encontra-se o avanço da fronteira com triangulação de Delaunay, com Voronoï, Quadtree, com Octree, com Sphere Packing, entre outros.

O aspecto principal do método do avanço da fronteira é a geração de elementos triangulares a partir do contorno de um domínio Ω, onde a fronteira pode ser definida a partir de qualquer ponto do domínio, podendo iniciar de uma ou mais frentes ou de um ou mais pontos do domínio. No momento em que esta fronteira é totalmente preenchida, diz-se que foi fechada a região, esta região será um novo contorno, gerando desta maneira, mais uma fronteira até que o domínio esteja totalmente preenchido. Neste capítulo, será descrito como foi desenvolvido o algoritmo para a geração da malha de elementos finitos com o uso do método do avanço da fronteira.

3.2 Característica da Malha

Para gerar a malha de elementos triangulares em domínio poligonal plano, não são levadas em consideração as condições de contorno, as quais são caracterizadas pela restrição dos movimentos axiais e pelas cargas aplicadas sobre o domínio. Neste caso é gerada somente a malha de elementos triangulares do tipo gradual ou uniforme. Chama-se de malha gradual, a malha que tem como característica o aumento gradual da distância entre os nós sobre o contorno do domínio, podendo-se, assim, ser definido o tamanho desejado do elemento a partir dos vértices do domínio. No algoritmo, quatro tamanhos de elementos foram fixados e a eles designados códigos e pesos. Se for identificado o tipo de elemento 1, este terá peso 10. Caso seja identificado como 2, terá peso 15 e para o tipo de elemento 3, o peso será de 20. O último tipo de elemento recebe o código 4 e terá peso 25. Atribuindo-se diferentes tipos de elementos aos vértices do contorno original do domínio, gera-se uma malha gradual. Para representar uma malha uniforme no domínio, esta é definida atribuindo-se um único tipo de elemento aos vértices do contorno. Exemplos de malha uniforme e gradual podem ser vistos pelas Figuras 19 e 20, respectivamente.

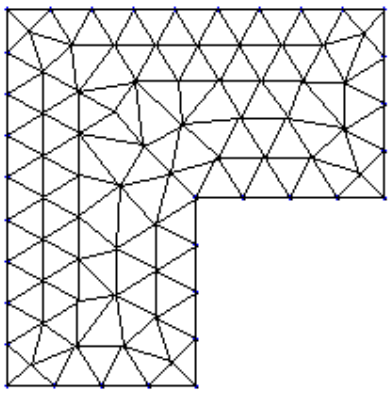

Figura 19 - Malha Uniforme

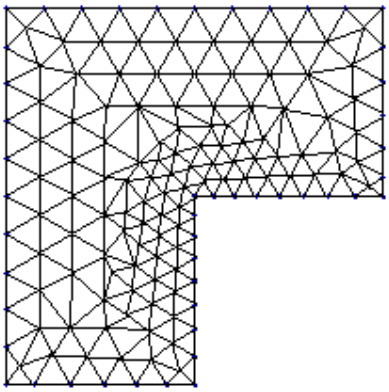

Figura 20 - Malha Gradual

Para melhorar as características dos elementos inscritos na malha, um método de suavização foi implementado, resolvendo assim alguns problemas dos elementos, como pode ser visto no Capítulo 4, porém, não foi implementado nenhum método de refino de malha.

3.3 Identificando o Contorno

A primeira fase, conforme apresentado no Anexo A no seu item A1.10, é identificar os nós ou vértices do domínio original, e isso se faz a partir da estrutura de dados do tipo lista encadeada chamada aqui de MGObject, e que também está apresentada no Anexo A. Com estes dados é possível saber qual é o ponto de origem, ou seja, o primeiro nó do domínio inserido e montar a seqüência dos demais nós sobre o contorno do domínio Ω. Esta seqüência de nós é definida no momento da criação do domínio poligonal plano, utilizando-se para isso os vértices do domínio original. Com essa seqüência tem-se a lista de nós criada (Tabela 1), esta lista contém o índice (seqüência numérica dos nós), as coordenadas nodais, o tipo de vértice, o tamanho do elemento inscrito a partir do nó de origem e o ângulo formado entre os segmentos de reta, possibilitando navegar sobre a lista de nós. De posse desta informação, é possível identificar quais são os nós vizinhos (anterior ou posterior) de um determinado nó. A Tabela 1 também é utilizada para determinar os

vértices côncavos ou convexos de um determinado polígono, assim como o ângulo formado pelos segmentos de retas associados aos nós. Para saber se um vértice é côncavo ou convexo, é necessário identificá-lo, isto é feito calculando-se a área do triângulo gerado entre três nós consecutivos. Ou seja, se a área calculada for negativa então o vértice é convexo e foi definido o valor 0 para identificá-lo, agora se a área for positiva, então é um vértice côncavo e terá o valor 3, isso pode ser visto na coluna vértice da Tabela 1.

Tabela 1 - Tabela de Nós (X,Y)

Índice	Pos. X	Pos. Y	Vértice	Tensão	Ângulo
1	650	300	0	1	90
2	850	300	0	4	90
3	850	450	0	4	90
4	650	450	0	1	90

Com a identificação do contorno e dos nós, definiu-se que a ordenação destes nós segue a seqüência de inserção dos pontos do polígono inscrito, ou seja, sentido horário a partir do primeiro vértice desenhado do domínio. A Figura 21 representa um domínio poligonal plano com quatro vértices convexos cuja seqüência de nós está representada numericamente na Tabela 1. Esta seqüência será utilizada para a criação da fronteira inicial, partindo-se do primeiro vértice para a inclusão dos elementos triangulares.

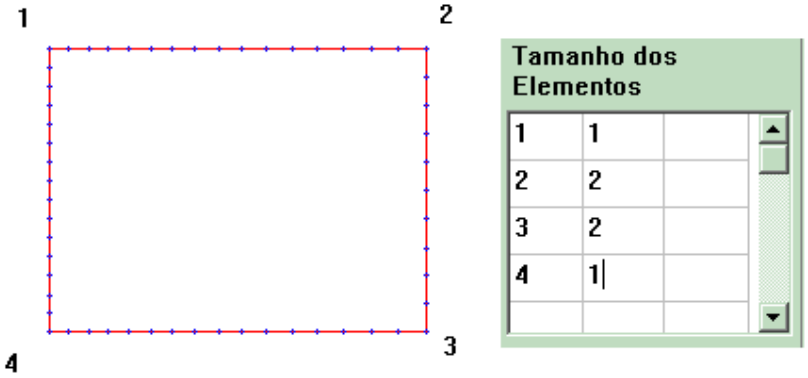

Figura 21 - Identificação dos Vértices e o Tamanho do Elemento em cada Vértice

3.4 Definição dos Nós sobre o Contorno Original (Fronteira Inicial)

Para iniciar o método do avanço da fronteira é necessário identificar os nós sobre o contorno original, ou contorno do domínio (Tabela 1). Sobre esse contorno, devem ser colocados os nós que possibilitaram a criação dos elementos segundo o tamanho desejado, a partir de cada um dos vértices do domínio, como mostra a Figura 22.

A inserção dos nós sobre o contorno começa pelos vértices de índice 1 e o vértice de índice 2. No algoritmo foi definido que o ponto de origem como sendo o ponto B, e o próximo como D, e o anterior é o nó A. De posse das coordenadas geométricas em relação aos pontos B e D, pode-se encontrar a distância ideal para adicionar uma quantidade exata de nós sobre este segmento BD, isso para o caso de uma malha uniforme. No caso de uma malha gradual, os nós devem ser colocados de forma a permitir o completo preenchimento do segmento BD, de forma a possibilitar um aumento gradual da distância entre os nós definidos.

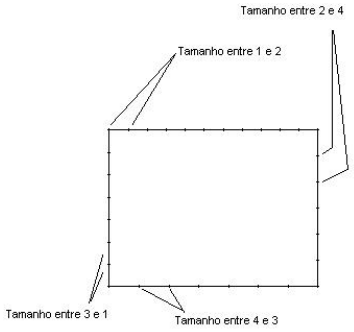

Figura 22 - Diferença entre os Pontos Inscritos

Para encontrar o tamanho exato entre cada um dos nós, foi utilizado o cálculo da PA conforme apresentado no Anexo A. Já (FANCELLO, 1993), (FANCELLO, 1990) e (ZHU at. al., 1991), utilizaram a integral de linha sobre todo o contorno para encontrar o tamanho e a posição ideal de cada um dos nós sobre o contorno. Neste trabalho, poderia ter sido utilizado o mesmo recurso. Porém, como é feito o cálculo

para cada um dos segmentos do contorno original, optou-se por continuar com a fórmula da PA. O algoritmo utilizado no cálculo da PA está descrito no Anexo A, e o algoritmo que preenche o contorno, no Anexo B.

3.5 Criando os Elementos a partir da Fronteira Aberta

Após serem inseridos todos os nós sobre o contorno como mostra a Figura 23, passa-se para o preenchimento da fronteira aberta. Neste caso, a fronteira é todo o contorno do domínio, semelhante ao que (FANCELLO, 1993) e (BARBIERI, 2000) descreveram. Mas, isso somente para a identificação dos nós sobre o contorno original do domínio.

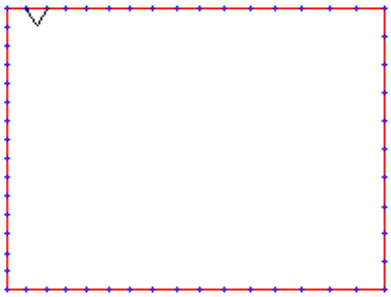

Figura 23 - Nós Definidos sobre o Contorno

No caso deste trabalho, serão adicionados somente elementos triangulares utilizando o método do avanço da fronteira. O ideal em relação aos elementos inscritos é que esses sejam triângulos eqüiláteros ou o mais próximo possível, como pode ser visto na Figura 24. Os elementos triangulares adicionados na primeira iteração do contorno partem do primeiro nó do contorno até o último, em sentido horário (ver Tabela 1).

Sendo assim, o algoritmo ao identificar o próximo nó do contorno, inicia a inserção dos elementos triangulares, levando-se em consideração a sua altura.

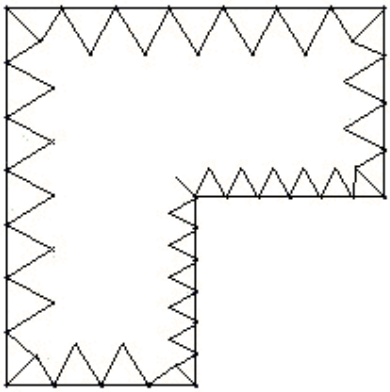

Figura 24 - Construção da Fronteira

Para ser um triângulo eqüilátero perfeito, é necessário que a altura seja dada pela fórmula:

$$altura = \frac{l\sqrt{3}}{2} \cong 0{,}86\,,$$

onde *l* é o tamanho do segmento AB. É equivalente afirmar que, se multiplicar o segmento AB por *0,86*, tem-se o mesmo resultado da fórmula acima apresentada. De posse do segmento AB, calcula-se o ponto de Steiner (possível ponto que permite a criação de um elemento triangular) ou nó C, conforme explicitado por (BERN & EPPSTEIN, 2000) e (BERN at. al., 1995), e que é necessário para completar o triângulo, construindo-se assim o triângulo ABC. Para encontrar este novo nó, utiliza-se a técnica de rotação de eixos, neste caso a rotação do segmento AB, conforme descrito no Anexo A item A1.9, produzindo-se assim, o que seria a inserção de círculos a partir dos pontos de origem do segmento AB. Partindo-se do ponto A e do ponto B inscrevem-se dois círculos, um com o centro em A e outro com centro em B. A interseção destes dois círculos produzirá dois pontos como pode ser visto na Figura 25, o nó C e o nó C'. Para identificar qual dos dois nós é o ideal, é necessário determinar a área do triângulo gerado pelos pontos ABC e ABC'.

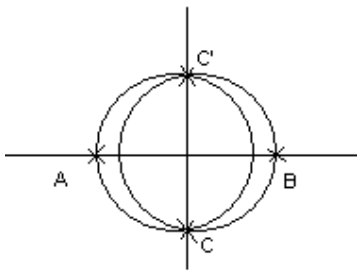

Figura 25 - Pontos de Intersecção entre os Círculos

Se a área for positiva, o triângulo estará dentro do domínio, mas, se a área for negativa, o triângulo estará fora do domínio. A equação utilizada para cálculo da área está definida no Anexo A, item A.4 – Algoritmo de MELKMANN.

De posse do ponto C, deve-se verificar se o mesmo pode ser inscrito na posição definida. Para isso, é feita a verificação dos ângulos gerados entre os segmentos de retas que constituem as arestas dos triângulos. No início, ou seja, na primeira iteração da inserção dos elementos triangulares sobre o contorno do domínio original, os segmentos de retas estão alinhados (vide segmentos ABD e BDE na Figura 26).

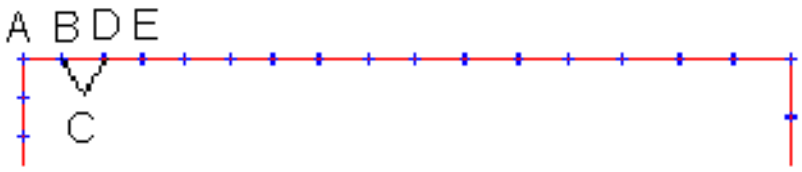

Figura 26 - Segmentos de Reta

Nos próximos itens deste capítulo, serão descritas cada uma das possibilidades de ângulos formados e sua característica conforme a Tabela 2, onde os ângulos são calculados a partir do segmento de reta ABD e BDE. O ponto B é a origem do ângulo formado entre os segmentos AB e BD, e é também um nó base para a inserção do elemento triangular. O próximo ângulo é encontrado pelo segmento BD e DE, onde o ponto D é a origem este ângulo, e será o outro nó base

para a inserção do elemento triangular. A Figura 26 representa um elemento triangular sendo constituído com os nós BDC, possuindo um ângulo de 180º entre os segmentos AB e BD, e o próximo ângulo sendo também de 180º entre os segmentos BD e DE.

Em seu livro, (GEORGE, 1991) descreve a possibilidade de inserir um triângulo em somente três situações, como descrito no Capítulo 2. Já o artigo de (ZHU at. al., 1991) provê outras formações de ângulos para a inserção de elementos triangulares. Sendo assim, optou-se por estabelecer outras variações com ângulos que possibilitassem a inserção de um elemento triangular.

Tabela 2 - Ângulos Identificados

	Segmento ABD entre		Segmento BDE entre	
1	157,5º	180º	157,5º	180º
2	157,5º	180º	135º	157,5º
3	157,5º	180º	112,5º	135º
4	157,5º	180º	90º	112,5º
5	157,5º	180º	67,5º	90º
6	157,5º	180º	45º	67,5º
7	135º	157,5º	157,5º	180º
8	135º	157,5º	135º	157,5º
9	135º	157,5º	112,5º	135º
10	135º	157,5º	90º	112,5º
11	112,5º	135º	157,5º	180º
12	112,5º	135º	135º	157,5º
13	112,5º	135º	112,5º	135º
14	100º	112,5º	157,5º	180º
15	90º	100º	157,5º	180º
16	90º	112,5º	135º	157,5º
17	90º	112,5º	112,5º	135º
18	90º	112,5º	90º	112,5º
19	90º	112,5º	0º	90º
20	67,5º	90º	157,5º	180º
21	50º	86º	112,5º	135º
22	67,5º	90º	67,5º	112,5º
23	0º	67,5º	157,5º	180º
24	0º	67,5	0º	157,5º

3.5.1 Segmento ABD entre 157,5º e 180º e segmento BDE entre 157,5º e 180º

Se os ângulos formados entre os segmentos de reta ABD e BDE estiverem entre 157,5º e 180º, conforme Figura 27, um triângulo eqüilátero poderá ser inscrito. Com isso, deve ser validado o ponto C com as funções BuscaNoProx e DefiPonto apresentadas no Anexo A itens A1.13 e A1.14.

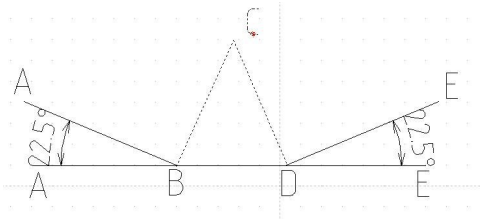

Figura 27 - Ângulos entre 157,5º e 180º

Após ter sido validado o ponto C, deve-se verificar o ângulo formado entre o segmento CB e BF (ver Figura 28), o nó F faz parte do triângulo inscrito anteriormente. Para encontrar o ângulo entre dois segmentos de reta, é utilizada a função BuscaAngulo definida no Anexo A item A1.15. Esta função retorna o ângulo formado entre o nó F do elemento triangular anterior, o nó B e o nó C atual. O ângulo α deverá estar entre 30º e 80º, ou entre 93º e 98º ou entre 108º e 170º , o que pode ser visto na Figura 28. Com isso, o ponto C é um nó válido, e deve ser inserido na tabela de nós, como pode ser visto na Tabela 1. O elemento triangular criado a partir dos nós B, D e C são armazenados na tabela de elementos. Esta tabela contém como informação respectivamente a coordenada do nó C, a coordenada do nó B e a coordenada do nó D, conforme Tabela 3.

Tabela 3 - Tabela de Elementos Inscritos

Posição	x1	y1	x2	y2	x3	y3
1	680,5064	317,0180	670	300	690	300
2	700,5064	317,0180	690	300	710	300
3	720,5064	317,0180	710	300	730	300
4	740,5064	317,0180	730	300	750	300
5	760,5064	317,0180	750	300	770	300

Caso, o ângulo formado entre os segmentos de reta CB e BF, não esteja entre os ângulos acima definidos, então, só é armazenado o nó B conforme a Tabela 1, e o elemento não é criado.

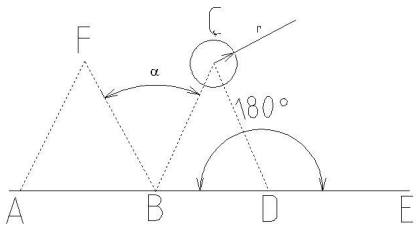

Figura 28 - Triângulo inscrito nos nós B e D com ângulo de 180º

3.5.2 Segmento ABD entre 157,5º e 180º e segmento BDE entre 135º e 157,5º

Se no segmento de reta ABD forem formados ângulos entre 157,5º e 180º, e para o segmento BDE os ângulos estiverem entre 135º e 157,5º ocorrerão duas situações. Na primeira, deve-se levar em consideração a posição do nó D. No caso de ser o último nó da fronteira da fronteira criada, e este já ter sido utilizado para adicionar um triângulo, então será traçada uma única reta entre o nó B e o nó F do triângulo inscrito, produzindo assim, um elemento triangular com os nós B, D e F. Este elemento triangular será inserido na tabela de elementos conforme Tabela 3. O nó de origem vai ser armazenado como um nó válido na Tabela 1.

No caso de ser ou não o último nó e que não tenha sido inscrito um elemento no primeiro nó, pode-se traçar um elemento triangular. Este elemento vai levar em consideração os nós B, D e C, como pode ser visto na Figura 29, onde é traçado um elemento triangular a partir desses nós. Então, fica armazenado o elemento triangular gerado, e os nós B e D como os novos nós na tabela de nós.

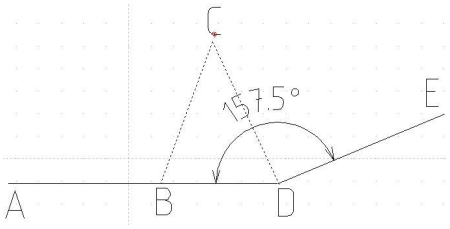

Figura 29 - Triângulo Inscrito nos nós B e D e ângulo de 157,5º

Estes novos nós farão parte da próxima fronteira a ser criada. Os ângulos que possibilitam este tipo de triângulo podem ser observados na Figura 30.

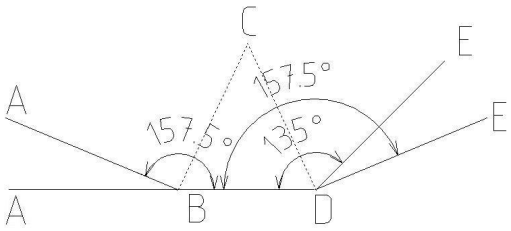

Figura 30 – Segmento ABD entre 157,5º e 180º e segmento BDE entre 135º e 157,5º

Agora, se a função DefiPont retornar falso, então não será desenhado o elemento triangular e o nó A será armazenado na Tabela de nós (Tabela 1) para ser utilizado na próxima fronteira.

3.5.3 Segmento ABD entre 157,5º e 180º e Segmento BDE entre 112,5º e 135º

Neste caso, existem três possibilidades de criação de elementos triangulares. A primeira opção acontece no caso do nó ser o último nó da fronteira aberta. Se isso ocorrer, então, será traçada uma linha do nó de origem A até o nó C do triângulo

gerado pela primeira posição da fronteira. Será armazenado o elemento triangular gerado por esta união (Tabela 3), e o nó de origem A, na tabela de nós (Tabela 1).

Agora, se não for o último nó da fronteira, este pode ser um vértice côncavo ou convexo. Se for um vértice côncavo, então pode inserir um elemento triangular utilizando os nós B e D. Se o nó C for válido, conforme visto do item 3.5.1, então um elemento triangular é inserido a partir dos nós B e D conforme Figura 31, criando-se, assim, um elemento na tabela de elementos. Os nós B e D serão adicionados à tabela de nós da próxima fronteira. Se o nó C não for válido, então somente o nó B será inserido na tabela de nós.

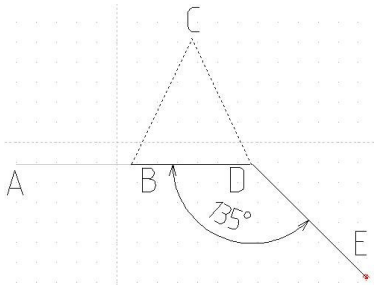

Figura 31 - Triângulo inscrito nos nós B e D e ângulo de 135º côncavo

No caso de ser um vértice convexo, deve-se utilizar a função BuscaAngulo com os segmentos DC e DC' para encontrar o ângulo formado entre os segmentos. Se o ângulo α for maior que 50º, então é calculado o ponto médio C'' entre o nó C atual e o nó C' do próximo triângulo. Para validar este novo nó C'', que é o nó médio, usa-se a função BuscaNoProx. Com o ponto C'' validado, são traçados dois elementos triangulares, um em relação aos nós B, D e C'' e o outro em relação aos nós D, E e C'', conforme Figura 32.

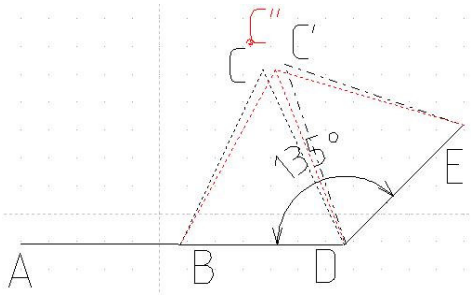

Figura 32 - Triângulos inscritos nos nós B, D e E e ângulo de 135º convexo

Porém, caso o ângulo seja inferior a 50º, então dois elementos triangulares são gerados a partir do nó C', ou seja, do nó C anterior. Em ambos os casos, os dois elementos criados são armazenados na tabela de elementos. Os nós B e C' são armazenados na tabela de pontos.

Os ângulos formados entre os segmentos ABD e BDE são vistos na Figura 33.

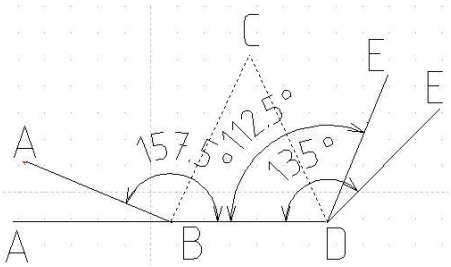

Figura 33 – Segmento ABD entre 157,5º e 180º e Segmento BDE entre 112,5º e135º

3.5.4 Segmento ABD entre 157,5º e 180º e Segmento BDE entre 90º e 112,5º

O primeiro passo para este algoritmo é encontrar o ponto médio C'' entre o nó C e o nó C', onde C' é o ponto C do triângulo anterior em relação ao segmento DE. Encontrado o nó C'', o mesmo deve ser validado pela função BuscaNoProx. Em relação ao nó C, deve-se calcular o ângulo α formado entre os segmentos BC e BC' com a função BuscaAngulo.

Este algoritmo pode gerar seis opções diferentes, onde cada valor depende da necessidade de compartilhamento dos elementos triangulares anteriores. No primeiro caso, se em tentativas anteriores não foi possível inserir um elemento triangular, então o nó B é armazenado na tabela de nós e não é criado nenhum elemento triangular.

No entanto, se foi adicionado um elemento triangular anterior e esse for compartilhado, ou o ângulo retornado pela função BuscaAngulo for menor que 20º, então existem duas opções. A primeira verifica se não houve inserção de um elemento triangular no primeiro nó em relação à tabela de nós. Com isso, são inseridos dois elementos triangulares em relação ao nó C'. São eles os elementos triangulares com os nós BDC' e DEC', conforme Figura 34, os quais são armazenados na tabela de elementos.

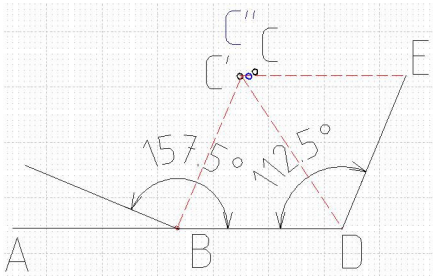

Figura 34 - Triângulos inscritos com base no nó C'

Agora, se os nós D ou E foram utilizados para a geração de um elemento triangular, então é desenhada uma linha que liga o nó B ao nó C' deste elemento triangular, criando assim um novo elemento na tabela de elementos.

Se não houver o compartilhamento de triângulos e o ângulo α encontrado pela função for maior que 100º, dois elementos são, então, adicionados na tabela de elementos. Um com as coordenadas de BDC'' e o outro DEC'', sendo armazenados na tabela de nós os nós B e C''.

Se o ângulo α for menor ou igual a 100º, tem-se duas opções. Encontrar um vértice côncavo ou um vértice convexo. No caso do vértice convexo, é necessário aumentar o segmento de reta BC, em 20 por cento (20%), para verificar se o tamanho do segmento de reta BC é menor que o segmento de reta DE. Neste caso, o nó C'' será calculado pela média entre os nós C e C'. De posse do nó C'', criam-se os elementos triangulares BDC'' e DEC'', conforme a Figura 35, os quais são armazenados na tabela de elementos. Os nós B e C'' são armazenados na tabela de nós.

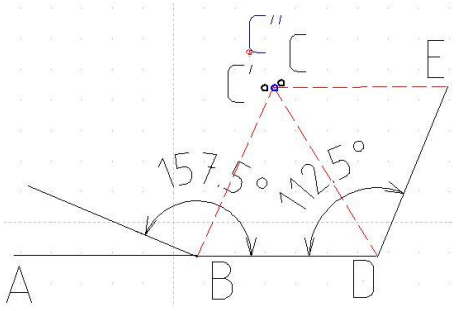

Figura 35 – Triângulos inscritos com base no nó C''

Para o vértice côncavo, é traçado um elemento triangular a partir do nó C, criando-se assim o triângulo BDC, o qual é armazenado na tabela de elementos. Os nós B e C são adicionados na tabela de nós.

A Figura 36 mostra o formato dos ângulos para este caso.

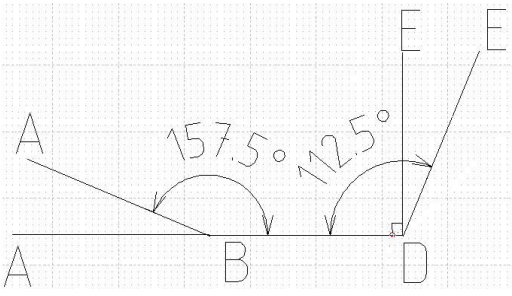

Figura 36 – Segmento ABD entre 157,5º e 180º e Segmento BDE entre 90º e 112,5º

3.5.5 Segmento ABD entre 157,5º e 180º e Segmento BDE entre 67,5º e 90º

Como o próximo ângulo (segmento BDE) encontrado tem um valor entre 67,5º e 90º, trata-se de um ângulo pequeno, como pode ser visto na Figura 37. Por isso, é utilizado para inserir somente elementos triangulares em vértices côncavos. Após a validação do nó C, um elemento triangular é inscrito no domínio, como mostra a Figura 38. Esse elemento possui os nós B, D e C e são armazenados na tabela de elementos, sendo os nós B e C armazenados na tabela de nós.

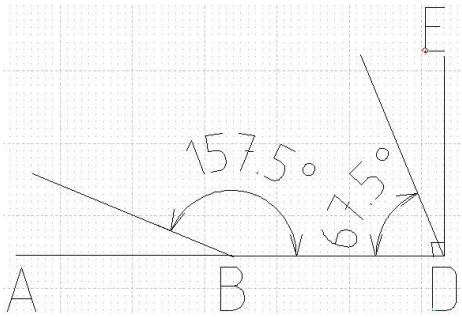

Figura 37 – Segmento ABD entre 157,5º e 180º e Segmento BDE entre 67,5º e 90º

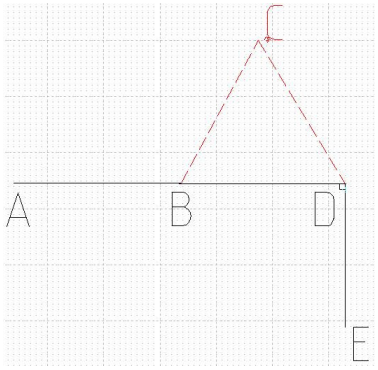

Figura 38 - Triângulo inscrito no vértice côncavo e ângulo de 90º

3.5.6 Segmento ABD entre 157,5º e 180º e Segmento BDE entre 45º e 67,5º

Pelo tamanho do ângulo formado entre os segmentos BD e DE, também só é possível utiliza-lo como um vértice côncavo. O algoritmo utilizado para estes ângulos segue o que foi definido no item 3.5.5. A Figura 39 mostra o formato do ângulo gerado.

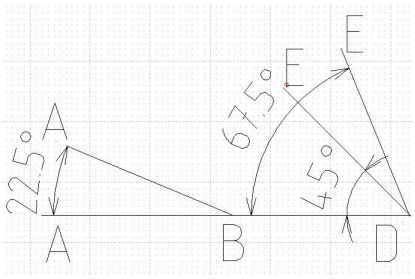

Figura 39 - Ângulos entre 157,5º e 180º e entre 45 e 67,5º

3.5.7 Segmento ABD entre 135º e 157,5º e Segmento BDE entre 157,5º e 180º

Os segmentos com estes ângulos possibilitam algumas variações. Isto pode ser visto pela Figura 40, a qual representa os ângulos formados por cada um dos segmentos utilizados. Primeiro encontra-se o ponto médio C'' em relação ao nó C do segmento BD, e o nó C' do segmento AB. Com o nó C'' definido, o mesmo deve ser validado com a função BuscaNoProx.

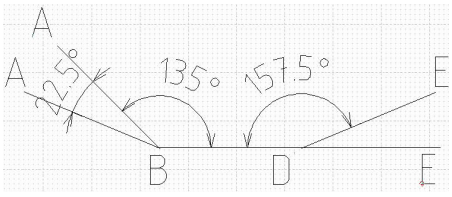

Figura 40 – Segmento ABD entre 135º e 157,5º e Segmento BDE entre 157,5º e 180º

O algoritmo é dividido de duas maneiras, e é levado em consideração o nó C do último elemento triangular inscrito. Se o triângulo anterior não foi identificado como complementar, então se tem o primeiro caso. Com isso, o ângulo α formado pelos segmentos de retas BC e BC''' deve ser calculado. Neste caso, C''' é o nó C encontrado para o segmento de reta DE, ou seja, do próximo segmento de reta. Se o ângulo calculado for menor que 40º, existem três possibilidades. A primeira ocorre se o nó A é um vértice convexo. Então, são produzidos dois elementos triangulares um elemento possui os nós ABC'', e o outro, os nós BDC'', como pode ser visto na

Figura 41 e os nós armazenados na tabela de elementos. Os nós a serem armazenados na tabela de nós dependerão do tipo do elemento triangular inscrito. Ou seja, se houver a necessidade de utilizar o nó A como base para a criação de um elemento, o nó A será colocado na tabela de nós, junto com o nó C''. Do contrário, somente o nó C'' será adicionado à tabela de nós.

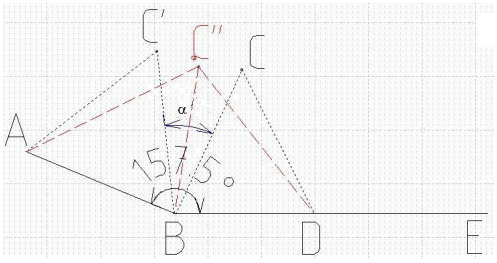

Figura 41 - Triângulos inscritos nos nós A, B e D

Se o vértice for côncavo, então será inscrito um elemento triangular a partir dos nós BDC, conforme a Figura 42. Os nós serão, então, armazenados na tabela de elementos, e os nós D e C armazenados na tabela de nós. Se o nó não for validado pela função DifePont, então não será gerado elemento triangular. Sendo assim, somente o nó B é armazenado na tabela de nós.

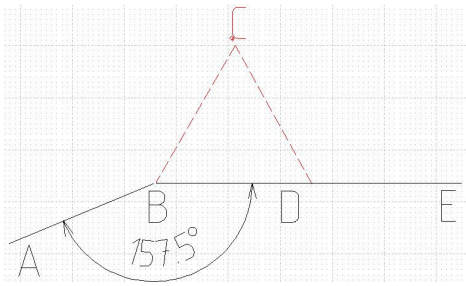

Figura 42 - Triângulo inscrito nos nós B e D

Agora, encontrando um ângulo maior que 40º, ocorrerá dois casos. Será levado em consideração se o triângulo formado pelos nós BDE está direcionado para dentro da fronteira ou para fora da fronteira. No primeiro caso, o triângulo deve estar para fora da fronteira, e assim será traçada uma linha do nó C até o nó D, este nó C é do triângulo anterior, como pode ser visto na Figura 43, criando-se um

elemento com os outros nós da fronteira. O elemento BDC será armazenado na tabela de elementos. Nenhum dos nós será armazenado na tabela de nós. No segundo caso, será desenhado um elemento triangular com os nós BDC, os quais são armazenados na tabela de elementos, e o nó B e C armazenados na tabela de nós.

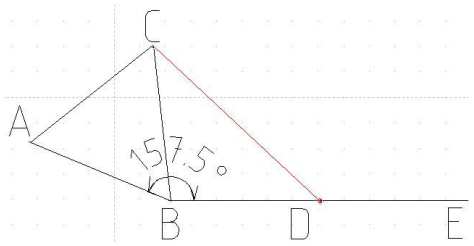

Figura 43 - Triângulo inscrito a partir do nó D

Ao utilizar um elemento triangular inscrito anteriormente como base, existe duas formas de criar este elemento. Sendo o nó B um vértice convexo, então, é traçada uma linha do nó C do elemento triangular anterior ao nó B, criando-se um elemento triangular com os nós BDC. Esses nós são armazenados na tabela de elementos, não sendo armazenado nenhum nó na tabela de nós. Caso contrário, se o nó B for um vértice côncavo, o nó C deve ser validado. O nó C não sendo um ponto válido, o nó B será armazenado na tabela de nós. Se for validado, então, um elemento triangular é inscrito pelos nós B, D e C, como na Figura 42, e os nós são armazenados na tabela de elementos. Na tabela de nós, serão armazenados os nós B e C. Neste caso, o ponto será armazenado como um vértice côncavo.

3.5.8 Segmento ABD entre 135º e 157,5º e Segmento BDE entre 135º e 157,5º

Para ser possível inserir um elemento triangular com estes ângulos, o ângulo formado pelos segmentos BC e BC' deve ser calculado, onde C' é o próximo nó C em relação ao segmento DE. Sendo o ângulo maior ou igual a 35º, então há a

possibilidade de inserir um elemento triangular. Para isso, é necessário validar o nó C. Se o nó C não for um nó válido e se a inserção de triângulos não estiver bloqueada, então um elemento é inserido, conforme a Figura 44. Quando a inserção de elemento triangular estiver bloqueada, significa que é a última fronteira criada, e o bloqueio de inserção de novos elementos é utilizado para que não ocorra a sobreposição de novos elementos triangulares.

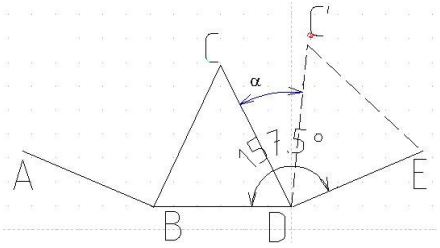

Figura 44 - Triângulo inscrito nos nós B e D

O elemento triangular gerado tem os nós B, D e C, como pode ser visto na Figura 45, onde C é o nó C do triângulo anterior inscrito. Caso contrário, somente o nó B é armazenado na tabela de nós e não havendo elemento triangular inscrito.

Porém, se o nó C for um nó válido, então, um triângulo é construído com os nós BDC, sendo os mesmos armazenados na tabela de elementos, e os pontos B e C, armazenados na tabela de nós. Os ângulos formados podem ser visualizados pela Figura 46.

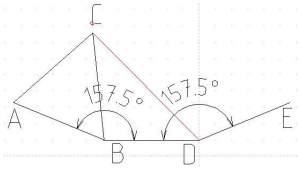

Figura 45 - Triângulo inscrito a partir do nó D

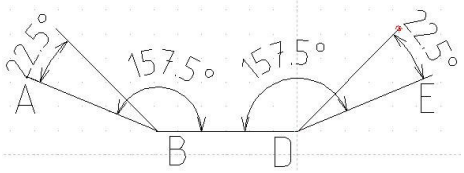

Figura 46 – Segmento ABD entre 135º e 157,5º e Segmento BDE entre 135º e 157,5º

3.5.9 Segmento ABC entre 135º e 157,5º e Segmento BDE entre 112,5º e 135º

Para este tipo de ângulo, o vértice pode ser côncavo ou convexo. Tanto para o vértice côncavo quanto para o convexo, o nó C deve ser validado, este nó C é definido em relação ao segmento DE. Se não for um nó válido, então somente será armazenado o nó B na tabela de nós.

Se for um vértice convexo e o nó C válido, deve-se então, encontrar o ângulo gerado entre os segmentos BC e BC', onde C' é o nó C do triângulo inscrito anteriormente. Se o ângulo for maior que 40º, então dois elementos triangulares são desenhados, conforme a Figura 47. O primeiro tem os nós BDC e o segundo os nós DEC, sendo ambos armazenados na tabela de elementos. Os nós B e C são, então, armazenados na tabela de nós. Possuindo um ângulo menor ou igual a 40º, os triângulos são formados utilizando-se o nó C'. Geram-se, então, dois elementos triangulares, um com os nós ABC' e outro BDC', sendo estes nós armazenados na tabela de elementos. Não há nós a serem armazenados na tabela de nós.

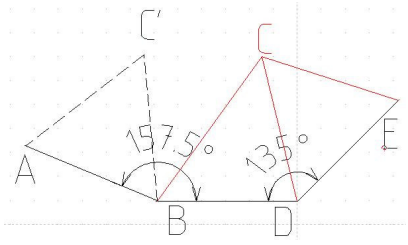

Figura 47 - Triângulos inscritos nos nós B, D e E

Se o vértice for côncavo e o nó C válido, então, é criado um elemento triangular com os nós ABC, conforme Figura 48. Esses nós são armazenados na tabela de elementos. Na tabela de nós, são adicionados os nós A e C. Contudo o nó C deve ser inserido como um vértice côncavo. A Figura 49 representa os ângulos formados entre os segmentos.

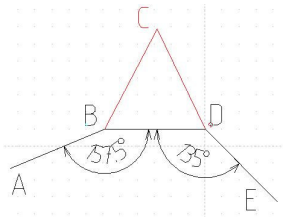

Figura 48 - Triângulo inscrito nos nós B e D e vértice côncavo

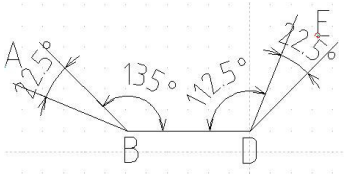

Figura 49 – Segmento ABD entre 135º e 157,5º e segmento BDE entre 112,5º e 135º

3.5.10Segmento ABD entre 135º e 157,5º e Segmento BDE entre 90º e 112,5º

Para saber se é possível adicionar um elemento triangular com estes ângulos, é necessário calcular o ponto médio C''. Este ponto médio é calculado com os nós C e C', onde o nó C' é o nó C do triângulo anterior formado pelo segmento AB, e A é o nó anterior a B. De posse do nó C'', o mesmo deve ser validado pela função BuscaNoProx.

Por possuir vértices tanto convexos quanto côncavos, foram definidas duas variações para eles. Na primeira, leva-se em consideração o vértice convexo, onde existem três variações possíveis, e para isso, é necessário calcular o ângulo formado entre os segmentos de reta BC e BC'. O nó C' é o nó C do triângulo anteriormente inscrito no domínio. Se o ângulo for maior ou igual a 40º, e o elemento triangular anterior estar ativo, então, são inseridos dois elementos triangulares. Esses elementos têm como coordenadas os nós BDC'' e DEC'', onde C'' é o nó C do elemento triangular criado a partir do segmento DE (ver Figura 50), os quais são armazenados na tabela de elementos. Na tabela de nós, são armazenados os nós B e C''. Contudo, se for o último nó da fronteira, tendo sido um elemento adicionado no primeiro nó, então, é utilizado o nó C, que é obtido a partir do segmento DE.

Posteriormente, este ponto C é validado com a função BuscaNoProx, possibilitando assim a geração de dois elementos triangulares. Os elementos BDC e DEC são adicionados na tabela de elementos, enquanto os nós B e C são armazenados na tabela de nós.

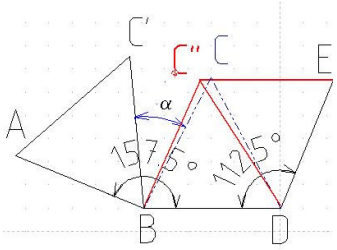

Figura 50 - Triângulos inseridos nos nós B, D e E

Caso isto o ângulo seja inferior a 40º, é traçado uma reta entre os nós C'D e C'E, definindo assim, dois novos elementos triangulares com os nós BC'D e DC'E, e são armazenados na tabela de nós. Agora, se for encontrado um vértice côncavo, o elemento triangular é criado com os nós BDC, conforme a Figura 51. Esses nós são armazenados na tabela de elementos, e também na tabela de nós.

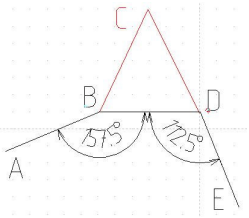

Figura 51 - Triângulo inscrito nos nós B e D e vértice côncavo

A Figura 52 mostra as características dos ângulos formados, e sua necessidade de diferentes tipos de alocação de elementos triangulares.

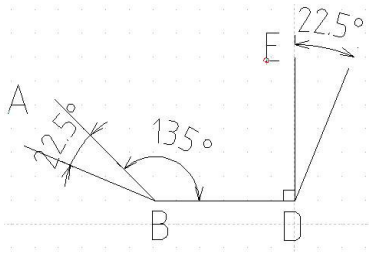

Figura 52 – Segmento ABD entre 135º e 157,5º e segmento BDE entre 90º e 112,5º

3.5.11Segmento ABD entre 112,5º e 135º e Segmento BDE entre 157,5º e 180º

Sendo estes os ângulos formados entre os segmentos ABD e BDE, há duas possibilidades para posicionamento do triângulo definido pelo segmento BDE. O triângulo pode estar ser criado em um vértice côncavo ou em um vértice convexo. No caso de estar posicionado sobre um vértice convexo, existem três formas para se inserir um elemento triangular. Primeiro deve-se verificar o ângulo formado pelos segmentos de reta BC e BC' (ver Figura 53), sendo que C' é o nó C definido a partir dos nós D e E. Caso o ângulo α esteja entre 5º e 30º ou seja maior que 90º, encontra-se um novo ponto médio C'' entre o nó C e o nó C'''. O nó C''' é obtido a partir do segmento AB, onde A é o nó anterior de B na tabela de nós. Se este nó não for válido, então é somente armazenado o nó B na tabela de nós.

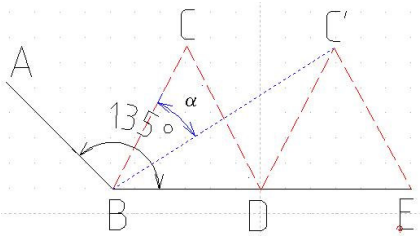

Figura 53 - Ângulo α entre os segmentos BC e BC'

Se o ângulo formado pelos segmentos BC e BC' for maior que 40º, dois elementos são inseridos no domínio utilizando o nó médio C''. O primeiro elemento tem como nós ABC'', e o segundo os nós BDC'', sendo esses armazenados na

tabela de elementos. Para este caso, somente o nó C'' será armazenado na tabela de nós. Caso o nó B seja o primeiro da tabela de nós, o nó A é marcado como utilizado. Caso o ângulo seja menor que 40º, são ligados os nós C''' e D, criando assim um elemento triangular BDC''' (ver Figura 54). Nenhum dos nós neste caso será armazenado na tabela de nós.

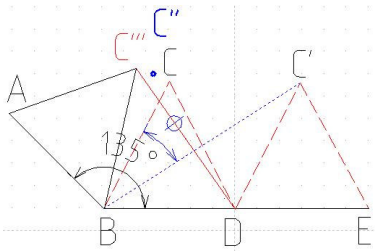

Figura 54 - Triângulo inscrito nos nós B, C''' e D

Dado que o triângulo esteja em um vértice côncavo, então o nó a ser utilizado é o nó C, o qual deve ser validado. Não sendo um nó válido, é somente armazenado o nó B na tabela de nós. Se for válido, um elemento triangular é inserido. Este elemento triangular tem como nós BDC, os quais são armazenados na tabela de elementos. O nó B e o nó C são armazenados na tabela de nós.

A Figura 55 mostra as características em relação aos ângulos formados entre os segmentos ABD e BDE.

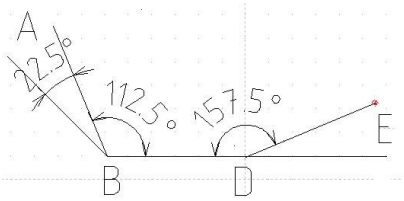

Figura 55 – Segmento ABD entre 112,5º e 135º e segmento BDE entre 157,5º e 180º

3.5.12Segmento ABD entre 112,5º e 135º e Segmento BDE entre 135º e 157,5º

Para esta formação de ângulo, deve ser levado em consideração se o nó anterior foi utilizado ou não. No caso de ser utilizado, e tendo o nó B ou o nó D como vértice côncavo, pode-se desenhar um elemento triangular. Mas para isto, o nó C deve ser válido. Se ele não for válido, o nó B será armazenado na tabela de nós. No entanto, se ele for válido, os nós B e D serão utilizados para a criação de um elemento triangular em conjunto com o nó C, conforme mostra a Figura 56. Estes nós são armazenados na tabela de elementos, e os nós B e C são armazenados na tabela de nós, sendo que o nó C deve ser adicionado como um vértice côncavo.

Se os nós B e D não forem vértices côncavos, uma linha é traçada do nó C' até o nó D, onde C' é o nó C do triângulo inscrito anteriormente, criando assim, um elemento triangular BC'D e armazenando-se os nós B, C' e D na tabela de elementos e nenhum dos nós é armazenado na tabela de nós.

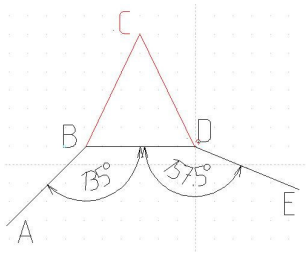

Figura 56 - Triângulo inscrito nos nós B e D

Se o nó anterior não for utilizado, então é necessário identificar o ponto médio C'' entre o nó C e o nó C', o nó C' é gerado a partir do segmento AB. Se esse nó C'' não for válido, o nó B deve ser armazenado na tabela de nós. Agora, se for um ponto válido, dois elementos triangulares são inseridos no domínio. Cada elemento triangular terá os nós A, B e C' e os nós B, C' e D que podem ser vistos na Figura

57. Esses nós são armazenados na tabela de elementos e os nós A e C' são armazenados na tabela de nós.

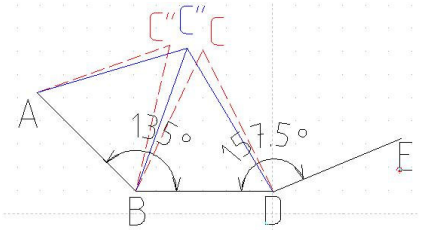

Figura 57 - Triângulos inscritos nos nós A, B e D

Para identificar melhor a necessidade deste tipo de inserção, a Figura 58 mostra os ângulos formados pelos segmentos ABD e BDE.

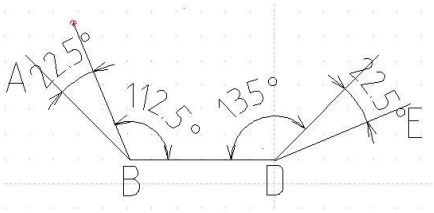

Figura 58 – Segmento ABD entre 112,5º e 135º e segmento BDE entre 135º e 157,5º

3.5.13Segmento ABD entre 112,5º e 135º e Segmento BDE entre 112,5º e 135º

Existem duas opções para inserção de elementos triangulares neste tipo de segmentos. A Figura 59 caracteriza melhor a provável formação dos elementos triangulares.

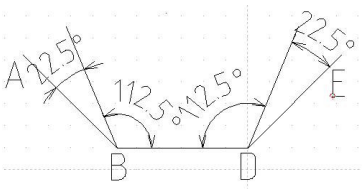

Figura 59 – Segmento ABD entre 112,5º e 135º e segmento BDE entre 112,5º e 135º

Para que uma das opções seja utilizada, é necessário que o nó médio C'' seja validado. O nó C'' é encontrado tendo com base os nós C e C', onde C' é calculado

em relação ao segmento AB. Sendo o nó C'' um nó válido, o ângulo formado pelos segmentos BC e BC' deve ser maior que 35. Com isso, são adicionados três elementos triangulares, com os nós ABC'', BDC'' e DEC'', conforme Figura 60. Todos estes nós são armazenados na tabela de elemento e somente os nós A e C'' são armazenados na tabela de nós.

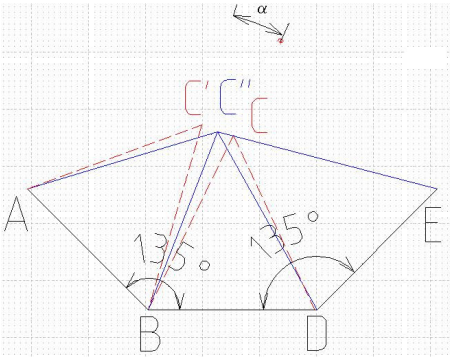

Figura 60 - Triângulos inscritos nos nós A, B, D e E

Caso o ângulo seja inferior a 35º, então é traçada uma linha entre o nó C', e o nó D, criando, assim, o elemento com os nós B, D e C' e nenhum dos nós serão armazenados na tabela de nós.

3.5.14 Segmento ABD entre 100 e 112,5º e Segmento BDE entre 157,5º e 180º

O mais importante para esta formação de ângulos é o ângulo obtido através dos segmentos BC e BC', onde C' é o nó C do triângulo anterior, conforme a Figura 61.

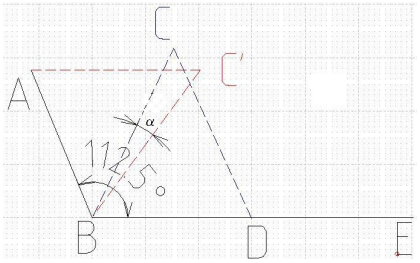

Figura 61 - Ângulo α formado entre os segmentos BC' e BC

Se esse ângulo for maior que 50º, é necessário calcular o ponto médio C''. O ponto médio C'' é o calculado com os nós C e C', sendo o nó C' o nó C do segmento anterior AB. Com o nó C'' validado e o nó A sendo um vértice convexo, os elementos triangulares com os nós ABC'' e BDC'' são criados, conforme a Figura 62. Esses pontos são armazenados na tabela de elementos, e os nós A e C'' são adicionados à tabela de nós. Caso o vértice seja côncavo, o elemento a ser inscrito possui os nós BDC, os quais são colocados na tabela de elementos. Os nós B e C são inseridos na tabela de nós.

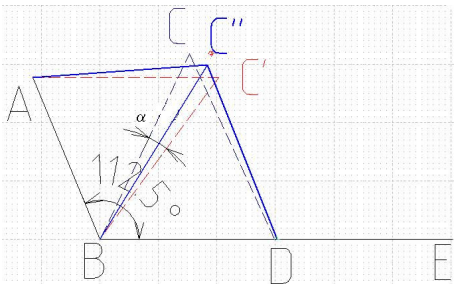

Figura 62 - Triângulos inscritos nos nós A, B e D e ângulo de 180º

No entanto, se o ângulo for menor que 50º, é utilizado o nó C' como base para a construção do elemento, criando-se assim, dois triângulos com os nós ABC' e BDC'. Eles são armazenados na tabela de elementos, não possuindo nós a serem armazenados na tabela de nós.

Os ângulos formados pelos segmentos ABD e BDE podem ser vistos na Figura 63.

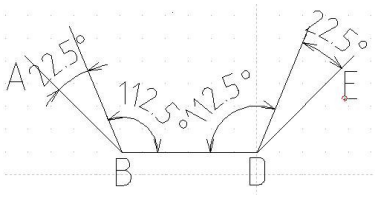

Figura 63 – Segmento ABD entre 112,5º e 135º e segmento BDE entre 100º e 112,5º

3.5.15 Segmento ABD entre 90º e 100º e Segmento BDE entre 157,5º e 180º

Conforme pode ser visto na Figura 64, os ângulos começam a diminuir muito, sendo necessário encontrar outras soluções que sejam adequadas para a geração da malha.

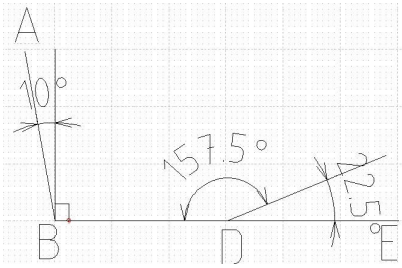

Figura 64 – Segmento ABD entre 90º e 100 e segmento BDE entre 157,5º e 180º

Se o nó C não for válido, então uma linha é traçada do nó A até o nó D, como pode ser visto na Figura 65. Os nós que produzem o elemento triangular são ABD e, por isso, são armazenados na tabela de elementos. Para esta opção, não há inserção de nós na tabela de nós.

Com o nó C válido, podem-se ter duas opções. Se for um vértice convexo, utiliza-se o nó C do triângulo anterior. Traça-se uma linha do nó C até o nó D, criando-se o elemento triangular pelos nós B, D e C. Esses nós são armazenados na tabela de elementos.

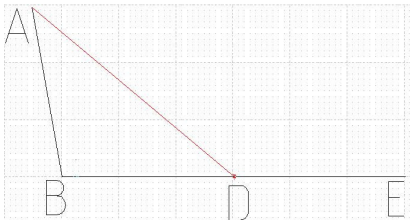

Figura 65 - Triângulo inscrito nos nós A e D

Se for um vértice côncavo, um elemento triangular é gerado pelos nós BDC. Esses nós são armazenados na tabela de elementos, sendo os nós B e C armazenados na tabela de nós.

3.5.16 Segmento ABD entre 90º e 112,5º e Segmento BDE entre 135º e 157,5º

O elemento triangular neste caso é criado tanto para vértices convexos, quanto para os vértices côncavos, sendo o formato dos ângulos apresentado na Figura 66.

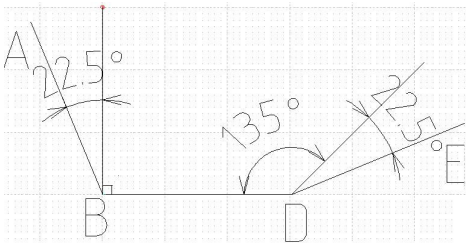

Figura 66 – Segmento ABD entre 90º e 112,5º e segmento BDE entre 135º e 157,5º

No caso de ser um vértice convexo há três possibilidades de inserir um triângulo. Se o ângulo α gerado entre os segmentos de reta BC e BC' não estiver entre 50º e 70º, podem ocorrer duas possibilidades, onde C' é nó C do triângulo inscrito anteriormente. A primeira possibilidade ocorre se o nó B não for o último da lista de nós, e o primeiro nó da fronteira não foi utilizado para gerar um elemento triangular. Neste caso é desenhada uma linha entre o nó C' e o nó D, conforme a Figura 67. Cria-se um elemento triangular com os nós C',B e D, que são

armazenados na tabela de elementos. Os nós não são adicionados na tabela de nós. Na segunda possibilidade, somente o elemento B é armazenado na tabela de nós, bloqueando a geração de nova fronteira.

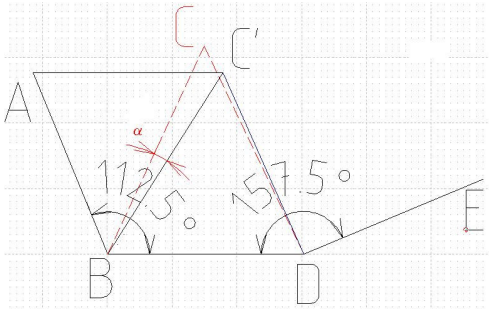

Figura 67 - Triângulo inscrito a partir do nó D

Se o ângulo estiver entre 50º e 70º, o nó B não for o último, e o primeiro nó não foi utilizado para criar um elemento, então, é possível desenhar um elemento triangular sobre o domínio. Desdobra-se em dois casos possíveis, onde no primeiro é verificado se o nó anterior do contorno foi utilizado. Se não foi, é encontrado o ponto médio C'' entre os nós C e C', onde C' é o nó C do triângulo anteriormente inscrito. De posse do nó C'', deve-se buscar o melhor nó com a função BuscaNoProx. Com o nó C'' validado, são inscritos dois elementos, e o primeiro elemento possui os nós C'', A e B, o segundo C'', B, D, conforme mostra a Figura 68. O nó C'' é inserido na tabela de nós, e os nós utilizados para gerar os triângulos são armazenados na tabela de elementos.

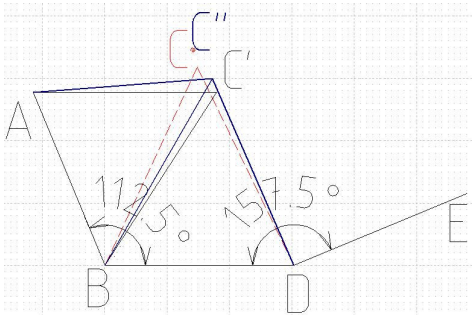

Figura 68 - Triângulos inscritos nos nós A, B e D

No entanto, se for encontrado um vértice côncavo, existirá duas possibilidades de se inserir um elemento triangular. Para isto, deve-se verificar se o nó anterior necessita de complemento. Se não for necessário, então um elemento triangular é inscrito no domínio com os nós C, B e D, como na Figura 69. Armazenam-se esses nós na tabela de elementos, e os nós B e C na tabela de nós. Caso contrário, é necessário utilizar os nós anteriores. Então, o nó C' será o nó C do triângulo anterior, possibilitando desenhar o elemento triangular com os nós C', B e D. Os nós que geraram o triângulo são armazenados na tabela de elementos, não havendo nó a ser armazenado na tabela de nós.

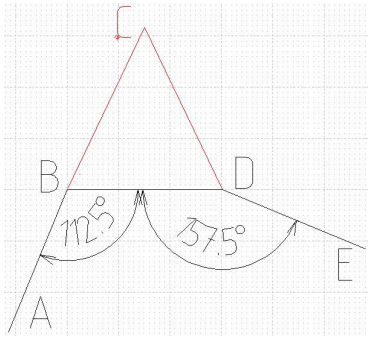

Figura 69 - Triângulo inscrito nos nós B e D e vértice côncavo

3.5.17 Segmento ABD entre 90º e 112,5º e Segmento BDE entre 112,5º e 135º

Para desenhar um triângulo com estes segmentos de reta, é necessário verificar se o nó anterior necessita de complemento. Caso não seja necessário, somente o nó A é armazenado na tabela de nós, não sendo criado nenhum elemento e bloqueando a geração de uma nova fronteira. Se for necessário utilizar o nó anterior, calcula-se o ponto médio C'' pelos nós C e C', onde C' é o nó encontrado do segmento de reta AB, e C é encontrado do segmento de reta BD. Sendo assim, é possível introduzir dois elementos triangulares, o primeiro com os nós C''AB e outro com os nós C''BD, conforme a Figura 70. Esses pontos são

armazenados na tabela de elementos e o nó C'' é armazenado na tabela de nós. Os ângulos que caracterizam a inserção deste elemento são representados pela Figura 71.

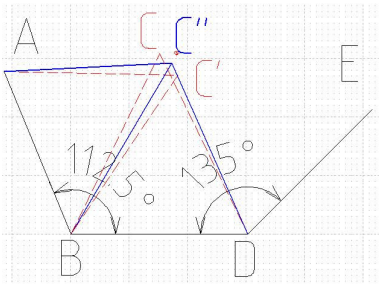

Figura 70 - Triângulos inscritos nos nós A, B e D

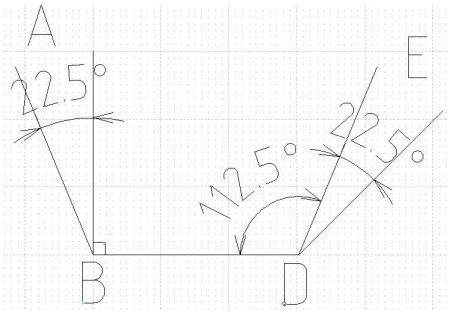

Figura 71 – Segmento ABD entre 90º e 112,5º e segmento BDE entre 112,5º e 135º

3.5.18 Ângulos entre 90º e 112,5º e Próximo entre 90º e 112,5º

A diferença deste algoritmo para o definido no item 3.5.17 é que o ponto médio é calculado duas vezes, no primeiro são levadas em consideração os segmentos AB e BD, obtendo assim o nó C''. O algoritmo utiliza este nó C'' e um novo nó C, sendo este definido pelo segmento de reta DE, para encontrar um novo ponto médio C''. Depois de encontrado o novo nó C'', o restante do algoritmo é o mesmo utilizado no item 3.5.17. A Figura 72 apresenta a formação dos ângulos que possibilitam a introdução deste tipo de elemento triangular.

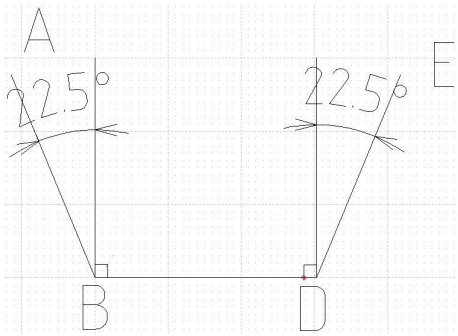

Figura 72 – Segmento ABD entre 90º e 112º e segmento BDE entre 90º e 112,5º

3.5.19 Segmento ABD entre 90º e 112,5º e Segmento BDE entre 0º e 90º

Neste caso, não é calculado o nó C, ou seja, são utilizados somente os nós da tabela de nós. Como os segmentos de reta BD e DE formam ângulos pequenos e/ou retos, como na Figura 73, o elemento triangular é desenhado com os nós B e E, conforme a Figura 74. Os nós BDE são armazenados na tabela de elementos, e o nó B é armazenado na tabela de nós.

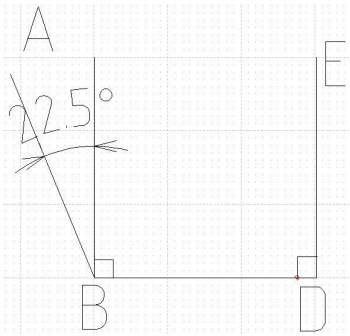

Figura 73 – Segmento ABD entre 90º e 112,5º e segmento BDE entre 0º e 90º

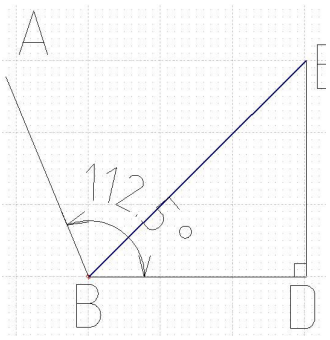

Figura 74 - Triângulo inscrito nos nós B, D e E

3.5.20 Segmento ABD entre 67,5º e 90º e Segmento BDE entre 157,5º e 180º

Para estes tipos de ângulos, são levados em consideração os vértices formados. Ou seja, se o vértice for côncavo, encontra-se a bissetriz em relação aos segmentos de reta AB e BD, porém sendo traçada em sentido contrário. Para encontrar a posição do nó F em relação a bissetriz definida, deve levar em conta a altura de um triângulo eqüilátero. Para isto, utiliza-se a norma do segmento AB para encontrar a posição ideal de F, conforme mostra a Figura 75.

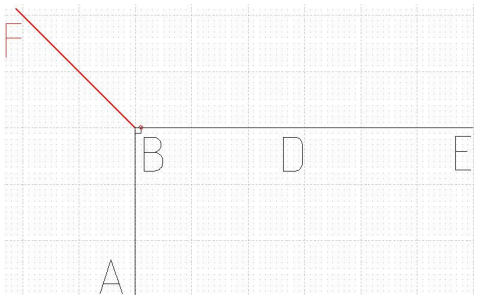

Figura 75 - Bissetriz em relação aos segmentos AB e BD

Após definir o nó F em relação à bissetriz, verifica-se se este nó é válido. Se for válido, um elemento triangular é traçado utilizando os nós C, B e D. O nó C foi encontrado em relação ao segmento de reta BD, como mostra a Figura 76. Na tabela de nós, são armazenados os nós B, F, onde esse é um novo vértice côncavo, e os nós B e C. Os nós C, B e D são armazenados na tabela de elementos. Se o nó

F não for válido, então o nó D é armazenado na tabela de nós, e a fronteira é bloqueada.

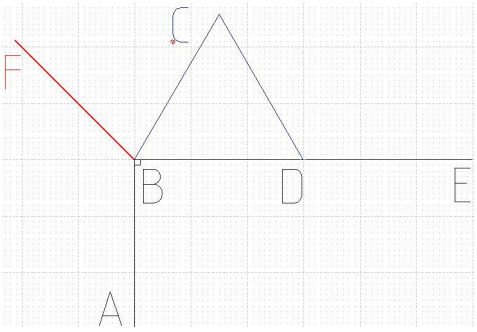

Figura 76 - Triângulo inscrito nos nós B e D e ângulo de 180º

Se for encontrado um vértice convexo, existem três formas de adicionar um elemento triangular. Para isto, é necessário calcular o ângulo α formado pelos segmentos BC e BC', conforme a Figura 77, onde C' é o nó C do triângulo inscrito anteriormente. Se o ângulo for igual a 0º, ou não necessitar complementar um triângulo anterior, dois elementos triangulares serão inscritos.

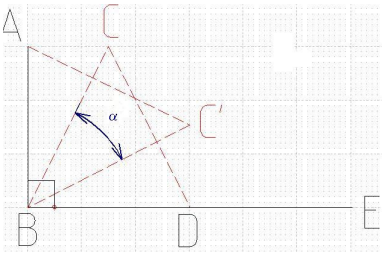

Figura 77 - Ângulo formado pelos segmentos BC e BC'

Para isto, deve-se encontrar o ponto médio C'' entre os nós A e D. Após encontrar o nó C'', o mesmo deve ser validado, ou seja, deve-se verificar se em volta do nó C'' não há outro nó inscrito. Traçam-se então duas retas, uma ligando os nós A e D, e a outra ligando os nós C'' e B, como pode ser visto na Figura 78. Os nós armazenados na tabela de elementos são A, B e C'' e C'', B e D. Os nós A e C'' são adicionados na tabela de nós.

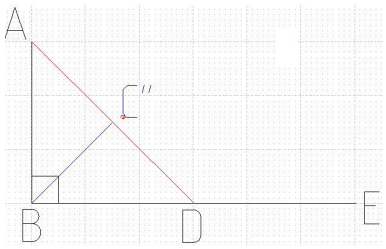

Figura 78 - Triângulos inscritos nos nós A, B e D

Caso o ângulo encontrado esteja entre 5º e 15º, e o nó anterior necessite de complemento, o ponto médio C'' é encontrado a partir dos nós C e C'. O nó C' é definido pelo segmento de reta AB. Com o nó C'', verificam-se os nós próximos. Assim, são definidos dois elementos triangulares, onde o primeiro é formado os nós C'', B e A, e o segundo pelos nós C'', D e B, os quais são armazenados na tabela de elementos. Os nós A e C'' são armazenados na tabela de nós.

Caso não seja um ângulo definido anteriormente, uma linha é traçada entre os nós D e C', onde C' é o ponto C do triângulo anterior. Os nós C',D e B são armazenados na tabela de elementos, não havendo nós para adicionar à tabela de nós.

A Figura 79 mostra as características dos ângulos formados.

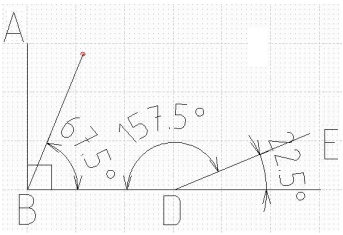

Figura 79 – Segmento ABD entre 67,5º 90º e segmento BDE entre 157,5º e 180º

3.5.21 Segmento ABD entre 50º e 86º e Segmento BDE entre 112,5º e 135º

Nesta situação, uma linha é traçada entre os nós A e D, pois A é o nó anterior de B na tabela de nós, conforme Figura 80.

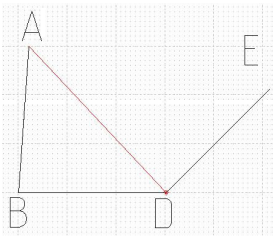

Figura 80 - Triângulo gerado a partir do nó A

Cria-se, então, um elemento triangular com os nós A, B e D, sendo estes nós armazenados na tabela de elementos. Na tabela de nós será armazenado o nó A. Bloqueiam-se assim novas inserções de elementos triangulares. A Figura 81 mostra a formação dos ângulos deste problema.

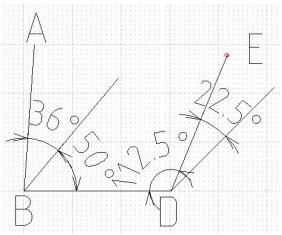

Figura 81 – Segmento ABD entre 67,5º 90º e segmento BDE entre 112,5º e 135º

3.5.22 Segmento ABD entre 67,5º e 90º e Segmento BDE entre 67,5º e 112,5º

Nesta situação, cria-se um elemento triangular com os nós A, B e D. Para isto, é traçada uma linha do nó A até o nó D, semelhante à Figura 80. Esses nós são inseridos na tabela de elementos. Na tabela de nós, inclui-se somente o nó A. A Figura 82 mostra o formato dos ângulos.

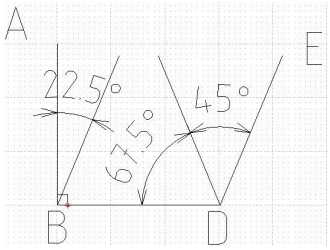

Figura 82 – Segmento ABD entre 67,5º e 90º e segmento BDE entre 67 e 112,5º

3.5.23 Segmento ABD entre 0º e 67,5º e Segmento BDE entre 157,5º e 180º

Este tipo de ângulo é utilizado para vértice côncavos e convexos, como pode ser visto na Figura 83. No caso de vértice côncavo é gerado um ângulo extremamente fechado. Assim, três novos nós são criados a partir dos segmentos de reta AB e BD. O primeiro nó tem como base os segmentos AB e BD, pois por ele é possível traçar a primeira bissetriz, que tem sentido contrário do segmento AB e BD.

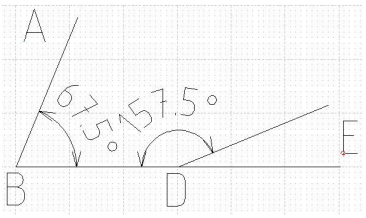

Figura 83 - Ângulos entre 0º e 60º e entre 157,5º e 180º

Com a bissetriz definida é possível definir a posição do nó F em relação ao nó B. Para encontra a posição ideal do nó F é utilizado o tamanho do segmento AB. Esse segmento garante a distância desejada entre os nós B e F, considerando a altura ideal do triângulo eqüilátero. De posse do nó F, o próximo passo é encontrar o nó C que tem como base os nós B e D (ver Figura 84). Sendo C um nó válido, os nós B, F e B, C são armazenados na tabela de nós. Porém, o nó F deve ser armazenado como um vértice côncavo.

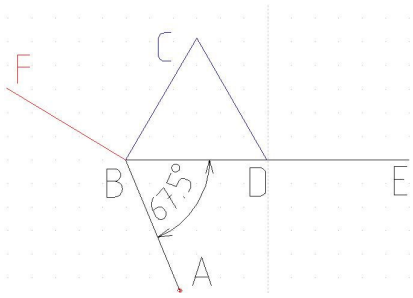

Figura 84 – Nó F a partir da Bissetriz e Triângulo inscrito no nó B

Se o ponto B for um vértice convexo, será traçada uma linha do nó A até o nó D, criando-se um elemento triangular com os nós A, B e D e que serão armazenados na tabela de elementos. O nó A é o único ponto a ser colocado na tabela de nós.

3.5.24 Segmento ABD entre 0º e 67,5º e Segmento BDE entre 0º e 157,5º

Pelas características apresentadas pela Figura 85, é possível construir um único elemento com os nós A, B e D. Neste caso, é traçada uma linha entre os nós A e D. Construída a linha, tem-se então o elemento triangular com os nós A, B e D, os quais são armazenados na tabela de elementos. O nó A é então guardado na tabela de nós.

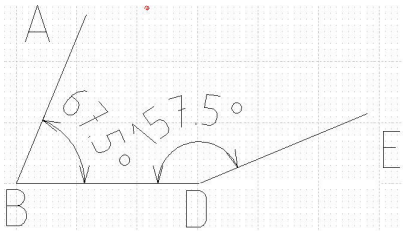

Figura 85 – Segmento ABD entre 0 e 60 e segmento BDE entre 0 e 157,5º

3.6 Definindo uma Nova Fronteira Dentro do Domínio

Após a criação dos elementos triangulares a partir da fronteira aberta, deve-se então definir uma nova fronteira, a qual se constituirá em um novo contorno. Este novo contorno será uma nova fronteira e assim sucessivamente até o preenchimento

total do domínio. Para que isto ocorra, é necessário ter a tabela de nós atualizada com os últimos nós inseridos no domínio. Estes nós, ou seja, todos os nós que foram utilizados para criar os triângulos na fase anterior. Com esta informação, é necessário verificar, em relação ao segmento ABD, qual o formato do triângulo inserido, como pode ser visto na Figura 86. Dependendo do tipo de triângulo cria-se ou não um novo elemento triangular. Para saber o tipo de triângulo, basta utilizar o algoritmo de MELKMANN. Este retornará um valor positivo (Figura 86 (c)), ou negativo (Figura 86 (b)), ou ainda zero (Figura 86 (a)). Somente será criado um novo elemento triangular caso o valor seja negativo, como mostrada em vermelho na Figura 86 (b).

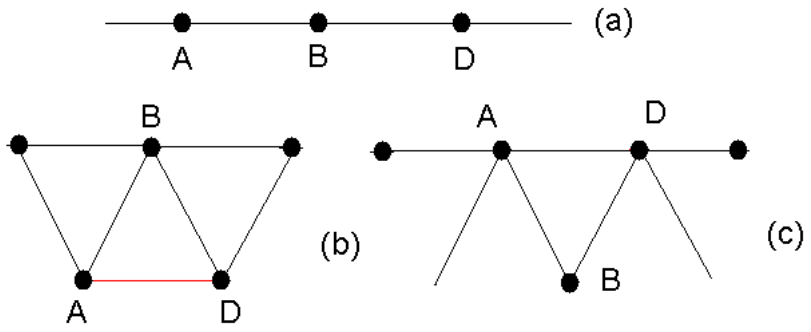

Figura 86 – Definindo uma nova fronteira

Este novo elemento triangular deve ser armazenado na tabela de elementos. Ao passar por todos os nós, é então dito que uma nova fronteira foi definida. Em alguns casos esta geração da nova fronteira cria elementos que podem ser preenchidos. Ou seja, cria possíveis elementos triangulares que podem fazer parte da nova fronteira. Para isto, é necessário verificar novamente a tabela de nós (vide Figura 87).

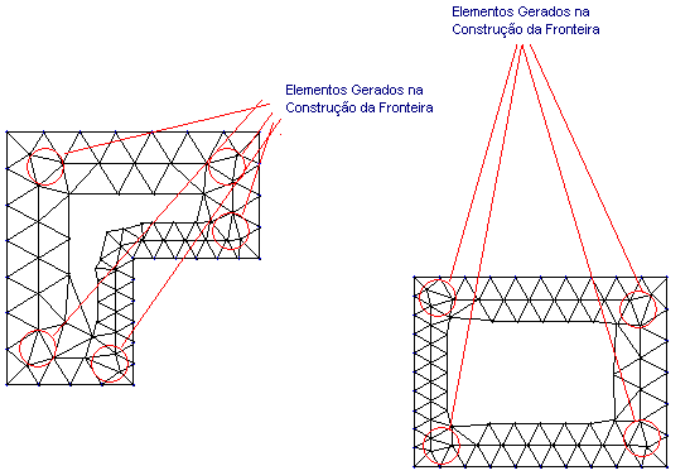

Figura 87 - Elementos gerados na construção da nova fronteira.

O algoritmo de criação de nova fronteira é apresentado no Anexo B.

3.7 Número de Fronteiras

O número de fronteiras vai depender do tamanho do domínio original, pois, a cada nova fronteira, é calculada a área do domínio. Sendo o domínio um polígono irregular, tem-se então a seguinte fórmula para cálculo da área:

$$Área = 1/2\left(\begin{vmatrix} x_1 & x_2 \\ y_1 & y_2 \end{vmatrix} + \begin{vmatrix} x_2 & x_3 \\ y_2 & y_3 \end{vmatrix} + ... + \begin{vmatrix} x_n & x_1 \\ y_n & y_1 \end{vmatrix}\right) \qquad (12)$$

que pode ser escrita da seguinte forma:

$$Área = 1/2\sum_{i=0}^{n-1}\left(x_i * y_{i+1} - y_i * x_{i+1}\right) \qquad (13)$$

Esta área é utilizada para garantir a quantidade de fronteiras em um determinado domínio. O algoritmo somente irá parar quando a área for negativa ou igual à zero, pois nestes casos, não haverá mais área útil para se construir uma nova frente. O algoritmo abaixo mostra como calcular a área do polígono irregular.

```
//---------------------------------------------------------------------------
// equação para calcula da área do polígono irregular
// área =  1/2 * somatório de i = 0 até o número de vértices onde,
// (x(i)*y(i+1) - y(i)*x(i+1)
double TMDIChild::AreaPoliIrre(int m, int n){
  MyPoint P0,Pos;
```

```
  double soma;
  int cont = 0;

  soma = 0;

  for (int i = 0; i < n; i++){
    P0.x = StrToFloat(StringGrid2->Cells[m+1][i]);
    P0.y = StrToFloat(StringGrid2->Cells[m+2][i]);
    Pos = buscaproximo(i,n,m);
    soma  = soma + ((P0.x*Pos.y)-(P0.y*Pos.x));
  }

  soma = soma / 2.0;

  return soma;
}
```

3.8 Parando a Inserção de Elementos

Existem duas formas para encerrar o algoritmo do avanço da fronteira. Na primeira, quando a área do domínio for menor ou igual a zero, o algoritmo de geração de novas fronteiras é concluído. A segunda forma de encerrar o algoritmo se dá no momento de ativação de um bloqueio de fronteira. Se esse bloqueio foi ativado, então o algoritmo de geração de novas fronteiras para. Ao parar de gerar novas fronteiras por qualquer um dos motivos acima mencionados, é ativado o algoritmo da costura. Esse algoritmo serve para preencher os vazios do domínio com elementos triangulares, considerando-se os nós armazenados. Para definir um elemento triangular neste caso, deve-se verificar o ângulo formado entre os segmentos AB e BD (ângulo um), AB e AD (ângulo dois) e DA e DB (ângulo três).

Sendo assim, para garantir a inserção de um elemento triangular, faz-se necessário que os ângulos formados possuam os seguintes valores, onde o ângulo um deve estar entre 30º e 120º, o ângulo dois entre 26º e 105º, e o ângulo três entre 25º e 110º. Se os ângulos encontrados estiverem compreendidos entre os ângulos definidos, então um elemento é inscrito e armazenado na tabela de elementos. Nenhum nó neste caso é armazenado na tabela de nós. Somente será adicionado um nó A na tabela de nós se os ângulos não estiverem compreendidos entre os

valores definidos acima. O algoritmo encerra quando não for mais possível adicionar elementos triangulares, conforme algoritmo descrito no Anexo B.

Se ao final do algoritmo definido acima o domínio não estiver completamente preenchido, o quê é verificado pela quantidade de nós restantes na tabela de nós, faz-se necessário completar o vazio. Se existirem mais de três nós, o algoritmo da costura final é ativado. A diferença desse algoritmo para o algoritmo da costura é dada pelos ângulos utilizados para gerar um elemento triangular. Neste caso, são utilizados os ângulos entre 12º e 120º para o ângulo um, entre 20º e 120º para o ângulo dois, e entre 20º e 125º para o ângulo três. O restante do algoritmo segue o mesmo procedimento do algoritmo da costura. A partir de então se tem o domínio preenchido completamente. As malhas obtidas através deste algoritmo são apresentadas no Capítulo 5.

Capítulo 4

4 Suavização

Para garantir uma boa qualidade da malha gerada, foi implementado um método de melhoria conhecido como suavização Laplaciana, e neste caso não foi implementada o refino da malha. A suavização Laplaciana, é largamente utilizada para melhoria dos elementos de uma malha, e com esta finalidade a melhoria acorre pelo reposicionamento do nó central, levando em consideração os nós vizinhos. Seguindo a formulação matemática apresentada por (Zhou, 2000), esta formulação foi implementada dentro da função Suavização.

A função Suavização é dividida em três partes, a primeira monta a tabela conectividade, pois conforme apresentado no livro de (Chandrupatla, 1997) a tabela conectividade contém somente o número do elemento e os nós relacionados ao elemento. Neste caso, a tabela conectividade possui a seguinte característica: Índice, nó 1, nó 2 e nó 3 por trabalhar somente com elementos triangulares conforme exemplo na Tabela 4.

Tabela 4 - Tabela conectividade

Elemento	Nó 1	Nó 2	Nó 3

Porém, para completar a tabela conectividade é necessária a tabela de nós, a qual tem o formato identificado pela Tabela 5 conforme (Chandrupatla, 1997) com a adição do campo tipo. Este campo é utilizado para identificar se o nó faz parte do contorno ou ele é interno ao domínio.

Tabela 5 - Tabela de nós

Nó	Posição em X	Posição em Y	Tipo

Com as informações nas duas tabelas, a função Suavização() busca identificar todos os relacionamentos de um determinado nó, ou seja, todos os nós vizinhos, criando assim a tabela conectividade. Para esta tabela não é possível identificar com exatidão o tamanho do vetor a ele relacionado, pois um nó pode estar relacionado a dois ou mais nós vizinhos. A tabela tem a sua estrutura definida conforme a “Tabela 6”.

Tabela 6 - Tabela conectividade auxiliar

Nó	Qtde nós vizinhos	Nó 1	....	Nó i

Com a Tabela 6 preenchida, é possível então, fazer o cálculo da suavização Laplaciana e atualizar os nós na Tabela 5, conforme pode ser observado na função Suavização(). Após completar o cálculo do reposicionamento do nó central, utiliza-se a Tabela 6 e a Tabela 5 para desenhar novamente o domínio atualizado, o qual pode ser visto pela Figura 88.

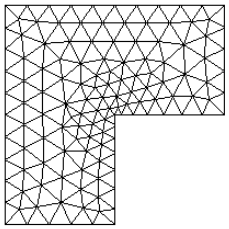

Figura 88 - Domínio após a suavização Laplaciana

Abaixo esta listado o algoritmo da suavização Laplaciana.

```
TMDIChild::Suavizacao1Click(TObject
*Sender)
{
  MyPoint P1,P2;
  //int contlap;
  int x,y,l,m,n,o;
  int ele1, ele2;
  int linhas;
  fSuavi->ShowModal();
  ReDraw();
  nIteracao = fSuavi->nPoint;
  // Apaga figura
  ApagarTudo1Click(Sender);
  linhas = 0;
```

```
  // verifica nos iguais
  // for para todos os nodos dentro da
tabela de nodos
  //-----------------------------------------------
-
  // montagem da matriz conectividade
auxiliar
  //-----------------------------------------------
-
  for (int k = 0 ; k < contlinh; k++){
    ele1 = StrToInt(StringGrid5-
>Cells[0][k]);
    StringGrid4->RowCount = k+1;
    StringGrid4->Cells[0][k] = k+1;

    StringGrid4->Cells[1][k] =
StringGrid5->Cells[0][k];
    int col = 3;
    int status = 0;
    ele2 = 0;
    for (int j = 1 ; j <= 3; j++){
      for (int i = 0 ; i < contlinhaux; i++){
        if (StrToInt(StringGrid6->Cells[j][i])
== ele1){
          status = 1;
          if (j == 1){
            if (EncontraNo(k,2,i,col) == 0) {
              StringGrid4->Cells[col][k] =
StringGrid6->Cells[2][i];
              col++;
              ele2++;
            }
            if (EncontraNo(k,3,i,col) == 0) {
              StringGrid4->Cells[col][k] =
StringGrid6->Cells[3][i];
              col++;
              ele2++;
            }
          }
          if (j == 2){
            if (EncontraNo(k,1,i,col) == 0) {
              StringGrid4->Cells[col][k] =
StringGrid6->Cells[1][i];
              col++;
              ele2++;
            }
            if (EncontraNo(k,3,i,col) == 0) {
              StringGrid4->Cells[col][k] =
StringGrid6->Cells[3][i];
              col++;
              ele2++;
            }
          }
          if (j == 3){
            if (EncontraNo(k,1,i,col) == 0) {
              StringGrid4->Cells[col][k] =
StringGrid6->Cells[1][i];
              col++;
              ele2++;
            }
            if (EncontraNo(k,2,i,col) == 0) {
              StringGrid4->Cells[col][k] =
StringGrid6->Cells[2][i];
              col++;
              ele2++;
            }
          }
          StringGrid4->Cells[2][k] =
IntToStr(ele2);
        }
      }
    }
    if (status == 0) StringGrid4-
>Cells[2][k] = 0;
  }
  //-----------------------------------------------
-
  // calculo do nos por suavizacao
laplaciana
  //-----------------------------------------------
-
  for (int k = 0; k < nIteracao; k++){
    for (int i = 0; i < contlinh; i++){
      P1.vertice =
StrToFloat(StringGrid5->Cells[4][i]);
      int totalx = 0;
      int totaly = 0;
      if (P1.vertice != 0 &&
StrToInt(StringGrid4->Cells[2][i]) != 0){
        for (int l = 0; l <=
StrToInt(StringGrid4->Cells[2][i]); l++){
          for (int t = 0; t < contlinh; t++){
            if (StringGrid4->Cells[l+3][i] ==
StringGrid5->Cells[0][t]){
              P2.x =
StrToFloat(StringGrid5->Cells[1][t]);
              P2.y =
StrToFloat(StringGrid5->Cells[2][t]);
              // acumula valor de x e y
              totalx = totalx + P2.x;
              totaly = totaly + P2.y;
              break;
            }
          }
```

```
        }
        // encontra novo valor de x e y
        totalx = (int) (totalx /
StrToInt(StringGrid4->Cells[2][i]));
        totaly = (int) (totaly /
StrToInt(StringGrid4->Cells[2][i]));
        //atualiza o valor de x e y
        StringGrid5->Cells[1][i] =
IntToStr(totalx);
        StringGrid5->Cells[2][i] =
IntToStr(totaly);
      }
    }
  }
  // busca o no principal na tabela de
nos
  //-----------------------------------------------
-
  // criaçao dos elementos a partir da
matriz conectividade
  // e da tabela de nós
  //-----------------------------------------------
-
  for (int i = 0; i < contlinh; i++){
    P1.x = StrToFloat(StringGrid5-
>Cells[1][i]);
    P1.y = StrToFloat(StringGrid5-
>Cells[2][i]);

    for (int l = 0; l < StrToInt(StringGrid4-
>Cells[2][i]); l++){
      for (int t = 0; t < contlinh; t++){
        if (StringGrid4->Cells[l+3][i] ==
StringGrid5->Cells[0][t]){
          P2.x = StrToFloat(StringGrid5-
>Cells[1][t]);
          P2.y = StrToFloat(StringGrid5-
>Cells[2][t]);
          DeseLinh(P1, P2);
          break;
        }
      }
    }
  }
}
```

Capitulo 5

5 Resultados Obtidos

Este capítulo apresenta os resultados obtidos com o programa desenvolvido. O programa possui recursos para gerar malhas uniformes e malhas graduais. Os resultados estão divididos em quatro tipos, o domínio quadrado, o domínio em L e o domínio retangular e um caso particular, o domínio V-Notch gerado somente com uma malha uniforme. Com isto, para cada uma das categorias foi gerado um tipo de malha gradual e uniforme.

No caso da geração de uma malha uniforme, o programa utiliza um tamanho único de elemento a ser inserido no domínio, tamanho este que está agregado ao código do programa.

Para o segundo caso, implementou-se um algoritmo que permite geração de malhas graduais em diferentes domínios. Conforme mencionado neste trabalho, vários problemas da física-matemática, notadamente aqueles com regiões de gradientes elevados, necessitam de malhas graduais para seus modelamentos. A graduação dos elementos sobre o domínio depende da escolha de tamanhos para os vértices do mesmo. Designa-se interativamente um código numérico para cada vértice, cada código estando associado a um tamanho diferente de elemento, e, a partir desta informação, o algoritmo gera a malha gradual.

No decorrer deste capítulo alguns resultados obtidos com o algoritmo de geração de malha gradual. Observam-se alguns problemas de transição entre elementos na região central dos domínios, que é onde o algoritmo do método do avanço da fronteira faz o fechamento da malha. A solução deste problema depende

de uma suavização da malha através do reposicionamento de seus nós. Um algoritmo de suavização não foi implementado.

Os resultados do domínio quadrado são apresentados abaixo. Onde em primeira instância, pode ser observado na Figura 89 (malha 1) que o domínio quadrado foi preenchido com uma malha uniforme com 148 elementos. Pode-se observar a geração de triângulos eqüiláteros ou quasi-equiláteros em todo o domínio.

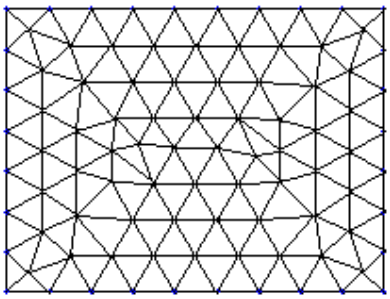

Figura 89 – Malha 1 - Preenchimento do Domínio com Malha Uniforme

A Figura 90 mostra uma malha gradual (malha 2) sobre o domínio retangular contendo 214 elementos. O tamanho dos elementos aumenta da aresta esquerda para a resta direita, sendo a transição de tamanho entre elementos bastante aceitáveis, não possuindo elemento muito grande ou muito pequeno em relação próximo ao elemento definido. A Figura 91 mostra uma malha gradual (malha 3) mais refinada (contendo 448 elementos) sobre o mesmo domínio. Apesar de se obter a graduação desejada, ocorre um defeito de conclusão, com elementos muito grandes próximos ao centro da malha, evidenciando a necessidade de se aplicar algum tipo de suavização para correção da mesma. A Figura 92 mostra uma malha gradual (malha 4) sobre o mesmo domínio, contendo 313 elementos, com uma graduação mais acentuada que as anteriores. Ao contrário da malha anterior, a transição de tamanho dos elementos está adequada, pois a evolução do tamanho

dos elementos não aparenta distorções significativas. Para a malha 2 (Figura 90) foi utilizado o algoritmo da suavização, criando assim uma nova malha gradual, o que pode ser visto na Figura 93. Neste caso, o que se observa é uma melhoria significativa no formato dos elementos mais internos próximos à área de costura.

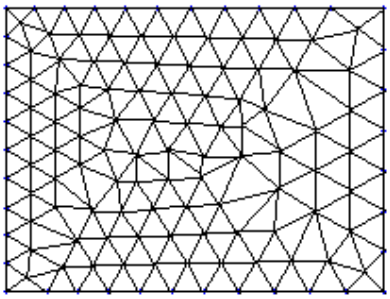

Figura 90 - Malha 2

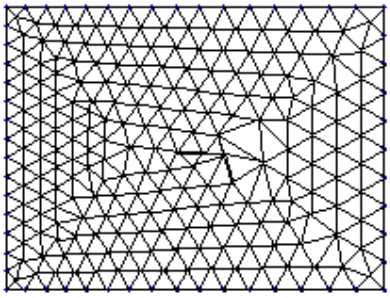

Figura 91 - Malha 3

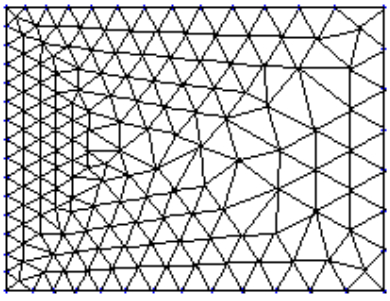

Figura 92 - Malha 4

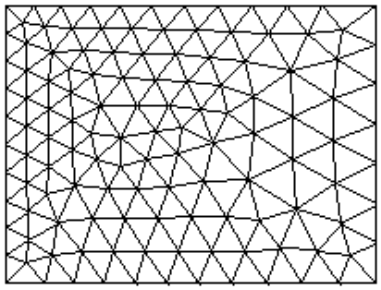

Figura 93 – Malha 2 após a suavização

Para o domínio em L os resultados podem ser avaliados a seguir. A Figura 94 (malha 5) mostra uma malha uniforme para um domínio em forma de L contendo 126 elementos. As Figuras 95 e 96 apresentam malhas graduais (malha 6 e malha 7) com 216 e 257 elementos, respectivamente, sobre o domínio em forma de L. O artigo de (LADEVEZE, COFFIGNAL & PELLE, 1986) apresenta uma malha sobre um domínio semelhante ao dessas figuras, mostrado abaixo na Figura 97 e aqui definido como malha 8. A malha 8 possui 182 elementos e é considerada como uma malha ótima pelos autores, pois se utilizou o refino adaptativo da malha com um erro prescrito de 0,05 e o erro computado foi de 0,052. Nota-se uma variação no tamanho dos elementos, porém não de forma ordenada. Nas malhas 6 e 7, construídas pelo presente algoritmo, houve a preocupação de se gerar uma malha em que o tamanho dos elementos decrescesse em direção à reentrância, como que simulando um problema de concentração de tensões. Este objetivo é atingido através dos resultados mostrados, no caso da malha 6 foi utilizado o algoritmo da suavização (ver Figura 98). Mostrando assim, que, o reposicionamento dos nós já muda significativamente a malha final diminuindo também as transições bruscas de tamanho de elementos.

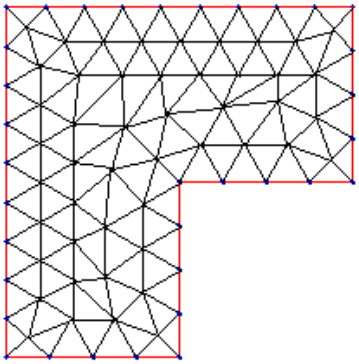

Figura 94 - Malha 5 – Domínio em forma de L

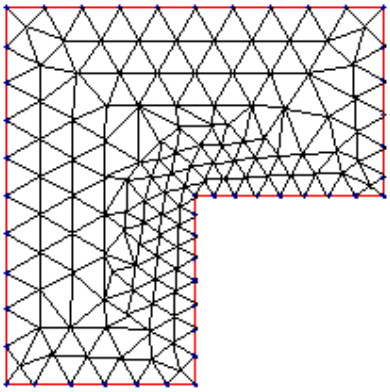

Figura 95 - Malha 6

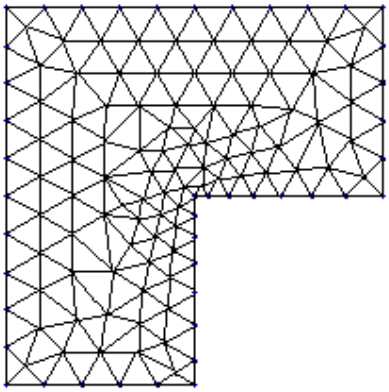

Figura 96 - Malha 7

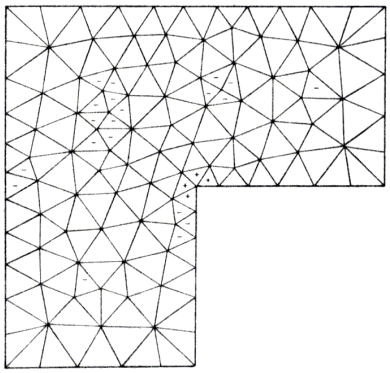

Figura 97 - Malha 8 - Malha em Domínio em L - (LADEVEZE, COFFIGNAL & PELLE, 1986).

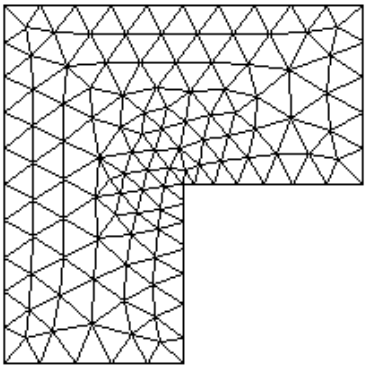

Figura 98 – Malha 6 após a suavização Laplaciana

Para o domínio retangular foram geradas duas malhas, uma uniforme e uma gradual. A Figura 99 (malha 9) mostra uma malha uniforme para domínios retangulares com 148 elementos. Pode-se observar a geração de triângulos eqüiláteros ou quasi-equiláteros por todo o domínio. A Figura 100 mostra uma malha (malha 10) sobre um domínio retangular contendo 293 elementos, com aumento gradual do tamanho desses elementos da aresta esquerda para a direita. Devido à falta de uma suavização, elementos da região central da malha apresentam sérias distorções.

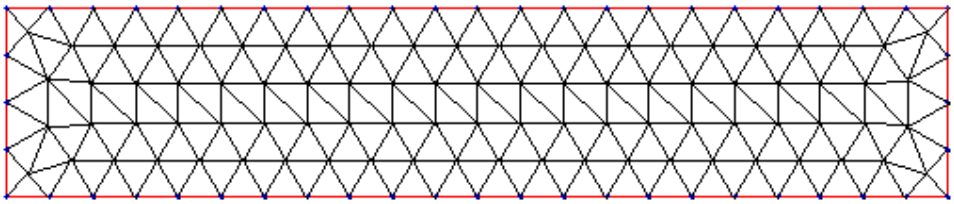

Figura 99 - Malha 9 – Domínios Retangulares

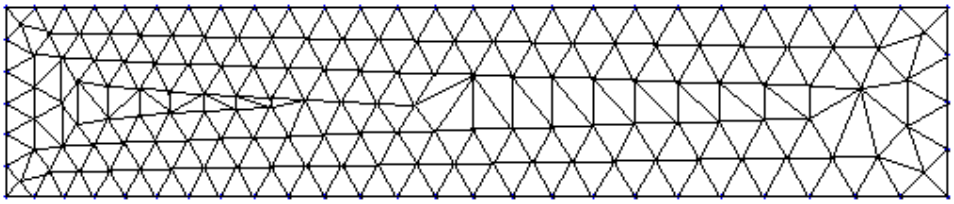

Figura 100 - Malha 10

O último caso é apresentado pela Figura 101 mostra uma malha uniforme para um domínio com um entalhe em V contendo 226 elementos. Onde se podem identificar problemas de fechamento. Observa-se na região central, próxima ao entalhe, a presença de elementos maiores que os demais elementos da malha, que também pode ser parcialmente resolvido utilizando uma suavização da malha. Na Figura 102, observa-se que os problemas levantados tiveram uma melhoria significativa após a suavização.

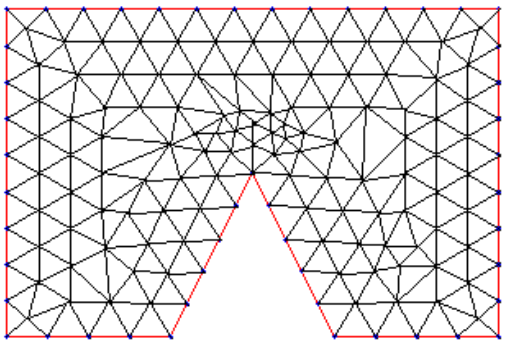

Figura 101 - Malha 11 – Domínio V-Notch

Algumas malhas graduais apresentam bruscas transições de tamanho de elemento. Como isso não ocorre em todas as malhas, conclui-se que o problema é

dependente da graduação adotada para gerar a malha, pois como pode ser observado nas Figuras 99 e 100, quanto menor for o tamanho do elemento maior é à possibilidade de possuir elementos triangulares não eqüiláteros no centro do domínio. Parece certo que, a relação entre as dimensões dos elementos e as dimensões do domínio também influi nesse resultado. Mesmo que outros fatores não tenham influência sobre este resultado, parece ser difícil, se não impossível, de se definir uma estratégia que evite a ocorrência do problema, tornando a suavização de malha um recurso essencial.

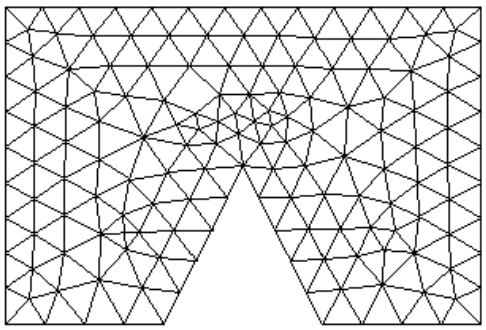

Figura 102 – Malha 11 após a suavização Laplaciana

Capítulo 6

6 Conclusão e Sugestões

O objetivo principal desta dissertação foi o desenvolvimento de um programa para geração de malhas em domínios bidimensionais. Da revisão de literatura realizada, concluiu-se que o método do avanço da fronteira é o mais simples de ser implementado e também o mais robusto. Assim, optou-se pela adoção desse método para a construção do programa.

Cabe comentar que uma escolha anterior havia sido feita. Durante o curso da revisão bibliográfica, um método novo, denominado circle packing chamou a atenção pela promessa de permitir a geração de uma primeira malha muito próxima de uma configuração ótima para problemas que exigem variação gradual no tamanho dos elementos (por exemplo, problemas de trinca). Implementou-se um algoritmo para o circle packing, seguindo-se as orientações da literatura, inclusive aproveitando-se do algoritmo para geração do cirlce packing fornecido por um dos autores (BERN, MITCHELL e RUPPERT, 1995). Infelizmente, não houve sucesso na implementação do método, o quê pode significar que o circle packing ainda carece de desenvolvimento.

O método do avanço da fronteira gera a malha com a inserção de elementos a partir do contorno do domínio, concluindo-a com a inserção de elementos no miolo do mesmo. O algoritmo desenvolvido parte deste princípio e está implementado para gerar malhas de elementos triangulares, buscando, na medida do possível, inserir triângulos eqüiláteros.

Aprendeu-se que o desenvolvimento de um algoritmo para geração de malhas não é uma tarefa trivial, necessitando-se aplicar noções de geometria analítica e

geometria descritiva, além de recursos da computação gráfica. Alguns conceitos de geometria necessários que foram implementados são: bisecção de segmentos de reta, intersecção de retas, rotação de segmentos de reta, norma de vetores, distância de um ponto a uma reta, entre outros.

Encontrou-se dificuldade também na construção de um algoritmo geral para o método do avanço da fronteira, ou seja, que funcione para qualquer domínio, e que gere elementos com tamanhos e orientações diferentes. Outra particularidade que impõe dificuldades é a capacidade de geração de malhas graduais, ou seja, aquelas no qual o tamanho dos elementos varia de acordo com determinadas regras. Problemas como distorção excessiva de elementos e descontrole sobre o tamanho dos mesmos no fechamento da malha foram encontrados na geração de malhas graduais, como mostrado no capítulo 5.

O programa desenvolvido limita-se a gerar malhas com elementos triangulares e somente sobre domínios planos mais simples, ou seja, aqueles sem arestas curvas e sem orifícios. Além disso, um algoritmo de suavização da malha foi implementado, mostrando que o simples reposicionamento de nós já melhora sensivelmente a qualidade da malha. Apesar destas limitações, esta dissertação atinge os objetivos propostos no capítulo 1, que são adquirir conhecimento sobre geração de malhas e construir um programa para geração de malhas, visando o seu acoplamento a um programa doméstico de elementos finitos.

As limitações do programa mencionadas acima indicam a continuação natural deste trabalho. Primeiramente, deve-se buscar aprimorar o algoritmo de geração de malhas graduais de modo a evitar ou minimizar os problemas de distorção encontrados.

Entre os recursos adicionais a serem implementados, o primeiro é a capacidade de gerar malhas em domínios de geometria arbitrária, incluindo aqueles com arestas curvas e reentrâncias diversas. Deve-se também implementar a capacidade de gerar malhas em domínios com orifícios internos. Por fim, seria desejável implementar no programa a capacidade da realização de refinos localizados, o quê permitiria a modelagem de importantes problemas de engenharia.

Referências Bibliográficas

BATHE, K.J. Finite Element Procedures. Prentice Hall, 1996.

BARBIERI, R. GerTri – Gerador de Triângulos. Tese para obtenção do título de Professor Titular. Pontifícia Universidade Católica do Paraná. 2000.

BARSOTI, L. Geometria Analítica e Vetores. Editora Artes Gráficas Unificado, 3ª Edição, 1984.

BERN, M. e EPPSTEIN, D. Quadrilateral Meshing by Circle Packing. International Journal of Computational Geometry & Applications, vol 10, pp 347-360:2000.

BERN, M. e MITCHELL, S. e RUPPERT, J. Linear-size Nonobtuse Triangulation of Polygons. Discrete & Computational Geometry, vol 14, pp 411-428:1995.

BERN, M e PLASSMANN, P. Mesh Generation. Handbook of Computational Geometry, Elsevier Science, 1999.

BOLDRINI, J.L. et.all. Álgebra Linear. Editora Harbra, 3ª Edição, 1996.

CAVALCANTE NETO, J.B. e CARVALHO, M.T. e MARTHA,L.F. Um Algoritmo para Geração de Malha não-Estruturada Tetraédrica para Regiões Arbitrárias. 21st Iberian Latin American Congress on Computational Methods in Engineering, vol 3, pp 11.1-11.21:2000.

CHANDRUPATLA, T.R., & BELEGUNDU, A.D.. Introduction to Finite Elements in Engineering. Prentice Hall, Second Edition, 1997.

CHENG,B. e TOPPING,B.H.V. Improved Adaptive Quadrilateral Mesh Generation Using Fission Elements. Advances in Engineering Software, vol 29, pp 733-744:1998.

DREYER. D.R. e OVERTON, M.L. Two Heuristics for the Steiner Tree Problem. Journal of Global Optimization, vol 31(1), pp 95-106:1998.

FANCELLO, E.A. e SALGADO, A.C. e FEIJÓO, R.A. Aranha: Generador de Malhas em 2D para Elementos Finitos Triangulares de 3 e 6 nós. XI Congresso Ibero Latino Americano sobre Métodos Computacionais para Engenharia, Rio de Janeiro, Brasil, 2:983-996, 1990º.

FANCELLO, E.A. Análise de Sensibilidade, Geração Adaptativa de Malhas e o Método de Elementos Finitos na Otimização de forma em Problemas de Contato e Mecânica da Fratura. Tese submetida ao corpo docente da Coordenação do PPGE da UFRJ, Rio de Janeiro, 1993.

FEITOSA, M.O. Cálculo Vetorial e Geometria Analítica – Exercícios Propostos e Resolvidos. Editora Atlas, 4ª Edição, 1996.

FELKEL, P e OBDRZÁLEK, S. Straight Skeleton Implementation. 14th Spring Conference on Computer Graphics (SCCG'98), pp 210-218:1998.

FIGUEIREDO, L. H. e DE CARVALHO, M.T.M. Geração Auto-Adaptativa de Malhas com Sistemas de Partículas. Anais do IX SIBGRAPI, pp 335-336: 1996.

FOLEY, J.D. e VAN DAN, A. e FEINER, S.K. e HUGHES, J.F. Computer Graphics – Principles ad Practice. Addison-Wesley, 1996.

GEORGE, P.L. Automatic Mesh Generation – Application to Finite Element Methods. John Wiley & Sons, 1991.

LADEVEZE P., COFFIGNAL G., PELLE J.P. Accuracy of Elastoplastic and Dynamic Analysis. BABUSKA, I.- ZIENCKIEWICZ, O.C.- GAGO, J., OLIVEIRA, E.R. de, A., Accuracy Estimates and Adaptative Refinements in Finite Element Computations, pp 181-204, John Wiley, 1986.

LI, X.Y. e TENG, S.H. e ÜNGÖR, A. Biting: Advancing Front Meets Sphere Packing. International Journal for Numerical Methods in Engineering, vol 49, pp 61-81:2000.

LISEIKIN, V.D. Grid Generation Methods. Springer, 1999.

LOBER, R.R. e TAUTGES, T.J. e CAIRNCROSS, R.A. The Parallelization of in Advancinf-front, All-quadrilateral Meshing Algorithm for Adaptive Analysis. 4th International Meshing Roundtable, Sandia National Laboratories, pp 59-70:1995.

MARTHA, L.F. e CAMPOS, J.L. e CAVALCANTE NETO, J. On the Use of Finite Elements in gOcad. 21st GOCAD Meeting, vol 1, pp 1-12:2001.

MCALLISTER, M. e KIRKPATRICK, D e SHOEYINK, J. A Compact Peicewise-Linear Voronoi Diagram for Convex Sites in the Plane. IEEE Symposium on Foundations of Computer Science, pp 573-582,1993.

MELKMANN, A.A. On-line construction of the convex hull of a simple polyline. http://geometryalgorithms.com/Archive/algorithm_0203/algorithm_0203.htm#Pseudo-Code: Melkman Algorithm. Último acesso em 23/10/2003.

QUADROS, W.R. e RAMASWAMI, K. a PRINZ, F.B. e GURUMOORTHY, B. LayTracks: A New Approach to Automated Quadrilateral Mesh Generation using MAT. 9th International Meshing Roundtable – Sandia National Laboratories. Pp 239-250:2000.

REES, M. Combinig Quadrilateral and Triangular Meshing Using the Advancing Front Approach. 6th IMR, pp 337-348:1997.

REIS, H. L. e MAGALHÃES, J. M. e RUTHNER, M. P. Geração de Malhas Triangulares em 2D. http://www.inf.puc-rio.br/~heloreis/#Qualidade. Último acesso em 04/11/2003.

SCHÖBERL, J..NETGEN - An Advancing Front 2D/3D - Mesh Generator Based on Abstract Rules. Computing and Visualization in Science, vol 1,pp 41-52, 1997.

SHEWCHUK, J.R. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. First Workshop on Applied Computational Geometry - ACM, pp 124-133:1996.

TRISTANO, J.R. e OWEN, S.J. e CANANN, S.A. Advancing Front Surface Mesh Generation in Parametric Space Using a Riemannian Definition. 7th International Meshing Roundtable, pp 429-445:1998.

Zhou, T. & Shimada, K.,Na. Angle-Based Approach to Two-Dimensional Mesh Smoothing. 9th International Meshing Roundtable, Sandia National Laboratories, pp. 373-384, 2000.

ZHU, J.Z. e ZIENKIEWICZ, O.C. e HINTO, E. e WU, J. A New Approach to the Development of Automatic Quadrilateral Mesh Generation. International Journal for Numerical Methods in Engineering, vol 32, pp 849-866:1991.

YAMAKAWA, S. e SHIMADA, K. Quad-Layer: Layered Quadrilateral Meshing of Narrow Two-Dimensional Domains by Bubble Packing and Chordal Axis Transformation. Proceedings of 2001 DAC – 27th Design Automation Conference, 2001.

GREAVES, D.M. e BORTHWICK, A.G.L. Hierarchical Tree-based Finite Element Mesh Generation. International Journal for Numerical Methods in Engineering, vol 45, pp 447-471, 1999.

Anexos

Anexo A – Funções Utilizadas

A Funções

O programa de geração de malhas desenvolvido nesta dissertação implementa o método do avanço da fronteira usando a linguagem de programação C++. Este anexo apresenta funções concebidas e implementadas no curso desta dissertação, essenciais para a construção do programa. O objetivo é o de evidenciar detalhes de programação.

A.1 Criando um Domínio

Um domínio é uma representação geométrica de um corpo físico. Sendo assim, é necessário representá-lo com linhas ou retas, círculos, arcos, etc. No momento de representar computacionalmente este corpo, devem-se armazenar as suas coordenadas cartesianas, sendo também necessário relaciona-lo com as medidas reais do corpo a ser representado. Como neste trabalho está sendo utilizado somente polígono irregular planos, este é caracterizado por retas ou linhas. O algoritmo abaixo descrito cria um domínio poligonal irregular plano em um ambiente gráfico.

```
case POLIG:                                          // polígono
{
        Poli * pl = new Poli(&VP);
        pl->NumPoints = mg->obj.pl->NumPoints;
        pl->P = (MyPoint *) malloc(sizeof(MyPoint) * pl->NumPoints);
        for (i = 0; i < pl->NumPoints; i++){
                pl->P[i].x = VP.MapX((float)VP.MapStandardX(mg->obj.pl->P[i].x));
                pl->P[i].y = VP.MapY((float)VP.MapStandardY(mg->obj.pl->P[i].y));
        }
        pl->PoligonoDraw();
        free(pl->P);
        delete pl;
        break;
}
```

onde pl é uma classe do tipo polígono e PoligonoDraw() é a função para desenhar um polígono irregular. A classe polígono é descrita abaixo:

```
class Poli {
public:
   // atributos
   MyPoint * P;          // vetor de pontos
   int NumPoints;       // número de pontos
   int vertice;        // valor 0-não interno 1-interno
   Marcadores oMarc;          // marcador
   ViewPort * VP;       // viewport
   // métodos
   Poli(ViewPort * vp);
   ~Poli();
   void DefineMarc(int x);
   void PolilinhaDraw();
   void PolimarcaDraw();
   void PoligonoDraw();
};
```

A função PoligonoDraw() desenha um polígono irregular, conforme a quantidade de vértices desejada, sendo necessários à inserção dos pontos de cada vértice.

```
void Poli::PoligonoDraw() {
   Line oLine(VP);            // linha a ser desenhada
   PolilinhaDraw();
   oLine.P1.x = P[NumPoints - 1].x;
   oLine.P1.y = P[NumPoints - 1].y;
   oLine.P2.x = P[0].x;
   oLine.P2.y = P[0].y;
   oLine.MidPointDraw();
}
```

A.2 Selecionando um Domínio

A seleção do domínio deve conter todas as retas que fazem parte dele. O que pode ser vista na Figura A.103. Ou seja, uma vez armazenados os pontos e o tipo

do domínio geométrico desenhado, fica mais simples buscar na lista de figuras e então marcá-lo, mudando a cor do contorno do domínio para vermelho.

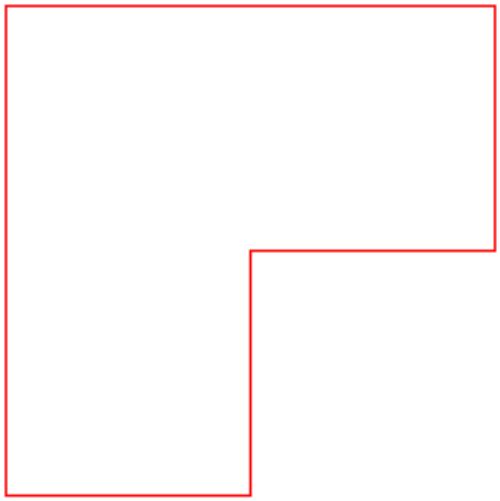

Figura A.103 – Seleção de um Domínio

Abaixo o algoritmo para seleção de um domínio qualquer:

```
MGObject * mg;                          // objeto auxiliar
float x1, y1, x2, y2;                   // pontos auxiliares
float xaux, yaux;                       // pontos de troca

mg = mgo;
while (mg != NULL){                     // caminha pela lista
      switch (mg->type){                // tipo do objeto
        case POLIG:                     // polígono
                {
                        // encontrar os pontos de maximo e minimo
                        x1 = x2 = mg->obj.pl->P[0].x;
                        y1 = y2 = mg->obj.pl->P[0].y;
                        for (int i = 0; i < mg->obj.pl->NumPoints; i++){
                                // econtra maximo em x
                                if (x2 <= mg->obj.pl->P[i].x)
                                x2 = mg->obj.pl->P[i].x;
                                // econtra minimo em x
                                if (x1 >= mg->obj.pl->P[i].x)
                                x1 = mg->obj.pl->P[i].x;
                                // econtra maximo em y
                                if (y2 <= mg->obj.pl->P[i].y)
                                y2 = mg->obj.pl->P[i].y;
                                // econtra minimo em y
                                if (y1 >= mg->obj.pl->P[i].y)
                                y1 = mg->obj.pl->P[i].y;
                        }
                        break;
                }
```

```
        }
                // troca de pontos
            if (x1 > x2){ xaux = x1; x1 = x2; x2 = xaux; }
            if (y1 > y2){ yaux = y1; y1 = y2; y2 = yaux; }
            mg->contorno = false;
        // objeto selecionado
        if ((x >= x1) && (x <= x2) && (y >= y1) && (y <= y2)){
            mgo_select = mg;
            mg->color = clRed;
            MGODraw(mg);
            mg->color = MainForm->UserColor;
            if ((mgo_actual != NULL) && (mgo_actual != mgo_select))
                    MGODraw(mgo_actual);
            mgo_actual = mgo_select;
            mg->contorno = true;
            return;
    }
    mg = mg->next;

    // atualização
    if (mgo_actual != NULL)  MGODraw(mgo_actual);
    mgo_select = NULL;
    mgo_actual = NULL;
```

A.3 Estrutura Utilizada

Esta estrutura possibilita definir quais são os pontos de vértices sobre o contorno. Após esta seleção, é necessário chamar a função de identificação dos vértices externos. Esta identificação é feita utilizando os vértices do elemento selecionado, pois todos os elementos são armazenados em uma estrutura do tipo lista encadeada. Uma lista encadeada armazena em seqüência todos os elementos que foram desenhados (a lista pode ser vista na Figura A.104) e também qual é o próximo componente, a sua altura ou sua largura, e se o contorno está ou não selecionado. A estrutura representada na Figura A.104 também mostra que existem vários tipos de componentes que podem ser adicionados à lista, como polígonos, retângulos, círculos entre outros. Com os dados armazenados nesta lista é possível identificar os vértices de cada componente, pois cada tipo de componente possui um formato definido. Por exemplo, o retângulo possui quatro vértices A, B, C e D, sendo que ao armazenar dois vértices A e D os mesmos são suficientes para gerar um

retângulo, pois A é o vértice superior esquerdo e D é o vértice inferior direito. Os outros dois vértices são identificados pela relação dos vértices armazenados.

```
// objeto geométrico (lista encadeada)
typedef struct MGObjecttag {
   int type;                                  // tipo
   TColor color;              // cor do objeto
   union {                   // definição do objeto
      Marcadores * pt;
      Line * l;
      MyRectangle * r;
      Poli * pl;
      PoligonoReg * pr;
      Circle * c;
      Elipse * e;
      Curve * cv;
   } obj;
   struct MGObjecttag * next;          // ponteiro para próximo objeto
   float altura;              // altura do objeto
   float largura;              // largura do objeto
   bool contorno;
} MGObject;
```

Figura A.104 – Lista Encadeada de Componentes

A.4 Vértices Côncavos

Para gerar uma malha inicial é necessário encontrar todos os vértices côncavos. Eles são responsáveis por definir os possíveis pontos de concentração de tensões na malha. Todo vértice côncavo que for encontrado deve ser marcado. A forma definida para identificar qual é o vértice côncavo dentro de uma coleção de pontos, foi informar que, ao encontrar o vértice côncavo, este está relacionado a sua coordenada.

O algoritmo para encontrar os pontos côncavos foi definido sobre um polígono. A escolha deste tipo de componente deu-se pelo fato que no polígono é possível ocorrer qualquer variação em sua forma, podendo existir vértices convexos e vértices côncavos com diferentes ângulos de abertura. Para tanto, foi utilizado o algoritmo de MELKMAN, que encontra os vértices côncavos de um determinado polígono.

O algoritmo de MELKMAN pode ser compreendido na Figura A.105:

```
// Copyright 2002, softSurfer (www.softsurfer.com)
// This code may be freely used and modified for any purpose
// providing that this copyright notice is included with it.
// SoftSurfer makes no warranty for this code, and cannot be held
// liable for any real or imagined damage resulting from its use.
// Users of this code must verify correctness for their application.

// Assume that a class is already given for the object:
//    Point with coordinates {float x, y;}
//===================================================================

// isLeft(): test if a point is Left|On|Right of an infinite line.
//    Input:  three points P0, P1, and P2
//    Return: >0 for P2 left of the line through P0 and P1
//            =0 for P2 on the line
//            <0 for P2 right of the line
//    See: the January 2001 Algorithm on Area of Triangles
inline float
isLeft( Point P0, Point P1, Point P2 )
{
    return (P1.x - P0.x)*(P2.y - P0.y) - (P2.x - P0.x)*(P1.y - P0.y);
}


// simpleHull_2D():
//    Input:  V[] = polyline array of 2D vertex points
//            n   = the number of points in V[]
//    Output: H[] = output convex hull array of vertices (max is n)
//    Return: h   = the number of points in H[]
int
simpleHull_2D( Point* V, int n, Point* H )
{
    // initialize a deque D[] from bottom to top so that the
    // 1st three vertices of V[] are a counterclockwise triangle
    Point* D = new Point[2*n+1];
    int bot = n-2, top = bot+3;   // initial bottom and top deque indices
    D[bot] = D[top] = V[2];       // 3rd vertex is at both bot and top
    if (isLeft(V[0], V[1], V[2]) > 0) {
        D[bot+1] = V[0];
        D[bot+2] = V[1];          // ccw vertices are: 2,0,1,2
    }
    else {
        D[bot+1] = V[1];
        D[bot+2] = V[0];          // ccw vertices are: 2,1,0,2
    }

    // compute the hull on the deque D[]
    for (int i=3; i < n; i++) {   // process the rest of vertices
        // test if next vertex is inside the deque hull
        if ((isLeft(D[bot], D[bot+1], V[i]) > 0) &&
            (isLeft(D[top-1], D[top], V[i]) > 0) )
            continue;         // skip an interior vertex

        // incrementally add an exterior vertex to the deque hull
        // get the rightmost tangent at the deque bot
        while (isLeft(D[bot], D[bot+1], V[i]) <= 0)
            ++bot;                // remove bot of deque
        D[--bot] = V[i];          // insert V[i] at bot of deque

        // get the leftmost tangent at the deque top
        while (isLeft(D[top-1], D[top], V[i]) <= 0)
            --top;                // pop top of deque
        D[++top] = V[i];          // push V[i] onto top of deque
    }

    // transcribe deque D[] to the output hull array H[]
    int h;        // hull vertex counter
    for (h=0; h <= (top-bot); h++)
        H[h] = D[bot + h];
```

```
delete D;                                   }
return h-1;
```

Figura A.105 – Algoritmo de MELKMAN

Para o caso desejado, foi criada uma variação do código definido por MELKMAN em C++. O ponto principal do algoritmo é selecionar três pontos seqüenciais (três vértices) no sentido anti-horário de um polígono e, então, encontrar a área deste triângulo gerado por estes três vértices. Se o resultado for maior que zero, o ponto está dentro do domínio e é um vértice côncavo, se o resultado for igual a zero, o vértice está no contorno do polígono e produz uma linha reta com os três vértices, e se o resultado for menor que zero, o ponto esta fora do polígono e foi encontrado um vértice convexo.

No momento que o vértice é identificado como côncavo este não será utilizado para efeitos de geração de um novo cálculo.

A fórmula utilizada para encontrar a área do triângulo é dada por:

$$Área = (P1.x - P0.x) \times (P2.y - P0.y) - (P2.x - P0.x) \times (P1.y - P0.y) \qquad (1)$$

A Figura A.106 mostra uma tela com seus respectivos componentes, o vértice côncavo sendo selecionado por um círculo em vermelho, o qual foi gerado com o objetivo de mostrar que somente os vértices côncavos são selecionados.

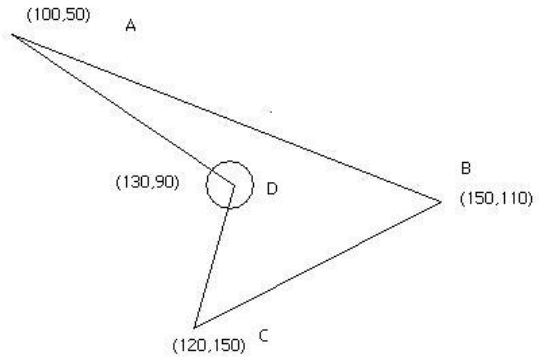

Figura A.106 – Ângulo Côncavo Selecionado

A.5 Calculando o Ângulo Entre dois Eixos

Com as coordenadas globais dos três vértices é gerado um sistema de coordenadas locais para calcular o ângulo do vértice gerado entre eles. Este vértice é o centro deste novo sistema de coordenadas. O ângulo é calculado sobre as coordenadas x e y dos dois pontos (anterior e posterior). Para calcular sobre estas novas coordenadas x e y, todo o sistema é tratado por geometria analítica, ou seja, todas as coordenadas são vetorizadas em relação aos pontos dados, gerando desta forma dois vetores unitários *u* e *v*. Com vetores, é possível calcular o ângulo das retas dadas, pelas expressões abaixo:

$$\arccos\theta = \frac{u.v}{\|u\|\|v\|} \qquad (2)$$

$$u = (u_1; u_2) \ ; \ v = (v_1; v_2) \qquad (3)$$

$$\|u\| = \sqrt{(u_1^2 + u_2^2)} \ ; \ \|v\| = \sqrt{(v_1^2 + v_2^2)} \qquad (4)$$

Com esta informação, foi possível gerar uma função para calcular o ângulo de um vértice qualquer como esta representada na Figura A.107. Para isto, é necessário passar as coordenadas do ponto anterior e posterior. A função retornará o ângulo gerado entre estes pontos.

```
double TMDIChild::EncontraAngulo(float p1x, float p1y, float p2x, float p2y){

double escalar=0, vetorial=0, angulo=0;
// calcular produto escalar

escalar = (p1x * p2x) + (p1y * p2y);
vetorial = sqrt(pow(p1x,2)+pow(p1y,2))*
sqrt(pow(p2x,2)+pow(p2y,2));
// encontra o angulo do vertice
angulo = acos(escalar / vetorial);
angulo = (angulo*180º)/3.1415;

return angulo;
}
```

Figura A.107 – Função para Calcular Ângulo

A.6 Encontrando a Bissetriz do Vértice Qualquer

Para encontrar a bissetriz de um vértice, devem-se encontrar as retas que fazem parte deste vértice. Para a geração da bissetriz pelo método geométrico, utiliza-se o compasso, o esquadro e o transferidor. A bissetriz é traçada a partir da geração de pontos em cada uma das retas com a mesma distância entre o vértice e o ponto. Com os pontos identificados e com um compasso, pode-se a partir deles encontrar o ponto que divide o ângulo. Mas, para criar computacionalmente uma bissetriz é necessário ter os pontos que definem o vértice. Podendo assim, dividir o ângulo formado pelo vértice desejado. Na Figura A.108 pode-se ver o algoritmo implementado para gerar qualquer bissetriz para qualquer vértice dado.

```
//-------------------------------------------------
// P0 - ponto inicial
// P1 - próximo ponto
// P2 -  ponto anterior
// tipo - tipo do calculo 0 - pontos iniciais; 1
- pontos em rel. a bissetriz B1
MyPoint
TMDIChild::EncontrarBissetriz(MyPoint
P0,MyPoint P1,MyPoint P2, int tipo){

  MyPoint P3, P4;
  double ux, uy, normau;
  double vx, vy, normav;
  // encontrar os pontos dos vetores
  ux = P1.x - P0.x;
  uy = P1.y - P0.y;
  // encontrar a norma do vetor u
  normau = sqrt(pow(ux,2)+pow(uy,2));
  // econtrar a direção
  ux = ux / normau; uy = uy / normau;
  // encontrar os pontos dos vetores
  vx = P2.x - P0.x;
  vy = P2.y - P0.y;
  // encontrar a norma do vetor v
  normav = sqrt(pow(vx,2)+pow(vy,2));
  // Encontrar a directo
  vx = vx / normav; vy = vy / normav;

  P3.x = P0.x + 8.87 * (ux + vx);
  P3.y = P0.y + 8.87 * (uy + vy);

   // dado o ponto medio encontra os
pontos entre P0P3
  if (tipo == 0){
    // encontrar os pontos dos vetores
    ux = P3.x - P0.x;
    uy = P3.y - P0.y;
    // encontrar a norma do vetor u
    normau = sqrt(pow(ux,2)+pow(uy,2));
    // encontrar a direção
    ux = ux / normau; uy = uy / normau;

    P4.x = P0.x - 30 * ux;
    P4.y = P0.y - 30 * uy;

    P4.vertice = 0;
    P4.angulo = 0;
    P4.b1x = 0;
    P4.b1y = 0;
    P4.b2x = 0;
    P4.b2y = 0;
    P4.b3x = 0;
    P4.b3y = 0;

    return P4;
  }
  else{
    P3.vertice = 0;
    P3.angulo = 0;
    P3.b1x = 0;
```

```
    P3.b1y = 0;                                    P3.b3y = 0;
    P3.b2x = 0;
    P3.b2y = 0;                                    return P3;
    P3.b3x = 0;                                  }
}
```

Figura A.108 - Algoritmo da Bissetriz

O algoritmo gerado segue a seguinte fórmula definida no livro de (FEITOSA,1996):

$$X = A + t \text{ x } (\vec{u} \text{ x } \vec{v}) \qquad (5)$$

onde *A* é o vértice que contém o ângulo com suas respectivas coordenadas x e y, *t* é uma variável que pode receber qualquer valor que faça parte do eixo x, e *u* e *v* são os vetores gerados a partir do vértice.

Para encontrar os vetores *u* e *v* é necessário utilizar as equações *(3)* e *(4)*, ficando desta forma:

$$\vec{u} = u / \|u\| \text{ e } \vec{v} = v / \|v\| \qquad (6)$$

Com isto, encontram-se os pontos da bissetriz em relação ao vértice dado. Contudo, o ponto encontrado esta fora do componente principal, sendo necessário adicionar este ponto dentro do domínio. Para isto, utilizam-se as regras de vetores para posicionar a nova coordenada com a seguinte equação:

$$\vec{u} = k \text{ x } \vec{v} \qquad (7)$$

se *k* for negativo, será um vetor com sentido contrário, e assim, colocando o ponto encontrado dentro do domínio principal (-B1), gerando a Figura A.109. As equações *(2)*, *(3)*, *(4)*, *(6)* e *(7)* foram definidas em (BOLDRINI et.all, 1996).

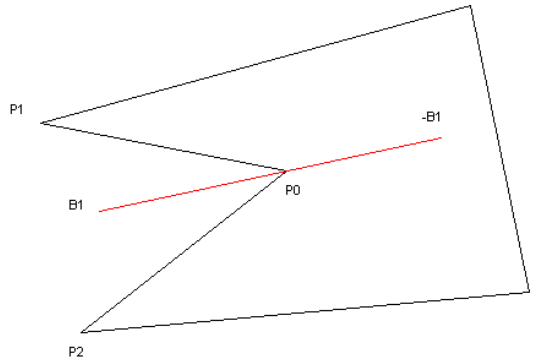

Figura A.109 - Pontos B1 e -B1 Encontrados

A.7 Encontrando Bissetrizes em Vértices Convexos

Para encontrar as bissetrizes dos demais vértices, utiliza-se o mesmo algoritmo apresentado anteriormente, contudo serão geradas as bissetrizes em todos os vértices do polígono. O algoritmo inicia a partir do primeiro vértice, seguindo em sentido horário até que complete todos os vértices, conforme a Figura A.110.

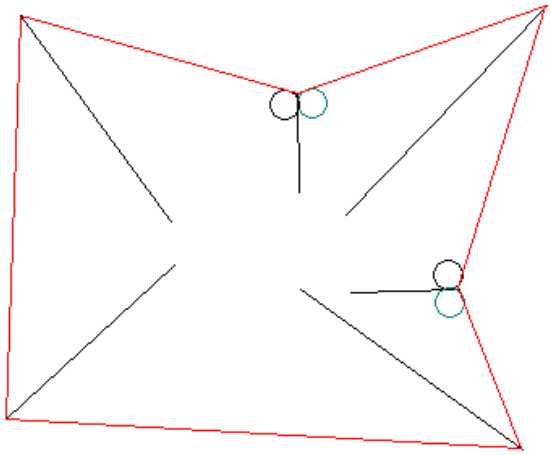

Figura A.110 - Bissetrizes em Todos os Vértices

A.8 Encontrar a Intersecção entre as Duas Retas Definidas

Na construção da malha com elementos triangulares, após encontrar as retas necessárias para construir um triângulo eqüilátero é necessário identificar a

intersecção entre elas. Esta interseção definirá os pontos de prováveis triângulos dentro do domínio. Para encontrar o ponto de interseção entre as duas retas dadas, deve-se encontrar as equações destas retas. Ou mais certo, é definir as coordenadas dos pontos P0 e P1 que estabelecem as retas. Estes pontos são definidos através de suas coordenadas x e y, ou seja, P0.x, P0.y, P1.x e P1.y. A função DefiEqua apresentada na Figura A.111, encontra a equação da reta dada às coordenadas desejadas.

```
//---------------------------------------------------------------------------
Equacao TMDIChild::DefiEqua(MyPoint P0,MyPoint P1){
quacao equa;
equa.c = (P0.x * P1.y) - (P1.x * P0.y); // valor de c
equa.a = P0.y - P1.y; // valor de x
equa.b = P1.x - P0.x; // valor de y
return equa;
}
```

Figura A.111 - Função para Montar a Equação da Reta

De posse desta informação é necessário encontrar os pontos de intersecção entre as duas retas dadas. Para isto (BARSOTI, 1992) e (BOLDRINI et.all, 1996) descrevem como é a construção de um sistema de equações. As duas equações fornecem os pontos de intersecção como apresentado na equação abaixo:

$$a \times x = b \qquad (8)$$

em *(8)* temos as seguintes soluções:

1) $a \neq 0$ → uma única solução;

2) $a = 0$ e $b = 0$ → várias soluções;

3) $a = 0$ e $b \neq 0$ → não existe solução;

Para resolver o sistema de equações mais facilmente, a forma proposta foi utilizar matrizes para encontrar a solução do problema como é apresentado por (BOLDRINI et.all, 1996). Sendo assim, utilizou-se a seguinte fórmula:

$$x = A^{-1} \text{x } b \qquad (9)$$

onde, A^{-1} é a matriz inversa, a qual contém as incógnitas x e y, e b é o vetor resultante das equações dadas. Para visualizar melhor o que está sendo exposto, abaixo é apresentado um exemplo em relação à formulação apresentada:

Dado os pontos A(2, 4) e B(7,9) que fazem parte da reta R1 e C(10,2) e D(8,5) da reta R2, encontrar a interseção entre as duas retas dadas.

Encontrando a equação das duas retas temos:

Equação1 em relação aos pontos A(2, 4) e B(7,9) - (-5x + 5y = 10*),*

Equação2 em relação aos pontos C(10,2) e D(8,5) - (-3x – 2y = 34*),*

onde a matriz A é $\begin{bmatrix} -5 & 5 \\ -3 & -2 \end{bmatrix}$ *e o vetor b é* $\begin{bmatrix} 10 \\ 34 \end{bmatrix}$

o determinante de A é – det(A) = 25,

a transposta de A é – $A^t = \begin{bmatrix} -2 & -5 \\ 3 & -5 \end{bmatrix}$, *então* $A^{-1} = \begin{bmatrix} -0,08 & -0,2 \\ 0,12 & -0,2 \end{bmatrix}$,

aplicando os dados em (9), tem-se:

$x = \begin{bmatrix} -7,6 \\ -5,6 \end{bmatrix}$, *que são os pontos de interseção entre as duas bissetrizes.*

Com os valores obtidos de x é possível verificar se este ponto faz parte do domínio ou não. E para ter uma melhor visualização do que foi descrito no exercício apresentado, o algoritmo que resolve este problema é mostrado na Figura A.112.

```
//---------------------------------
// encontra a interseção entre dois pontos
e adiciona um circulo para identificar
// o ponto de interseção


                MyPoint
TMDIChild::InterceccaoP0P1(MyPoint P0,
  MyPoint P1, MyPoint P2, MyPoint P3){
              Marcadores oMark;
                MyPoint P;
            Equacao equa1, equa2;
              Matriz MA, MT;
                Vetor VB;
              double det = 0;

          // monta equacao da reta
          equa1 = DefiEqua(P0,P1);
      // monta segunda equação da reta
          equa2 = DefiEqua(P2,P3);
// carrega matriz com a equacao 1
  MT[0][0] = MA[0][0] = equa1.a;
  MT[0][1] = MA[0][1] = equa1.b;
        VB[0] = -(equa1.c);
// carrega matriz com a equação 2
  MT[1][0] = MA[1][0] = equa2.a;
  MT[1][1] = MA[1][1] = equa2.b;
        VB[1] = -(equa2.c);
    //transposta da matriz MA
        MTransposta(MT);
  det = MInversa(MA,MT,det);
          if (det != 0){
   P = MMatrInveVeto(MA,VB);
      // desenha circulo
          vertice = 2;
          return P;
            }

           }
```

Figura A.112 - Algoritmo para Encontrar os Pontos de Interseção

O resultado deste algoritmo pode ser visto na Figura A.113, que representa graficamente o ponto C gerado pela intersecção de duas retas.

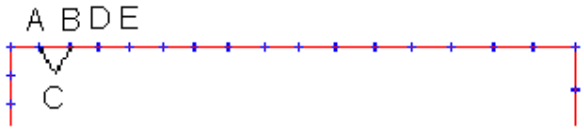

Figura A.113 - Ponto C Gerado

A.9 Criando um Triângulo no Domínio

Por se tratar de um gerador de malha triangular, é necessário inserir elementos triangulares no domínio. Porém, é necessário encontrar o ponto C do triângulo, uma vez que existe somente o segmento de reta AB, como pode ser visto na Figura A.113. Sendo assim, a partir do segmento de reta AB, é traçada uma reta pela rotação do eixo AB em relação ao ponto A. Sendo esta rotação de 45. Da mesma forma, é traçada uma reta em relação ao ponto B. O grau de rotação utilizado para este caso foi de 315.

Conforme apresentado por (Foley at. al., 1996), os cálculos utilizados seguem a sua orientação. Para isto é necessário primeiro montar a matriz identidade *(I)*.

$$I = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Dê posse da matriz identidade, das coordenadas do ponto de origem, e do ângulo desejado, é calculada a matriz de rotação MR, dada por:

$$MR = \begin{pmatrix} \cos(a) & \text{sen}(a) & 0 \\ -\text{sen}(a) & \cos(a) & 0 \\ A.x*(1-\cos(a)) + Ay*\text{sen}(a) & Ay*(1-\cos(a) - Ax*\text{sen}(a) & 1 \end{pmatrix}$$

onde Ax é a coordenada em x e Ay é a coordenada em y do ponto A. Após ter encontrado a matriz resultado, esta é multiplicada pelas coordenadas do ponto A e do ponto B. Com isto, tem-se então dois novos pontos, gerando assim duas novas retas. Com estas duas novas retas é possível calcular o ponto de Intersecção entre elas. Este ponto de intersecção mostrará o possível ponto C do triângulo desejado ABC.

De posse deste ponto e do ponto médio do segmento AB, é calcula então a real coordenada do ponto. Esta coordenada deve possibilitar a criação de um triângulo eqüilátero. Para garantir que será um triângulo eqüilátero a altura do triângulo deverá ser de 0,86 vezes o tamanho do segmento AB. O algoritmo gerador do triângulo é apresentado abaixo.

```
MyPoint TMDIChild::EncoPontTria(MyPoint P0, MyPoint Pos){
  MyPoint P1, P2, P3;
  Equacao equareta, equaperp;
  float normaP0, normaP2;
  Mat MT;  // Matriz
  Vec * v; // vetor
                                        // primeiro vetor encontrado
                                        Identy(MT); // matriz identidade
                                        Rotation(MT, P0.x, P0.y, 45); //rotacao
                                        v = (Vec *) malloc(sizeof(Vec) * 2);
                                        v[0][0] = P0.x;
                                        v[0][1] = P0.y;
                                        v[0][2] = 1;
                                        v[1][0] = Pos.x;
```

```
  v[1][1] = Pos.y;
  v[1][2] = 1;
  MatVecMult(v[0], MT);
  MatVecMult(v[1], MT);
  P1.x = v[1][0];
  P1.y = v[1][1];
  // segundo vetor encontrado
  Identy(MT); // matriz identidade
  Rotation(MT, Pos.x, Pos.y, 315);
//rotacao
  v[0][0] = Pos.x;
  v[0][1] = Pos.y;
  v[0][2] = 1;
  v[1][0] = P0.x;
  v[1][1] = P0.y;
  v[1][2] = 1;
  MatVecMult(v[0], MT);
  MatVecMult(v[1], MT);
  P2.x = v[1][0];
  P2.y = v[1][1];
  // limpa memoria
  free(v);
  // ponto de intersecção entre os dois
vetores encontrados
  P2 = InterceccaoP0P1(P0, P1, Pos, P2);
  // ponto médio entre P0 e Pos
  P1.x = (P0.x + Pos.x)/2.0;
  P1.y = (P0.y + Pos.y)/2.0;
  normaP0 = norma(P0,Pos) * 0.86;
  normaP2 = norma(P1,P2);
  P3.x = P1.x + (((P2.x -
P1.x)*normaP0)/normaP2) ;
  P3.y = P1.y + (((P2.y -
P1.y)*normaP0)/normaP2);
   //P3.x = (P2.x * normaP0)/normaP2;
  //P3.y = (P2.y * normaP0)/normaP2;
  return P3;
}
```

A.10 Definindo a Quantidade de Elementos no Contorno

Para iniciar a geração de malha, é necessário primeiro separar o contorno em segmentos. Estes segmentos podem ser iguais se ela for do tipo uniforme. E se for desejado uma malha gradual, os segmentos devem aumentar gradativamente. Porém, o valor do tamanho do elemento é fixado, garantindo assim a qualidade da malha. Para isto, foi montado um algoritmo para calcular estes segmentos. Para encontrar o tamanho do segmento no contorno foi utilizado a fórmula da PA. Sendo assim, um segmento inicia com um tamanho 10, podendo ter o tamanho 15, 20 ou 25 unidades de tamanho. A fórmula da PA é dada por:

$$Sn = ((a1 + na) \times n) / 2,$$

Onde Sn é o tamanho total do segmento desejado, a1 é a norma do primeiro ponto, na é a norma do último ponto e n é o número de segmentos. O algoritmo abaixo apresenta a função para encontrar a quantidade de pontos necessários para garantir a qualidade da malha.

```
void TMDIChild::CalcDiam(float normaP0,
float tipoP0, float tipoPos, VetorNos vnos){
  float quantnosini, quantnosfim,  nelem;
  //valorx = norma(Pos,P0);
  // PA
  // encontrar o numero de elementos
  // formula PA => Sn = ((a1 + an)*n)/2
  // sendo assim, n = (Sn*2)/(a1+an);
  if (tipoP0 == 1) quantnosini =  10;
  if (tipoP0 == 2) quantnosini =  15;
  if (tipoP0 == 3) quantnosini = 20;
  if (tipoP0 == 4) quantnosini = 25;
  if (tipoPos == 1) quantnosfim =  10;
  if (tipoPos == 2) quantnosfim =  15;
  if (tipoPos == 3) quantnosfim = 20;
  if (tipoPos == 4) quantnosfim = 25;
  vnos[0] = quantnosini;
  vnos[1] = quantnosfim;
  vnos[2] = (int) (normaP0 *
2.0)/(quantnosini+quantnosfim);
  if (quantnosini == quantnosfim)
    vnos[3] = normaP0 / (int) vnos[2];
  else
    vnos[3] = (quantnosfim-
quantnosini)/(vnos[2]-1);
}
```

A.11 Encontrado a Equação da Reta

Para encontrar os pontos de intersecção entre duas retas, é necessário definir a equação da reta. Esta equação leva em consideração os pontos A e B de um segmento de reta. Sendo que a equação da reta é dada por:

$$\frac{x - Ax}{Bx - Ax} - \frac{y - Ay}{By - Ay} = 0,$$

Com esta informação, foi possível criar um algoritmo para encontrar a equação da reta em relação a dois pontos dados, o que pode ser visto abaixo.

```
Equacao TMDIChild::DefiEqua(MyPoint P0,MyPoint P1){
  Equacao equa;
  equa.c = -(P0.x * P1.y) + (P0.x * P0.y) + (P0.y * P1.x) - (P0.y * P0.x); // valor de c
  equa.a = P1.y - P0.y; // valor de x
  equa.b = -(P1.x - P0.x); // valor de y
  return equa;
}
```

A.12 Busca de um Provável Nó Próximo

Durante a definição dos pontos necessários para criar o triângulo eqüilátero desejado no nó C, deve-se validar este ponto. Para isso, ao encontrar o nó C do triângulo BDC conforme Figura A.113, é necessário identificar em relação aos

elementos já inscritos se o segmento BC e DC interceptam algum outro elemento, ou, se existe algum outro nó muito próximo. Foi então, definido que, a partir da coordenada do nó C é verificado se existe outro nó próximo, para isso, varre-se a primeira e segunda coluna da tabela de elementos inscritos (Tabela 3), e estas colunas possuem a coordenadas dos nós C inscritos anteriormente. Com esta informação é verificado se à distância do nó C desejado é maior que um por cento ao nó C armazenado. Caso essa distância seja inferior, então o nó armazenado será utilizado como o nó para a geração do triângulo, conforme algoritmo abaixo. Caso não seja encontrado nenhum nó próximo, o ponto C continua sendo um nó válido. Para o algoritmo abaixo, o ponto P2 é o nó C, e x e y são as coordenadas dos nós já armazenados na tabela de elementos (Tabela 3).

```
MyPoint
TMDIChild::BuscaNoProx(MyPoint P2){
  int cont;
  double x, y;

  pontoOk = false; // semaforo

  // gera um nova entrada na tabela de
pontos ao encontrar um vértice concavo
  cont = 0;
  while (cont < contlinhaux){
    x = StrToFloat(StringGrid1-
>Cells[1][cont]);
    y = StrToFloat(StringGrid1-
>Cells[2][cont]);


    if ((int)(P2.x - (P2.x * 0.01)) <= (int) x &&
(int) x <= (int)(P2.x * 1.009))
      if ((int)(P2.y - (P2.y * 0.01)) <= (int) y
&& (int) y <= (int)(P2.y * 1.01)){
        pontoOk = true;
        break;
      }
      else
        pontoOk = false;

    cont++;
  }

  if (pontoOk) {
    P2.x = x;
    P2.y = y;
  }

  return P2;
}
```

A.13 Verificar a distância entre o nó C e os nós do contorno

Esta função busca verificar se à distância do nó C em relação aos demais nós do contorno, para isto, calcula-se 80% da distância do nó C até o nó B (seu ponto de origem, e no algoritmo o ponto P0). Com a norma do segmento CB obtida calcula-se os 80% desta, podendo a partir de então procurar os nós do contorno conforme

exposto na Tabela 1. A procura parte do primeiro nó da Tabela 1, sendo ele armazenado em uma variável chamada de Pos1, sendo um nó já marcado e que faz parte da fronteira aberta. De posse do nó Pos1 é possível calcular a norma do segmento CPos1, e esta norma encontrada deve ser maior que a norma do segmento CB. Caso a norma do segmento CPos1 seja inferior a 80% da norma do segmento CB o nó C não é utilizado para a geração de um elemento triangular. O código visualizado abaixo busca os nós marcados de uma tabela de nós, como pode ser visto na Tabela 1.

```
// calcular norma em relação ao demais pontos, para descobrir se
// o ponto não esta próximo a outros pontos
bool TMDIChild::DifePont(MyPoint P2, float normaP2, int m, int n, int cont){
  MyPoint Pos1;
  float normaP2P1;
  int i = 0;

  for (int j = 0; j < n; j++){
    Pos1.x = StrToFloat(StringGrid2->Cells[m+1][j]);
    Pos1.y = StrToFloat(StringGrid2->Cells[m+2][j]);

    normaP2P1 = norma(P2,Pos1);

    if (normaP2P1 < normaP2 * 0.80){
      i++;
    }
  }
  if (i >= 1) return false;
  else return true;
}
```

A.14 Busca Ângulo

Esta função tem como característica encontrar o ângulo formado entre três nós da fronteira. Esta função retorna o ângulo formado entre os três nós, conforme o algoritmo abaixo.

```
int TMDIChild::BuscaAngulo(MyPoint P0, MyPoint Ant, MyPoint Pos){

  int angulo;
  float p1x,p1y,p2x,p2y;

  p1x = Ant.x - P0.x;
```

```
  p1y = Ant.y - P0.y;
  p2x = Pos.x - P0.x;
  p2y = Pos.y - P0.y;


  angulo = (int) EncontraAngulo(p1x,p1y,p2x,p2y);

  return angulo;
}
```

Anexo B – Código Fonte.

B.1 Definição dos nós sobre o contorno

```
void TMDIChild::DefiPontCont(){
  MyPoint P0, Ant, Pos, Dia, P1,P2,P3,P4;
  int l,m,n, contador = 0, cont;
  float tamseg, normaP1, normaP0;
  VetorNos vnos;
  // vnos
  // posicao 0 -> tamanho do primeiro elemento
  // posicao 1 -> possivel tamanho do ultimo elemento
  // posicao 2 -> quantidade de elementos
  // posicao 3 -> incremento do tamanho do elemento, no
caso da posicao 0
  //                for igual a posicao 1 entao o tamanho do
elemento é fixo.

  // subdivisão dos vértices em relação ao dominio
principal
  l = 6 * contint;
  m = l - 6;
  n = contlin;
  cont = 1;
  // gera um nova entrada na tabela de pontos ao
encontrar um vértice concavo
  contint++;
  contlin = 0;
  contcol = contcol + 6;
  StringGrid2->ColCount = StringGrid2->ColCount + 6;


  P0.x = StrToFloat(StringGrid2->Cells[m+1][0]);
  P0.y = StrToFloat(StringGrid2->Cells[m+2][0]);
  P0.vertice = StrToFloat(StringGrid2->Cells[m+3][0]);
  P0.angulo = StrToFloat(StringGrid2->Cells[m+4][0]);
  P0.b1x = StrToFloat(StringGrid2->Cells[m+5][0]);
  Pos = buscaproximo(0, n, m);

  while (cont <= n){


    // verifica qual é o tipo de tensão entre os pontos
    // se a tensão é a mesma faz subdivisão entre os
valores com o mesmo
    // tamanho, caso contrario divide e adiciona os pontos
na forma gradual
    normaP0 = norma(P0,Pos);

    //if (P0.angulo == Pos.angulo){
    //tamseg =
    CalcDiam(normaP0, P0.angulo, Pos.angulo, vnos);
    P1.angulo = P0.angulo;
    // ponto auxiliar
    P2 = P0;
    // Desenha Circulo
    //DeseCirc(P0,1.0);
    //MontTabePontTang(P0, P0.vertice, P0.angulo);

    contador = 0;
    normaP1 = normaP0;
    if (vnos[0] != vnos[1]) tamseg = vnos[0];
    else tamseg = vnos[3];
    // calcula a norma
    //normaP0 = norma(P0,Pos);// - tamseg;
    //contador = vnos[2];//tamseg;
    // marca pontos
    while (normaP1 > 0){//contador <= vnos[2]){ //normaP0){

      P1.x = Pos.x - P0.x;
      P1.y = Pos.y - P0.y;
      // calcula a norma
      //normaP1 = norma(P2,Pos);
      // encontra os novos pontos em relação ao raio
      P1.x = P0.x + ((P1.x * tamseg)/normaP1);
      P1.y = P0.y + ((P1.y * tamseg)/normaP1);
      P1.b1x = P0.b1x;

      P0 = P1;

      if ((int) (normaP1 - (normaP1 * 0.30)) > (int) tamseg){
        DeseCirc(P1,1.0);
        MontTabePontTang(P1, P0.vertice, P0.angulo);
      }
      else {
        DeseCirc(Pos,1.0);
        MontTabePontTang(Pos, Pos.vertice, Pos.angulo);
        break;
      }

      normaP1 = normaP1 - tamseg;

      if (vnos[0] != vnos[1]){
        tamseg = tamseg + vnos[3];
      }

    }
    P0 = Pos;
    Pos = buscaproximo(cont, n, m);
    cont++;
  }
}
```

B.2 Preenchimento da Fronteira

```
void TMDIChild::GeraPontInte(){
  MyPoint P0, Pos, Pos1, Ant, PAuxP2, P1,P2, P3, P4, P5,
P2Ant, P2Pos, P0Aux;
  int l,m,n, cont, angulo, anguloaux, anguloant, status,
anguloprox;
  double maxx;
  float p1x,p1y,p2x,p2y;
  double normaP0, normaP1;
  bool dif, pontoP0;
  bool pontoant;

  pontoant = false;
  // subdivisão dos vértices em relação ao dominio
principal
  l = 6 * contint;
  m = l - 6;
  n = contlin;

  dif = true;
  difstatus = true;
  pontoP0 = true;

  // gera um nova entrada na tabela de pontos ao
encontrar um vértice concavo
  contint++;
  contlin = 0;
  contcol = contcol + 6;
  StringGrid2->ColCount = StringGrid2->ColCount + 6;

  P0.x = StrToFloat(StringGrid2->Cells[m+1][0]);
  P0.y = StrToFloat(StringGrid2->Cells[m+2][0]);
  P0.vertice = StrToFloat(StringGrid2->Cells[m+3][0]);
  P0.angulo = StrToFloat(StringGrid2->Cells[m+4][0]);
  Pos = buscaproximo(0,n,m);
  Pos1 = buscaproximo(1,n,m);
  Ant = buscaanterior(0,n,m);
  PAuxP2.x = 0; PAuxP2.y = 0; PAuxP2.vertice = 0;
PAuxP2.angulo = 0;

  cont = 1;
```

```
  status = 0;

  while (cont <= n){
     // encontrar o angulo
     // posiciona em coordenadas locais para calculo do
angulo
     p1x = Ant.x - P0.x;
     p1y = Ant.y - P0.y;
     p2x = Pos.x - P0.x;
     p2y = Pos.y - P0.y;

     anguloant = (int) EncontraAngulo(p1x,p1y,p2x,p2y);

     // encontrar o angulo
     // posiciona em coordenadas locais para calculo do
angulo
     p1x = P0.x - Pos.x;
     p1y = P0.y - Pos.y;
     p2x = Pos1.x - Pos.x;
     p2y = Pos1.y - Pos.y;

     angulo = (int) EncontraAngulo(p1x,p1y,p2x,p2y);


     // ponto em relação a altura do triangulo
     P2 = EncoPontTria(P0, Pos);
     P2Ant = EncoPontTria(Ant,P0);
     P2Pos = EncoPontTria(Pos,Pos1);

     // encontrar o angulo
     // posiciona em coordenadas locais para calculo do
angulo
     p1x = P2Pos.x - P0.x;
     p1y = P2Pos.y - P0.y;
     p2x = P2.x - P0.x;
     p2y = P2.y - P0.y;

     anguloprox = (int) EncontraAngulo(p1x,p1y,p2x,p2y);

     // area do triangulo se for positiva é um tirangulo
interno e se for
     // negativa é externo.
     maxx = ((P0.x - Pos.x)*(Ant.y - Pos.y))-((Ant.x -
Pos.x)*(P0.y - Pos.y));

     if (157,5º < anguloant && anguloant <= 180º && 157,5º
< angulo && angulo <= 180º  ){

        P2 = BuscaNoProx(P2);

        normaP0 = norma(P0,P2);
        if (dif = DifePont(P2, normaP0, m, n, cont)){

          P3.x = (P0.x + Pos.x)/2.0;
          P3.y = (P0.y + Pos.y)/2.0;

          normaP0 = norma(P2,P3);
          normaP1 = norma(P2,PAuxP2);
          // noventa por cento da normaP1
          normaP0 = normaP0 * 0.9;

          anguloprox = BuscaAngulo(P0, P2, PAuxP2);

          if (difstatus == true || normaP1 >= normaP0 &&
((anguloprox >= 30 && anguloprox <= 80)||(anguloprox >
93 && anguloprox <= 98)||(anguloprox > 108 &&
anguloprox <= 170))){
          // cria elemento p2, pos, p0
            DeseLinh(P2,P0);
            DeseLinh(P2,Pos);
            // Armazena elemento na tabela
            MontTabeElem(P2, P0, Pos);
            // Novo ponto tabela de pontos (dominio)

            MontTabePontTang(P0, 0, 0);
            MontTabePontTang(P2, 0, 0);

            status = 0;
            PAuxP2 = P2;
            if (cont == 1){
              P0Aux = P3;
              pontoP0 = false;
            }
          }
          else {
            MontTabePontTang(P0, 0, 0);
            difstatus = false;
            PAuxP2 = P2;
            status = 0;
          }
        } else {
          MontTabePontTang(P0, 0, 0);
          difstatus = false;
        }
      }
      if (157,5º < anguloant && anguloant <= 180º && 135º <
angulo && angulo <= 157,5º  ){
        P2 = BuscaNoProx(P2);

        normaP0 = norma(P0,P2);
        if (dif = DifePont(P2, normaP0, m, n, cont)){
          // cria elemento p2, pos, p0
          if (cont == n-1 && pontoP0 == false){
            DeseLinh(P0Aux,P0);
            // Armazena elemento na tabela
            MontTabeElem(P0Aux, P0, Pos);
            // Novo ponto tabela de pontos (dominio)
            MontTabePontTang(P0, 0, 0);

          } else {
            DeseLinh(P2,P0);
            DeseLinh(P2,Pos);
            // Armazena elemento na tabela
            MontTabeElem(P2, P0, Pos);
            // Novo ponto tabela de pontos (dominio)
            MontTabePontTang(P0, 0, 0);
            // Armazena elemento na tabela
            if (Pos.vertice == 3)
               MontTabePontTang(P2, 3, 0);
            else
               MontTabePontTang(P2, 0, 0);

            PAuxP2 = P2;
            status = 1;
          }
        } else {
          MontTabePontTang(P0, 0, 0);
          difstatus = false;
        }
      //} else
        status = 0;
      }
      if (157,5º < anguloant && anguloant <= 180º && 112,5º
< angulo && angulo <= 135º  ){
        if (cont == n - 1){
          // cria elemento p2, pos, p0
          DeseLinh(P0, P0Aux);
          // Armazena elemento na tabela
          MontTabeElem(P0Aux, P0, Pos);
          // Novo ponto tabela de pontos (dominio)
          MontTabePontTang(P0, 0, 0);
          MontTabePontTang(P0Aux, 0, 0);
          // chegou no final
          break;
        }
        if (Pos.vertice == 3){
          P2 = BuscaNoProx(P2);

          normaP0 = norma(P0,P2);
          if (dif = DifePont(P2, normaP0, m, n, cont)){
            // cria elemento p2, pos, p0
```

```
          DeseLinh(P2,P0);
          DeseLinh(P2,Pos);
          // Armazena elemento na tabela
          MontTabeElem(P2, P0, Pos);
          // Novo ponto tabela de pontos (dominio)
          MontTabePontTang(P0, 0, 0);

          MontTabePontTang(P2, 0, 0);

          PAuxP2 = P2;
          status = 1;
        } else {
          MontTabePontTang(P0, 0, 0);
          difstatus = false;
        }
      } else {
        anguloprox = BuscaAngulo(P0, P2, PAuxP2);

        if (anguloprox > 50){
          P3.x = (P2.x + P2Pos.x)/2.0;
          P3.y = (P2.y + P2Pos.y)/2.0;
          P2Pos = BuscaNoProx(P3);
          // cria elemento p2, pos, p0
          DeseLinh(P2Pos,P0);
          DeseLinh(P2Pos,Pos);
          DeseLinh(P2Pos,Pos1);
        } else {
          DeseLinh(P2Pos,P0);
          DeseLinh(P2Pos,Pos);
          DeseLinh(P2Pos,Pos1);
          //MontTabePontTang(P0, 0, 0);
          //MontTabePontTang(P2Pos, 0, 0);
        }
        // Armazena elemento na tabela
        MontTabeElem(P2Pos, P0, Pos);
        MontTabeElem(P2Pos, Pos, Pos1);
        // Novo ponto tabela de pontos (dominio)
        MontTabePontTang(P0, 0, 0);

        MontTabePontTang(P2Pos, 0, 0);

        PAuxP2 = P2Pos;
        Pos = Pos1;
        cont++;
        status = 1;
      }
    }
    if (157,5º < anguloant && anguloant <= 180º && 90º <
angulo && angulo <= 112,5º  ){
      P3.x = (P2.x + P2Pos.x)/2.0;
      P3.y = (P2.y + P2Pos.y)/2.0;

      P3 = BuscaNoProx(P3);

      anguloprox = BuscaAngulo(P0, P2, PAuxP2);

      if (anguloprox < 20 || status == 1) status = 1;
      else status = 0;

      if (difstatus == true) {
        // cria elemento p2, pos, p0
        if (status == 0) {
          if (angulo > 100 ){
              DeseLinh(P3,P0);
              DeseLinh(P3,Pos);
              DeseLinh(P3,Pos1);
              // Armazena elemento na tabela
              MontTabeElem(P3, P0, Pos);
              MontTabeElem(P3, Pos, Pos1);
              // Novo ponto tabela de pontos (dominio)
              MontTabePontTang(P0, 0, 0);

              MontTabePontTang(P3, 0, 0);

              PAuxP2 = P3;
              Pos = Pos1;
              cont++;
              status = 1;
          } else {
            //P2 = BuscaNoProx(P2);
            if (Pos.vertice != 3){
              if ((norma(P0,P2) * 1.02) > norma(Pos,Pos1)){
                P3.x  = (P2.x + P2Pos.x) / 2.0;
                P3.y  = (P2.y + P2Pos.y) / 2.0;
                //P3 = BuscaNoProx(P3);
                P2 = P3;
              }
              DeseLinh(P2,P0);
              DeseLinh(P2,Pos);
              DeseLinh(P2,Pos1);
              // Armazena elemento na tabela
              MontTabeElem(P2, P0, Pos);
              // Novo ponto tabela de pontos (dominio)
              MontTabePontTang(P0, 0, 0);
              MontTabePontTang(P2, 0, 0);
              PAuxP2 = P2;
              Pos = Pos1;
              cont++;
              status = 1;
            } else {
              DeseLinh(P2,P0);
              DeseLinh(P2,Pos);
              // Armazena elemento na tabela
              MontTabeElem(P2, P0, Pos);
              // Novo ponto tabela de pontos (dominio)
              MontTabePontTang(P0, 0, 0);
              MontTabePontTang(P2, 0, 0);
              PAuxP2 = P2;
              status = 0;
            }
          }
        }else {
          if (pontoP0 == true){
            DeseLinh(PAuxP2,P0);
            DeseLinh(PAuxP2,Pos);
            DeseLinh(PAuxP2,Pos1);
            // Armazena elemento na tabela
            MontTabeElem(PAuxP2, P0, Pos);
            MontTabeElem(PAuxP2, Pos, Pos1);

            Pos = Pos1;
            cont++;
            status = 0;
            if (cont == 1){
              P0Aux = P3;
              pontoP0 = false;
            }
          }
          else {
            DeseLinh(P0Aux,P0);
            MontTabeElem(P0Aux, P0, Pos);
          }
        }
      } else {
        MontTabePontTang(P0, 0, 0);
        status = 1;
        pontoant = true;
      }
    }

    if (157,5º < anguloant && anguloant <= 180º && 67,5º
< angulo && angulo <= 90º  ){
      if (Pos.vertice == 3){

        P2 = BuscaNoProx(P2);

        normaP0 = norma(P0,P2);
        if (dif = DifePont(P2, normaP0, m, n, cont)){
          // cria elemento p2, pos, p0
          DeseLinh(P2,P0);
          DeseLinh(P2,Pos);
          // Armazena elemento na tabela
```

```
          MontTabeElem(P2, P0, Pos);
          // Novo ponto tabela de pontos (dominio)
          MontTabePontTang(P0, 0, 0);

          MontTabePontTang(P2, 0, 0);

        }
       }
      }
      if (157,5º < anguloant && anguloant <= 180º && 45 <
angulo && angulo <= 67,5º  ){
        if (Pos.vertice == 3){

         P2 = BuscaNoProx(P2);

         normaP0 = norma(P0,P2);
         if (dif = DifePont(P2, normaP0, m, n, cont)){
           // cria elemento p2, pos, p0
           DeseLinh(P2,P0);
           DeseLinh(P2,Pos);
           // Armazena elemento na tabela
           MontTabeElem(P2, P0, Pos);
           // Novo ponto tabela de pontos (dominio)
           MontTabePontTang(P0, 0, 0);

           MontTabePontTang(P2, 0, 0);

           PAuxP2 = P2;
         }
        }
      }
      if (157,5º < anguloant && anguloant <= 180º && 22.5 <
angulo && angulo <= 45  ){
      }
      if (135º < anguloant && anguloant <= 157,5º && 157,5º
< angulo && angulo <= 180º  ){
        P3.x = (P2.x + P2Ant.x)/2.0;
        P3.y = (P2.y + P2Ant.y)/2.0;

        P3 = BuscaNoProx(P3);

        if (status == 0){
          if (anguloprox < 40){
            normaP0 = norma(P0,P3);
            if (dif = DifePont(P3, normaP0, m, n, cont)){
              if (Ant.vertice != 3){
                // cria elemento p2, pos, p0
                DeseLinh(P3,Ant);
                DeseLinh(P3,P0);
                DeseLinh(P3,Pos);
                // Armazena elemento na tabela
                MontTabeElem(P3, P0, Ant);
                MontTabeElem(P3, Pos, P0);
                // Novo ponto tabela de pontos (dominio)
                if (pontoant == false){
                  MontTabePontTang(Ant, 0, 0);
                } else pontoant = false;

                MontTabePontTang(P3, 0, 0);

                //MontTabePontTang(P3, 0, 0);
                if (pontoP0 == true){
                   P0Aux = P3;
                   pontoP0 = false;
                }
              } else {
                DeseLinh(P2,P0);
                DeseLinh(P2,Pos);
                // Armazena elemento na tabela
                MontTabeElem(P2, Pos, P0);
                // Novo ponto tabela de pontos (dominio)
                MontTabePontTang(P0, 0, 0);
                MontTabePontTang(P2, 0, 0);
              }
            } else {
              MontTabePontTang(P0, 0, 0);
              difstatus = false;
            }
          } else {
            if (maxx < 0){
              P3 = PAuxP2;
              DeseLinh(P3,Pos);
              // Armazena elemento na tabela
              MontTabeElem(P3, Pos, P0);
              status = 0;
            } else {
              DeseLinh(P2,P0);
              DeseLinh(P2,Pos);
              // Armazena elemento na tabela
              MontTabeElem(P2, Pos, P0);
              // Novo ponto tabela de pontos (dominio)
              MontTabePontTang(P0, 0, 0);
              MontTabePontTang(P2, 0, 0);
            }
          }
        } else {
          if (P0.vertice != 3){
            P3 = PAuxP2;
            DeseLinh(P3,Pos);
            // Armazena elemento na tabela
            MontTabeElem(P3, Pos, P0);
            status = 0;
          } else {

            P2 = BuscaNoProx(P2);

            normaP0 = norma(P0,P2);
            if (dif = DifePont(P2, normaP0, m, n, cont)){
              // cria elemento p2, pos, p0
              DeseLinh(P2,P0);
              DeseLinh(P2,Pos);
              // Armazena elemento na tabela
              MontTabeElem(P2, P0, Pos);
              // Novo ponto tabela de pontos (dominio)
              MontTabePontTang(P0, 0, 0);

              MontTabePontTang(P2, 3, 0);

              //MontTabePontTang(P2, 3, 0);
              PAuxP2 = P2;
              status = 1;
            } else {
              MontTabePontTang(P0, 0, 0);
              difstatus = false;
            }
          }
        }
      }
      if (135º < anguloant && anguloant <= 157,5º && 135º <
angulo && angulo <= 157,5º  ){
        if (anguloprox >= 35){

          P2 = BuscaNoProx(P2);

          normaP0 = norma(P0,P2);
          if (dif = DifePont(P2, normaP0, m, n, cont)){
            // cria elemento p2, pos, p0
            DeseLinh(P2,P0);
            DeseLinh(P2,Pos);
            // Armazena elemento na tabela
            MontTabeElem(P2, Pos, P0);
            MontTabePontTang(P0, 0, 0);

            MontTabePontTang(P2, 0, 0);

            //MontTabePontTang(P2, 0, 0);
          } else {
            MontTabePontTang(P0, 0, 0);
            difstatus = false;
          }
        } else {
          if (difstatus == true){
```

```
          DeseLinh(PAuxP2,Pos);
          // Armazena elemento na tabela
          MontTabeElem(PAuxP2, Pos, P0);
          status = 1;
        } else {
          MontTabePontTang(P0, 0, 0);
          pontoant = true;
        }
      }
    }
    if (135º < anguloant && anguloant <= 157,5º && 112,5º
< angulo && angulo <= 135º  ){
      if (Pos.vertice != 3){

        P3 = BuscaNoProx(P2Pos);

        normaP0 = norma(P0,P2);
        if (dif = DifePont(P2, normaP0, m, n, cont)){
          // cria elemento p2, pos, p0
          if (cont != n && pontoP0 != true){
            anguloprox = BuscaAngulo(P0, P2, PAuxP2);

            if (anguloprox > 40){
               DeseLinh(P3,P0);
               DeseLinh(P3,Pos);
               DeseLinh(P3,Pos1);
               // Armazena elemento na tabela
               MontTabeElem(P3, P0, Pos);
               MontTabeElem(P3, Pos,Pos1);
               // Novo ponto tabela de pontos (dominio)
               MontTabePontTang(P0, 0, 0);

               MontTabePontTang(P3, 0, 0);
               PAuxP2 = P3;
            } else {
              DeseLinh(PAuxP2,P0);
              DeseLinh(PAuxP2,Pos);
              DeseLinh(PAuxP2,Pos1);
              // Armazena elemento na tabela
              //MontTabeElem(P3, Ant, P0);
              MontTabeElem(PAuxP2, P0, Pos);
              MontTabeElem(PAuxP2, Pos,Pos1);
            }

            Pos = Pos1;
            status = 1;
            cont++;
          }
        } else {
          MontTabePontTang(P0, 0, 0);
          difstatus = false;
        }
      } else {
        P2 = BuscaNoProx(P2);

        normaP0 = norma(P0,P2);
        if (dif = DifePont(P2, normaP0, m, n, cont)){
          // cria elemento p2, pos, p0
          DeseLinh(P2,P0);
          DeseLinh(P2,Pos);
          // Armazena elemento na tabela
          MontTabeElem(P2, P0, Pos);
          // Novo ponto tabela de pontos (dominio)
          MontTabePontTang(P0, 0, 0);

          MontTabePontTang(P2, 3, 0);

          PAuxP2 = P2;
          status = 1;
        } else {
          MontTabePontTang(P0, 0, 0);
          difstatus = false;
        }
      }
    }
    if (135º < anguloant && anguloant <= 157,5º && 90º <
angulo && angulo <= 112,5º  ){
      P3.x = (P2.x + P2Ant.x)/2.0;
      P3.y = (P2.y + P2Ant.y)/2.0;

      P3 = BuscaNoProx(P3);

      normaP0 = norma(P0,P3);
      //if (dif = DifePont(P3, normaP0, m, n, cont)){
        // cria elemento p2, pos, p0
      if (Pos.vertice != 3){
        anguloprox = BuscaAngulo(P0, P2, PAuxP2);

        if (status == 1 && anguloprox >= 35){
          DeseLinh(P3,P0);
          DeseLinh(P3,Pos);
          DeseLinh(P3,Pos1);
          // Armazena elemento na tabela
          MontTabeElem(P3, Ant, P0);
          MontTabeElem(P3, P0, Pos);
          MontTabeElem(P3, Pos,Pos1);
          // Novo ponto tabela de pontos (dominio)
          MontTabePontTang(P0, 0, 0);

          MontTabePontTang(P3, 0, 0);
          //MontTabePontTang(P3, 0, 0);

          PAuxP2 = P3;
          Pos = Pos1;
          cont++;
        }else if (status == 1 && cont == n-1 && pontoP0 ==
false && difstatus == true) {
          P3 = P2Pos;

          P3 = BuscaNoProx(P3);

          DeseLinh(P3,P0);
          DeseLinh(P3,Pos);
          DeseLinh(P3,Pos1);
          // Armazena elemento na tabela
          MontTabeElem(P3, P0, Pos);
          MontTabeElem(P3, Pos,Pos1);
          // Novo ponto tabela de pontos (dominio)
          MontTabePontTang(P0, 0, 0);
          MontTabePontTang(P3, 0, 0);

          PAuxP2 = P3;
          Pos = Pos1;
          cont++;
        } else {
          DeseLinh(PAuxP2,Pos);
          MontTabeElem(PAuxP2, P0, Pos);
          if (Pos.vertice != 3){
            DeseLinh(PAuxP2,Pos1);
            MontTabeElem(PAuxP2, Pos, Pos1);
            Pos = Pos1;
            cont++;
          }
          status = 0;
        }
      }
      if (Pos.vertice == 3){
        P3 = BuscaNoProx(P2);

        DeseLinh(P3,P0);
        DeseLinh(P3,Pos);
        // Armazena elemento na tabela
        MontTabeElem(P3, P0, Pos);
        // Novo ponto tabela de pontos (dominio)
        MontTabePontTang(P0, 0, 0);
        MontTabePontTang(P3, 0, 0);

        PAuxP2 = P3;
      }
      //else {
        //MontTabePontTang(P0, 0, 0);
```

```
        //difstatus = false;
       //}
     }
     if (135º < anguloant && anguloant <= 157,5º && 0 <
angulo && angulo <= 90º  ){
     }
     if (112,5º < anguloant && anguloant <= 135º && 157,5º
< angulo && angulo <= 180º  ){

       if (maxx < 0){
        if (anguloprox >= 5 && anguloprox <= 30 && status
== 0 || anguloprox > 90º){
          P3.x = (P2.x + P2Ant.x)/2.0;
          P3.y = (P2.y + P2Ant.y)/2.0;

          P3 = BuscaNoProx(P3);

          normaP0 = norma(P0,P3);
          if (dif = DifePont(P3, normaP0, m, n, cont)){

            anguloprox = BuscaAngulo(P0, P2, PAuxP2);

            if (anguloprox > 40){
              // cria elemento p2, pos, p0
              DeseLinh(P3,Ant);
              DeseLinh(P3,P0);
              DeseLinh(P3,Pos);
              // Armazena elemento na tabela
              MontTabeElem(P3, P0, Ant);
              MontTabeElem(P3, Pos, P0);
              // Novo ponto tabela de pontos (dominio)
              //MontTabePontTang(Ant, 0, 0);

              MontTabePontTang(P3, 0, 0);

              if (cont == 1){
                 P0Aux = P3;
                 pontoP0 = false;
              }
            } else {
              DeseLinh(PAuxP2,P0);
              DeseLinh(PAuxP2,Pos);
              // Armazena elemento na tabela
              MontTabeElem(PAuxP2, P0, Ant);
              MontTabeElem(PAuxP2, Pos, P0);
            }
          } else {
            MontTabePontTang(P0, 0, 0);
            difstatus = false;
          }
        } else {
          //normaP0 = norma(P0,P2);
          //if (dif = DifePont(P2, normaP0, m, n, cont)){
            // cria elemento p2, pos, p0
          if (difstatus == true){
            DeseLinh(PAuxP2,Pos);
            // Armazena elemento na tabela
            MontTabeElem(PAuxP2, Pos, P0);
            status = 1;
          } else {
            MontTabePontTang(P0, 0, 0);
            difstatus = false;
          }
        }
       } else {
        P2 = BuscaNoProx(P2);

        normaP0 = norma(P0,P2);
        if (dif = DifePont(P2, normaP0, m, n, cont)){
          // cria elemento p2, pos, p0
          DeseLinh(P2,P0);
          DeseLinh(P2,Pos);
          // Armazena elemento na tabela
          MontTabeElem(P2, Pos, P0);
          // Novo ponto tabela de pontos (dominio)
          MontTabePontTang(P0, 0, 0);

          MontTabePontTang(P2, 0, 0);

          //MontTabePontTang(P2, 0, 0);
        } else {
          MontTabePontTang(P0, 0, 0);
          difstatus = false;
        }
       }
     }
     if (112,5º < anguloant && anguloant <= 135º && 135º <
angulo && angulo <= 157,5º  ){
       if (status == 1){
        status = 0;
        if (Pos.vertice == 3 || P0.vertice == 3){
          P2 = BuscaNoProx(P2);

          normaP0 = norma(P0,P2);
          if (dif = DifePont(P2, normaP0, m, n, cont)){
            // cria elemento p2, pos, p0
            DeseLinh(P2,P0);
            DeseLinh(P2,Pos);
            // Armazena elemento na tabela
            MontTabeElem(P2, P0, Pos);
            // Novo ponto tabela de pontos (dominio)
            MontTabePontTang(P0, 0, 0);

            MontTabePontTang(P2, 3, 0);
            //MontTabePontTang(P2, 3, 0);

            PAuxP2 = P2;
            if (P0.vertice == 3) status = 0;
            else status = 1;
          } else {
            MontTabePontTang(P0, 0, 0);
            difstatus = false;
          }
        } else {
            //DeseLinh(PAuxP2,P0);
            DeseLinh(PAuxP2,Pos);
            // Armazena elemento na tabela
            MontTabeElem(PAuxP2, P0, Pos);
            // Novo ponto tabela de pontos (dominio)
            //MontTabePontTang(P0, 0, 0);

            //MontTabePontTang(P2, 3, 0);
            //MontTabePontTang(P2, 3, 0);

            //PAuxP2 = P2;
        }
       } else {
          P3.x = (P2.x + P2Ant.x)/2.0;
          P3.y = (P2.y + P2Ant.y)/2.0;
          P3 = BuscaNoProx(P3);

          normaP0 = norma(P0,P3);
          if (dif = DifePont(P3, normaP0, m, n, cont)){
            // cria elemento p2, pos, p0
            DeseLinh(P3,Ant);
            DeseLinh(P3,P0);
            DeseLinh(P3,Pos);
            // Armazena elemento na tabela
            MontTabeElem(P3, Ant, P0);
            MontTabeElem(P3, P0, Pos);
            // Novo ponto tabela de pontos (dominio)
            MontTabePontTang(Ant, 0, 0);

            MontTabePontTang(P3, 0, 0);

            PAuxP2 = P3;
            //Pos = Pos1;
            //cont++;
          } else {
            MontTabePontTang(P0, 0, 0);
            difstatus = false;
          }
```

```
      }
    }
    if (112,5º < anguloant && anguloant <= 135º && 112,5º
< angulo && angulo <= 135º  ){

      P3.x = (P2.x + P2Ant.x)/2.0;
      P3.y = (P2.y + P2Ant.y)/2.0;
      P3 = BuscaNoProx(P3);

      normaP0 = norma(P0,P3);
      if (dif = DifePont(P3, normaP0, m, n, cont)){

        anguloprox = BuscaAngulo(P0, P2, PAuxP2);

        if (anguloprox > 35){
          // cria elemento p2, pos, p0
          DeseLinh(P3,Ant);
          DeseLinh(P3,P0);
          DeseLinh(P3,Pos);
          DeseLinh(P3,Pos1);
          // Armazena elemento na tabela
          MontTabeElem(P3, Ant, P0);
          MontTabeElem(P3, P0, Pos);
          MontTabeElem(P3, Pos,Pos1);
          // Novo ponto tabela de pontos (dominio)
          MontTabePontTang(Ant, 0, 0);

          MontTabePontTang(P3, 0, 0);

          PAuxP2 = P3;
          Pos = Pos1;
          cont++;
        } else {
          DeseLinh(PAuxP2,Pos);
          MontTabeElem(PAuxP2, P0, Pos);
          //MontTabePontTang(P0, 0, 0);
        }
      } else {
        MontTabePontTang(P0, 0, 0);
        difstatus = false;
      }

    }
    if (112,5º < anguloant && anguloant <= 135º && 90º <
angulo && angulo <= 112,5º  ){
    }
    if (112,5º < anguloant && anguloant <= 135º && 0 <
angulo && angulo <= 90º  ){
    }
    if (100 < anguloant && anguloant <= 112,5º && 157,5º
< angulo && angulo <= 180º  ){

      anguloprox = BuscaAngulo(P0, P2, PAuxP2);

      if (anguloprox > 50){
        if (P0.vertice != 3){
          P3.x = (P2.x + P2Ant.x)/2.0;
          P3.y = (P2.y + P2Ant.y)/2.0;
          P3 = BuscaNoProx(P3);

          DeseLinh(P3,Ant);
          DeseLinh(P3,P0);
          DeseLinh(P3,Pos);
          // Armazena elemento na tabela
          MontTabeElem(P3, P0, Ant);
          MontTabeElem(P3, Pos, P0);
          // Novo ponto tabela de pontos (dominio)
          MontTabePontTang(Ant, 0, 0);

          MontTabePontTang(P3, 0, 0);

          if (cont == 1){
            P0Aux = P3;
            pontoP0 = false;
          }
        } else {
          DeseLinh(P2,P0);
          DeseLinh(P2,Pos);
          MontTabeElem(P2, P0, Pos);
          MontTabePontTang(P0, 0, 0);
          MontTabePontTang(P2, 0, 0);
        }
      } else {
        DeseLinh(PAuxP2,P0);
        DeseLinh(PAuxP2,Pos);
        // Armazena elemento na tabela
        MontTabeElem(PAuxP2, P0, Ant);
        MontTabeElem(PAuxP2, Pos, P0);
      }
    }
    if (90º < anguloant && anguloant <= 100 && 157,5º <
angulo && angulo <= 180º  ){
      P3 = PAuxP2;

      normaP0 = norma(P0,P2);
      if (dif = DifePont(P2, normaP0, m, n, cont)){
        if (P0.vertice != 3){
          // cria elemento p2, pos, p0
          DeseLinh(P3,Pos);
          // Armazena elemento na tabela
          MontTabeElem(P3, Pos, P0);
        } else {
           // cria elemento p2, pos, p0
          DeseLinh(P2,P0);
          DeseLinh(P2,Pos);
          // Armazena elemento na tabela
          MontTabeElem(P2, Pos, P0);
          MontTabePontTang(P0, 0, 0);

          MontTabePontTang(P2, 0, 0);
        }
      } else {
        DeseLinh(Ant,Pos);
        MontTabeElem(Ant, Pos, P0);
        //MontTabePontTang(P0, 0, 0);
        difstatus = false;
      }
    }
    if (90º < anguloant && anguloant <= 112,5º && 135º <
angulo && angulo <= 157,5º  ){
      anguloprox = BuscaAngulo(P0, PAuxP2, Pos);

      if (P0.vertice != 3){
        if (anguloprox > 50 && anguloprox < 70){
          if (pontoP0 == false && cont < n){
            if (pontoant == true){
              P3.x = (P2.x + P2Ant.x)/2.0;
              P3.y = (P2.y + P2Ant.y)/2.0;
              P3 = BuscaNoProx(P3);

              DeseLinh(P3,Ant);
              DeseLinh(P3,P0);
              DeseLinh(P3,Pos);
              // Armazena elemento na tabela
              MontTabeElem(P3, P0, Ant);
              MontTabeElem(P3, Pos, P0);
              // Novo ponto tabela de pontos (dominio)
              //MontTabePontTang(Ant, 0, 0);

              MontTabePontTang(P3, 0, 0);
              pontoant = false;
            } else {
              DeseLinh(PAuxP2,Pos);
              MontTabeElem(P2, Pos, P0);
            }
          }
        } else {
          if (pontoP0 == true && cont < n){
            MontTabePontTang(P0, 0, 0);
            difstatus = false;
          } else if (cont < n){
            DeseLinh(PAuxP2,Pos);
```

```
            MontTabeElem(PAuxP2, P0, Pos);
          }
        }
      }
      if (P0.vertice == 3){
        if (status == 0){
          P3 = BuscaNoProx(P2);

          DeseLinh(P3,P0);
          DeseLinh(P3,Pos);
          // Armazena elemento na tabela
          MontTabeElem(P3, P0, Pos);
          // Novo ponto tabela de pontos (dominio)
          MontTabePontTang(P0, 0, 0);
          MontTabePontTang(P3, 0, 0);

          PAuxP2 = P3;
        } else {
          P3 = PAuxP2;

          DeseLinh(P3,P0);
          DeseLinh(P3,Pos);
          // Armazena elemento na tabela
          MontTabeElem(P3, P0, Pos);
          // Novo ponto tabela de pontos (dominio)
          //MontTabePontTang(P0, 0, 0);
          //MontTabePontTang(P3, 0, 0);

          PAuxP2 = P3;
        }
      }
    }
    if (90º < anguloant && anguloant <= 112,5º && 112,5º
< angulo && angulo <= 135º  ){
        if (status == 1){
          P3.x  = (P2.x + P2Ant.x)/2.0;
          P3.y  = (P2.y + P2Ant.y)/2.0;

          P3 = BuscaNoProx(P3);

          DeseLinh(P3,Ant);
          DeseLinh(P3,P0);
          DeseLinh(P3,Pos);
          // Armazena elemento na tabela
          MontTabeElem(P3, P0, Pos);
          // Novo ponto tabela de pontos (dominio)
          //MontTabePontTang(Ant, 0, 0);
          MontTabePontTang(P3, 0, 0);

          PAuxP2 = P3;
          status = 1;
        } else {
          MontTabePontTang(P0, 0, 0);
          difstatus = false;
        }
    }
    if (90º < anguloant && anguloant <= 112,5º && 90º <
angulo && angulo <= 112,5º  ){
        if (status == 1){
          P3.x  = (P2.x + P2Ant.x)/2.0;
          P3.y  = (P2.y + P2Ant.y)/2.0;

          P3.x  = (P3.x + P2Pos.x)/2.0;
          P3.y  = (P3.y + P2Pos.y)/2.0;

          P3 = BuscaNoProx(P3);

          DeseLinh(P3,Ant);
          DeseLinh(P3,P0);
          DeseLinh(P3,Pos);
          // Armazena elemento na tabela
          MontTabeElem(P3, P0, Pos);
          // Novo ponto tabela de pontos (dominio)
          //MontTabePontTang(Ant, 0, 0);
          MontTabePontTang(P3, 0, 0);

          PAuxP2 = P3;
          status = 1;
        } else {
          MontTabePontTang(P0, 0, 0);
          difstatus = false;
        }
    }
    if (90º < anguloant && anguloant <= 112,5º && 0 <
angulo && angulo <= 90º  ){
        DeseLinh(P0,Pos1);
        MontTabeElem(P0, Pos, Pos1);
        MontTabePontTang(P0, 0, 0);
        Pos = Pos1;
        cont++;
        PAuxP2 = P0;
        //MontTabePontTang(P0, 0, 0);
        difstatus = false;
    }
    if (67,5º < anguloant && anguloant <= 90º && 157,5º <
angulo && angulo <= 180º  ){
      // cantos internos de noventa
      if (P0.vertice == 3){
        // encontrar bissetriz
        P3 = GeraBissetriz(P0, Ant, Pos, 1);

        // tamanho do eixo da bossetriz gerada
        normaP1 = norma(Pos,P3);
        // altura máxima do triangulo equilatero = eixo AB *
0.86
        normaP0 = norma(Pos,P0) * 0.86;
        // nova coordenada para preenchimento
        P1.x = P0.x - P3.x;
        P1.y = P0.y - P3.y;
        // encontra os novos pontos em relação a norma
        P3.x = P0.x - (P1.x * normaP0) / normaP1;
        P3.y = P0.y - (P1.y * normaP0) / normaP1;

        P3 = BuscaNoProx(P3);
        P2 = BuscaNoProx(P2);

        normaP0 = norma(P0,P3);
        if (dif = DifePont(P3, normaP0, m, n, cont)){
          // BIssetriz
          DeseLinh(P0,P3);
          DeseLinh(P0,P2);
          DeseLinh(Pos,P2);

          MontTabePontTang(P0, 0, 0);
          MontTabePontTang(P3, 3, 0);
          //MontTabePontTang(P3, 3, 0);

          MontTabePontTang(P0, 0, 0);
          MontTabePontTang(P2, 0, 0);
          //MontTabePontTang(P2, 0, 0);

          MontTabeElem(P2, Pos, P0);
        } else {
          MontTabePontTang(Pos, 0, 0);
          difstatus = false;
        }
      }
      /// cantos em noventa  externos
      if (P0.vertice != 3){

        anguloprox = BuscaAngulo(P0, P2, PAuxP2);

        if (anguloprox == 0 || status == 0){
          // divide linha
          P2.x = (Ant.x + Pos.x)/2.0;
          P2.y = (Ant.y + Pos.y)/2.0;
          P2 = BuscaNoProx(P2);
          // cria elemento ant, p2, p0
          // e p2, p0, pos
          DeseLinh(Ant,Pos);
          // Armazena elemento na tabela
          MontTabeElem(Pos, P0, Ant);
```

```
          // Novo ponto tabela de pontos (dominio)
          MontTabePontTang(Ant, 0, 0);
          DeseLinh(P2,P0);
          MontTabeElem(P2, Pos, P0);
          // Novo ponto tabela de pontos (dominio)
          MontTabePontTang(P2, 1, 0);
        } else if (anguloprox >= 5 && anguloprox <= 15 &&
status == 1){
          P2.x = (P2.x + P2Ant.x)/2.0;
          P2.y = (P2.y + P2Ant.y)/2.0;
          P2 = BuscaNoProx(P2);
          // cria elemento ant, p2, p0
          // e p2, p0, pos
          DeseLinh(P2,Ant);
          DeseLinh(P2,P0);
          DeseLinh(P2,Pos);
          // Armazena elemento na tabela
          MontTabeElem(P2, P0, Ant);
          MontTabeElem(P2, Pos, P0);
          // Novo ponto tabela de pontos (dominio)
          MontTabePontTang(Ant, 0, 0);
          MontTabePontTang(P2, 0, 0);

        } else {
          DeseLinh(Pos,PAuxP2);
          MontTabeElem(PAuxP2, Pos, P0);
        }
       }
     }
     if (67,5º < anguloant && anguloant <= 90º && 112,5º <
angulo && angulo <= 135º  ){
       if (anguloant > 50 && anguloant < 86){
         DeseLinh(Ant,Pos);
         MontTabeElem(Ant, Pos, P0);
         MontTabePontTang(Ant, 0, 0);
         difstatus = false;
       }
     }
     if (67,5º < anguloant && anguloant <= 90º && 90º <
angulo && angulo <= 112,5º  ){
         DeseLinh(Ant,Pos);
         MontTabeElem(Ant, Pos, P0);
         MontTabePontTang(P0, 0, 0);
         difstatus = false;
     }
     if (67,5º < anguloant && anguloant <= 90º && 67,5º <
angulo && angulo <= 90º  ){
         DeseLinh(Ant,Pos);
         MontTabeElem(Ant, Pos, P0);
         MontTabePontTang(Ant, 0, 0);
         difstatus = false;
     }
     if (0 < anguloant && anguloant <= 67,5º && 157,5º <
angulo && angulo <= 180º  ){
       if (P0.vertice == 3){
         // encontrar bissetriz
         P3 = GeraBissetriz(P0, Ant, Pos, 1);
         // tamanho do eixo da bossetriz gerada
         normaP1 = norma(Pos,P3);
         // altura máxima do triangulo equilatero = eixo AB *
0.86
         normaP0 = norma(Pos,P0) * 0.86;
         // nova coordenada para preenchimento
         P1.x = P0.x - P3.x;
         P1.y = P0.y - P3.y;
         // encontra os novos pontos em relação a norma
         P3.x = P0.x - (P1.x * normaP0) / normaP1;
         P3.y = P0.y - (P1.y * normaP0) / normaP1;
         P3 = BuscaNoProx(P3);
         // segunda bissetriz
         // encontrar bissetriz
         P4 = GeraBissetriz(P0, PAuxP2, P3, 0);
         // tamanho do eixo da bossetriz gerada
         normaP1 = norma(Pos,P4);
         // altura máxima do triangulo equilatero = eixo AB *
0.86
         normaP0 = norma(Pos,P0) * 0.86;
         // nova coordenada para preenchimento
         P1.x = P0.x - P4.x;
         P1.y = P0.y - P4.y;
         // encontra os novos pontos em relação a norma
         P4.x = P0.x - (P1.x * normaP0) / normaP1;
         P4.y = P0.y - (P1.y * normaP0) / normaP1;

         P4 = BuscaNoProx(P4);
         // Terceira bissetriz
         // encontrar bissetriz
         P5 = GeraBissetriz(P0, P3, P2, 0);
         // tamanho do eixo da bossetriz gerada
         normaP1 = norma(Pos,P5);
         // altura máxima do triangulo equilatero = eixo AB *
0.86
         normaP0 = norma(Pos,P0) * 0.86;
         // nova coordenada para preenchimento
         P1.x = P0.x - P5.x;
         P1.y = P0.y - P5.y;
         // encontra os novos pontos em relação a norma
         P5.x = P0.x - (P1.x * normaP0) / normaP1;
         P5.y = P0.y - (P1.y * normaP0) / normaP1;

         P5 = BuscaNoProx(P5);
         P2 = BuscaNoProx(P2);
         // BIssetriz
         //DeseLinh(P0,P4);
         // BIssetriz
         DeseLinh(P0,P3);
         // BIssetriz
         //DeseLinh(P0,P5);

         normaP0 = norma(P0,P3);
         if (dif = DifePont(P3, normaP0, m, n, cont)){
           DeseLinh(P0,P2);
           DeseLinh(Pos,P2);
           //MontTabePontTang(P0, 0, 0);
           //MontTabePontTang(P4, 3, 0);
           //MontTabePontTang(P4, 3, 0);

           MontTabePontTang(P0, 0, 0);
           MontTabePontTang(P3, 3, 0);
           //MontTabePontTang(P3, 3, 0);

           //MontTabePontTang(P0, 0, 0);
           //MontTabePontTang(P5, 3, 0);
           //MontTabePontTang(P5, 3, 0);

           MontTabePontTang(P0, 0, 0);
           MontTabePontTang(P2, 0, 0);
           //MontTabePontTang(P2, 0, 0);

           MontTabeElem(P2, Pos, P0);
         } else {
            MontTabePontTang(Pos, 0, 0);
            difstatus = false;
         }
       } else {
          //alteracao em 09/04/2003
          DeseLinh(Ant,Pos);
          MontTabePontTang(Ant, 0, 0);
          //MontTabePontTang(Pos, 0, 0);
          difstatus = false;
       }
       status = 0;
     }
     if (0 < anguloant && anguloant <= 67,5º && 0 < angulo
&& angulo <= 157,5º  ){
         DeseLinh(Ant,Pos);
         MontTabeElem(Ant, Pos, P0);
         MontTabePontTang(Ant, 0, 0);
     }

     P0 = Pos;
```

```
    Pos = buscaproximo(cont, n, m);
    Pos1 = buscaproximo(cont+1, n, m);
    Ant = buscaanterior(cont, n, m);
    cont++;
    if (cont == n)Pos1 = buscaproximo(0, n, m);
  }
}
```

B.3 Algoritmo para Fechamento da Fronteira

```
//---------------------------------------------------------------------------
void TMDIChild::FechFron(){
  MyPoint P0,Pos, Ant, P1,P3;
  int l,m,n;
  double maxx, normaP1, normaP0;
  int angulo, status;
  float p1x,p1y,p2x,p2y;

  if (difstatus == true){
    // subdivisão dos vértices em relação ao dominio
principal
    l = 6 * contint;
    m = l - 6;
    n = contlin;

    // gera um nova entrada na tabela de pontos ao
encontrar um vértice concavo
    contint++;
    contlin = 0;
    contcol = contcol + 6;
    StringGrid2->ColCount = StringGrid2->ColCount + 6;

    int i = 0;

    status = 0;

    while (i < n){
      P0.x = StrToFloat(StringGrid2->Cells[m+1][i]);
      P0.y = StrToFloat(StringGrid2->Cells[m+2][i]);
      Pos = buscaproximo(i,n,m);
      Ant = buscaanterior(i,n,m);

      maxx = ((P0.x - Pos.x)*(Ant.y - Pos.y))-((Ant.x -
Pos.x)*(P0.y - Pos.y));

      if (maxx < -0.001 ){
        p1x = Ant.x - P0.x;
        p1y = Ant.y - P0.y;
        p2x = Pos.x - P0.x;
        p2y = Pos.y - P0.y;

        angulo = (int) EncontraAngulo(p1x,p1y,p2x,p2y);

        if (angulo <= 87 || (angulo >= 90º && angulo < 100)){
          DeseLinh(Ant,Pos);
          MontTabePontTang(Pos, Pos.vertice, Pos.angulo);
          MontTabeElem(Ant,P0,Pos);
          status = 1;
        } else if ((angulo > 87 && angulo <= 160) && status
== 0) {
          // encontrar bissetriz
          P3 = GeraBissetriz(P0, Ant, Pos, 0);
          // tamanho do eixo da bossetriz gerada
          normaP1 = norma(Pos,P3);
          // altura máxima do triangulo equilatero = eixo AB *
0.86
          normaP0 = norma(Pos,P0) * 0.86;
          // nova coordenada para preenchimento
          P1.x = P0.x - P3.x;
          P1.y = P0.y - P3.y;
          // encontra os novos pontos em relação a norma
          P3.x = P0.x - (P1.x * normaP0) / normaP1;
          P3.y = P0.y - (P1.y * normaP0) / normaP1;
          // BIssetriz
          DeseLinh(P0,P3);

          DeseLinh(Ant,P3);

          DeseLinh(P3,Pos);

          MontTabePontTang(P3,3, 0);
          MontTabePontTang(Pos, Pos.vertice, Pos.angulo);
          MontTabeElem(Ant,P0,P3);
          MontTabeElem(P3,P0,Pos);
          status = 1;
        } else status = 0;
      } else {
        status = 0;
      }
      i++;
    }


    // subdivisão dos vértices 9em relação ao dominio
principal
    l = 6 * contint;
    m = l - 6;
    n = contlin;

    // gera um nova entrada na tabela de pontos ao
encontrar um vértice concavo
    contint++;
    contlin = 0;
    contcol = contcol + 6;
    StringGrid2->ColCount = StringGrid2->ColCount + 6;

    i = 0;

    status = 0;

    while (i < n){
      P0.x = StrToFloat(StringGrid2->Cells[m+1][i]);
      P0.y = StrToFloat(StringGrid2->Cells[m+2][i]);
      P0.vertice = StrToFloat(StringGrid2->Cells[m+3][i]);
      Pos = buscaproximo(i,n,m);
      Ant = buscaanterior(i,n,m);

      maxx = ((P0.x - Pos.x)*(Ant.y - Pos.y))-((Ant.x -
Pos.x)*(P0.y - Pos.y));

      if (maxx < -0.001 ){

        p1x = Ant.x - P0.x;
        p1y = Ant.y - P0.y;
        p2x = Pos.x - P0.x;
        p2y = Pos.y - P0.y;

        angulo = (int) EncontraAngulo(p1x,p1y,p2x,p2y);

        if (status == 0 && angulo > 38 && angulo <= 85){
          DeseLinh(Ant,Pos);
          MontTabeElem(Ant,P0,Pos);
          status = 1;
        } else {
           MontTabePontTang(P0, P0.vertice, 0);
           status = 0;
        }
          //MontTabePontTang(Pos, 0, 0);
        //} else if ((P0.vertice == 0 || P0.vertice == 3) &&
(Pos.vertice == 0 || Pos.vertice == 3)){
          //MontTabePontTang(Pos, Pos.vertice, 0);
        //}
      } else {
         MontTabePontTang(P0, P0.vertice, 0);
         status = 0;
```

```
    }
    i++;
  }
```

B.4 Algoritmo para Costura da Malha

```
void TMDIChild::Costura(){
  MyPoint P0, Pos, Pos1, Ant, P2;
  int l,m,n, cont, angulo, anguloant, status, anguloprox;
  double maxx;
  float p1x,p1y,p2x,p2y;
  double area;
  int statusaux = 0;
  int i = 0, j =0;
  bool pontoP0;

  // subdivisão dos vértices em relação ao dominio
principal
  l = 6 * contint;
  m = l - 6;
  n = contlin;

  area = AreaPoliIrre(m,n);

  while (area > 0){

    // gera um nova entrada na tabela de pontos ao
encontrar um vértice concavo
    contint++;
    contlin = 0;
    contcol = contcol + 6;
    StringGrid2->ColCount = StringGrid2->ColCount + 6;

    P0.x = StrToFloat(StringGrid2->Cells[m+1][0]);
    P0.y = StrToFloat(StringGrid2->Cells[m+2][0]);
    P0.vertice = StrToFloat(StringGrid2->Cells[m+3][0]);
    P0.angulo = StrToFloat(StringGrid2->Cells[m+4][0]);
    Pos = buscaproximo(0,n,m);
    Ant = buscaanterior(0,n,m);
    Pos1 = buscaproximo(1,n,m);

    cont = 1;

    status = 0;

    pontoP0 = false;

    while (cont <= n){

      // encontrar o angulo
      // posiciona em coordenadas locais para calculo do
angulo
      p1x = Ant.x - P0.x;
      p1y = Ant.y - P0.y;
      p2x = Pos.x - P0.x;
      p2y = Pos.y - P0.y;

      anguloant = (int) EncontraAngulo(p1x,p1y,p2x,p2y);

      p1x = P0.x - Pos.x;
      p1y = P0.y - Pos.y;
      p2x = Ant.x - Pos.x;
      p2y = Ant.y - Pos.y;

      angulo = (int) EncontraAngulo(p1x,p1y,p2x,p2y);

      // encontrar o angulo
      // posiciona em coordenadas locais para calculo do
angulo
      p1x = P0.x - Ant.x;
      p1y = P0.y - Ant.y;
      p2x = Pos.x - Ant.x;
      p2y = Pos.y - Ant.y;

      anguloprox = (int) EncontraAngulo(p1x,p1y,p2x,p2y);

      // area do triangulo se for positiva é um tirangulo
interno e se for
      // negativa é externo.
      maxx = ((P0.x - Pos.x)*(Ant.y - Pos.y))-((Ant.x -
Pos.x)*(P0.y - Pos.y));


      if (maxx < 0){
        if (anguloant > 30 && anguloant <= 120){
          if (angulo > 26 && angulo <= 105){
            if (anguloprox > 25 && anguloprox <= 110){
              if (j == 0){
                /// valida angulo formado entre o ponto gerado e
o proximo no
                // encontrar o angulo
                // posiciona em coordenadas locais para calculo
do angulo
                p1x = Ant.x - Pos.x;
                p1y = Ant.y - Pos.y;
                p2x = Pos1.x - Pos.x;
                p2y = Pos1.y - Pos.y;

                anguloprox = (int)
EncontraAngulo(p1x,p1y,p2x,p2y);

                //if (cont == 1) pontoP0 = true;

                if (anguloprox > 30 && cont != 1){ //pontoP0 ==
false){
                  DeseLinh(Ant, Pos);

                  MontTabeElem(Ant,P0,Pos);
                  status = 1;
                  j = 1;
                  //pontoP0 = true;
                } else {
                  MontTabePontTang(P0, 0, 0);
                  j = 0;
                }
              } else {
                MontTabePontTang(P0, 0, 0);
                j = 0;
              }
            }else {
              MontTabePontTang(P0, 0, 0);
              j = 0;
            }
          }else {
            MontTabePontTang(P0, 0, 0);
            j = 0;
          }
        } else {
          MontTabePontTang(P0, 0, 0);
          j = 0;
        }
      } else {
        MontTabePontTang(P0, 0, 0);
        j = 0;
      }

      P0 = Pos;
      Pos = buscaproximo(cont, n, m);
      Ant = buscaanterior(cont, n, m);
      Pos1 = buscaproximo(cont+1, n, m);
      cont++;
      if (cont == n)Pos1 = buscaproximo(0, n, m);
    }

    //if (status == 0) MontTabePontTang(Pos, 0, 0);
  } // fim if dif == true

}
```

```
l = 6 * contint;
m = l - 6;
n = contlin;

//if (pontoP0 == true) status = 0;

if (status == 0) {
    ReorganizaNo(l,m,n);
    l = 6 * contint;
    m = l - 6;
    n = contlin;

    ReorganizaNo(l,m,n);
    l = 6 * contint;
    m = l - 6;
    n = contlin;
    statusaux++;
  }

  i++;
  if (statusaux > 2) break;

  area = AreaPoliIrre(m,n);

  if (n <= 3) break;
 }
}
```