José Fernando Mantovani Micali

Modelagem de Figuras Planas e Geração de Malhas Usando Quadtree

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

São Paulo
1994

José Fernando Mantovani Micali

Modelagem de Figuras Planas e Geração de Malhas Usando Quadtree

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

Área de Concentração:
Engenharia de Estruturas

Orientador:
João Cyro André

São Paulo
1994

A essas duas pessoas que tanto amo,
meus pais, Fernando e Edir.

Agradecimentos

Devo creditar grande parcela de responsabilidade pela minha trajetória profissional ao Prof. João Cyro André. Há exatos dez anos, então meu professor e orientador da graduação, apresentou-me um mundo rico de caminhos que me excitavam percorrer. De lá para cá, grande tem sido a oportunidade de estar ao seu lado e conferir suas qualidades. O Prof. João Cyro tem esse grande poder de catálise, aglutinando as pessoas em torno de objetivos comuns.

Ao Prof. José Antônio Lerosa de Siqueira - pessoas inteligentes descobrem logo caminhos. Legal tê-los compartilhado comigo.

Ao Prof. José Antônio Verderesi - a paixão pela Matemática envolve e faz sonhar. Difícil é reestabelecer a rota.

Ao Prof. Benedikt Kepczynski - esse emaranhado de consoantes traduz grande conhecimento da Língua, por incrível que pareça, Portuguesa.

Aos amigos do Laboratório de Mecânica Computacional da Escola Politécnica da USP, Mário e Januário, sempre prestativos.

Aos companheiros da Fundação Paulista de Tecnologia e Educação, mantenedora da Escola de Engenharia de Lins, em especial à equipe do Centro de Computação Gráfica : Adalberto (Binha), Henrique ('s), Fábio (Rato), Rogério, Roberto - grande força.

À Marina e à Sandra.

E ao pessoal da gráfica da FPTE, os últimos a quem agradecerei também pessoalmente nesse domingo que para eles seria de descanso.

José Fernando Mantovani Micali
Outono, 1994.

Sumário

Resumo

A modelagem geométrica e a geração automática de malhas são componentes básicos de um ambiente informatizado para modelagem de elementos finitos utilizando análise adaptativa.

Neste trabalho, apresentam-se os conceitos utilizados no desenvolvimento do GAMA2D, que é composto por um modelador de figuras planas e por um gerador automático de malhas de elementos finitos. O modelador foi desenvolvido com base no Método de Representação pela Fronteira. A geração de malhas é realizada empregando-se o método do Quadtree. Neste processo os elementos de geometria e os parâmetros de controle de malha, fornecidos pelo usuário, são utilizados para discretizar o domínio em um conjunto de quadrantes que são posteriormente transformados em elementos finitos.

O desenvolvimento deste software de Computação Gráfica observou um projeto completo contendo as diversas etapas - conceitual, semântica, sintática e léxica - que se consideram fundamentais para a sua qualidade. A portabilidade foi alcançada através da utilização da linguagem C e da criação de camadas entre os algoritmos da aplicação e a estrutura de dados, e entre os algoritmos da aplicação e as rotinas gráficas.

Abstract

Geometric modeling and automatic mesh generation are basic components of a computational environment to develop adaptative finite element analysis.

In this work, one presents all needed concepts to develop the software GAMA2D, which is composed of a geometric planar modeler and an automated finite element mesh generator. The modeler is based on boundary representation method. The Quadtree method is employed to generate meshes. Geometrical elements and mesh control parameters, defined by users, are employed to discretize the domains in quadrants. In the sequence they are changed in finite elements.

The implementation of this graphical computation software followed a complete design. In order to guarantee high quality the developed design contains conceptual, semantic, syntactic, and lexical steps. Portability is obtained by using C-language and creating interfaces between algorithms and data structures, and between algorithms and graphical routines.

Capitulo 1

Introdução

Dentro da engenharia, pertencente ao ramo das ciências fatuais, raciocinamos com base em entidades abstratas que são associadas às entidades reais, através de regras de representação. Essa abstração nos permite, através de simplificações, representar o mundo dos fatos, através de modelos teóricos. Por exemplo, através de simplificações, podemos modelar a geometria, a reologia e o comportamento mecânico de uma viga-parede através de uma chapa elástica.

O modelo teórico pode, por sua vez, através de discretização, ser transformado em um modelo numérico. Por exemplo, para determinar o campo de deslocamentos da viga-parede pode-se buscar a função que minimiza o funcional energia potencial total. Como um funcional pode ser entendido como uma função de infinitas variáveis [22], podemos, através de discretização, transformar o funcional em uma função de um número finito de deslocamentos generalizados e o problema de estacionarizar um funcional se transforma em um problema de buscar o mínimo de uma função, podendo ser resolvido pelo Cálculo Diferencial, dando origem ao denominado Método Variacional Direto.

O Método Variacional Direto mais utilizado dentro da Mecânica das Estruturas é o Método dos Elementos Finitos. No Método dos Elementos Finitos o

domínio é subdividido em subdomínios denominados elementos que se ligam através dos nós, onde são definidos os parâmetros generalizados a serem determinados.

Satisfeitas certas condições de convergência, as respostas do Método dos Elementos Finitos tornam-se tanto mais precisas quanto maior o número de variáveis adotadas, sendo que no limite alcançaríamos a solução exata. Ou seja, quanto maior o número de pontos ou quanto mais discretizada a malha mais próximo estaremos da solução exata. Logicamente por se tratar de um método numérico e aproximado deve-se considerar que existe um número finito de elementos que leve à solução do problema dentro da precisão desejada e do tempo esperado.

Tradicionalmente o Departamento de Estruturas e Fundações tem tratado de temas ligados à Teoria das Estruturas, tanto na formulação como no tratamento numérico por elementos finitos sem que se desenvolvam atividades de pesquisa associadas a pré e pós-processadores.

Observa-se, por outro lado, na área de análise estrutural cada vez mais o uso de recursos de Computação Gráfica que torna mais amigável e confiável a entrada de dados e permite maior profundidade de análise a saída de resultados. Pode-se mesmo dizer que um programa de análise não pode prescindir de pré e pós-processador que utilize os recursos de hardware e software disponíveis no mercado.

Dentro desse contexto procurou-se no âmbito do Laboratório de Mecânica Computacional do Departamento de Estruturas e Fundações, então sob coordenação do Prof. Siqueira, tratar inicialmente da utilização de recursos de Computação Gráfica para desenvolver um pré-processador.

Os objetivos principais desse trabalho são o de apresentar a base conceitual e, a partir dela, gerar um sistema que possibilite a modelagem geométrica de entidades que pelas regras de representação possam ser tratadas como um modelo teórico plano e permitir que esse modelo plano contínuo seja discretizado em elementos finitos, através de um processo de geração automática da malha.

O sistema deve permitir que a geometria do modelo seja introduzida de forma amigável e interativa, utilizando-se de conceitos de Computação Gráfica e, a seguir, baseado nas densidades de malha desejada pelo usuário, que seja gerada automaticamente, a malha de elementos finitos.

Utilizando-se desse pré-processamento evita-se a tediosa tarefa de definir a geometria e as coordenadas da malha com possibilidade extremamente menor de cometer erros e gerar uma malha não válida, além de permitir a visualização imediata do modelo gerado. Essa automatização dá ao engenheiro a possibilidade de testar diferentes modelos estruturais, diferentes malhas e dedicar-se a tarefas menos rotineiras e mais analíticas.

Este texto tem o claro objetivo de servir de base conceitual para o pré-processador gráfico desenvolvido e está dividido da seguinte maneira : o **capítulo 2** contém a teoria necessária para se desenvolver um modelador de figuras planas, tratando então dos elementos topológicos e das condições de invariância no plano. No **capítulo 3** apresenta-se a estrutura de dados do modelador, bem como uma rápida descrição de modelagem de sólidos usando operadores de Euler. No **capítulo 4** uma classificação dos métodos de geração de malha é mostrada, situando o método do quadtree. Logo após são apresentados os conceitos básicos desse método. No **capítulo 5** trata-se do método de geração de malhas escolhido, apresentando todos os algoritmos e a estrutura de dados. No **capítulo 6** elaboram-se os conceitos para construção de um projeto de interface gráfica utilizando exemplos extraídos do GAMA2D. No **capítulo 7** alguns exemplos de domínios discretizados são mostrados. No **capítulo 8** são apresentadas as conclusões finas. O **apêndice** traz um manual de utilização passo-a-passo do GAMA2D, construindo o exemplo básico utilizado durante todo o texto para explanar os conceitos de geração de malha.

Um sistema automático completo de análise por elementos finitos se compõe dos seguintes módulos : modelagem geométrica, geração automática de malha, análise por elementos finitos, estimativa de erros, adaptação da geometria, atualização da malha e pós-processador gráfico. O presente trabalho se concentra nos dois primeiros itens, preocupando-se em desenvolver os conceitos e as ferramentas que dêem suporte e sejam facilmente integráveis aos módulos seguintes, que devem ser tratados por um projeto global em andamento no Laboratório de Mecânica Computacional.

Como o texto aborda temas ainda não consagrados pela literatura de língua portuguesa, é inevitável que se deparem com termos técnicos, ou classificações, de difícil tratamento. Nessas situações, sem querer criar neologismos, mas para clareza do texto usou-se, o termo em inglês com padrão itálico. Palavras como software e hardware já foram incorporadas à lingua portuguesa através de empréstimo da língua inglesa. Outros termos do cotidiano da engenharia e ainda não assimilados pela língua, como o verbo discretizar, foram usados sem nenhuma distinção em especial.

Capítulo 2

Modelagem de figuras planas

Neste capítulo apresentam-se os conceitos básicos para a modelagem de figuras planas, utilizando-se o método de representação pela fronteira. É apresentada a terminologia dos elementos topológicos empregados, bem como a equação que estabelece as condições de invariância de modelos planos, contemplando as possibilidades de apresentarem furos e múltiplos componentes. Por fim estudam-se as condições de invariância do modelo.

O modelo de representação pela fronteira, utilizado nesse trabalho, é uma adaptação para o plano do B-rep (Boundary representation), originalmente construído para modelagem de sólidos. A representação pela fronteira descreve o domínio através de uma fronteira orientada composta de arestas, vértices e faces.

Definidas as operações topológicas a serem utilizadas, através de operadores de pode-se construir, utilizando-se o conjunto de arestas, vértices e faces, domínios que mantenham as condições de invariância preservadas. Operadores podem também ser implementados de modo a estabelecer uma interface de alto nível entre os algoritmos e a estrutura de dados do sistema.

2.1 O modelo de representação pela fronteira

Segundo Hoffman [09], no estudo da representação de sólidos são encontrados dois modelos principais: o C.S.G. (Constructive Solid Geometry) e o B-rep (Boundary-representation).

No C.S.G. um sólido é representado por um conjunto de expressões booleanas definindo operações entre modelos sólidos simples, denominados primitivos. Sendo assim, tanto a superfície quanto o interior do objeto são definidos implicitamente.

O método de representação pela fronteira baseia-se no fato de que o domínio pode ser representado de forma não ambígua através da descrição da sua fronteira. Essa fronteira deve ser orientada de modo que seja possível identificar, para qualquer ponto, se ele está ou não contido no domínio por ela definido. Esta descrição envolve duas etapas, a descrição topológica, estabelecendo as relações de adjacências e a orientação dos vértices, arestas e faces, e a representação analítica, que insere esses elementos num plano coordenado, introduzindo-lhes atributos métricos.

A descrição topológica especifica vértices, arestas e faces, indicando suas incidências e as relações de adjacências. A representação analítica especifica as coordenadas dos pontos relacionados aos vértices e as equações das curvas ligadas às arestas.

2.2 Terminologia

Serão estudadas regiões planares que podem ser decompostas em vértices, arestas e faces. É importante notar que inerentes às definições dos elementos topológicos estão as restrições que reduzem o universo das figuras àquelas de interesse a este trabalho, estabelecendo o campo de aplicação do modelador de figuras planas desenvolvido.

Um **vértice** num plano é simplesmente um ponto.

Uma **aresta** é um segmento de curva no plano, tal que ele não se auto-intercepte, partindo de um vértice e chegando a um vértice, tal como ilustra a figura 2.1.

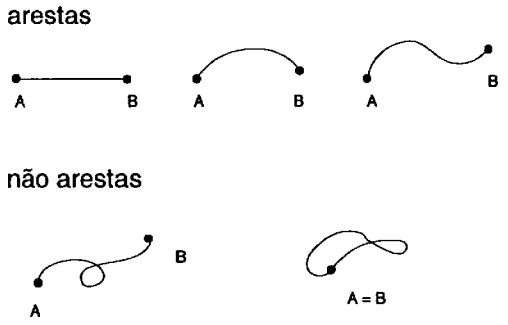


Figura 2.1 Exemplos de arestas e não arestas

A **fronteira** (Fr) de uma aresta é constituída por suas extremidades. Por exemplo, na figura 2.2 temos:

$$Fr(A_1A_2) = A_1 + A_2$$

$$Fr(A_1A_2 + A_2A_3 + A_3A_4) = A_1 + A_4$$

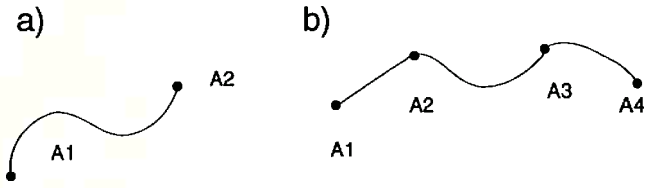


Figura 2.2 Exemplo de fronteira de arestas

Um **ciclo** é uma seqüência de arestas que não têm fronteira.
Verificamos na figura 2.3 que :

$$Fr(A_1A_2 + A_2A_3 + A_3A_1) = 0$$

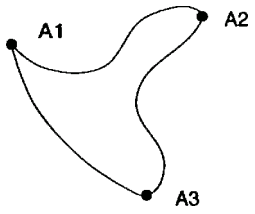


Figura 2.3 Exemplo de ciclo (Fr = 0)

Uma **face** é a região do plano limitada por um ciclo externo, que é sua fronteira externa e eventualmente por um ou mais ciclos internos, que constituem sua fronteira interna. A figura 2.4 mostra exemplos de faces válidas e não válidas.

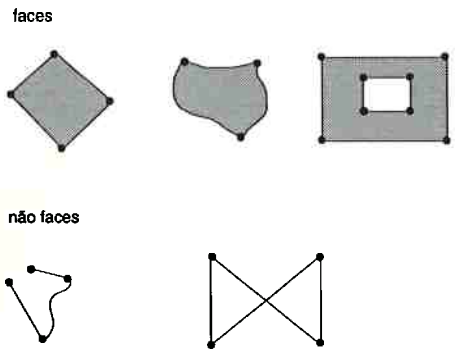


Figura 2.4 Exemplos de faces e de não faces

Define-se o **domínio** como sendo uma região contínua do plano composta por faces, arestas e vértices. Nota-se na figura 2.5 que o exemplo de **não domínio** não caracteriza um domínio por não definir uma região contínua do plano. Evidentemente, cada parte, quando tratada isoladamente, caracteriza um domínio. A continuidade do domínio é uma exigência do algoritmo de geração de malhas que é abordado nesse texto.

Um **buraco** é uma região do espaço que não pertence ao domínio mas é totalmente cercado por este.

Verificamos na figura 2.6, a existência de um buraco, determinado a seguir:

$$Fr(A_1A_2 + A_2A_3 + A_3A_4 + A_4A_1) = 0$$

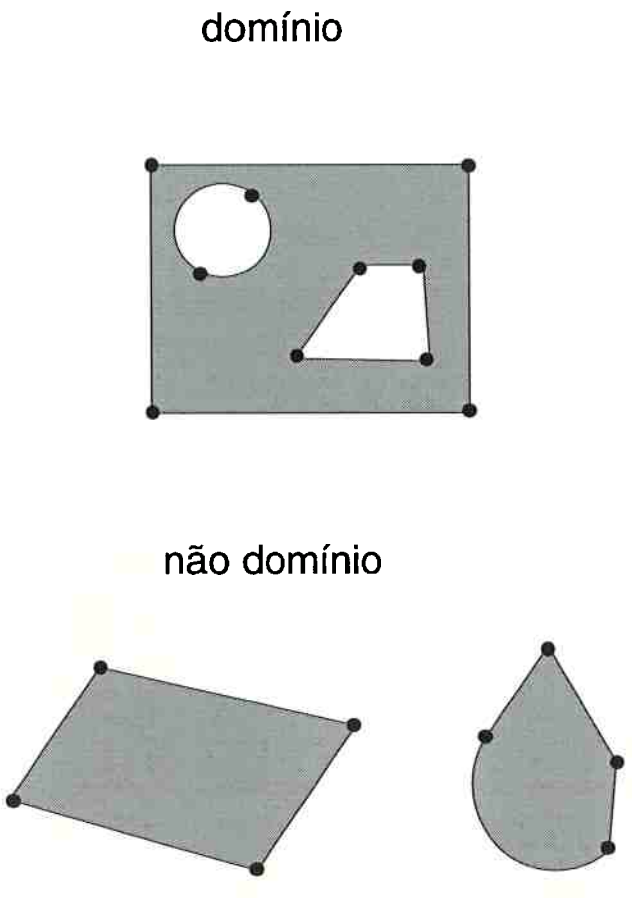


Figura 2.5 Exemplo de domínio e de não domínio

A **fronteira** de uma região poligonal é formada pelas arestas dos polígonos. Na figura 2.6 a fronteira do domínio é dada por:

$$Fr(D) = B_1B_2 + B_2B_3 + B_3B_4 + B_4B_1 + A_1A_2 + A_2A_3 + A_3A_4 + A_4A_1$$

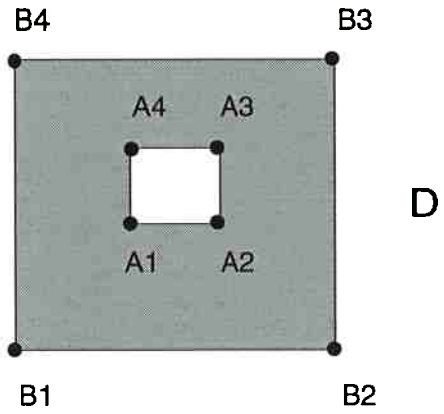


Figura 2.6 Ilustração de fronteira de domínio

2.3 Condições de invariância para as figuras definidas

Chamando de f o número de faces, e (de *edge*) o número de arestas e v o número de vértices da região, para uma região poligonal temos:

$$v - e + f = 1$$

No exemplo apresentado na figura 2.7, tem-se: $v = 4$, $e = 4$ e $f = 1$, que verifica a expressão.

No entanto, a equação mais geral que cobre todas as figuras definidas no item anterior deve permitir furos. Essa equação é apresentada a seguir sendo que o número de furos é representado por h (*hole*) e o número de ciclos dado por ℓ (*loop*).

$$v - e + f - (\ell - f) = 1 - h$$

Temos, por exemplo, que para a figura 2.7 $v = 4$, $e = 4$, $f = 1$, $h = 0$ e $\ell = 1$, valores que satisfazem a expressão anterior, onde já se contempla a possibilidade de existência de furos no domínio.

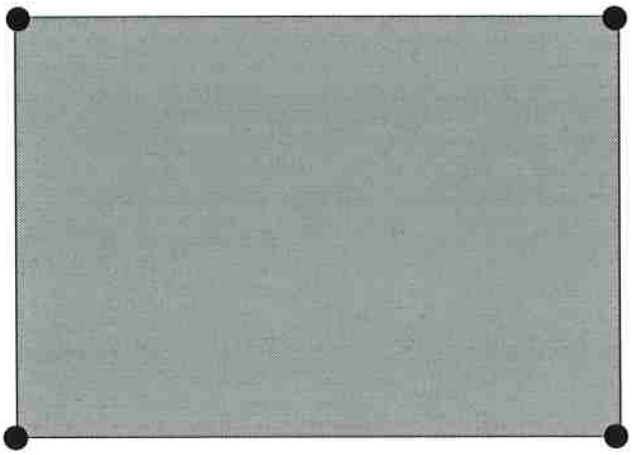


Figura 2.7 Exemplo de domínio

Com o intuito de melhor explicar a utilização da expressão

$$v - e + f - (\ell - f) = 1 - h$$

ela será verificada para mais três exemplos.

Na figura 2.8 a), pode-se contar: $v = 5$, $e = 5$, $f = 1$, $h = 0$ e $\ell = 1$; que verifica a expressão.

Na figura 2.8 b), tem-se: $v = 7$, $e = 8$, $f = 1$, $h = 1$ e $\ell = 1$; que verifica a expressão.

Na figura 2.8 c), determina-se: $v = 7$, $e = 7$, $f = 1$, $h = 1$ e $\ell = 2$; que verifica a expressão.

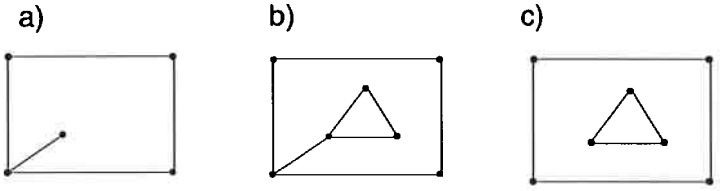


Figura 2.8 Exemplos de domínios que observam a condição de invariância

Capítulo 3

Estrutura de dados do modelador de figuras planas

Conceitualmente, um sistema de modelagem de sólidos deve possuir três níveis de abstração [9].

O nível mais alto de abstração é aquele que se refere à interação do usuário com o sistema e é chamado de **interface do usuário** . Essa interação se dá através da linguagem de comandos que pode ser gráfica, textual ou ambas. Neste nível se encontram as ferramentas para construção, modificação, arquivamento e eliminação dos objetos que compõem o projeto. O sistema de modelagem desenvolvido usa, na interface com o usuário, linguagem gráfica e textual, sendo que os comandos podem ser fornecidos acessando-se ícones apresentados na tela, selecionando menus ou digitando-se diretamente o comando desejado. Maiores detalhes da interface com o usuário do sistema desenvolvido são apresentados no capítulo 6.

Um nível de abstração inferior ao primeiro é o chamado **infra-estrutura algorítmica e do modelo de dados,** que compreende as implementações das operações disponíveis na interface do usuário, ferramentas auxiliares necessárias para essas operações, bem como o modelo de dados. Este nível é abordado nesse capítulo, principalmente no que se refere ao modelo de dados.

O nível mais baixo é denominado **substrato** e se preocupa com detalhes da implementação física do algoritmo em determinado equipamento, servindo como primitiva para a infra-estrutura algorítmica. Por exemplo, a capacidade de manipulação de aritmética de ponto flutuante do hardware ou a tarefa de ativar pixel no monitor de vídeo são atribuições do substrato. Adotou-se no GAMA2D, software desenvolvido nesse trabalho, o substrato oferecido pela linguagem C, mais especificamente a versão 9.0 da WATCOM.

3.1 Conceitos básicos sobre estruturas de dados

"**Programas**, afinal de contas, são formulações concretas de **algoritmos** abstratos e **estrutura de dados**", argumenta Wirth, no prefácio de [21], sobre a importância das estruturas de dados, colocando-as no mesmo patamar dos algoritmos.

Na verdade, os dados, em uma primeira instância, representam abstrações do fenômeno real, sendo formulados, preferencialmente, como estruturas abstratas não necessariamente implementadas nas linguagens usuais de programação. No processo de desenvolvimento do projeto do sistema, a representação de dados é refinada de modo gradual para que haja uma aderência cada vez maior aos vínculos impostos pela linguagem de computação escolhida.

Os dados representam uma abstração da realidade na medida que certas propriedades e características do objeto real são desprezadas por serem inexpressivas ou irrelevantes para a solução do problema. No campo da modelagem de sólidos utilizando o método de representação pela fronteira, essas abstrações consagraram alguns modelos de dados tais como o *winged-edge* [12,20] e o *half-edge* [20].

No nível da implementação física do modelo de dados utilizando-se uma linguagem de computação, pode-se dizer que o recurso mais versátil de que se dispõe é o das listas ligadas. Este tipo de representação possui as seguintes características:

- os átomos da lista ligada não estão necessariamente armazenados seqüencialmente na memória;
- o tamanho da lista não precisa ser fixo;
- os átomos podem ser de tipos heterogêneos.

A perda de contigüidade de armazenamento exige que se armazene junto a cada átomo a informação de onde acessar o átomo seguinte. Algumas linguagens, tais como C e PASCAL, possuem um tipo especial de dado, o ponteiro (*pointer*), que pode armazenar endereço de memória. Esquematicamente, as listas ligadas são compostas por dois campos: o primeiro que armazena as informações relativas àquele átomo e, o segundo, do tipo ponteiro, que estabelece as relacões de vizinhança daquele átomo.

As relações de vizinhança definem a topologia da lista ligada. Sob este aspecto existem três tipos básicos de lista ligada: a simplesmente ligada, a duplamente ligada e a circular.

Na lista simplesmente ligada, cada átomo é composto por um campo de informações e outro contendo um ponteiro que referencia a próxima ocorrência, conforme mostra a figura 3.1. É importante notar que o campo **prox** é do tipo ponteiro, o campo **info** pode ser composto, a variável **inicio**, que também é do tipo ponteiro, guarda a cabeça da lista e o símbolo de terra indica valor nulo para ponteiro.

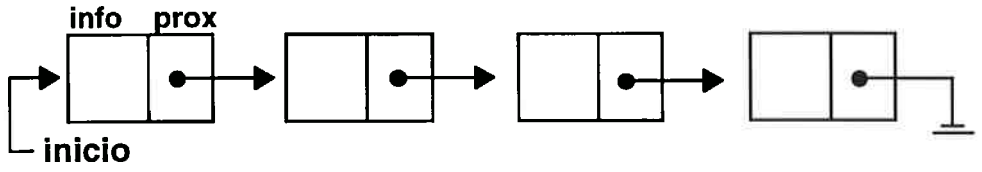


Figura 3.1 Representação de lista simplesmente ligada

Na lista duplamente ligada, cada átomo possui referência tanto para a próxima ocorrência quanto para a anterior. Nas circulares, o último elemento ao invés de possuir um ponteiro nulo, aponta para o primeiro átomo.

Além da clareza da representação e da flexibilidade, as listas ligadas exibem outra vantagem que é a possibilidade de se usar alocação dinâmica de memória. Linguagens de programação como C e PASCAL possuem funções específicas para esse tipo de alocação de memória. A alocação dinâmica permite que a reserva de memória seja efetuada para componentes individuais, no instante em que eles começam a existir, durante a execução do programa. Este fato contrasta fortemente com a alocação estática onde a reserva é realizada sempre com tamanho fixo, no início da execução do programa. A utilização de alocação dinâmica permite que se explore de maneira eficiente toda a memória disponível.

Até o momento definiu-se a forma estrutural de listas ligadas, mas não como operar com elas. Por exemplo, as seguintes operações básicas poderiam ser utilizadas para listas ligadas:

- visitar todos os átomos da lista;
- inserir um novo dado no fim da lista;
- remover um átomo do início da lista;
- buscar um dado na lista.

As funções e procedimentos que manipulam uma estrutura de dados são chamados de operadores daquela estrutura. A definição completa de uma estrutura de dados inclui tanto sua forma estrutral como a disciplina de acesso (o que define sua coleção de operadores válidos).

Deve-se notar que as operações básicas definidas para o exemplo apresentado definem que a disciplina de acesso utilizada foi a fila, isto é, um novo dado é sempre introduzido no fim da lista e a remoção é feita sempre pelo início. Por questão de disciplina e boa forma de programação, uma estrutura de dados deve ser manipulada apenas pelos seus operadores.

3.2 Estruturas de dados para modelagem de sólidos

Na modelagem de sólidos usando-se o método de representação pela fronteira, a elaboração de uma estrutura de dados remete-nos aos seguintes questionamentos, conforme descreve Turner [20]. Dada uma coleção de faces, arestas e vértices que compreendem a fronteira de um modelo sólido, quais são as possibilidades de formular uma estrutura de dados para representar a estrutura de dados do modelo? Duas características gerais parecem ser significantes: primeira, quais tipos de acessos devem ser suportados, e segundo quais dos possíveis relacionamentos devem ser armazenados explicitamente? Se um particular relacionamento não foi armazenado, qual a facilidade que se tem de realizá-lo? Qual a relação entre custo e benefício em termos de tempo de processamento e espaço de armazenamento?

Uma enumeração dos possíveis relacionamentos entre as três entidades topológicas nos leva a concluir que elas são em número de nove, conforme mostra a

figura 3.2. Pode-se notar que, com excessão do E→V e do E→F, que são do tipo 1:2, todos os outros relacionamentos são do tipo 1:n.

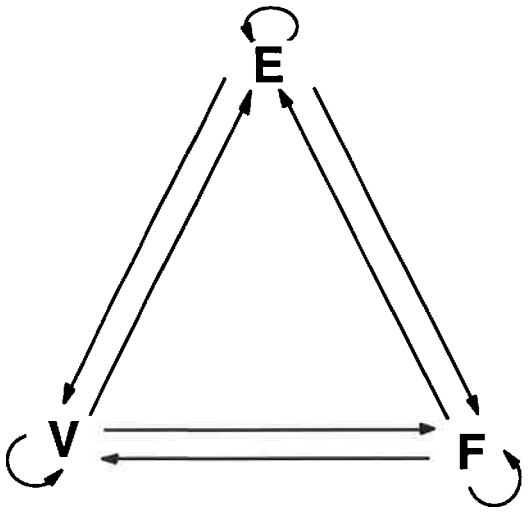


Figura 3.2 Possíveis relacionamentos entre elementos topológicos

Turner apresenta um estudo onde compara os tipos de armazenamento em termos de tempo de processamento e espaço de armazenamento. Em um dos extremos encontra-se a implementação com dois relacionamentos gerando alto custo de processamento e baixo de armazenamento. No outro, a implementação com os nove relacionamentos leva a um alto custo de armazenamento e baixo de tempo. A curva por ele apresentada indica claramente que nenhum desses extremos representa uma boa solução.

Forma estrutural do modelo *winged-edge*

A mais popular estrutura de dados para modelagem de sólidos é a *winged-eged* [20,12]. Ela apresenta um número de 5 relacionamentos, conforme demonstra a figura 3.3, situando-se numa posição intermediária da curva discutida anteriormente. É composta por nós de face, nós de aresta e nós de vértice. O elemento principal é o

nó de aresta, o qual esquematicamente parece ter asas (*wings*), de onde deriva seu nome.

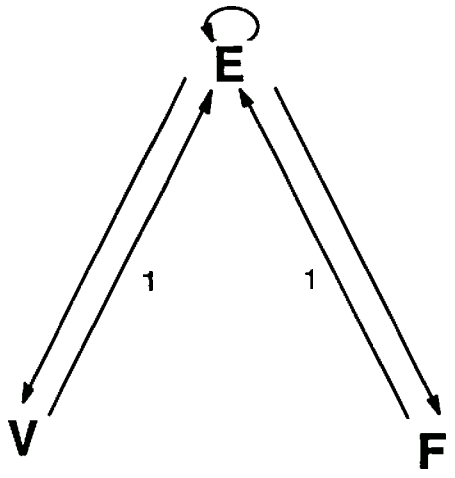


Figura 3.3 Relacionamentos da estrutura *winged-edge*

A maioria das informações sobre as ligações, conforme mostra a figura 3.4, estão nele contidas, ou seja:

- ele identifica os dois vértices que são pontos finais da aresta;
- ele identifica as duas faces que utilizam essa aresta;
- ele identifica a próxima aresta e a anterior que podem ser encontradas em cada uma das duas faces das quais ele faz parte.

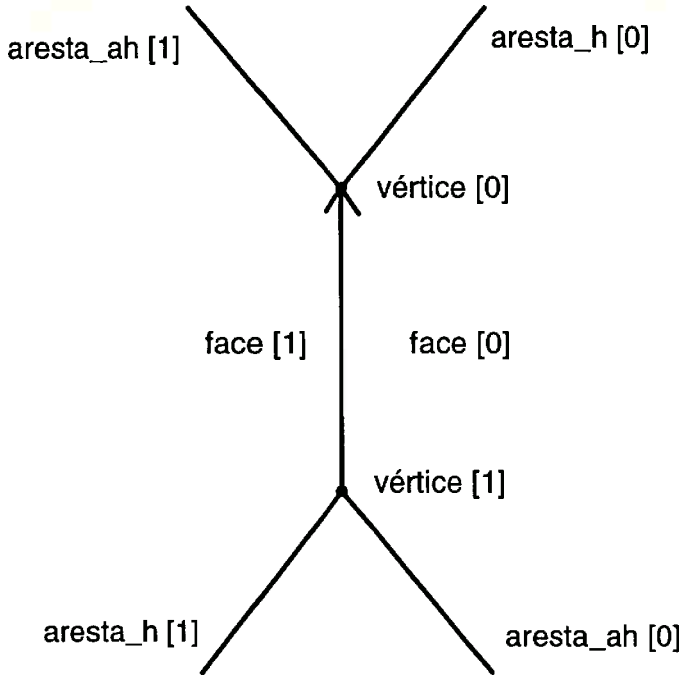


Figura 3.4 Modelo de dados esquemático do *winged-edge*

Deve-se observar que, na figura 3.4, o **ah** de **aresta_ah** indica sentido anti-horário e o **h** de **aresta_h** indica sentido horário.

Outra estrutura de dados para modelagem de sólidos que merece ser citada é a *half-edge* . Ela é apresentada por Mantila [12] como uma uma variação do *winged-edge*.

Operadores de Euler

Já se viu anteriormente que a definição completa de uma estrutura de dados inclui tanto sua forma estrutural como a disciplina de acesso (que define sua coleção de operadores válidos). Apresentou-se como forma estrutural para modelagem de sólidos o *winged-edge*.

A disciplina de acesso mais conhecida para os modelos sólidos são os operadores de Euler. Eles manipulam a forma estrutural de modo a preservar as condições de invariância do modelo.

Para tornar mais clara a função dos operadores de Euler pode-se estabelecer uma analogia entre a estrutura de dados para modelagem de sólidos e a estrutura de dados de uma fila. A forma estrutural de uma fila pode ser a lista ligada, a disciplina de acesso se faz através de operações que devem obedecer a regra FIFO (*first in first out*), isto é, o primeiro a entrar deve ser o primeiro a sair. Os operadores que manipulam a lista ligada, obedecendo a disciplina de acesso FIFO, são chamados operadores de fila.

Na modelagem de sólidos, a forma estrutural pode ser o *winged-edge*, a disciplina de acesso deve ser feita através de operações que garantam as condições de invariância do modelo. Os operadores que manipulam o winged-edge, obedecendo a disciplina de acesso ditada pelas condições de invariância, são chamados operadores de Euler.

A seguir descrevem-se sucintamente alguns operadores de Euler necessários para a construção de um modelo sólido simples.

MVFS - cria o modelo plano esquelético [12], isto é, cria uma instância na estrutura de dados de um vértice, uma face e um sólido.

MEV - cria uma aresta e um vértice.

MEF - cria uma aresta e uma face.

As operações para construir o modelo da figura 3.5, podem ser as seguintes:

- aplicar o operador MVFS para criar o modelo plano esquelético. O resultado é representado em a);
- aplicar três vezes MEV, três vértices e arestas serão adicionados ao modelo, conforme mostrado em b);
- aplicar MEF para unir os vértices finais do modelo da letra b) e criar uma aresta e uma face, apresentados na letra c);
- aplicar quatro vezes MEV para obter d);
- aplicar quatro vezes MEF para obter a forma final do modelo desejado, apresentado em e).

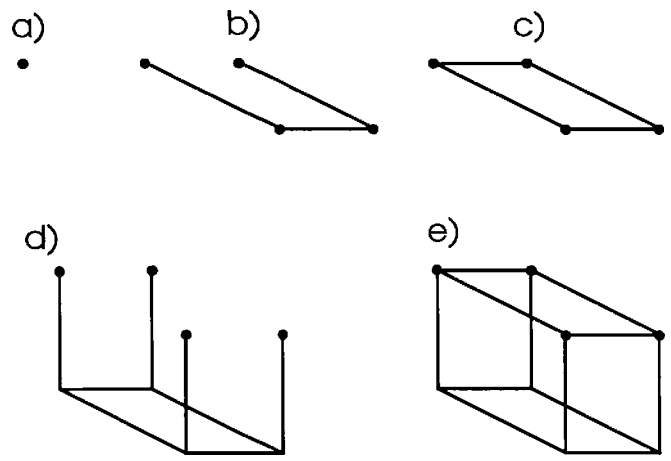


Figura 3.5 Construção de modelo sólido usando-se operadores de Euler

O intuito desse exemplo é de apenas ilustrar a filosofia de utilização dos operadores de Euler. Assim sendo, muitos detalhes foram suprimidos, mas não se deve perder de vista que em cada etapa as condições de invariância do modelo estavam garantidas.

3.3 A estrutura de dados do GAMA2D

O modelo de dados do GAMA2D fundamenta-se no estudo da modelagem de dados para sólidos. Ela cria uma estrutura básica topológica que se liga à estrutura geométrica conforme indica a figura 3.6. As entidades topológicas consistem de: faces, ciclos de face, ciclos, arestas de ciclo, arestas e vértices. A estrutura topológica é composta por pontos e curvas.

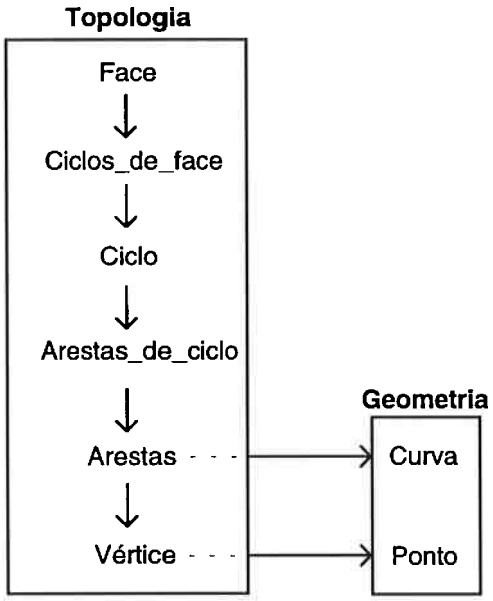


Figura 3.6 Modelo de dados da geometria/topologia do GAMA2D

Cada face possui uma referência para uma lista de todos os ciclos que a definem e que são chamados ciclos de face. Cada ciclo de face aponta para um ciclo. O ciclo aponta para uma lista de arestas que o constituem, chamada arestas de ciclo. Cada aresta de ciclo, por sua vez, aponta para uma aresta. Cada aresta possui referências para seus dois vértices. A estrutura de dados da geometria contém as informações geométricas da topologia. Desta forma, cada aresta aponta para a entidade geométrica curva e cada vértice aponta para a entidade geométrica ponto.

Quanto à forma estrutural da implementação física do modelo de dados, foi adotada a lista duplamente ligada. As listagens apresentadas a seguir foram extraídas de um dos arquivos de cabeçalho do GAMA2D. Observa-se que as variáveis globais e a estrutura de dados dos atributos que originalmente constam desse módulo foram eliminadas nessa listagem.

Quanto à disciplina de acesso, criaram-se operadores para manipulação de listas ligadas que permitem que se façam as operações básicas de inserção, remoção e pesquisa, para cada elemento topológico ou geométrico, descrito no modelo de figura plana adotado para o GAMA2D. Para cada manipulação pode-se checar as condições de invariância de modo a estabelecer se aquela é ou não uma operação válida.

```
/*************************************************************************
*   Sistema              :  GAMA2D
*   Modulo               :  GLOB.H
*   Descricao            :  Definicao da estrutura de dados e das variaveis  globais do
*                           sistema
*   Modulos chamados     :  nenhum
*   Ultima atualizacao   :  11/09/93 por Fernando
*   Proximas alteracoes  :  analise da estrutura de curva para contemplar outros tipos de
*                           curvas
*************************************************************************/

/*_____ Estrutura de dados da TOPOLOGIA _____*/

struct vertice_st
{
  ponto_t ponto;
  int deletado, num;
  struct vertice_st *seg;
  struct vertice_st *ant;
  atrib_t           *atrib;
};
typedef struct vertice_st  vertice_t;

vertice_t    *vvertice      = NULL,      /*End. inicial da lista de vert. de trabalho    */
    *pvertice_in  = NULL,                /*Endereco inicial da poligonal de trabalho */
    *pvertice_ant = NULL,
    *pvertice      = NULL;               /*Endereco do ultimo vert. de trab. definido */

int  ind_pvertice = 0;                   /*Indice do ultimo vert. de trab. definido    */
 struct aresta_st
{
  struct aresta_st *seg;
  struct aresta_st *ant;
  curva_t        curva;
  pert_t per;
};
typedef struct aresta_st  aresta_t;
```

```
aresta_t    *aresta_cor = NULL,          /*Endereco da aresta corrente       */
   *aresta_ant = NULL,
   *aresta_in   = NULL;                  /*Endereco inicia da lista de arestas */

struct ares_ciclo_st
{
  struct ares_ciclo_st *seg;
  aresta_t             *aresta;
};
typedef struct ares_ciclo_st  ares_ciclo_t;
ares_ciclo_t       *cicloa    = NULL,
          *cicloa_in = NULL;
struct ciclo_st
{
  struct ciclo_st *seg;
  ares_ciclo_t       *ares_ciclo;
};
typedef struct ciclo_st ciclo_t;
ciclo_t     *ciclo = NULL,
      *ciclo_in  = NULL;

struct circ_face_st
{
  struct circ_face_st  *seg;
  ciclo_t              *ciclo;
};
typedef struct ciclo_face_st ciclo_face_t;
ciclo_face_t       *ciclof    = NULL,
          *ciclof_in = NULL;
struct face_st
{
  struct face_st  *seg;
  ciclo_face_t    *ciclo_face;
  atrib_t         *atrib;
  int deletado;
};
typedef struct face_st  face_t;
```

```
face_t  *face      = NULL,                /*Endereco da face corrente          */
        *face_in   = NULL;                /*Endereco inicial da lista de faces */

/*_____________ Estrutura de dados da GEOMETRIA _____________________*/

 struct ponto_st
{
  double   x,
           y;
};
typedef struct ponto_st  ponto_t;

 struct curva_st
{
  struct vertice_t *vertice;
  int quant_pontos, tipo;
  int deletado, cor, cor_g;
  vertice_t *pt_def;
  ponto_t   pt_const[QMAXDIVCURVA + 1];
  atrib_t    *atrib;
};
typedef struct curva_st  curva_t;
```

Figura 3.7 Estrutura de dados da geometria/topologia do GAMA2D

Capítulo 4

Conceitos básicos da geração automática de malhas

Um sistema de geração automática de malhas de elementos finitos é aquele em que a interação com o usuário está limitada à definição do modelo geométrico e à especificação de um ou mais parâmetros de densidade de malha. Segundo Perucchio[17] um algoritmo de geração automática de malha deve ser considerado como uma função G que relaciona o modelo geométrico S (e uma série de parâmetros de densidade de malha l) com uma malha válida M, de tal forma que M = G(S, l). M é considerada uma malha válida de S se:

- M é uma malha correta de elementos finitos em um amplo sentido (isto é, cada um dos elementos não intercepta nenhum outro elemento, elementos são geometricamente corretos e assim por diante);
- nenhum elemento de M está fora de S;
- todos os nós de M estão em uma das duas situações: na fronteira ou no interior do domínio;
- todas as arestas e vértices de S estão representadas em M.

Este capítulo mostra uma classificação e breve descrição dos métodos de geração de malhas de elementos finitos, situando o Método do Quadtree Modificado. Logo após são apresentados os conceitos básicos desse método bem como o modelo de dados empregado.

4.1 Classificação dos métodos de geração de malhas

Poucas tentativas foram feitas até hoje no sentido de se classificar os métodos de geração de malhas de elementos finitos, seja ele automático ou não. Tanto Ho-Le[10] quanto Baehmann[01] citam como primeira tentativa de rever os métodos de geração, o trabalho publicado em 1973 por Buell e Bush[24], numa época em que se poderia considerar ainda incipiente o número de pesquisas desenvolvidas nessa área. Mais tarde, em 1980, Thacker[35] publica uma extensa bibliografia dos métodos sem no entanto apresentar nenhuma preocupação com taxionomia ou em estabelecer comparações entre eles.

Em 1988, K Ho-Le publica um artigo com o intuito principal de classificar os métodos de geração de malhas de elementos finitos, propondo uma primeira seleção baseada na ordem temporal em que o conjunto de nós ou elementos são criados, subdividindo a seguir esses grandes grupos conforme ilustra a figura 4.1. É interessante notar que Baehmann apesar de não apresentar uma preocupação explícita em classificar os métodos, agrupa-os com coerência em relação ao disposto em K Ho-Le. Apresentamos a seguir uma breve descrição dos grupos em que foram divididos os métodos de geração de malha de elementos finitos. Deve-se observar que as referências bibliográficas relativas aos métodos citados a seguir podem ser encontrados em [10] e foram transcritas para a bibliografia suplementar.

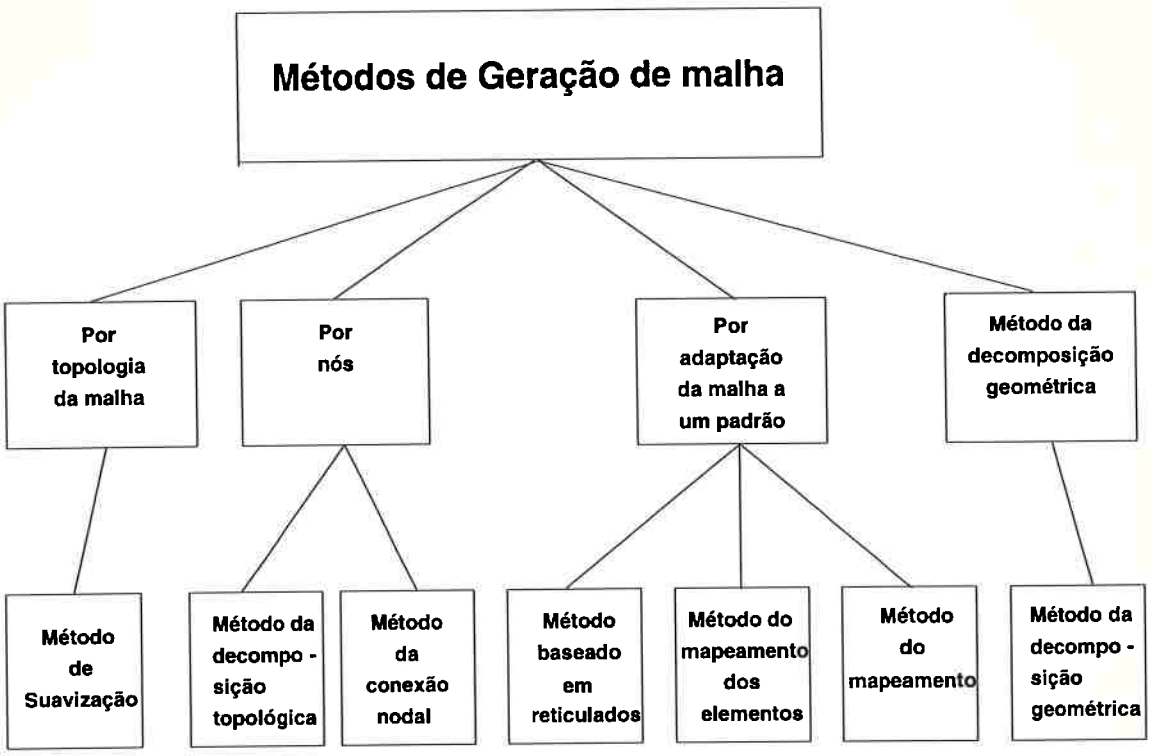


Figura 4.1 Classificação dos métodos de geração de malhas de elementos finitos[10]

4.1.1 Por topologia da malha (*Mesh topoly first*)

Segundo a classificação apresentada na figura 4.1, o primeiro grupo é aquele em que há, inicialmente, a determinação da topologia da malha. Posteriormente, através do método de suavização de malha, determinam-se as coordenadas nodais. Como não se conhece até o momento algoritmo capaz de criar a topologia como etapa inicial, esse método vem sendo empregado como auxiliar de outros tal como o método do quadtree modificado, classificado como método baseado em reticulados.

4.1.2 Por nós (*Nodes First*)

Neste método a etapa inicial se concentra na criação dos nós, que são posteriormente conectados para formar os elementos retangulares ou triangulares, podendo-se distinguir dois métodos : o Método da Decomposição Topológica e o Método da Conexão Nodal.

Método da decomposição topológica
(*Topology decomposition aproach*)

O método da decomposição topológica foi desenvolvido por Wordenweber[38, 39]. Neste método o objeto é representado por dois tipos de entidades: vértices e arestas. Cada entidade tem a geometria que especifica a sua posição e forma e a topologia que especifica seu relacionamento com as outras entidades. Através da conexão dos vértices o objeto é decomposto em uma série de triângulos de modo a cobrir toda a superfície do objeto. Esses elementos, que foram grosseiramente formados, são refinados posteriormente para atender à densidade de malha desejada.

Método da conexão nodal
(*Node conection approach*)

Neste método distinguem-se duas fases principais : a de geração dos nós e a de geração dos elementos.

A geração dos nós pode ser realizada de maneira randômica através da técnica de Cavendish[26] que propõe que nós sejam inicialmente adicionados à fronteira do objeto em intervalos regulares e, logo a seguir, outros nós sejam adicionados randomicamente ao interior do domínio respeitando a densidade de malha preestabelecida.

Existem duas técnicas de geração não randômica de nós. O método de Lo's[33] que usa a intersecção de retas horizontais com o contorno do objeto para gerar os nós, sendo que o espaçamento das retas é função do tamanho dos elementos a serem gerados. O método de Lee[32] que utiliza-se da representação CSG em duas dimensões para a geração nodal.

Na fase de geração dos elementos, os nós, definidos na etapa anterior, são conectados formando elementos que cubram inteiramente o domínio do objeto. A menos do método de Lee[24], que produz malha formada por quadriláteros, os outros produzem malhas triangulares. Podemos citar ainda os métodos de Frederick's[29] ,

Nelson's[34] e o de Delaunay . O método de Delaunay é um dos mais usados, pois a sua triangulação procura maximizar a soma dos menores ângulos de todos os triângulos criando, deste modo, uma melhor malha para uma dada série de pontos.

4.1.3 Por adapatação da malha a um padrão (*Adapted mesh template*)

Incluem-se nesse grupo os métodos em que um modelo da malha é gerado previamente e, logo após, adaptado ao objeto a ser discretizado. Este grupo é composto por três métodos descritos a seguir.

Método baseado em reticulados
(*Grid-based approach*)

O método baseado em reticulados baseia-se no fato de que um reticulado se parece com uma malha e que as células do retículado podem ser transformadas em elementos.

No método de Tacker[36], o objeto é inicialmente circunscrito por um triângulo eqüilátero preenchido por um reticulado triangular. Os pontos do reticulado que caem fora dos limites do objeto são eliminados, deixando a fronteira em forma de ziguezague. Os pontos do reticulado da fronteira em ziguezague são movidos para o contorno do objeto, para criar uma malha compatível com o contorno. Outros métodos que também se enquadram nessa classificação são os de Heighway[30], Kela[31] e Shephard[40]. Tanto os algoritmos quanto o modelo de dados usados na implementação de um programa de geração automática de malhas usando o método de Shepard são abordados com detalhes nesse trabalho.

Método do mapeamento dos elementos
(*Mapped element approach*)

Esse método impõe que um objeto seja subdividido manualmente em regiões simples, cada uma delas consistindo de três ou quatro lados, definindo macro elementos. O gabarito da malha é um quadrado unitário com malha quadrada definida no seu interior ou um triângulo unitário com malha triangular definidos no espaço paramétrico. O gabarito é mapeado para a região a ser discretizada via função

de forma. O número de divisões depende da densidade de malha desejada. As funções de forma podem ser encontradas em Ziennkiewicz [41] sendo que sua técnica apresenta a restrição de que dois lados opostos devem possuir o mesmo número de divisões. Essa limitação é superada por Cohen [28].

Método de Mapeamento
(*Conformal mapping approach*)

Nesse método constrói-se um polígono Q com o mesmo número de vértices da região a ser discretizada (representada por um polígono P). Determina-se então uma função que leva de Q a P, baseando-se na correspondência entre seus vértices. A malha construída no polígono Q é então mapeada para o objeto através dessas funções.

4.1.4 Por nós e elementos criados simultaneamente (*Nodes and elements simultaneously*)

Neste grupo enquadram-se os métodos em que não se pode fazer distinção entre as fases de criação dos nós e dos elementos.

Método da decomposição geométrica
(Geometric decomposition approach)

No método da decomposição geométrica são feitas considerações a respeito da forma e do tamanho dos elementos. No algoritmo descrito por Tracy[37] admite-se que o contorno do objeto tenha sido subdividido em segmentos de reta com comprimentos definidos pelos tamanhos desejados para os elementos. Os vértices que formam ângulo menor que 90 graus originam um triângulo através da criação de nova aresta. Os vértices com ângulo entre 90 e 180 graus originam dois triângulos com a criação de novo vértice e arestas. O método de Bykat's[25] também se enquadra nessa classificação.

4.2 Descrição do Método Quadtree Modificado

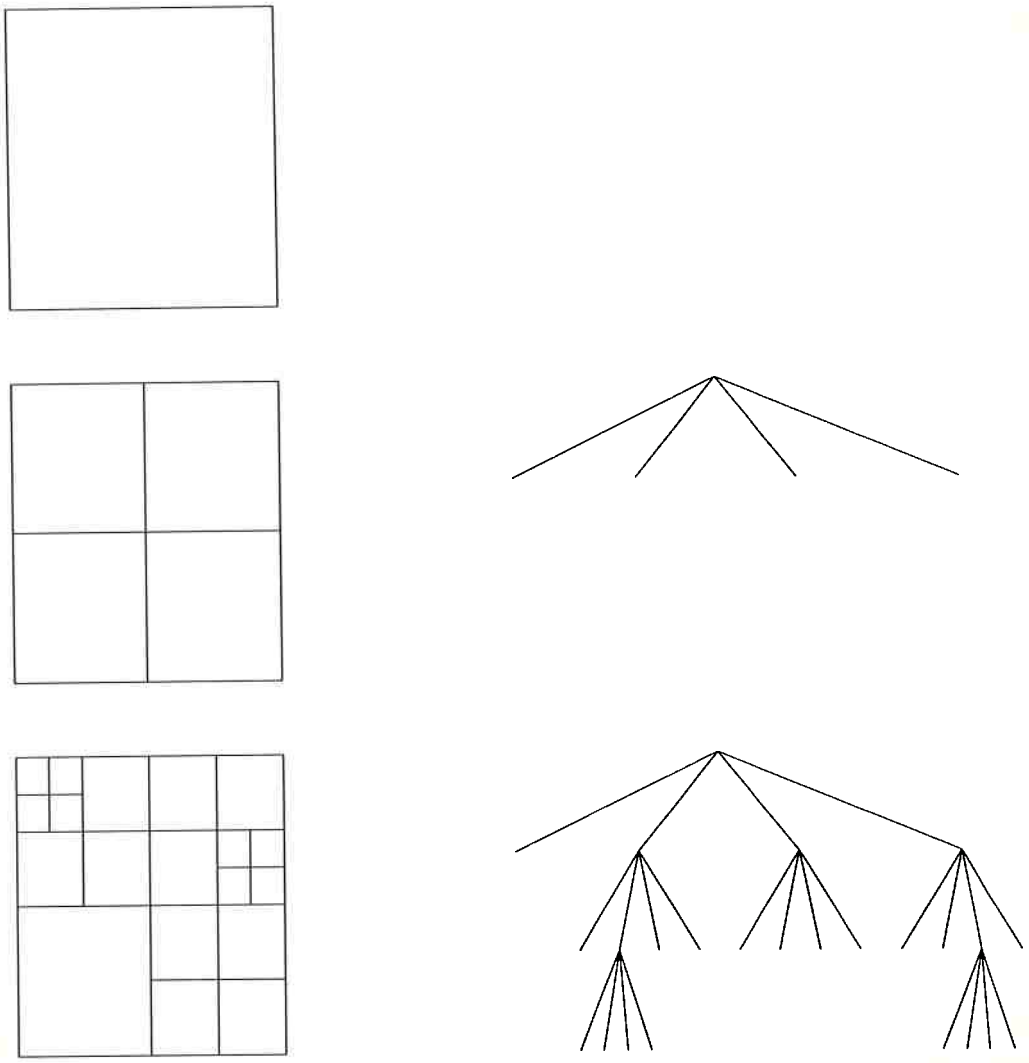
O gerador de malhas quadtree modificado é baseado em um método de representação de modelos bidimensionais usando decomposição espacial. O domínio do objeto é representado por um conjunto de células quadradas de vários tamanhos, denominadas quadrantes, que são armazenadas em uma estrutura de dados do tipo árvore hierárquica.

Inicialmente, o objeto a ser discretizado é circunscrito por um quadrado denominado universo do domínio do modelo. Este, que é o maior dos quadrantes e representa a raiz da árvore é então subdividido em quatro quadrantes que representam os seus quatro filhos. Cada um dos quatro quadrantes assim formados podem, por sua vez, ser subdivididos, formando um novo nível da árvore. Este procedimento pode ser realizado recursivamente até que os níveis definidos pelos parâmetros de controle de malha sejam atingidos. O parâmetro de controle de malha associa a determinadas regiões ou a elementos geométricos que definem o domínio um determinado valor inteiro que indica qual o nível hierárquico mínimo da região em relação à árvore de dados. A figura 4.2 mostra a representação espacial e gráfica de um quadtree.

Um quadrante que tenha sido subdividido é denominado quadrante de continuação. De outro modo, um quadrante cuja representação finaliza um ramo da árvore é chamado quadrante terminal e pode estar completamente contido no domínio, totalmente fora dele ou ainda parcialmente preenchido. Podemos notar que apenas os quadrantes parcialmente preenchidos é que têm a possibilidade de serem subdivididos. A figura 4.3 mostra a representação do quadtree modificado de um objeto.

Como vimos, o quadtree pode ser usado na representação de objetos bidimensionais através de um conjunto de células ou quadrantes terminais. Como uma malha de elementos finitos também se baseia na idéia de representar o modelo contínuo em um conjunto discreto de células, podemos imaginar que os quadrantes terminais poderiam ser usados como elementos finitos e os vértices dos quadrantes, como nós dos elementos finitos.

No capítulo 5 são descritos os passos necessários para a geração de malhas de elementos finitos através do método quadtree modificado.



a) espacial

b) gráfica

Figura 4.2 Representação do quadtree: ao alto) *quadrante universo*; no meio) *nível 2* e de baixo) *quadtree não uniforme de nível 4*.

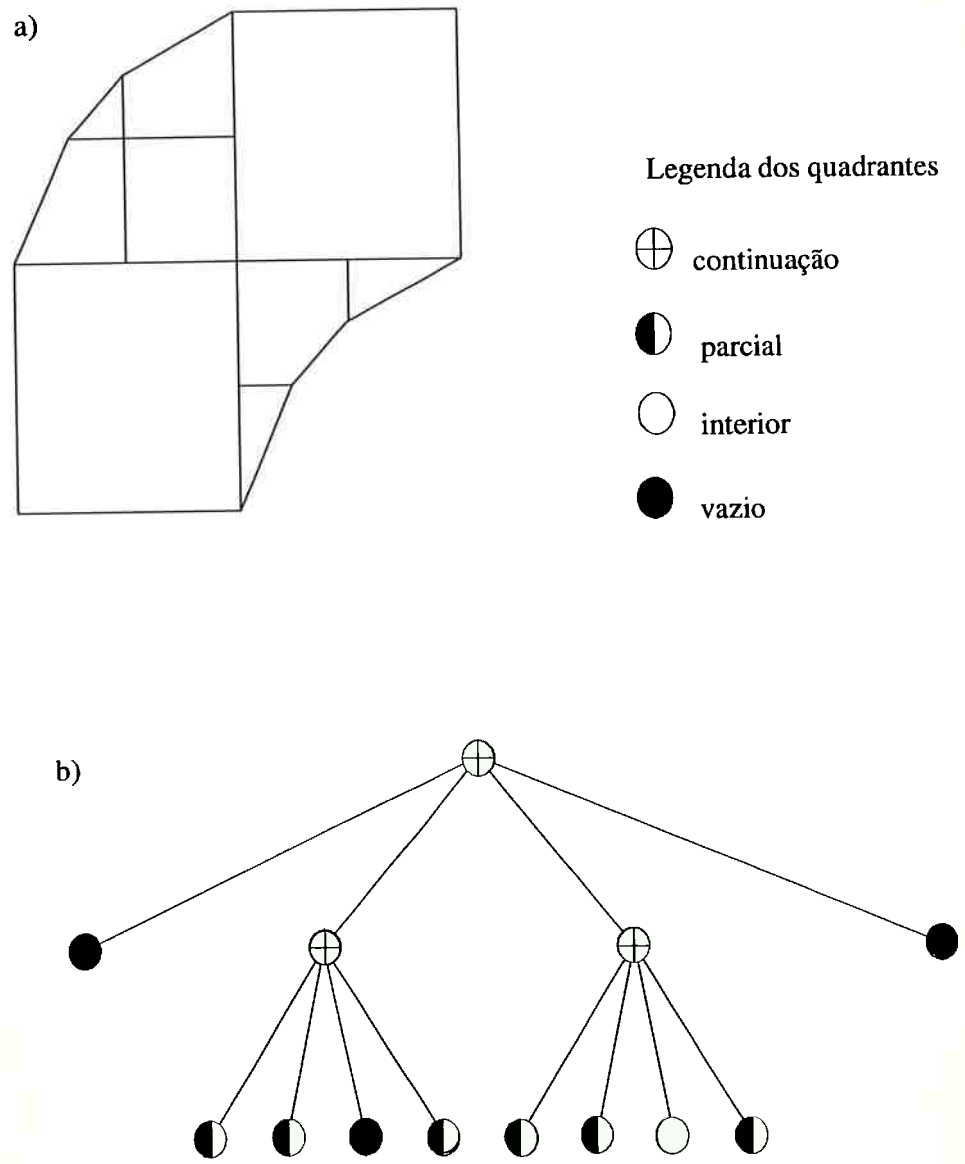


Figura 4.3 Quadtree modificado de um modelo mostrando os tipos de quadrantes:
a) representação espacial b) representação gráfica

4.3 Modelo de dados do Quadtree Modificado

O próprio nome quadtree indica a estrutura básica do modelo de dados que é a árvore quaternária. Uma árvore, de modo geral, é um grafo orientado conexo satisfazendo as seguintes propriedades:

- não contém circuitos;
- existe um vértice que é fonte chamado raiz;
- todo vértice, com exceção da raiz, tem grau de entrada igual a 1;
- existe um e apenas um caminho da raiz para cada vértice.

Uma árvore quaternária deve obedecer também à propriedade de que cada vértice pode ter no máximo quatro filhos.

Os operadores de árvore geram algoritmos recursivos. A recursividade, dentro da programação, refere-se à propriedade que algumas linguagens possuem de permitir que uma função invoque a si própria. A linguagem C permite que se implementem algoritmos recursivos.

Na figura 4.4 apresenta-se a listagem da estrutura de dados do quadtree, utilizada no GAMA2D. Pode-se notar que cada instância possui uma referência para seu pai e para seus quatro filhos estabelecendo a sua topologia. Possui também as coordenadas que definem sua dimensão, um inteiro que indica o nível de profundidade em que se encontra e um código indicando o seu tipo. A listagem apresentada, na verdade é um trecho do arquivo de cabeçalho que define a estrutura de dados do GAMA2D. Observa-se que o caráter *, indica que a variável é do tipo ponteiro.

```
struct quad_st
{
        struct quad_st *quadpai,
                       *subquad0,
                       *subquad1,
                       *subquad2,
                       *subquad3;
        float   x_esq, y_esq,
                x_dir, y_dir;
        int     nivel,
                tipo_quad; /*   0 = exterior, 1=continuacao,
                               2= fronteira,3=interior */
};
typedef struct quad_st quad_t;

quad_t *quadco          /* endereco do quadrante corrente  */
       *quad_in         /* endereco do quadrante raiz      */
```

Figura 4.4 Estrutura de dados do Quadtree

Capítulo 5

Geração de malhas usando o Método Quadtree Modificado

Um gerador automático de malhas de elementos finitos tem que ser capaz de interpretar as informações geométricas que definem o modelo e, a partir delas, gerar uma malha de elementos finitos válida. O processo de geração automática da malha de elementos finitos é constituído de duas etapas principais. Na primeira etapa o domínio é subdividido, através do método do quadtree, em células quadradas desconexas que cobrem todo o domínio do objeto. Cada célula contém as informações topológicas discretas relativas àquela porção do domínio coberta pela célula. Em uma segunda etapa os quadrantes terminais do quadtree são transformados em elementos finitos.

Este capítulo apresenta todas as etapas para a geração de malhas de elementos finitos através do método do quadtree modificado. Para cada uma dessas etapas são apresentados os algoritmos e uma ilustração do processo através de um exemplo básico. Tomou-se como exemplo básico uma figura plana de geometria simples, com poucos elementos e contornos retos que pudesse dar um caráter didático às ilustrações que nela se baseassem. Deve-se ressalvar porém que os

algoritmos apresentados são absolutamente gerais. A definição geométrica do exemplo básico se encontra na figura 5.1.

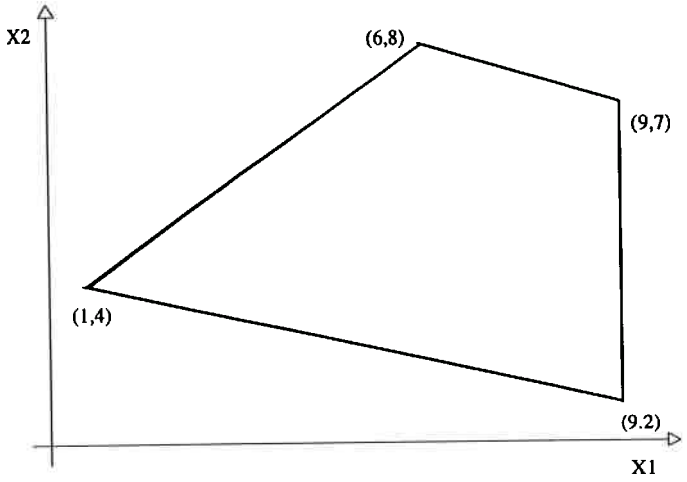


Figura 5.1 Geometria do exemplo básico

5.1 Controle dos níveis de refinamento de malha

Como a malha final é função do tamanho dos quadrantes, e este, função do nível em que cada quadrante se encontra na árvore, tem que haver uma maneira de se controlar o nível de subdivisão de cada localização do domínio do objeto. Isto é resolvido usando-se o parâmetro de controle de malha.

O parâmetro de controle de malha indica o nível mínimo que deve ser atribuído a uma determinada região do objeto. Ele pode ser fornecido interativamente pelo usuário ou então serem admitidos valores pré-estabelecidos pelo algoritmo.

O parâmetro de controle de malha corresponde a um valor inteiro que pode estar associado às curvas que compõe a fronteira, aos vértices ou a região do objeto e pode ser especificado de forma a definir de maneira **absoluta** o nível da árvore correspondente àquela região. Assim sendo um parâmetro de controle de malha de nível 1 define o maior dos quadrantes, ou seja, o universo do domínio do objeto. O nível 2 corresponde a quadrantes cujo lado possui a metade do comprimento do nível 1. O nível 3, portanto, possui quadrantes com comprimento de 1/4 do tamanho do nível 1. O número de níveis admitido é dependente da máquina em que o algoritmo será implementado.

A aplicação do parâmetro de controle de malha somente aos vértices e arestas pode não ser suficiente para se conseguir o refinamento necessário da malha. Deve ser possível então colocar-se parâmetros de controle de malha em qualquer posição do domínio. Essas posições se referem a pontos de refinamento de malha e definem de forma **relativa** o nível da árvore. Assim sendo, um ponto de refinamento de malha de valor n especifica que o quadtree deve ser refinado n níveis além daquele em que se encontra.

Os pontos de refinamento de malha também podem ser obtidos automaticamente de um programa de análise adaptativa após análise de estimativa de erro. Além disso, estão sendo desenvolvidas técnicas que permitem o reconhecimento de regiões críticas na forma do objeto e estabelecem automaticamente valores iniciais apropriados para o parâmetro de controle de malha.

A figura 5.2 mostra a aplicação do parâmetro de controle de malha aplicado a vértices e arestas do exemplo básico.

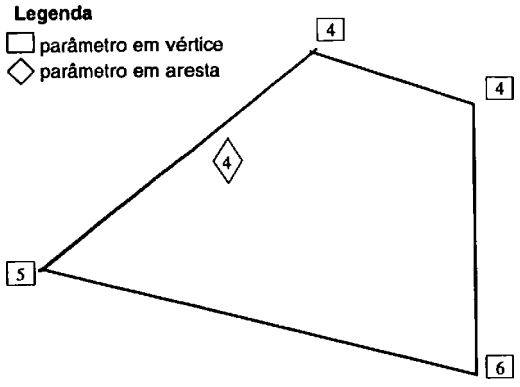


Figura 5.2 Parâmetros de controle de malha aplicados a vértices e arestas

5.2 Divisão inicial do quadtree

O primeiro passo na representação do domínio do objeto através do quadtree modificado é a subdivisão do quadtree nos níveis indicados pelos parâmetros de controle de malha.

Inicialmente é criado o quadrante do universo do modelo cujo lado corresponde à maior das diferenças de coordenadas nas direções x e y. Este quadrante corresponde ao nível 1, e é denominado raiz da árvore hierárquica. A seguir, verifica-se qual o maior parâmetro de controle de malha dentre as entidades contidas no quadrante ora criado. Se o nível do quadrante for menor do que o maior parâmetro de controle de malha, esse quadrante será subdividido em quatro outros. Cada quadrante criado possuirá lados com metade da dimensão do quadrante pai. Simultaneamente são incorporados na base de dados do quadtree quatro novos ramos da árvore. Agora tomando-se um dos quatro quadrantes recém criados verifica-se se o seu nível corresponde ao maior parâmetro de controle de malha, assim como feito anteriormente. Esse procedimento é realizado de maneira recursiva até que todos os quadrantes estejam criados nos níveis correspondentes ao maior dos parâmetros de controle de malha contidos no seu domínio. A figura 5.3 mostra a divisão inicial do quadtree do exemplo padrão.

Deve-se ressaltar que esse passo cria toda a árvore hierárquica, isto é, o espaço na estrutura de dados, sem no entanto armazenar qualquer informação relativa ao modelo que se quer representar.

O pseudocódigo do algoritmo relativo a divisão inicial do quadtree é apresentado na figura 5.4.

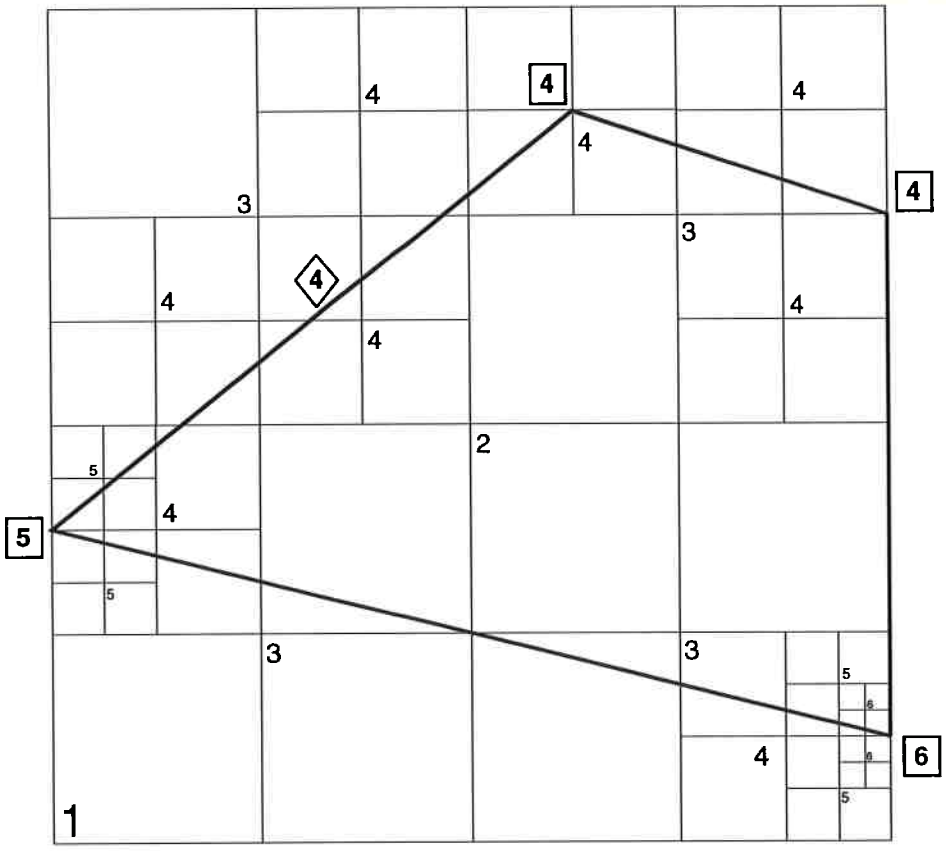


Figura 5.3 Divisão inicial do quadtree do exemplo básico

```
Função QUAD_UNIVERSO()
/* Objetivo:  obtém as dimensões e aloca espaço na estrutura de dados
              para o quadrante do universo do modelo. Dá início ao processo
              de refinamento dos quadrantes invocando a função REFINA( )*/
{
   Encontrar x_max, x_min, y_max, y_min do modelo
   l_quad                  ← máximo( (x_max-x_min),(y_max-y_min) )
   x_med                   ← x_min + ( x_max - x_min )/ 2
   y_med                   ← y_min + ( y_max - y_min )/ 2
   quad_univ               ← localização da raiz da árvore na estrutura de dados
   quad_univ->quadpai      ← NULO
   quad_univ->subquad0     ← NULO
   quad_univ->subquad1     ← NULO
   quad_univ->subquad2     ← NULO
   quad_univ->subquad3     ← NULO
   quad_univ->nivel        ← 1
   quad_univ->x_esq        ← x_med - l_quad/ 2
   quad_univ->x_dir        ← x_med + l_quad/ 2
   quad_univ->y_esq        ← y_med - l_quad/ 2
   quad_univ->y_dir        ← y_med + l_quad/ 2
   Refina( quad_univ , NULO )
}
Função REFINA( quad, reg )
/* Objetivo:  gera, recursivamente, a partir do quadrante do universo, todo o quadtree
              enquanto todos os parâmetros de controle de malha não forem atendidos*/
{
 nivel_quad        ← nível do quadrante quad
 param_max         ← máximo parâmetro de controle de malha na região de quad
 se          quad ≠ quad_univ
       então     Armazena na estr. de dados do quadtree as coord. de quad na região reg
                 Inicializa os 4 filhos do quadrante quad como nulos
                 Atribui ao nível do quadrante quad o nível do seu pai +1
 fim-se
 se        param_max > nivel_quad
       então     quad_seg ← localização na estrutura de dados do filho de quad
                 Refina( quad_seg , 0 )
                 Refina( quad_seg , 1 )
                 Refina( quad_seg , 2 )
                 Refina( quad_seg , 3 )
 fim-se
 }
```

Figura 5.4 Algoritmo relativo à divisão inicial do quadtree

5.3 Controle da razão de aspecto dos elementos

Para que na geração de malhas de elementos finitos não ocorram elementos que não tenham uma boa proporção entre seus lados, devemos garantir que quadrantes adjacentes não possuam diferença de nível maior do que um. Todos os quadrantes terminais são, então, analisados para assegurar que não haja grandes diferenças no tamanho de elementos finitos vizinhos na malha final.

O processo se inicia tomando-se o primeiro quadrante terminal que compõe a árvore hierárquica da estrutura de dados do quadtree. A seguir seus quadrantes vizinhos que também sejam terminais são encontrados através de um método que será abordado nessa mesma seção. Caso esse primeiro quadrante tenha mais de um nível de diferença em relação a algum de seus vizinhos ele será subdividido até alcançar um nível de diferença com seu vizinho de nível mais alto. É tomado então o próximo quadrante terminal da árvore hierárquica para o qual repete-se todo o processo descrito para o primeiro quadrante. Desse modo, todos os quadrantes terminais da árvore são analisados. Se durante esse processo algum deles for subdividido, a rotina é repetida tomando-se novamente o primeiro quadrante e analisando todos os quadrantes terminais. O fim do processo só se estabelece quando ao percorrermos todos os quadrantes terminais da árvore não tiver ocorrido nenhuma criação de novos quadrantes. A figura 5.6 ilustra através do exemplo básico, as divisões extras que garantem um nível de diferença entre quadrantes vizinhos.

O algoritmo que garante um nível de diferença entre quadrantes que compartilham um mesmo lado é apresentado na figura 5.5. Observe-se que os quadrantes adicionais, introduzidos para garantir um nível de diferença, estão representados em tracejado.

```
Função Um_nivel_viz()
/* Objetivo:     percorre os quadrantes terminais. A cada ocorrência testa a
                 diferença de nível ente ele e seus vizinhos. Se a diferença
                 for maior que 1 refina o quadrante terminal até que essa
                 diferença se torne igual a 1. O fim do processo se dá quando
                 após percorrer todos os quadrantes terminais nenhum
                 refinamento tiver sido efetuado
*/
{
 quad          ← primeiro quadrante terminal do quadtree
 refinou       ← V
 repetir-enquanto refinou
          refinou ← F
          repetir-enquanto   quad ≠ último quadrante teminal do quadtree
                             nivel_quad   ← nível do quadrante quad
                             Encontrar vizinhos de quad
                             max_nivel    ← maior nível dentre os quadrantes vizinhos
                                se      max_nivel > nivel_quad + 1
                                   então Refinar quadrante quad até nível
                                         nivel_quad - 1
                                         refinou ← V
                                fim-se
                                quad ← próximo quadrante terminal do quadtree
          fim-repetir
 fim-repetir
 }
```

Figura 5.5 Algoritmo que garante um nível de diferença entre quadrantes vizinhos

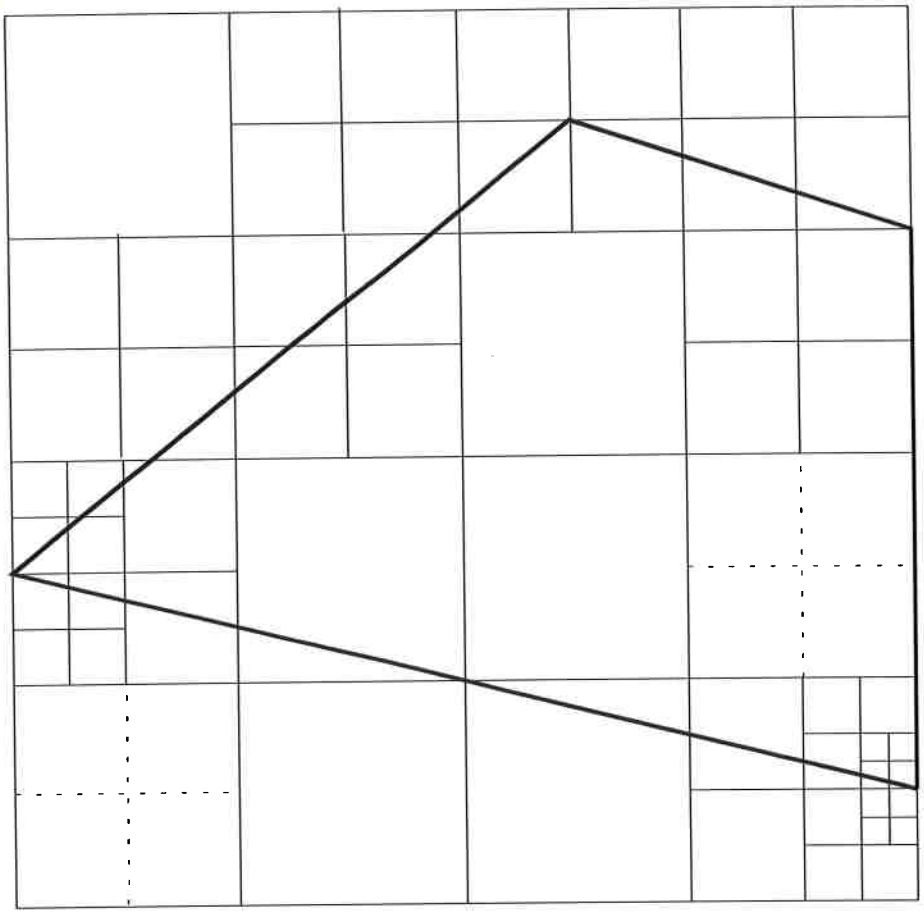


Figura 5.6 Divisões extras que garantem um nível de diferença entre quadrantes vizinhos

No algoritmo que garante um nível de diferença entre quadrantes vizinhos utiliza-se intensamente uma função que determina todos os vizinhos de um determinado quadrante. A implementação de um algoritmo eficiente na busca de vizinhos é importante para o bom desempenho do sistema.

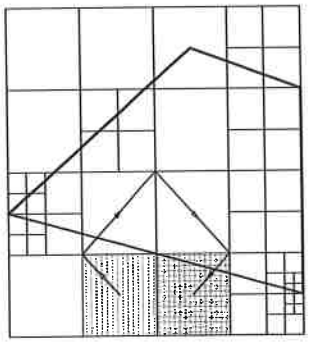
Encontra-se em SAMET[15] um método para encontrar os vizinhos nas direções horizontal e vertical de quadtree homogêneo, isto é, que possua todos os quadrantes terminais com o mesmo nível. O algoritmo apresentado a seguir é fundamentado nesse método, sendo, entretanto, mais geral, pois permite a busca para quadtree não homogêneo. Essa generalização do método não é encontrada na

literatura. O texto e o algoritmo se concentram na busca de vizinhos à direita, analogamente pode-se obter os vizinhos à esquerda, os superiores e os inferiores.

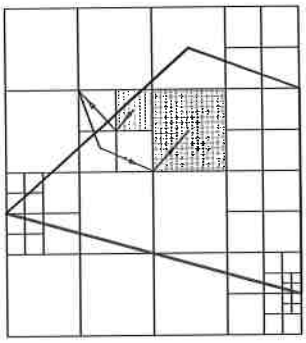
O método é composto por duas etapas: A primeira se concentra em, percorrendo a estrutura em árvore no sentido ascendente, encontrar o primeiro ancestral comum ao quadrante do qual se deseja determinar o vizinho, denominado quadrante origem, e ao quadrante vizinho. A segunda etapa se aplica em percorrer, a partir desse ancestral comum, um caminho descendente na árvore hierárquica até atingir o vizinho ou vizinhos procurados. Na primeira etapa, a partir do quadrante origem, percorre-se a lista de quadrantes pais, enquanto o caminho que levar até ele tiver a direção direita, já que se procura vizinho à direita, ou que o quadrante raiz tenha sido encontrado. Na segunda etapa, com o ancestral já determinado, e a partir dele, traça-se uma rota que é o caminho da primeira etapa espelhado na aresta comum ao quadrante origem e ao vizinho. Esta rota leva ao quadrante procurado.

Como o método prevê quadtree não homogêneo, deve-se contemplar a possibilidade do caminho que leva até o ancestral comum ter número de passos maior do que o do caminho espelhado, o que significa que há apenas um quadrante vizinho e que ele se encontra em nível inferior ao do quadrante original. Neste caso, o quadrante encontrado é o vizinho, conforme mostra a figura 5.7 a). Quando o caminho refletido apresenta número igual de passos, existem duas possibilidades. A primeira é a do quadrante encontrado estar no mesmo nível do quadrante que deu origem à busca e, nesse caso, o quadrante encontrado é o vizinho, conforme ilustra a figura 5.7 b). A segunda possibilidade é a do quadrante encontrado estar em nível superior e, portanto o quadrante origem apresentará mais de um vizinho, ou seja, os filhos do quadrante encontrado que estejam situados do lado esquerdo. É o caso ilustrado na figura 5.7 c). Na figura 5.8 é apresentado o algoritmo que determina o(s) vizinho(s) à direita de um determinado quadrante.

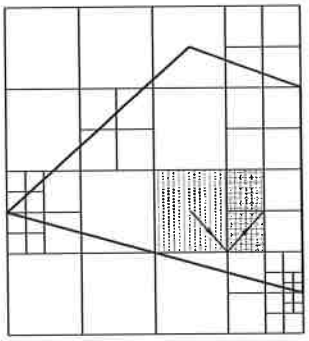
a)



b)



c)



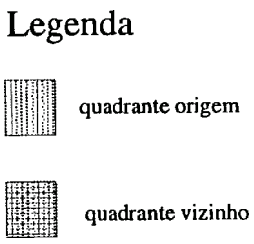


Figura 5.7 Caminho de busca de quadrantes vizinhos nas três situações possíveis

```
Função VIZ_DIR( quad_orig )
/* Objetivo:encontra o(s) vizinho(s) à direita de quad_orig armazenando-o(s) no
   vetor vizinho() */
{
passo     ← -1
npasso    ← 0
q         ← quad_orig

/*armazena no vetor rota() o caminho que leva até o ancestral comum */
repetir-enquanto passo ≠ 0  e  passo ≠ 1  e  q ≠ quad_univ
          passo       ← código da região[1]  (0,1,2 ou 3) em que q se encontra
          npasso      ← npasso + 1
          rota(npasso) ← passo
          q           ← localização de seu pai na estrutura de dados
fim-repetir

/* percorre um caminho que é o espelhamento de rota () */
repetir-enquanto npasso > 0 e filho de q na região 0 ≠ NULO
          passo ← rota( npasso )
          faça
              caso passo = 0
                    q  ← filho de q na
                         região2
              caso passo = 1
                    q  ← filho de q na
                         região3
              caso passo = 2
                    q  ← filho de q na
                         região0
              caso passo = 3
                    q  ← filho de q na
                         região1
          fim-faça
          npasso← npasso - 1
fim-repetir
```

1	3
0	2

[1] Os códigos das regiões, ou seja, dos quadrantes, se encontram no quadro ao lado do algoritmo.

```
vizinho(1)   ← q
se npasso = 0
      então nv     ← 1
                nvi ← 1
                repetir-enquanto nvi ≤ nv
                          q  ← vizinho(nvi)
                          se filho de q na região 0 ≠ NULO
                             então vizinho(nvi) ← filho de q na região 0
                                   nvi ← nvi + 1
                                   vizinho(nv) ← filho de q na região 1
                             senão nvi ← nvi + 1
                          fim-se
                fim-repetir
fim-se
}
```

Figura 5.8 Algoritmo para determinar os vizinhos a direita

5.4 Discretização do contorno

A discretização do contorno é efetuada percorrendo-se a lista de arestas que serão armazenadas de forma discreta, em posição adequada, dentro da estrutura de dados criada no item anterior.

Começando com o primeiro vértice, determina-se a intersecção da primeira aresta com o *box* de intersecção. O *box* de intersecção é composto por todos os lados de um, dois ou quatro quadrantes. Se a posição do vértice do contorno coincide com a do vértice do quadrante, o *box* de intersecção será composto pelos lados dos quatro quadrantes que compartilham daquele vértice, conforme ilustra a figura 5.9 a). Se o vértice coincide com o lado de um quadrante, o *box* de intersecção será formado pelos lados dos dois quadrantes adjacentes àquele lado, conforme ilustra a figura 5.9 b). O último caso é quando o vértice reside no interior do quadrante. Neste caso o *box* de intersecção será composto pelos lados desse quadrante, conforme ilustra a figura 5.9 c).

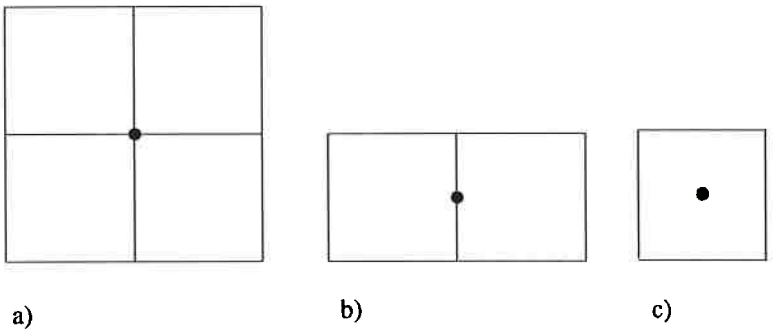


Figura 5.9 *Boxes* de intersecção usados para discretizar modelo de arestas.

Uma vez que o *box* de intersecção tenha sido determinado, a aresta é percorrida, a partir do primeiro vértice até que a primeira intersecção com o *box* seja encontrada. O vértice inicial mais o ponto encontrado formam o primeiro segmento, ou seja, a primeira informação discreta da aresta a ser armazenada na base de dados. Esta primeira intersecção será o ponto inicial do segmento seguinte e dará origem à busca do próximo ponto de intersecção ou vértice que comporá a segunda informação

de aresta discreta a ser armazenada. Dessa maneira, com o último ponto de intersecção encontrado servindo como ponto inicial do segmento seguinte, o processo continua até que a última aresta seja discretizada.

Admitindo uma forma parametrizada, podemos descrever as arestas através de suas componentes coordenadas como segue.

$$x_k = f_k(\xi) , \quad k=1,2$$

onde:

x_k é uma coordenada de um ponto pertencente a aresta

$f_k(\xi)$ é a função de mapeamento usada para expressar a coordenada x em termos do parâmetro x.

As coordenadas da próxima intersecção de uma aresta com um *box* de intersecção podem ser obtidas da seguinte maneira.

- Achar ξminimo com $\xi_n > \xi_m$, sendo

$$\xi_n = f^{-1}(X_j) , \; j = 1 \text{ ou } 2$$

onde:

ξ_n é a coordenada paramétrica da próxima intersecção da aresta com o *box* de intersecção.

ξ_m é a coordenada paramétrica da ultima intersecção da aresta com o *box* de intersecção.

$f^{-1}(X_j)$ é a função de mapeamento inverso que leva à j-ésima coordenada ao parâmetro x de um dos pontos da aresta.

X_j é a coordenada em x ou em y do lado do *box* que a aresta intercepta.

- Com o parâmetro de aresta do ponto de intersecção conhecido determina-se a outra coordenada através de:

$$X_i = f_i(\xi_n), \; i = 2 \text{ ou } 1$$

onde:

X_i é a coordenada em y ou em x da próxima intersecção da aresta com o *box* de intersecção

f_i é a função de mapeamento da i-ésima coordenada

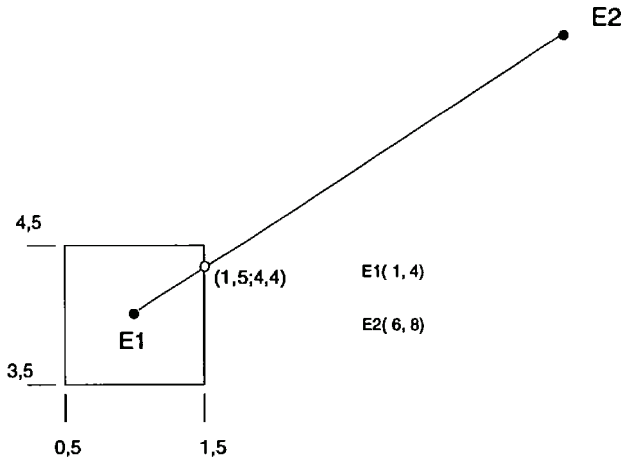


Figura 5.10 *Box* de intersecção e aresta do exemplo básico

Todo o desenvolvimento matemático para a construção de equação parametrizada de curvas cúbicas é apresentado por FOLEY[08], sendo que o procedimento matricial por ele apresentado pode ser generalizado para curvas de outros graus.

A seguir são apresentados todos os passos para determinação da intersecção de uma aresta, com equação de grau 1, com o *box* de intersecção. A geometria utilizada nesse exemplo está ilustrada na figura 5.10 e se trata de uma das arestas do exemplo básico e de um quadrante que foi classificado como *box* de intersecção, também do exemplo básico.

A aresta apresenta as equações paramétricas:

$x_1(\xi) = (-\xi + 1)\ 1 + \xi 6$
$x_2(\xi) = (-\xi + 1)\ 4 + \xi 8$

com $0 \leq \xi \leq 1$

Extraindo ξ, vem:

$\xi = (x_1 - 1)/\ 5$
$\xi = (x_2 - 4)/\ 4$

Testando as coordenadas dos lados do *box* de intersecção vem:

para $x_1 = 0{,}5 \Rightarrow \xi < 0$ (lado do *box* não intercepta aresta)
para $x_2 = 3{,}5 \Rightarrow \xi < 0$ (lado do *box* não intercepta aresta)
para $x_1 = 1{,}5 \Rightarrow \xi = 1/10$
para $x_2 = 4{,}5 \Rightarrow \xi = 1/8$

Portanto $\xi_n = 1/10$, logo $x_1 = 1{,}5$ e $x_2 = 4{,}4$.

O segmento de reta com extremidades dadas pelos pontos (1,0 , 4,0) e (1,5 , 4,5) é a primeira informação discretizada a ser armazenada na estrutura de dados junto ao quadrante a que pertence.

A figura 5.11 apresenta o algoritmo para discretização do contorno e a figura 5.12, o exemplo básico com o contorno discretizado.

```
Função DISCRETIZA_CONTORNO()
/* Objetivo:  percorre o modelo de arestas. Calcula para cada ocorrência sua
              intersecção com o box de intersecção . Armazena as
              informações relativas aà discretização do contorno no quadtree.
{
aresta← primeira aresta da lista de arestas
repetir-enquanto aresta ≠ NULO
        coo aresta(1)    ← coordenada na dir. x do primeiro vértice de aresta
        coo aresta(2)    ← coordenada na dir. y do primeiro vértice de aresta
        coo aresta(3)    ←coordenada na dir. x do segundo  vértice de aresta
        coo aresta(4)    ← coordenada na dir. y do segundo  vértice de aresta
        intersec         ← V
        repetir-enquanto intersec
                Encontrar box de intersecção de (coo_aresta(1),coo_aresta(2))
                box_inters() ← lista de quadrantes que compõe o box
                Calcular, se existir, a intersecção da aresta que   começa
                em (coo_aresta(1),coo_aresta(2)) com
                box_inters()
                se houve intersecção
                   então intersec ← V
                         coo inters()  ← coordenadas da intersecção
                         quad      ←quadrante onde ocorreu a   intersecção
                                   armazenar coo_inters() no quadtree em
                                   quad
                   senão intersec       ←F
                         armazenar(coo_aresta(3),coo_aresta(4))
                fim-se
        fim-repetir
        aresta ← próxima instância da lista de arestas
fim-repetir
}
```

Figura 5.11 Algoritmo de discretização do contorno

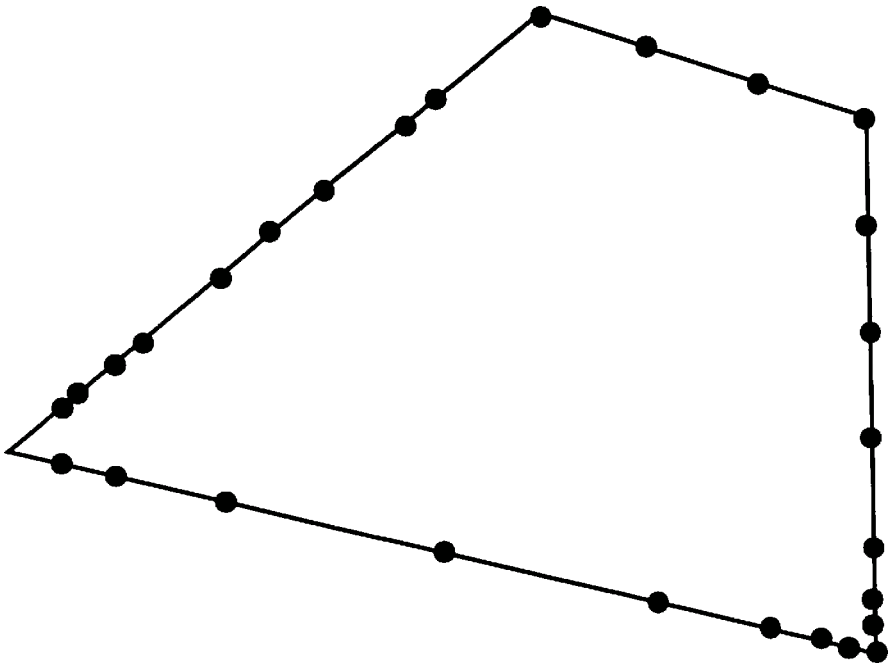


Figura 5.12 Exemplo básico com o contorno discretizado

No algoritmo de determinação da discretização do contorno pode-se acelerar o processo de cálculo dos pontos de intersecção, descartando as retas que garantidamente não interceptam o *box* de intersecção, usando o algoritmo de Cohen-Sutherland[08].

No método de Cohen-Sutherland, as linhas que definem o *box* de intersecção são prolongadas formando nove regiões. A cada uma dessas regiões é atribuido um código composto por quatro bites conforme ilustra a figura 5.13. É atribuido a cada ponto que define a reta a ser checada, o código da região em que ele reside. Realiza-se, então, uma operação lógica de conjunção (*and*) bite a bite. Se o resultado dessa operação resultar zero para todos os bites, a reta garantidamente não intercepta o *box* e pode ser rejeitada.

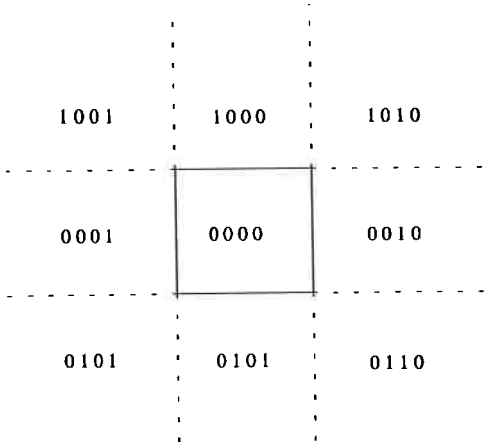


Figura 5.13 Código das regiões do método de Cohen-Surtherland

5.5 Determinação dos quadrantes do interior do domínio

A determinação dos quadrantes que definem o interior do domínio baseia-se em um algoritmo extraído da Matemática, cuja função é avaliar se um dado ponto pertence ou não a um certo domínio. Esse algoritmo encontra-se descrito a seguir.

Deseja-se determinar a partir da figura 5.13 se o ponto **P** está contido no interior do domínio nela definido. Inicialmente cria-se um retângulo que circunscreva o domínio, conforme figura 5.14. Toma-se, então, um ponto **Q** externo ao retângulo criado e, portanto, externo ao domínio. Liga-se o ponto **Q** ao ponto **P**, formando um segmento de reta. Percorre-se esse segmento de reta de **Q** para **P**. Como partiu-se de um ponto externo ao domínio, a primeira vez que o segmento **PQ** interceptar o domínio, estar-se-á entrando no domínio. Na próxima intersecção, estar-se-á saindo. A cada intersecção alterna-se essa condição até atingir-se o ponto **P**. Chega-se à conclusão, através desse algoritmo, que o ponto **P** pertence ao domínio.

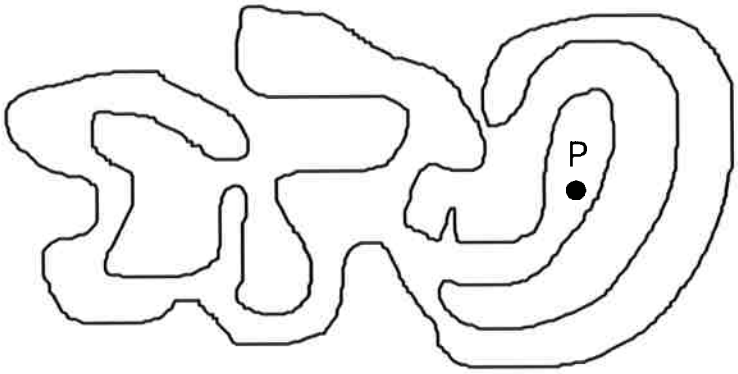


Figura 5.13 Problema: **P** está contido no domínio fechado?

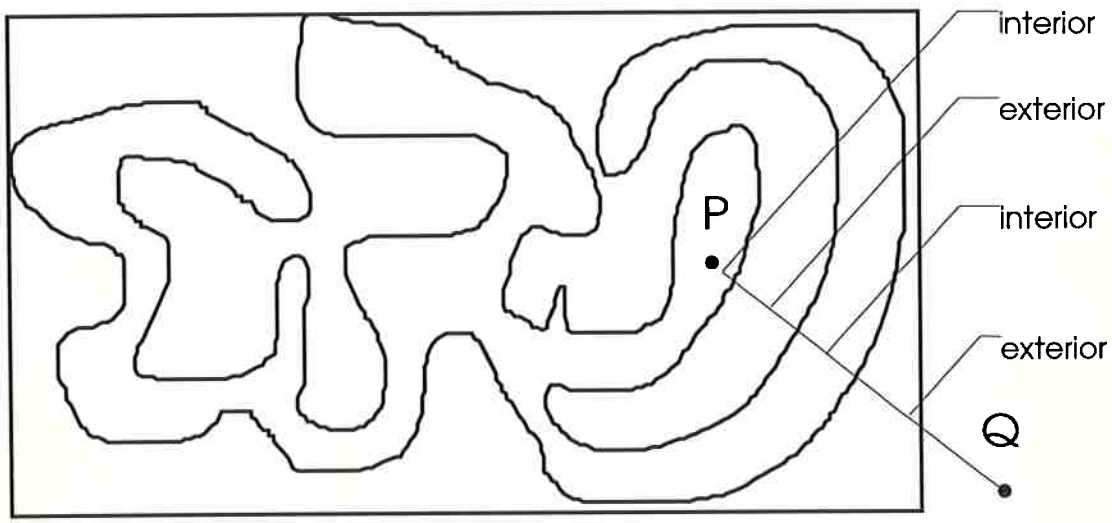


Figura 5.14 Ilustração do algoritmo para determinar se **P** pertence ao domínio

A adaptação desse algoritmo à determinação dos quadrantes do interior no método do quadtree é feita conforme descrito a seguir. Deve-se lembrar que um quadrante ou é do contorno, ou do interior ou está vazio. Os que pertencem ao contorno já foram determinados no passo anterior, esse algoritmo determinará os que estão vazios e os que estão totalmente contidos no interior.

Inicialmente toma-se o quadrante localizado mais à esquerda e ao alto. Percorre-se, a partir dele, todos os seus vizinhos à direita. Cada quadrante do contorno encontrado determinará a situação do seu vizinho à direita se ele não for também do contorno. Quando encontrado um quadrante do contorno verifica-se se houve intersecção do modelo de ciclo com um dos seus lados definidos por retas horizontais. O número de intersecções detectadas determinará o estado do próximo vizinho à direita, se ele não pertencer ao contorno. Isto é, se o último quadrante avaliado tivesse recebido o *status* de vazio, a próxima intersecção conferirá um *status* de interior para o próximo quadrante e vice-versa. Estes passos são repetidos tomando como origem cada um dos quadrantes localizados à esquerda no quadtree.

Outro algoritmo de definição dos quadrantes do interior pode ser encontrado em Baehmann [01]. O algoritmo aqui apresentado é exclusivo desse trabalho.

```
Função DEFINE_INTERIOR(quad)
/*  Objetivo : define o status dos quadrantes que não pertencem ao contorno
    como quadrantes do interior ou quadrantes vazios. Recebe como parâmetro
    inicial o quadrante localizado mais à esquerda e ao alto do quadtree. */
{
se  quad ≠ NULO
    então  viz    ← vizinho à direita de quad
           stat   ← status de quad
           repetir-enquanto viz ≠ NULO
                  statviz ← status de viz
                         se statviz = VAZIO e stat = CONTORNO
                                então  se  número de intersecções em quad for ímpar
                                              então  se statcor = VAZIO
                                                        então statcor = INTERIOR
                                                        senão statcor = VAZIO
                                                     fim-se
                                                     trocar status de viz para statcor
                                       fim-se
                         fim-se
                         stat← statcor
                         viz    ← vizinho à direita de quad
           fim-repetir
           quad   ← vizinho inferior de quad
           Define_interior( quad )
fim-se
}
```

Figura 5.15 Algoritmo de definição dos quadrantes do interior do domínio

CAPÍTULO 6

Interface do sistema com o usuário

Os anos 80 assistiram a um grande esforço da tentativa de otimizar dois escassos recursos de computação: tempo e memória. Com a evolução dos equipamentos e a conseqüente redução dos preços de hardware e software, interfaces homem/máquina de alta qualidade passaram à evidência como a última fronteira para a massificação do uso da informática. A tentativa de definir metodologias para o projeto de interface com o usuário e a necessidade de formalizar conceitos fez nascer uma nova sub-área da engenharia de software chamada engenharia de interface com o usuário.

A engenharia de interface com o usuário possui um caráter multidisciplinar, abrangendo áreas como a ciência de computação, as psicologias de percepção e cognitivas e os fatores de ergonomia (como interagir com equipamentos). Dentro de um outro enfoque, o da linguagem, poderíamos imaginar o estudo do diálogo homem/computador como sendo pertencente ao campo da semiótica (semeion = signo) que é definida como a ciência de toda e qualquer linguagem[1].

[1]Estudiosos da semiótica definem seu campo de investigação como sendo tão vasto que chega a cobrir o que chamamos de vida, visto que, desde que a descoberta da estrutura química do código genético,

O cuidado com o desenvolvimento de uma interface amigável é facilmente justificável se considerarmos os dados apresentados por Clarke e Muller [14]. Segundo esses autores o custo de uma solução típica de engenharia que envolve o método dos elementos finitos pode ser dividido em 80% de custos de engenharia e 20% de custos de computação. Os 80% dos custos de engenharia consistem em 65% para a preparação dos dados de entrada e 15% para interpretação dos resultados. O uso de computação gráfica interativa muda esses custos como segue: custos de preparação dos dados de entrada podem ser reduzidos para 20%, para interpretação dos resultados podem ser reduzidos para 10% e de computação sobem para 30%. A economia total[2] é então da ordem de 40%.

Apresentam-se, neste capítulo, os conceitos básicos de interface usuário/computador utilizados no projeto do GAMA2D. Deve-se lembrar que o pré-processador é constituído de dois módulos: o modelador de figuras planas e o gerador automático de malhas e que, embora os conceitos apresentados nesse capítulo sejam absolutamente gerais, os seus exemplos prendem-se ao modelador de figuras planas.

Abordam-se, inicialmente, as qualidades desejáveis a qualquer interface gráfica e que devem ser meta constante do projetista do sistema. A seguir, define-se o projeto da linguagem usuário/computador que atenda os quesitos de qualidade estabelecidos. O projeto da linguagem é apresentado em cada uma das suas etapas: projeto conceitual, projeto semântico, projeto sintático e projeto léxico, sendo que cada uma dessas etapas é ilustrada através de exemplos extraídos do projeto do modelador de figuras planas. Por fim, são discutidos o projeto da tela e as características de cada implementação física do modelador de figuras planas.

---

aquilo que chamamos de vida não é senão uma espécie de linguagem, isto é, a própria noção de vida depende da existência de informação no sistema biológico[Lúcia Santaella - O que é semiótica]

[2]Os dados levantados por Clarke e Muller basearam-se em sistemas de grande porte. Os custos de computação seriam reduzidos significativamente se fosse utilizado microcomputador. Qualitativamente, porém, os dados são bastante sugestivos.

6.1 A qualidade do diálogo computador-usuário

O cuidado com a definição do diálogo computador-usuário é essencial para o sucesso do projeto de um sistema interativo. Durante muitos anos, pela escassez de recursos de hardware, predominou o diálogo orientado pelo usuário, ou seja, o usuário digitava a frase completa e o computador antes de executá-la realizava uma análise semântica e sintática dos comandos. É facil inferir a quantidade de erros resultantes desse tipo de interação e a dificuldade de aprendizado na utilização do sistema. Em contrapartida no diálogo orientado pelo computador o sistema orienta as ações do usuário através de menus e *prompts* sugerindo as tarefas a serem realizadas. É evidente a superioridade desse tipo de diálogo no que se refere a facilidade de aprendizado e utilização e na minimização de erros.

Conseguem-se boas referências para a elaboração do projeto do diálogo orientado pelo computador, através da analogia que pode ser feita com o diálogo pessoa a pessoa. O homem desenvolveu várias maneiras de se comunicar, seja através de gestos ou palavras, ou mesmo através de gestos, sons e imagens, antes do domínio da fonética. A computação gráfica rompe com a restrição da linguagem puramente verbal, expandindo os horizontes da comunicação através do uso da imagem.

Os principais objetivos a serem alcançados pela interface com o usuário são: aumento da velocidade de aprendizado e da velocidade de utilização, redução da taxa de erro e aumento do poder de atração exercido em potenciais usuários. Foley e van Dan[08] enumeram os seguintes princípios a serem adotados quando da construção de um projeto de linguagem homem/máquina:

Seja consistente

Um sistema consistente é aquele no qual o modelo conceitual, a funcionalidade, as seqüências e a interação com o hardware são uniformes e seguem algumas poucas e simples regras. A proposta básica da consistência é permitir a generalização dos conhecimentos adquiridos em alguns aspectos do sistema para o sistema todo. Consistência permite também evitar a frustração resultante da interação com um sistema que não se comporta de maneira inteligível e lógica. A melhor maneira de se alcançar um sistema consistente é através de um cuidadoso projeto *top-down* do sistema. Apresentam-se a seguir alguns exemplos de consistência a serem observados no projeto da interface do sistema com o usuário:

Os mesmos signos para interação com o usuário são sempre empregados, por exemplo, as cores sempre codificam as informações da mesma maneira, o mesmo tom de vermelho significando pare e o mesmo tom de verde indicando prossiga.

As mensagens do estado atual do sistema são sempre mostradas em uma localização logicamente (nem sempre se encontram na mesma posição física) bem determinada.

Em relação à estrutura do menu, seus itens sempre aparecem fisicamente na mesma posição.

Outros exemplos de consistência, agora na porção da interface relativa à entrada do usuário são:

Os caracteres do teclado, como por exemplo: *tab*, *return*, *backspace* e *line feed* sempre têm a mesma função e podem ser usados sempre que um texto esteja sendo fornecido.

Comandos globais, como por exemplo: Ajuda e Cancela, podem ser ativados a qualquer momento.

Comandos genéricos, como por exemplo: Move, Copia e Apaga, estão disponíveis e podem ser ativados para qualquer tipo de objeto do sistema.

Forneça feedback

Deve ser frustrante conversar com uma pessoa que nunca esboça um sorriso ou demonstra qualquer descontentamento e responde apenas quando é forçada a fazê-lo. Esta frustração se deve ao fato de não se ter nenhuma indicação de que a pessoa esteja entendendo o que estamos tentando transmitir. O feedback é tão essencial no diálogo humano quanto na interação do homem com a máquina. A grande diferença é que na conversação com outra pessoa há muitas fontes de resposta (gestos, linguagem corporal, expressão facial, olhar) que são fornecidas inconscientemente pelos seus participantes. Pelo contrário, um computador fornece pouquíssimo feedback de maneira automática (apenas a luz de indicação de ligado), deste modo, todo o feedback deve ser planejado e programado.

Existem três formas de feedback correspondentes aos diferentes níveis da linguagem: léxico, sintático e semântico. O nível mais baixo é o léxico sendo que cada ação léxica na linguagem de entrada deve possuir uma correspondente resposta léxica na linguagem de saída. Por exemplo, o eco dos caracteres digitados em um teclado ou o movimento do cursor na tela quando o usuário movimenta o mouse.

O feedback no nível sintático ocorre quando cada parte da linguagem de entrada (comando, posição, objeto escolhido, etc.) é aceita pelo sistema. Por exemplo, no GAMA2D quando um botão é selecionado o ícone a ele relacionado é

ampliado e transferido para uma região da tela que informa ao usuário qual o comando corrente. Outro exemplo se refere à seleção de entidades geométricas que passam a ser identificadas através de linhas tracejadas.

O feedback semântico comunica ao usuário que a operação requisitada foi completada. Isso é feito através da apresentação simultânea da tela resultante dessa operação ou, o que as vezes se torna muito mais elucidativo, através da apresentação dos resultados parciais criando uma animação que ajuda o usuário a entender a conformação final da tela. É desejável também, em ações cujo tempo de resposta superem dois segundos, que sejam exibidas mensagens que indiquem que o computador está trabalhando sem problemas ou mesmo que informem qual o processo que está sendo executado.

Minimize as possibilidades de erros

Deve-se tentar projetar o sistema totalmente sensível ao contexto, isto é, o sistema deve guiar o usuário de modo a fazê-lo trabalhar conforme o contexto corrente, se ele existir, e tentar evitar ou tornar impossível ações que possam fazê-lo desviar-se de seus objetivos ou que apresentem efeitos colaterais desastrosos. Algumas observações importantes podem ser apresentadas:

Não ofereça opções de menu que possam acarretar uma mensagem do tipo "Seleção ilegal, comando não válido no momento".

Não deixe o usuário selecionar Apaga se não há nada a ser apagado.

Não deixe o usuário selecionar Copia quando nada foi selecionado para ser copiado

Forneça possibilidade de recuperação de erros

Há varias evidências que demonstram que o ser humano é mais produtivo se os seus erros puderem ser rapidamente corrigidos. Pode-se imaginar por exemplo o impacto na velocidade de digitação se fosse eliminada a tecla de *backspace* do computador. A cautela com que a digitação seria efetuada para não ocasionar nenhum erro a tornaria bem pouco produtiva. Outro fato, além da produtividade, se relaciona ao aprendizado. A possibilidade de recuperação de erros dá ao usuário a liberdade necessária para a exploração completa do sistema encorajando-o a experimentar e portanto aprender.

Preveja múltiplos níveis de interação

A interação com o usuário deve prever que o sistema possa ser utilizado por uma grande gama de usuários desde o totalmente inexperiente àquele que o utiliza durante várias horas por dia. Para os novos usuários é mais conveniente a utilização de menus e ícones que têm uma correspondência com atividades mais cotidianas da vida das pessoas e que portanto indiquem uma clara analogia com a ação que representam. Usuários mais experientes entretanto devem dar mais valor para a velocidade com que a comunicação das suas intenções são captadas. Para esse tipo de usuário é interessante que estejam disponíveis *hot keys* e mnemônicos de comandos.

O GAMA2D prevê que um mesmo comando possa ser ativado através da seleção de um item do menu suspenso ou dos ícones dos botões laterais, através da digitação do comando na região reservada à linha de comando, através de um mnemônico composto pelas três primeiras letras do comando ou por uma ou duas teclas (*hot key*) digitadas simultaneamente. O usuário menos experiente pode recorrer ao comando Ajuda sensível a contexto para conhecer as várias formas de ativar aquele mesmo comando.

Minimize a memorização

Muitas interfaces forçam uma memorização desnecessária do usuário. Quando da utilização de uma interface gráfica podemos, através do uso de signos padrões, fazer com que haja um reconhecimento e não uma memorização do comando pelo usuário. A padronização dos ícones e nomes de comandos vem fazendo com que usuários que aprenderam a utilizar um certo sistema sintam grande facilidade de transferir esses conhecimentos para outros sistemas que compartilham daquele mesmo tipo de interface.

6.2 Projeto da linguagem

Nos parágrafos anteriores desse capítulo estabeleceu-se a importância de uma interface com o usuário cuidadosamente projetada, bem como as qualidades desejáveis a essa linguagem computador/usuário. Com o objetivo de construir uma interface com o usuário que atinja todos os níveis de qualidade estabelecidos no item

anterior, detalha-se agora o projeto da linguagem, seguindo as quatro etapas sugeridas por Foley[08] que são: projeto conceitual, projeto semântico, projeto sintático e projeto léxico.

6.2.1 Projeto conceitual

O projeto conceitual trata da definição dos conceitos da aplicação que devem ser seguidos pelo usuário e pode ser chamado também de modelo do usuário. Consiste da definição dos objetos, das propriedades dos objetos, das relações entre eles e das operações válidas para cada um desses objetos definidos.

O projeto conceitual da interface com o usuário é descrito muitas vezes através de metáforas ou analogias às coisas com que o usuário típico já está habituado e deve ser tão simples e consistente quanto possível.

Procurou-se empregar, sempre que possível, no módulo relativo à modelagem de figuras planas do GAMA2D, conceitos que fossem similares à outros sistemas já consagrados e que possuissem sempre uma analogia com o processo convencional de desenho, isto é, aquele que se faz sem utilização de ferramentas de informática. Esses cuidados devem tornar o processo de aprendizado do modelador de figuras planas extremamente eficaz para o usuário típico do sistema, que é o engenheiro estrutural com alguma familiaridade em projeto assistido por computador.

Deve-se destacar novamente, que o preprocessador objeto desse trabalho, é composto de dois módulos: o modelador de figuras planas e o gerador de malhas de elementos finitos. Descreve-se na figura 6.1 a hierarquia dos objetos que compõe o modelador de figuras planas do pré-processador.

O objeto constituinte de um modelador de figuras planas é o seu modelo de faces, composto pelos ciclos. Os ciclos são compostos por arestas, vértices e orientação. A orientação é dada por uma lista de vértices ordenados, definindo um sentido de caminhamento. Estabeleceu-se que o ciclo define a região à direita do sentido de caminhamento. Associados aos vértices tem-se os entes geométricos pontos, definidos por suas duas coordenadas. Associadas às arestas tem-se as entidades geométricas curvas, cujas equações definem suas posições geométricas.

Na figura 6.2 é definida a lista de atributos dos objetos anteriormente apresentados. Os atributos de face referem-se às características físicas do material que as constitui. Sabendo-se que o modelador de figuras planas dará suporte a um gerador de malhas de elementos finitos, deve-se prever que o usuário possua controle

sobre o nível de refinamento da malha. Para tanto, as arestas e vértices têm que possuir um atributo extra, além da numeração, que é o parâmetro de controle de malha. Uma discussão detalhada dos conceitos de nível de refinamento e parâmetro de controle de malha pode ser encontrada no capítulo 6.

```
Preprocessador =
= { Modelador de Figuras Planas, Gerador de Malhas de Elementos Finitos }
    Modelador de Figuras Planas =
     = { modelo de faces }
     Modelo de Faces =
         = { lista de ciclos }
             Lista de Ciclos =
              = {  lista de arestas, orientação }
                   Lista de Arestas =
                    = { lista de vértices, lista de curvas }
                        Vértice =
                        = { ponto }
                            Ponto =
                            = { abscissa, ordenada }
                        Curva =
                         = {  pontos, equação }
                   Orientação =
                    = { lista de vértices ordenados }
```

Figura 6.1 Hierarquia dos objetos no modelo conceitual do modelador de figuras planas

```
Atributos =
= { atributos de face, atributos de aresta, atributos de vértice }
    Atributos de Face =
    = { material, espessura }
    Atributos de Aresta =
    = { numeração, parâmetro de controle de malha }
    Atributos de Vértice =
    = { numeração, parâmetro de contole de malha }
```

Figura 6.2 Lista de atributos dos objetos definidos no modelo conceitual

Definidos os objetos deve-se, agora, dentro do projeto conceitual, estabelecer as operações válidas para as várias classes de objetos. No modelador de figuras planas, as operações primárias válidas para faces, ciclos, arestas e vértices são:

EDITAR, CONSULTAR, VISUALIZAR, DEFINIR ATRIBUTOS, ARQUIVAR.

Cada uma dessas operações primárias pode ser decomposta em um conjunto de operações válidas para certas classes de objetos. As seguintes operações são válidas para todos os objetos:

EDITAR	: inserir, apagar.
VISUALIZAR	: enquadrar, *zoom*.
DEFINIR ATRIBUTO	: inserir, eliminar, ligar visibilidade, desligar visibilidade.
ARQUIVAR	: gravar, gravar "DXF"[3], carregar, carregar "DXF".

As operações ligadas a CONSULTAR são válidas para dois tipos de objetos: arestas e vértices. No caso das arestas tem-se:

CONSULTAR	: ponto final, ponto médio, comprimento, ângulo de inclinação.

No caso dos vértices tem-se:

CONSULTAR	: distância entre vértices.

Quando se deseja operar simultaneamente sobre várias instâncias de um mesmo tipo de objeto, ou até, sobre várias classes de objetos para as quais essa operação é válida, temos que dispor de um mecanismo eficiente de seleção. Apesar da seleção de objetos não ter por si só um efeito de transformação, podemos encará-la como uma operação. Trata-se de uma operação intermediária, é certo, mas deve ser definida como operação e, no nosso caso, é válida para todos os objetos.

---

[3]DXF refere-se a um padrão mundial de formatação de arquivos, usado inclusive pelo AutoCAD e é uma sigla extraída de *Draft eXtended File*.

SELECIONAR : um a um, por janela, por *crossing*.

Todas essas três operações de selecionar, são do tipo *pointing* ou por indicação. No primeiro caso, cada objeto apontado é selecionado. No segundo, todos aqueles que estiverem totalmente contidos dentro de uma janela definida, serão selecionados. No último caso, serão selecionados todos os objetos que estiverem contidos ou interceptarem as fronteiras da janela de seleção.

6.2.2 Projeto semântico

O projeto semântico é também conhecido como projeto funcional e está intimamente ligado ao projeto conceitual. Ele especifica detalhadamente o funcionamento da interface: quais informações são necessárias para cada operação sobre cada objeto, quais erros podem ocorrer, como esses erros devem ser tratados e quais os resultados esperados a cada operação. Deve-se notar que o projeto semântico define significados, não a forma ou seqüência com que as ações são conduzidas (que são tratadas pelo projeto sintático).

Apresentam-se, a seguir, dois exemplos de funções implementadas no modelador de figuras planas. Julga-se que esses exemplos sejam suficientemente esclarecedores e dispensem o detalhamento de todas as funções do sistema o que tornaria o trabalho extenso e monótono.

Exemplo 1

Função : Inserir Aresta (Linha)

Parâmetros : Abscissa e ordenada do vértice inicial, abscissa e ordenada do vértice final.

Descrição : A aresta ou linha é criada fornecendo-se as coordenadas do vértice inicial e as coordenadas do vértice final. Os valores numéricos das coordenadas podem ser digitados via teclado ou através do posicionamento do cursor na tela. O processo se repete até que haja um cancelamento explícito do usuário. A partir da segunda aresta a ser criada, usa-se como

coordenadas do seu vértice inicial os valores do vértice final da aresta anterior, evitando a repetição de digitação de dados. Essa função cria indiretamente os vértices, se eles não existirem.

Feedback : 1.As coordenadas fornecidas aparecem como uma pequena cruz e o desenho da aresta é mostrado instantaneamente na tela de desenho. 2.Se o vértice for fornecido através do posicionamento do cursor na tela, um efeito de *rubberbanding*[4] é ativado a partir do segundo vértice fornecido. 3.A região da tela que apresenta a posição do cursor é atualizada com os valores das coordenadas fornecidas. 4.A cor padrão para o desenho das arestas é o branco. Essa cor pode ser alterada pelo usuário.

Erros potenciais : Se as coordenadas fornecidas estiverem fora dos limites da janela de visualização do desenho, a aresta não será totalmente visível. Para conseguir-se a vizualização de todo o desenho pode-se ativar a função de enquadramento.

Exemplo 2

Função : Apagar aresta (linha).

Parâmetros : Aresta ou linha selecionada.

Descrição : Elimina a aresta ou linha selecionada. Notar que o operando pode ser um conjunto de entidades selecionadas e não apenas uma como descrito.

Feedback : 1. A aresta selecionada torna-se tracejada. 2. Logo após a função ser confirmada, a linha é eliminada do desenho.

Erros potenciais : Se nenhuma aresta tiver sido selecionada a função não apresentará nenhum resultado.

[4]O diagrama de estado, mostrado na figura 6.4, esclarece o conceito de *rubberbanding*.

6.2.3 Projeto sintático

O projeto sintático estabelece as seqüências de entrada e saída da interação. Na liguagem do usuário, define as regras pelas quais unidades de entrada, que por si só possuam siginificação, são formadas em sentenças completas. Chamando essas unidades de entrada de palavras, podemos dizer que o projeto sintático, na entrada, define as regras, pelas quais, seqüências de palavras são compostas formando frases.

Na saída, a noção de seqüência deve incluir fatores espaciais e temporais. Desse modo, a linguagem de saída inclui a organização da tela e as variações temporais que nele se apresentem.

Segundo Foley[08], as seqüências de ações permitidas ao usuário podem ser definidas através de diagramas de estado ou de transição e através de redes recursivas de transição. Paulino [14] utiliza também quadros de sintaxe para esclarecer a sintaxe de entrada do usuário na interação.

Apresenta-se, na figura 6.3, um exemplo de quadro de sintaxe aplicado à edição de entidades geométricas e topológicas. Essa ferramenta, pela sua proximidade com a língua vernácula, dispensa maiores comentários. Na seqüência, analisa-se um pouco mais detalhadamente os diagramas de estado.

SUJEITO	VERBO	OBJETO DIRETO	
O usuário	insere apaga	um(a) um conjunto de	face aresta(s) vértice(s) todos

Figura 6.3 Quadro de sintaxe para EDIÇÃO DA GEOMETRIA/TOPOLOGIA.

Na criação de uma interface do usuário sensível a contexto, a resposta do sistema às ações do usuário deve depender do estado atual da interface. Essa resposta pode: invocar uma ou várias rotinas, alterar o conteúdo de uma ou mais variáveis de estado, modificar as técnicas de interação ou exibir outros itens do menu em preparação para a próxima ação do usuário. Associada ao estado atual da interface temos as variáveis de estado cujo valor pode ser alterado por todas as rotinas que realizem alguma ação e que, portanto, afetem o comportamento da interface do usuário.

Os diagramas de transição possuem uma, e apenas uma, variável de estado. Esta variável guarda um valor inteiro que indica o estado corrente do sistema. As ações do usuário causam transições de um estado para outro. Cada uma dessas transições tem a ela associada nenhuma, uma ou mais rotinas que realizam alguma ação e que são chamadas assim que a transição ocorre.

Deve-se ressaltar a importância dos diagramas de estado também no que se refere à detecção de inconsistências nas seqüências de entrada e também na avaliação do número de passos necessários para que uma determinada tarefa seja completada. Sendo que um dos parâmetros usados para se avaliar o desempenho de uma interface é justamente o número de passos que levam a cabo uma determinada tarefa, os diagramas de estado servem para se fazer um prognóstico da qualidade da interface mesmo antes de implementá-la.

Apresentam-se, a seguir, alguns exemplos de diagramas de estado utilizados no projeto sintático do modelador de figuras planas desenvolvido nesse trabalho. O primeiro exemplo, diagrama de estado para criação de uma linha utilizando *rubberbanding*, foi especialmente escolhido para esclarecer o conceito de *rubberbanding*, citado anteriormente.

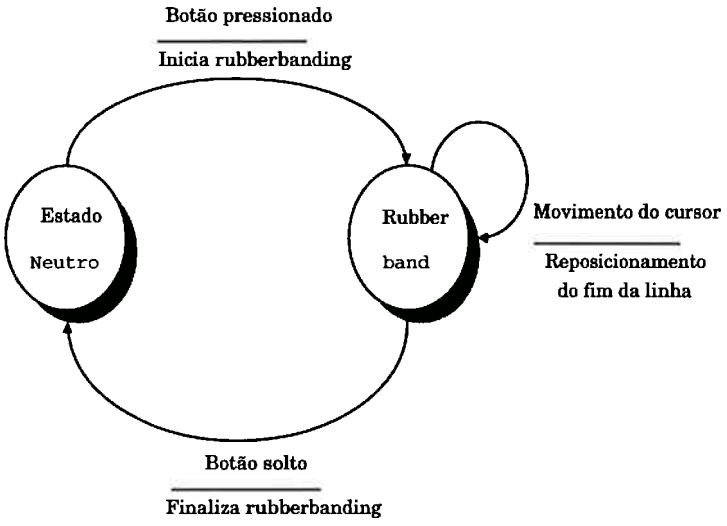


Figura 6.4 Diagramação de estado para a inserção de uma linha utilizando *rubberbanding*

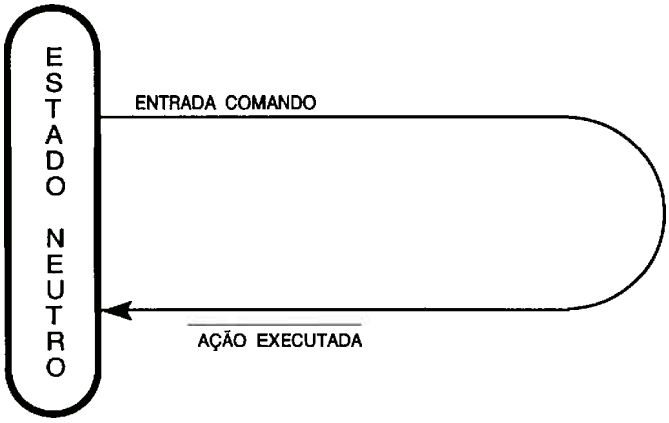


Figura 6.5 Diagrama de estado para função *Zoom* Total

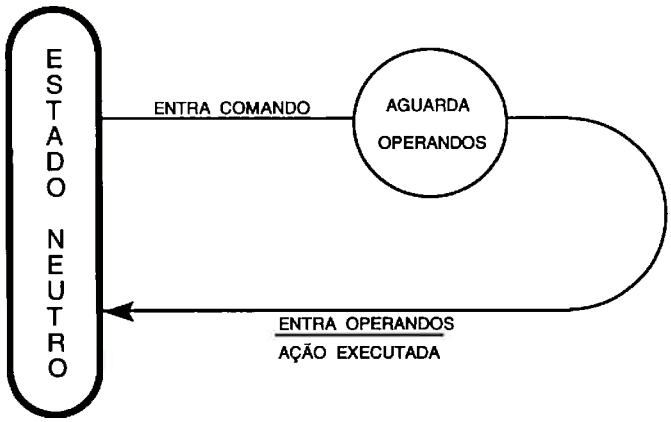


Figura 6.6 Diagrama de estado para função *Zoom Window*

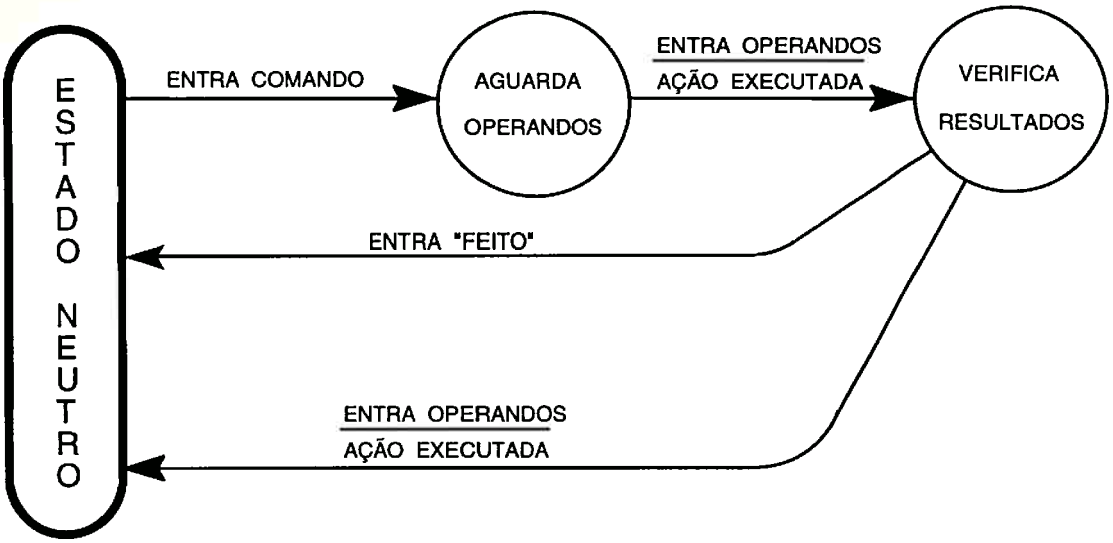


Figura 6.7 Diagrama de estado para função Retângulo

6.2.4 Projeto léxico

O projeto léxico especifica como cada palavra utilizada nas frases de entrada e saída são construídas em termos de primitivas de hardware[5]. Deve-se considerar que palavra e frase estão sendo utilizadas com o sentido estabelecido no projeto sintático. Primitivas de entrada são aquelas providas pelos dispositivos de entrada e primitivas de saída são as formas (tais como linhas e caracteres) e seus atributos (tais como cor e tipo de fonte) providos pelas subrotinas do pacote gráfico.

O projeto léxico recai então, no seu módulo de entrada, na escolha de uma técnica de interação. No módulo de saída, o projeto léxico, é construído através da combinação de primitivas de saída e seus atributos para formar ícones e outros símbolos. O projeto léxico incumbe-se, então, de fazer a ligação do projeto lógico,

[5]Primitivas de hardware são também chamadas de *lexemes*, ou em Português, lexema ou semantema.

que define a linguagem usada na interface independentemente do hardware, com as capacidades de hardware.

Essas técnicas de interação, citadas no parágrafo anterior, são divididas em tarefas básicas de interação que permitem que o usuário forneça uma unidade de informação (palavra) que seja inteligível dentro do contexto corrente da aplicação. As tarefas básica podem ser reunidas em cinco grupos: a tarefa de posicionamento, a tarefa de seleção dentro de um conjunto de tamanho variável, a de seleção dentro de um conjunto de tamanho fixo, a de entrada de textos e a tarefa de quantificação.

A tarefa de posicionamento envolve: o sistema de coordenadas, a resolução, os *grids*, o feedback, se há direção preferencial de trabalho e o tempo de aprendizado. A tarefa de seleção dentro de um conjunto de tamanho variável envolve a seleção de objetos por nome ou através do ato de apontar (*pointing*). A tarefa de seleção dentro de um conjunto de tamanho fixo se faz normalmente através de menus, envolvendo: organização dos itens do menu, escolha entre menus de um nível ou hierárquico, localização do menu, tamanho e forma dos seus itens, reconhecimento de padrões e utilização de teclas de funções. A tarefa de entrada de textos envolve o reconhecimento dos caracteres fornecidos. E, por fim, a tarefa de quantificação trata das técnicas de entrada de valores númericos com escopo definido, ou seja , com limites inferior e superior fixos. Foley[08] aborda com detalhes todas essas tarefas e eventos, incluindo técnicas de interação em 3D.

6.3 Projeto da tela

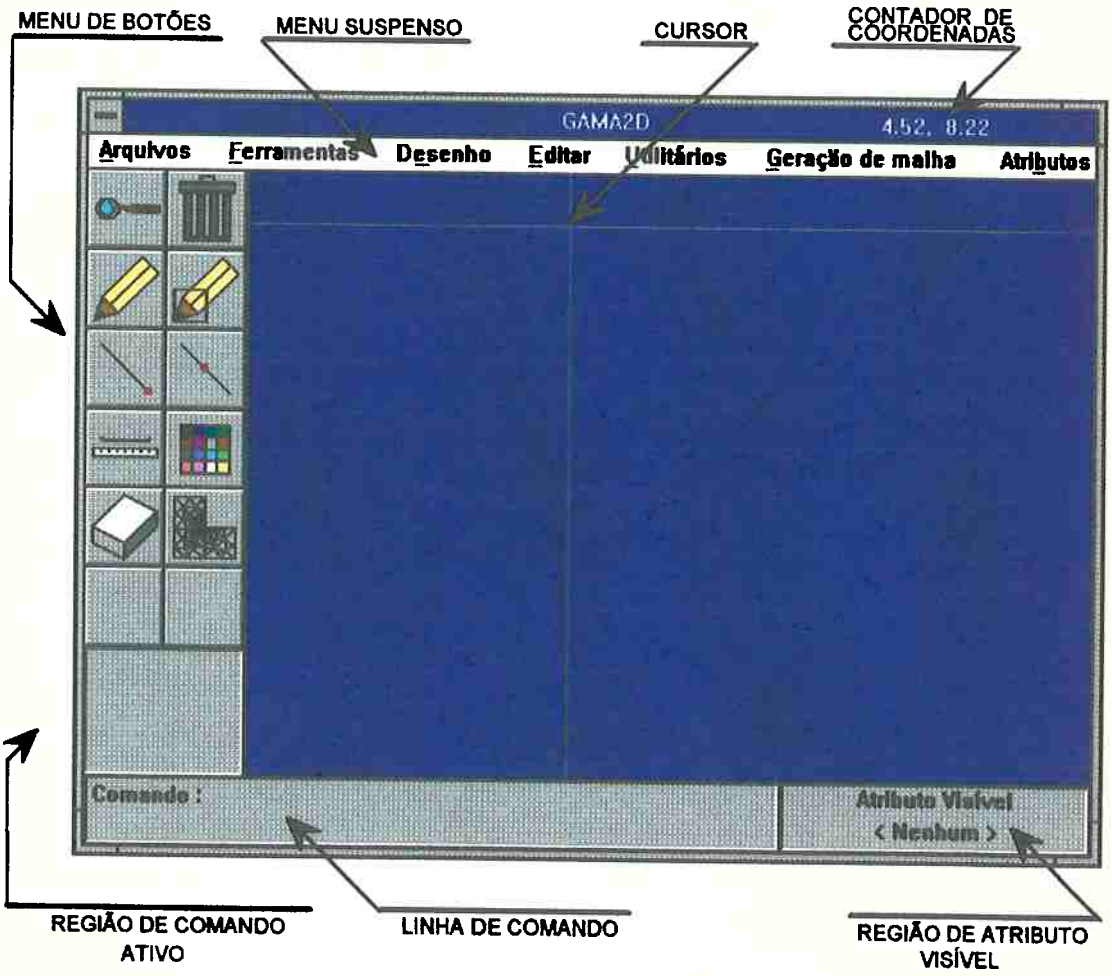
O projeto da tela é fundamental para o sucesso e aceitação de qualquer sistema de computação; é ainda mais importante para aqueles ligados à computação gráfica. A programação visual, resultado do projeto da tela, afeta tanto os usuários iniciantes quanto aqueles que devem passar horas utilizando o sistema. Esse projeto deve incluir a escolha do estilo da interface com o usuário, estilo e forma dos menus, uso das cores, codificação das informações e escolha da colocação das unidades de informação em relação ao conjunto. Os reultados a serem obtidos através de um bom projeto de tela são: clareza, consistência e aparência atrativa.

O estilo de interface adotado para o GAMA2D é o WYSIWYG (*What You See Is What You Get*), isto é, a representação com que o usuário interage na tela é essencialmente a mesma imagem que está sendo criada pela aplicação.

Na escolha do estilo e forma de menus optou-se pela convivência de dois tipos simultaneamente: o menu de textos e o de ícones. O do tipo texto, é um menu suspenso do tipo *push-down* com seus itens organizados e agrupados de acordo com sua funcionalidade. Optou-se por um projeto hierárquico na organização dos subitens sendo que a ordem adotada foi a de freqüência de utilização. O de ícones, é um menu de botões situado do lado esquerdo da tela apresentando os itens de maior utilização. Ele não possui subníveis e pode ser desligado a qualquer momento para aumentar a área de trabalho.

No projeto da tela, a codificação das informações se preocupa com a criação e a escolha de padrões visuais que distingam os vários tipos de objetos. Segundo Foley[08] as formas de codificar informações em ordem decrescente de eficiência são: cor, forma geométrica, espessura de linha, tipo de linha e intensidade. Para uma eficiência de 95% no reconhecimento dos padrões pode-se usar no máximo: 10 cores, 15 formas geométricas, 6 espessuras e 4 intensidades. Existe uma tendência de se usar codificação redundante para representar a mesma informação. Na elaboração da codificação visual do GAMA2D foram adotados os critérios descritos nesse parágrafo sendo que um bom exemplo é a seleção de linhas. O primeiro critério para se distinguir linhas selecionadas é o uso de cor. O GAMA2D pode estar sendo usado porém em monitor monocromático. Usou-se então codificação redundante para as linhas selecionadas: além da mudança de cor, adotou-se outro tipo de linha.

O último estudo que se deve fazer na elaboração do projeto da tela é o da escolha da colocação das unidades de informação em relação ao conjunto. Pode-se usar a técnica de dividir a superfície de visualização em regiões onde tipos diferentes de informações são apresentadas. Estudos indicam que para uma distribuição mais harmônica dessas regiões utilizem-se retângulos com uma das seguintes proporções entre seus lados: 1:1, 1:1.414, 1:1.618 ou 1:2. No GAMA2D a superfície de visualização foi dividida em 7 regiões: três regiões de informação do sistema (estado atual, comando ativado, atributo ligado, posição do cursor ...), três regiões para entrada de comando (menu de botões, menu suspenso e linha de comando) e a área de trabalho. A figura aprsentada a seguir mostra as regiões da superfície de visualização do GAMA2D.


MENU DE BOTÕES
MENU SUSPENSO
CURSOR
CONTADOR DE COORDENADAS
GAMA2D
4.52, 8.22
Arquivos
Ferramentas
Desenho
Editar
Utilitários
Geração de malha
Atributos
Comando :
Atributo Visível
< Nenhum >
REGIÃO DE COMANDO ATIVO
LINHA DE COMANDO
REGIÃO DE ATRIBUTO VISÍVEL

6.4 Implementação física do modelador de figuras planas

Linguagem e versões

Todas as versões do GAMA2D foram escritas em linguagem C. A primeira utilizando o C da Microsoft (MsC), versão 5.01, e uma biblioteca de funções gráficas chamada BC - Biblioteca Compugráfica. A BC foi criada por um grupo ligado à Escola Politécnica da Universidade de São Paulo. Esse grupo fundou a SoftCAD, transformando a BC em CLBC para dar suporte também a aplicações gráficas em Clipper. A CLBC passou a crescer, cada vez mais, em direção ao Clipper, sendo que, o módulo para linguagem C ficou estagnado.

A segunda versão do GAMA2D, também utilizava o C da Microsoft porém na sua versão 6.0. Pelo motivo apresentado anteriormente, foi substituída a CLBC pela própria biblioteca de funções gráficas do MsC 6.0.

A terceira e última versão do GAMA2D utiliza o C da Watcom e sua biblioteca gráfica. Esta modificação foi necessária devido ao tamanho do GAMA2D quando transformado em programa executável. O GAMA2D foi criado para dar suporte ao gerador automático de malhas e portanto deve prever que haja um espaço adicional de memória para esse outro módulo e para os dados. Como esses dois módulos, carregados juntos, estavam ocupando praticamente toda a memória convencional do microcomputador, sobrava muito pouco espaço para os segmentos de dados. A solução foi explorar a memória acima dos 640 Kbytes convencionais, recurso oferecido pelo Watcom e não disponível no MsC 6.0.

Portabilidade e Padronização

A portabilidade se refere a independência que uma dada aplicação consegue ter dos diferentes dispositivos físicos disponíveis, ou melhor, a portabilidade mede a facilidade com que se pode transportar um determinado programa de aplicação entre configurações diversas de equipamentos.

A portabilidade do aplicativo pode ser alcançada através da escolha de uma linguagem de programação que também possua essa característica e através da criação de interfaces de alto nível entre as várias camadas de software que constitui o aplicativo. Devem ser criadas interfaces entre as rotinas da aplicação e a estrutura de dados, entre as rotinas da aplicação e as rotinas gráficas e entre as rotinas gráficas e os dispositivos gráficos.

Na implementação do GAMA2D escolheu-se a linguagem C, como já exposto anteriormente. A escolha dessa linguagem deveu-se, entre outros fatores, à sua forte característica de portabilidade.

A interface das rotinas do GAMA2D com sua estrutura de dados se dá através de operadores. Esses operadores dedicam-se exclusivamente às tarefas de manipulação dos dados armazenados nas estruturas através das funções básicas de eliminação, inserção e busca. Qualquer algoritmo que necessite se comunicar com a estrutura de dados o faz através dos operadores. A portabilidade reside no fato de que se o modelo de dados tiver que ser alterado ou substituído, apenas os operadores terão que ser reformulados sem que se tenha que rever qualquer outra rotina do programa de aplicação.

Na interface das rotinas do GAMA2D com as rotinas gráficas, ou seja, com a bibioteca gráfica do C, criou-se uma outra camada chamada biblioteca gráfica da aplicação. A função dessa biblioteca pode ser melhor explicada através de um exemplo. Foi criada uma função chamada Linha que recebe como parâmetros as coordenadas dos dois extremos, o tipo de traço e a cor da linha. Essa função recebe os parâmetros e invoca a função da biblioteca gráfica que desenha linha. A grande vantagem da criação dessa interface é que caso se venha a substituir a biblioteca gráfica, apenas as rotinas da biblioteca da aplicação terá que ser reformulada, sem que se tenha que alterar uma linha do programa de aplicação.

A interface da rotina gráfica com os dispositivos físicos é feita diretamente pela biblioteca gráfica do C. Apenas o *driver* para controle de *mouse* teve que ser desenvolvido através das interrupções do DOS.

Outra característica importante de um software é a sua capacidade de se comunicar com outros softwares. Nesse aspecto, são muito importantes os padrões gráficos que vêm se estabelecendo. O GAMA2D possui a possibilidade de gravar e recuperar figuras no formato DXF (Data eXchange File) que vem se tornando um padrão mundial e encontra-se implementado na maioria dos aplicativos gráficos, incluindo o AutoCAD.

Capítulo 7

Exemplos

Dos quatro exemplos escolhidos para serem apresentados neste capítulo, três foram extraídos da literatura e um é o exemplo básico que serviu para explanação de todos os conceitos sobre geração de malha. Na maioria dos casos, a malha final foi conseguida utilizando-se pontos extras de refinamento localizado, isto é, além dos parâmetros de controle de malha definidos para as entidades topológicas, utilizou-se a função de refinamento que discretiza regiões do domínio.

Os três primeiros exemplos são apresentados, cada um deles, com apenas uma configuração dos parâmetros de controle de malha, que controlam os níveis de refinamento. O quarto exemplo é mostrado em quatro fases distintas, aumentando-se sucessivamente os parâmetros de controle de malha, alcançando-se uma progressiva discretização do modelo.

Exemplo 1

O primeiro exemplo foi extraído de Zienkiewicz [23] e representa o modelo de uma barragem de gravidade conforme mostra a figura 7.1.

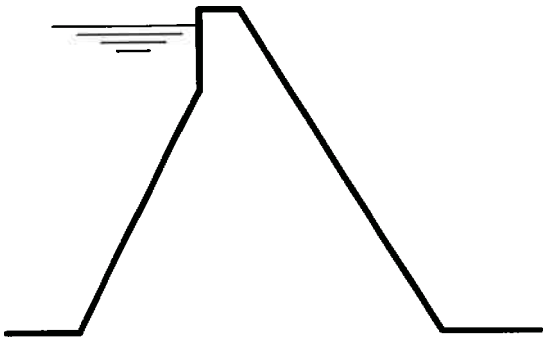


Figura 7.1 Modelo de barragem de gravidade

Os parâmetros de controle de malha escolhidos visaram alcançar os mesmos níveis de discretização utilizados na referência bibliográfica. A figura é apresentada a cores para se ter uma maior proximidade com a representação gráfica da tela do GAMA2D.

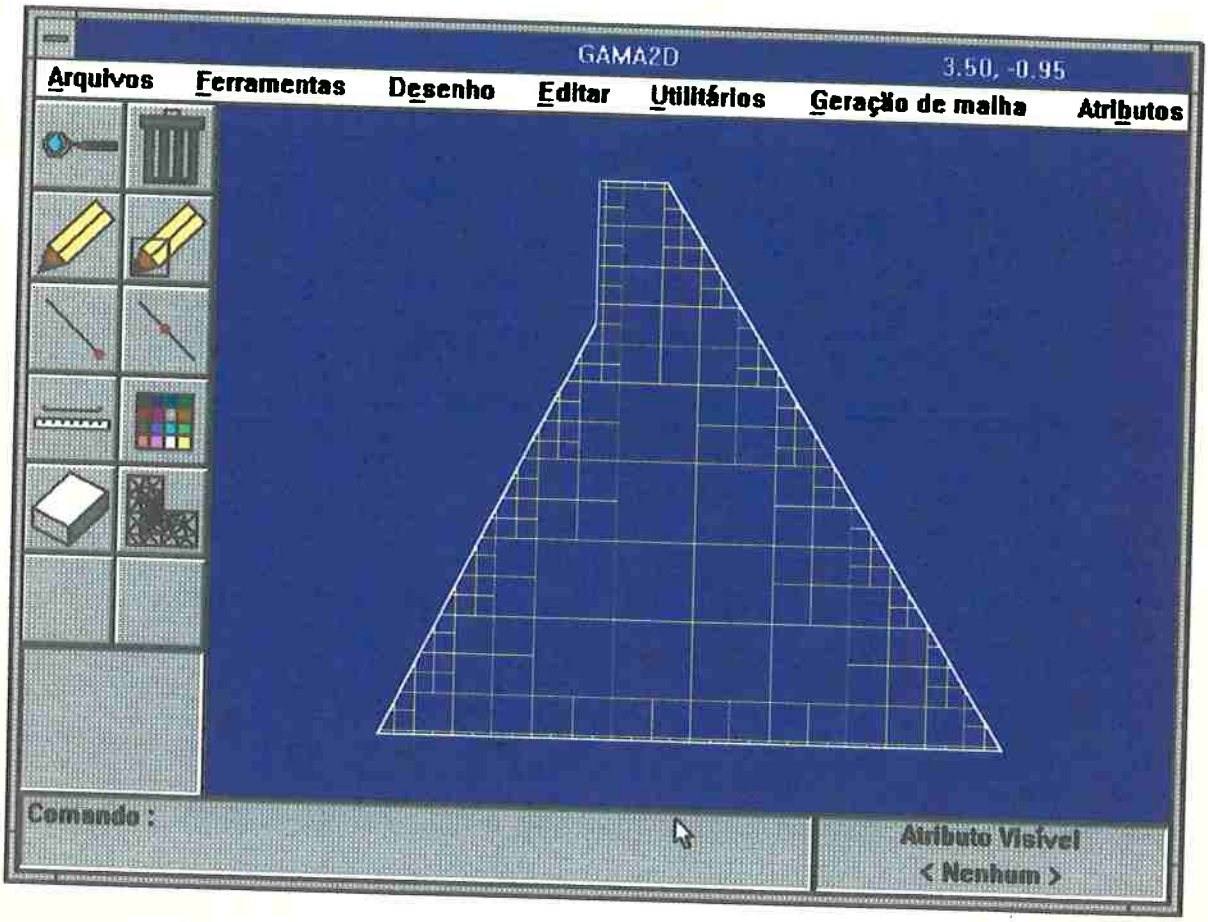


Figura 7.2 Barragem discretizada

Exemplo 2

O exemplo apresentado a seguir foi extraído de Baehmann [01] .Trata-se de um modelo plano quadrado, engastado em uma das extremidades e uniformemente carregado conforme apresentado na figura 7.3.

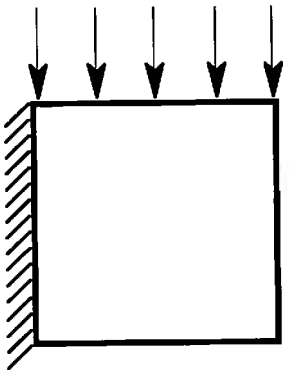


Figura 7.3 Modelo engastado com carga uniformemente distribuída

Na figura 7.4 é apresentada a discretização do modelo adotando-se parâmetro de controle de malha igual a 8 para o vértice superior esquerdo , igual a 5 para a aresta vertical esquerda e 4 para as outras arestas.

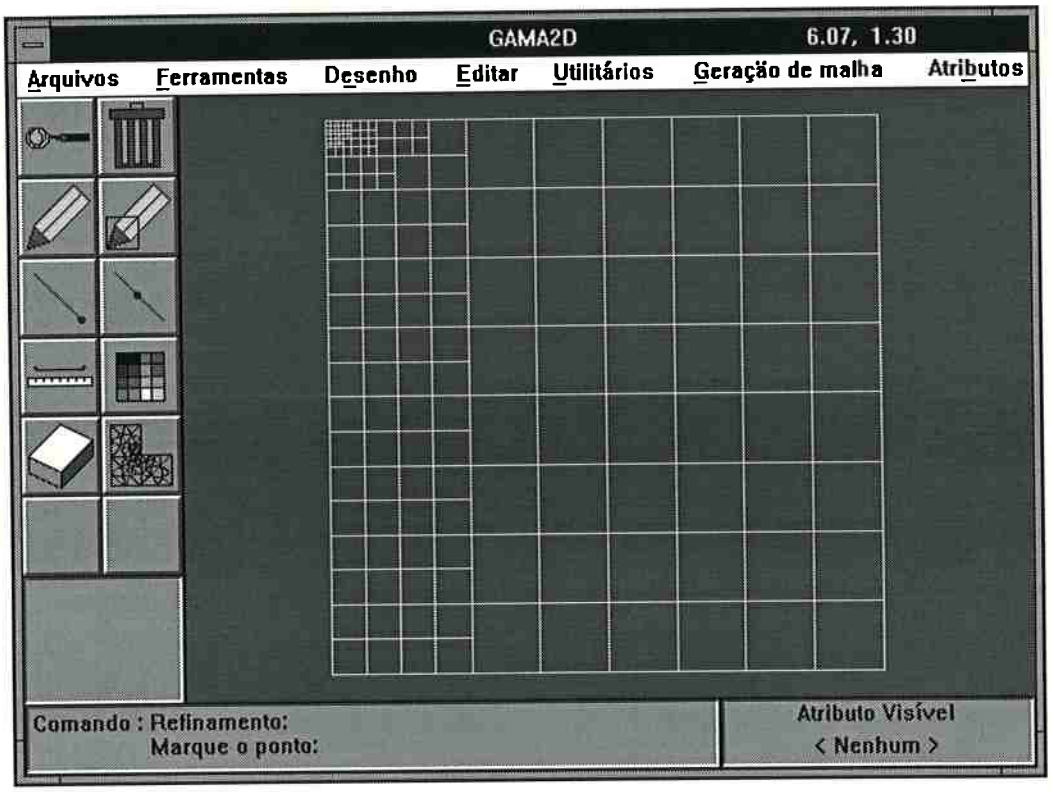


Figura 7.4 Modelo com nível de discretização máximo igual a 8

Exemplo 3

O exemplo 3 dispensa maiores comentários já que serviu como referência para a explanação de todos os conceitos de geração de malha. Apenas o modelo discretizado é apresentado. A geometria, bem como os parâmetros de controle de malha usados podem ser encontrados no capítulo 5.

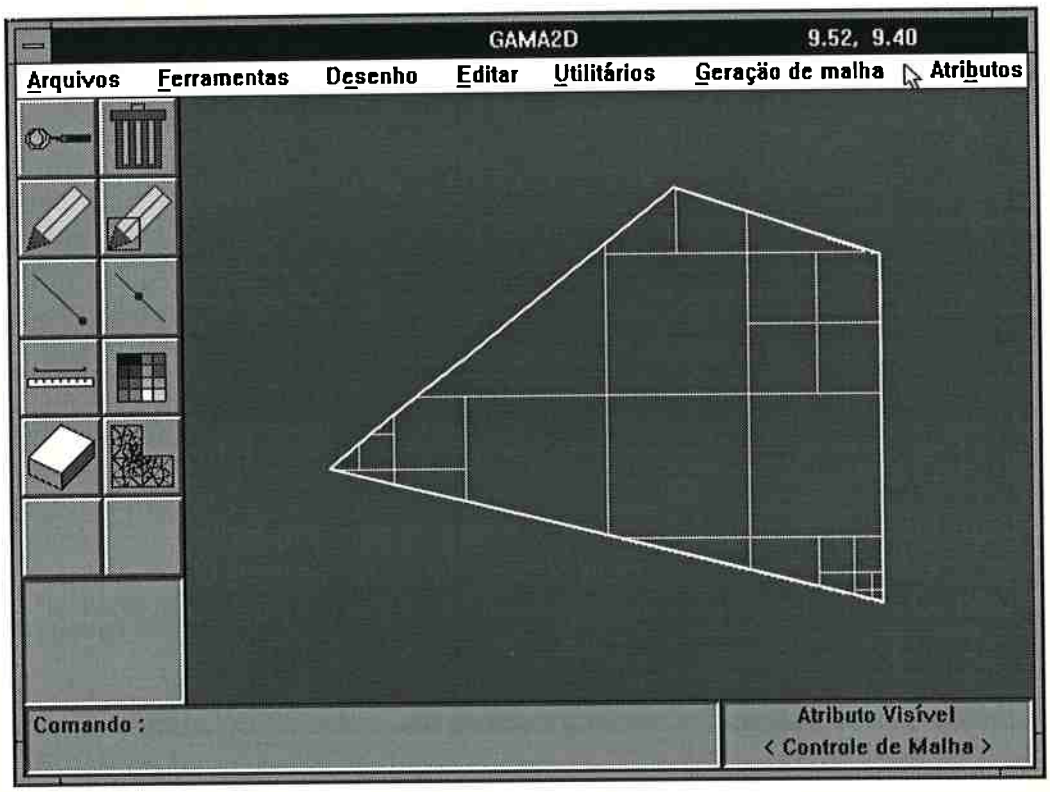


Figura 7.5 Exemplo básico discretizado

Exemplo 4

O exemplo 4 foi também extraído de Baehmann[1]. Trata-se de uma chapa quadrada com furo, sendo tracionada na direção de um dos eixos, como mostra a figura 7.5.

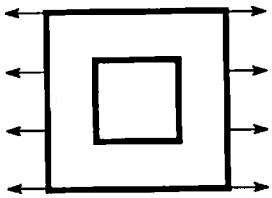


Figura 7.6 Modelo de chapa quadrada com furo sendo tracionada

A seqüência de figuras apresentadas a seguir apresentam níveis crescentes de refinamento. Na figura 7.7 foi escolhido parâmetro de controle de malha de valor 3 para todas as arestas.

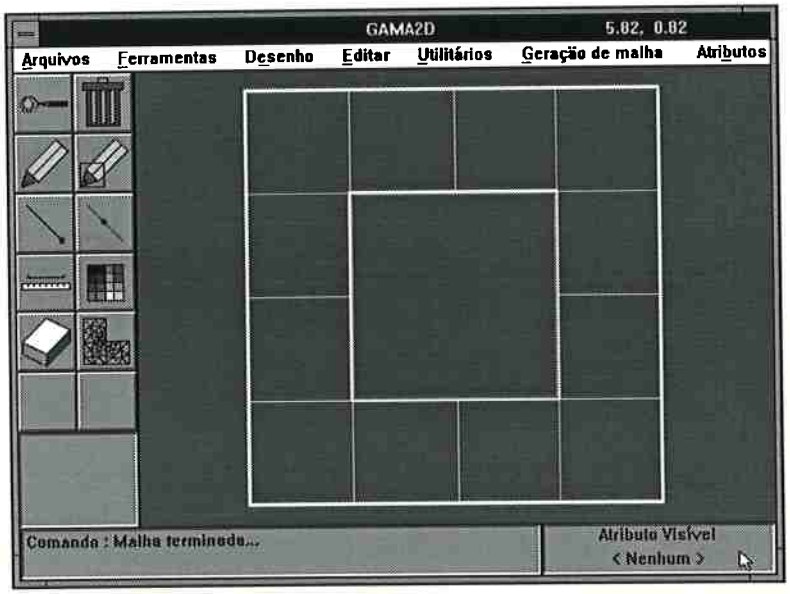


Figura 7.7 Discretização homogênea de nível 3

A figura 7.8 apresenta parâmetro de controle de malha igual a 3 para as arestas externas, igual a 4 para as arestas internas e 5 para os vértices internos.

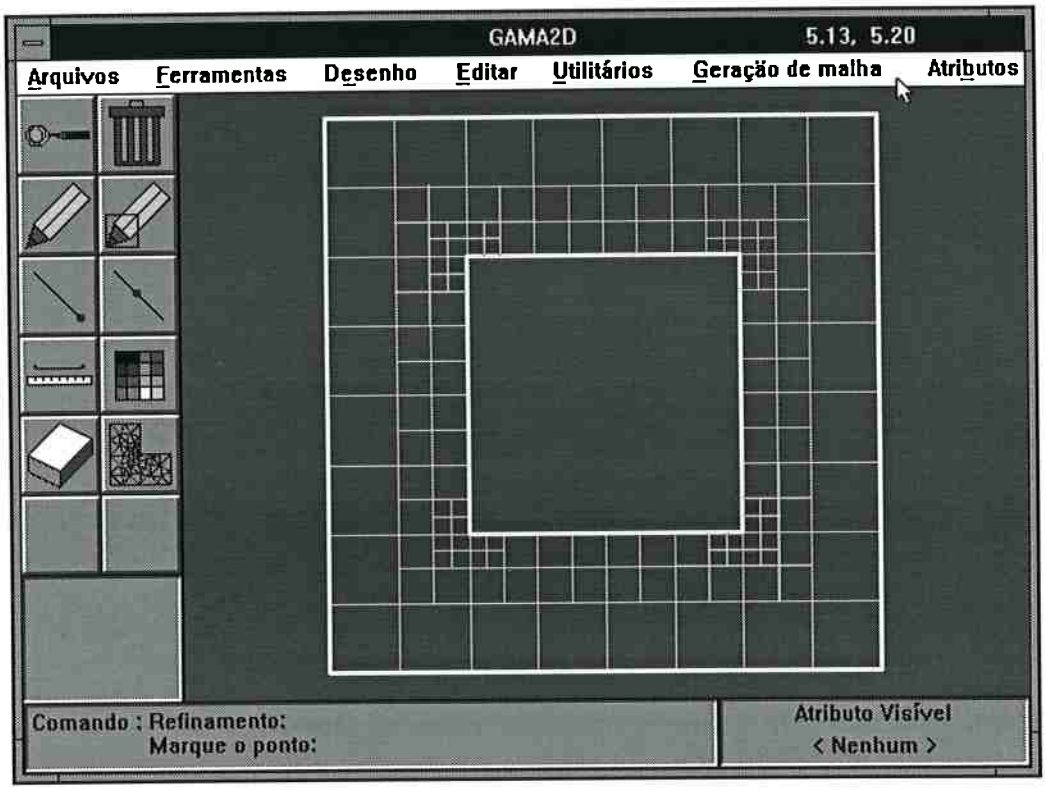


Figura 7.8 Discretização com parâmetro de controle máximo igual a 5

A figura 7.9 apresenta os seguintes parâmetros de malha: arestas externas iguais a 5, arestas internas iguais a 6 e vértices internos iguais a 8.

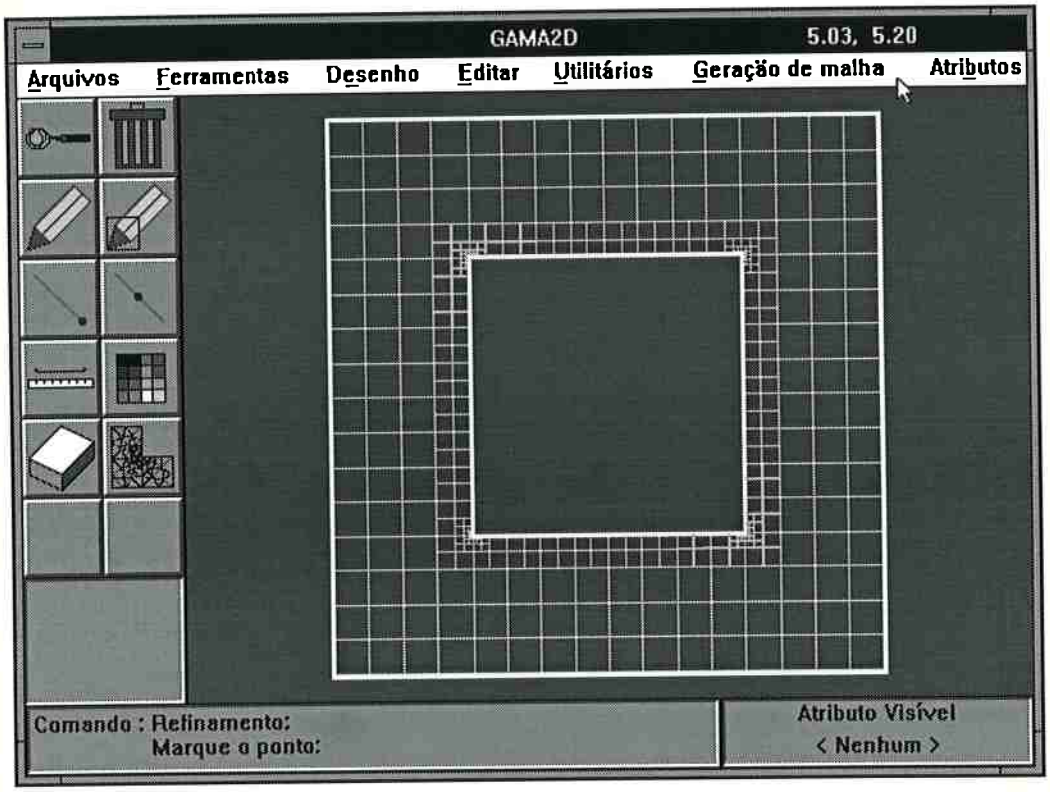


Figura 7.9 Discretização com parâmetro de controle máximo igual a 8

A figura 7.10 apresenta os seguintes parâmetros de malha: arestas externas iguais a 5, arestas internas iguais a 7 e vértices internos iguais a 8.

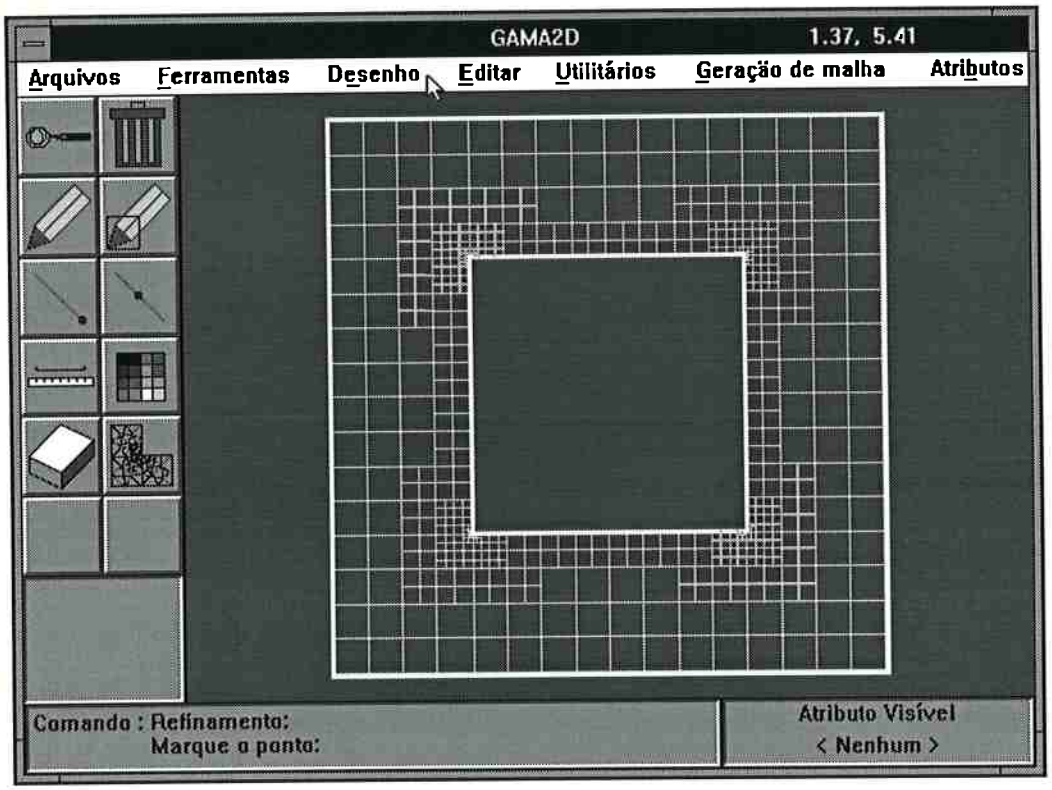


Figura 7.10 Discretização com parâmetro de controle máximo igual a 8 e refinamentos extras

Capítulo 8

Conclusões

Um sistema completo de análise adaptativa deve contar com componentes que sejam capazes de cumprir as seguintes funções: modelagem geométrica, geração de malha, análise por elementos finitos, estimativa de erros, atualização do modelo, atualização da malha e pós-processamento. A não ser na primeira etapa, quando o modelo a ser analisado é descrito, todas as outras podem ser realizadas de maneira totalmente automática. Há estudos que indicam que mesmo os parâmetros iniciais que controlam o nível de refinamento da malha possam ser atribuídos automaticamente de maneira bastante criteriosa.

Esse ambiente de análise adaptativa fornece inicialmente condições e ferramentas propícias para que se crie uma representação do domínio, topológica e geometricamente válida. A partir desse ponto o sistema prescinde de qualquer intervenção humana. O algoritmo de discretização gera uma malha válida levando em conta o grau de refinamento estabelecido inicialmente. Submete o modelo discretizado à análise por elementos finitos. É realizada então uma estimativa de erros. Nas regiões em que essas estimativas de erros ultrapassam o valor máximo admitido há um refinamento local da malha. Essa malha gerada é então submetida a nova análise até que se consiga uma malha que verifique totalmente a condição de

erro máximo estabelecido. Através da análise por elementos finitos desse último modelo discretizado podem ser obtidas, utilizando-se o pós-processador, as saídas necessárias para a interpretação e análise do especialista.

Praticamente todos os componentes integrantes desse ambiente de análise constituem-se em matérias bastante vastas e que por si só representam grandes desafios. Certamente o desenvolvimento de um sistema que produza os resultados descritos é tarefa para uma equipe. O trabalho de uma única pessoa seria certamente frustrada pela obsolescência decorrente da rápida evolução das técnicas envolvidas principalmente no que diz respeito à informática.

Os benefícios trazidos por um sistema de análise adaptativa são óbvios e contundentes: economia de tempo e conseqüentemente de custo, maior tempo para o exercício do papel analítico e criativo, reservados ao homem, além da confiabilidade dos resultados.

Dentro desse quadro geral apresentado e com essa visão do conjunto é que foi desenvolvido esse que pretende ser um elemento de sistema de análise adaptativa dentro do âmbito do Laboratório de Mecânica Computacional.

O rápido tempo de obsolescência evidencia a importância da portabilidade do sistema e a necessidade do trabalho em equipe e, portanto, o emprego de técnicas que permitam a modularização do trabalho. Essas características, portabilidade e modularidade, foram exaustivamente perseguidas durante todo o trabalho de elaboração dos componentes de modelagem de figuras planas e geração automática de malhas que são os objetivos desse trabalho.

8.1 Modelagem de figuras planas

Na construção do modelador de figuras planas foi adotado o método de representação pela fronteira. Nesse método o domínio é descrito através de uma fronteira orientada composta de vértices, arestas e faces. Os acessos à estrutura de dados foram feitos através de operadores que garantem a integridade dos dados e a validade da representação do domínio a qualquer momento. Através desses operadores criou-se uma camada entre a estrutura de dados e os algoritmos. Isso permite uma grande flexibilidade no uso de outros modelos de dados bastando a substituição das funcões que desempenham o papel de operadores.

Além dessa camada entre o algoritmo e a estrutura de dados, formada pelos operadores, foi criada uma outra entre os algoritmos do sistema e os algoritmos que interagem diretamente com os dispositivos gráficos. A construção de uma pseudo-biblioteca gráfica própria, contendo todas as funções que manipulam os dispositivos gráficos, permite que a mudança de ambiente gráfico possa ser realizada alterando-se apenas as rotinas que pertencem a essa camada sem nenhuma intervenção nos algoritmos do sistema.

O desenvolvimento de um modelador de figuras planas teve um caráter didático na consolidação de inúmeros conceitos tanto nas áreas da computação gráfica, metodologia de desenvolvimento de sistemas, técnicas de programação, como na de modelagem geométrica. A importância prática da implementação do modelador se dá na medida que a criação de um ambiente integrado para análise adaptativa deva começar com um modelador aberto, acessível, sobre o qual se tenha total domínio e que possa servir de plataforma para os outros módulos. Essa implementação abre espaço e dá suporte para futuras pesquisas explorando, por exemplo, outras técnicas de modelagem. Não se tem, entretando, a pretensão de criar um editor gráfico completo muito menos que venha a competir com produtos já consagrados. Pretende-se, sim, estabelecer com eles uma convivência harmoniosa que permita explorar as vantagens de cada um deles. A comunicação entre o modelador de figuras planas desenvolvido e a maioria dos softwares de computação gráfica existentes no mercado pode ser estabelecida através do intercâmbio de arquivo que compartilhe do protocolo DXF.

A metodologia utilizada para o desenvolvimento da interface com o usuário, inspirada no diálogo humano, inserida dentro de um projeto *top-down* que permita a visualização do conjunto, deu bons resultados. Essa metodologia de desenvolvimento de projeto permite que se inicie a elaboração do sistema em um nível de abstração bastante alto até chegar-se na última etapa a detalhes da implementação física.

A preocupação com a portabilidade levou à utilização da linguagem C. A escolha do compilador WATCOM versão 9.0 contou com um aspecto fundamental que é a sua capacidade de explorar a memória acima dos 640 Kbytes convencionais, o que não acontece com as versões 6.0 da Microsoft e 2.0 da Borland. Uma vantagem adicional é a facilidade de compatibilizar as rotinas compiladas no WATCOM com o AutoCAD, isto é, pode-se acessar rotinas escritas em C a partir do AutoCAD.

8.2 Geração automática de malha

O gerador automático de malhas foi construído baseando-se em um método de decomposição espacial denominado método do quadtree modificado. As entradas necessárias são apenas as descrições geométrica e topológica da representação do domínio a ser discretizado. O algoritmo gera uma malha válida levando em conta o grau de refinamento desejado, sem que qualquer intervenção do usuário seja necessária.

Há duas rotinas principais no processo de geração da malha. Inicialmente o domínio é representado através de um conjunto de células que são quadrantes terminais de um quadtree. O segundo passo consiste de a partir da topologia dos quadrantes terminais criar a malha de elementos finitos. Até o momento encontra-se implementado no GAMA2D os algoritmos da primeira etapa, isto é, a descrição do contínuo através de células.

O nível de refinamento da malha é função do tamanho dos quadrantes que por sua vez é função do nível em que se encontra representado na árvore. Controla-se, então, o grau de refinamento da malha através da variação do nível da árvore.

A escolha do método do quadtree modificado deveu-se basicamente aos seguintes fatores: o método é totalmente automatizável, permite refinamento localizado e já foi implementado com sucesso. O refinamento localizado é importante no desenvolvimento de um sistema de análise adaptativa. No processo adaptativo, os níveis da árvore são automaticamente gerados pelo processo de estimativa de erro que determina quais regiões do domínio necessitam de discretizações extras.

As críticas existentes ao método dizem respeito à qualidade dos elementos gerados no contorno [K Ho Le] e à dificuldade de se transportar os algoritmos para três dimensões [Perucchio]. Parece que com o desenvolvimento dos algoritmos de suavização de malha o primeiro ponto está equacionado.

8.3 Proposta de extensões

Algumas extensões são imediatas e podem ampliar bastante a gama de problemas contemplados.

A implementação de contornos curvos para o domínio depende quase que exclusivamente do desenvolvimento de algoritmos de intersecção da curva introduzida com um retângulo e de algoritmos de desenhos de curvas do tipo *spline* (*Bézier*, *NURBS*,...). A estrutura de dados já contempla curvas desse tipo. As rotinas de desenho de arcos encontram-se disponíveis na biblioteca gráfica utilizada. Em termos de geração de malha o único processo que se altera é a discretização do contorno. Essa alteração é feita facilmente com a introdução dos algoritmos de intersecção mencionados anteriormente.

A geração de malhas triangulares pode ser implementada utilizando-se os algoritmos apresentados em [1].

A criação de interfaces com vários programas de elementos finitos também é simples e pode ampliar bastante a utilização do programa.

Outras extensões tratam, na verdade, de atualizações com vista à rápida evolução da informática tais como a utilização de linguagem orientada para objeto e a criação de versão para o *Windows*.

As últimas extensões propostas são os componentes descritos na introdução desse capítulo e valem principalmente para reafirmar as metas traçadas e o sonho de participação em uma equipe de trabalho dentro do Laboratório de Mecânica Computacional que possa utilizar os elementos desenvolvidos nessa dissertação para pesquisa e desenvolvimento de um sistema integral de análise adaptativa.

Referências Bibliográficas

[01] BAEHMANN, Peggy L. **Automated Finite Element Modeling and Simulation.** Rensselaer Polytechnic Institute. Troy, New York, May 1989. 223p. Tese (Doutorado).

[02] BAEHMANN, Peggy L.; SHEPHARD, Mark S. **Adaptive multiple-level *h*-refinement in automated finite element analyses.** Engineering with Computers. Volume 5, n.3/4, p.235-247, 1989.

[03] BAKER, Timothy J. **Automatic Mesh Generation for Complex Three-Dimensional Regions using a constrained delaunay triangulation.** Engineering with Computers. Volume 5, n.3/4, p.161-175, 1989.

[04] FILGUEIRAS, Lucia V. L. et al. **Fundamentos de Computação Gráfica.** LTC - Livros Técnicos e Científicos Editora S.A. São Paulo, 1987.

[05] FILIP, D. J. **Determining the orientation of closed planar curves.** CAD. Volume 22, n.7, September 1990.

[06] FINNIGAN, P.M.; KELA, A.; DAVIS, J.E. **Geometry as a Basis for Finite Element Automation.** Engineering with Computers. Volume 5, n. 3/4, p. 147-160, 1989.

[07] FITZHORN, Patrick A. **Language of topologically valid bounding manifolds.** CAD. Volume 22, n.7, September 1990.

[08] FOLEY, James D. et al. **Computer Graphics principles and practice.** 2.ed., Addison-Wesley Publishing Company, 1990.

[09] **HOFFMAN, Cristoph M.** Geometric and solid modeling: an introduction. **Purdue University, 1993**

[10] HO-LE, K. **Finite element mesh generation methods: a review and classification.** CAD. Volume 20, n.1,1988.

[11] JAMSA, Kris. **The C library.** Osborne Mc Graw-Hill. Berkeley, California, 1976.

[12] MÄNTYLÄ, Martti. **An Introduction to Solid Modeling.** Helsinki University of Technology. Computer Science Press, 1988.

[13] Microsoft C 5.0. **Optimizing Compiler for the MS-DOS.** Operating system. User's Guide.

[14] PAULINO, Gláucio Hermógenes. **Pre-processamento de Estruturas Reticuladas Espaciais, com Reordenação Nodal, Usando Computação Gráfica Interativa.** Pontíficia Universidade Católica do Rio de Janeiro. Departamento de Engenharia Civil. Rio de Janeiro, Brasil, Dezembro 1988. 299p. Dissertação de Mestrado.

[15] SAMET, Hanan; WEBBER, Robert E. **Hierarchical Data Structures and Algorithms for Computer Graphics.** The Institute of Electrical and Electronics Engineers, INC. Volume 8, 1988.

[16] SAMET, Hanan. **Neighbor Finding in Images Represented by Octrees.** Computer vision, Graphics and image processing. P.367-386, 1989.

[17] SAPIDIS, N.; PERUCCHIO, R. **Advanced techniques for automatic finite element meshing from solid models.** CAD. Volume 21, n.4, p.248-253, May 1989.

[18] SCHILDT, Herbert. **Advanced C.** Osborne Mc Graw-Hill. Berkeley, California, 1975.

[19] TURNER, Joshua. **Boundary Representation for Solid Models.** Part I: Euler Operators. Notas de aula, 1992.

[20] TURNER, Joshua. **Boundary Representation for Solid Models.** Part II: Data Structures. Notas de aula, 1992.

[21] WIRTH, Niklaus. **Algoritmos e Estruturas de Dados**. Editora Prêndice-Hall do Brasil. Rio de Janeiro, 1989.

[22] ZAGOTTIS, Décio de. **Introdução à Teoria das Estruturas.** Capítulos 25 a 36: Elasticidade - Elementos Finitos. Escola Politécnica da Universidade de São Paulo, Departamento de Engenharia de Estruturas e Fundações, 1986.

[23] ZIENKIEWICZ, O.C.; CHEUNG, Y.K. **The Finite Element Method in Structural and Continuum Mechanics.** McGRAW-HILL Publishing Company Limited. London, 1968.

Referências Bibliográficas Suplementares

[24] BUELL, W. R. and BUSH, B. A. **Mesh generation - a survey.** Trans. ASME, J. Eng. Ind. February, 1973. P.332-338.

[25] BYKAT, A. **Design of a recursive, shape controlling mesh generator.** Int. J. Numer. Meth. Eng. Vol. 19. 1983. P.1375-1390.

[26] CAVENDISH, J. C. **Automatic triangulation of arbitrary planar domains for the finite element method.** Int. J. Numer. Method. Eng. Vol. 8. 1974. P. 679-696.

[27] CLARKE, C. V. and MULLER, R. **Gifts-1100: Graphics Oriented Interactive Element Time-Sharing System, In a Handbook of Element Systems.** C.A. Brebia (ed.). CML Publications, 1981.

[28] COHEN, H. D. **A method for the automatic generation of triangular elements on a surface.** Int. J. Numer. Meth. Eng. Vol. 15. 1980. 470-476.

[29] FREDERICK, C. O. et al. **Two-dimensional automatic mesh generation for structural analysis.** Int. J. Numer. Meth. Eng. Vol. 2. 1970. P.133-144.

[30] HEIGHWAY, E. A. and BIDDLECOMBE, C. S. **Two-dimensional automatic triangular mesh generation fo rthe finite element electromagnetics package PE2D.** IEEE Trans. on Mag. Vol. MAG-18, N° 2. March 1982. P. 594-598.

[31] KELA, A.; PERUCCHIO, R. et al. **Toward automatic finite element analysis.** Comput. Mech. Eng. Vol. 5, N° 1. July 1986.

[32] LEE, Y. T. **Automatic finite element generation based on constructive solid geometry.** PhD thesis Mechanical Engineering Dept, University of Leeds, Leeds, UK. 1983.

[33] LO, S. H. **A new mesh generation scheme for arbitrary planar domains.** Int. J. Numer. Meth. Eng. Vol. 21. 1985. P. 1403-1426.

[34] NELSON, J. M. **A triangulation algorithm for arbitrary planar domains.** Appl. Math. Modeling, Vol. 2. 1978. P. 151-159.

[35] THACKER, W. C. **A brief review of techniques for generating irregular computational grids.** Int. J. Numer. Meth. Eng. Vol. 21. 1985. P. 329-347.

[36] THACKER, W. C. et al. **A method for automating the construction of irregular computational grids for storm surge forecast models.** J. of Computational Physics. Vol.37. 1980. P. 371-387.

[37] TRACY, F. T. **Graphical pre- and post-processor for two-dimensional finite element method programs.** Proc. SIGGRAPH '77. Vol. 11, n° 2. 1977. P. 8-12.

[38] WORDENWEBER, B. **Finite Element mesh generation. Comput.-Aided Des.** Vol. 16, N° 5. September 1984. P. 285-291.

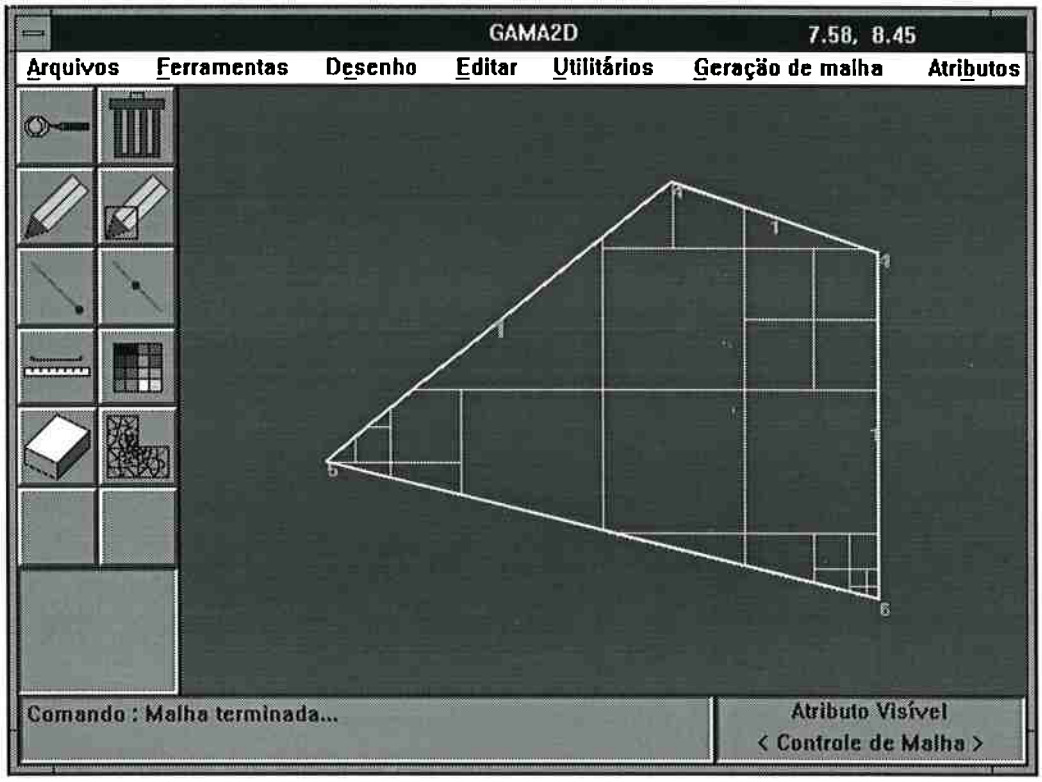
[39] WORDENWEBER, B. **Volume triangulation.** Technical Report University of Cambridge, UK, CAD Group Document N° 110. 1980.

[40] YERRY, M. A. ; SHEPHARD, M. S. **A modified quadtree approach to finite element mesh generation.** IEEE Comput. Graph. & Appl. Febreary 1983. P. 39-46.

[41] ZIENKIEWICZ, O. C.; PHILLIPS, D. V. **An automatic mesh generation scheme for plane and curved surfaces by 'isoparametric' co-ordinates.** Int. J. Numer. Meth. Eng. Vol. 3. 1971. P. 519-528.

Selecione Controle de Malha. Um sub-menu se abrirá.

Selecione Interior.



Saindo do GAMA2D

No menu superior selecione arquivo. Um submenu se abrirá

Selecione Sair. Abrirá uma janela.

Selecione OK.

APÊNDICE

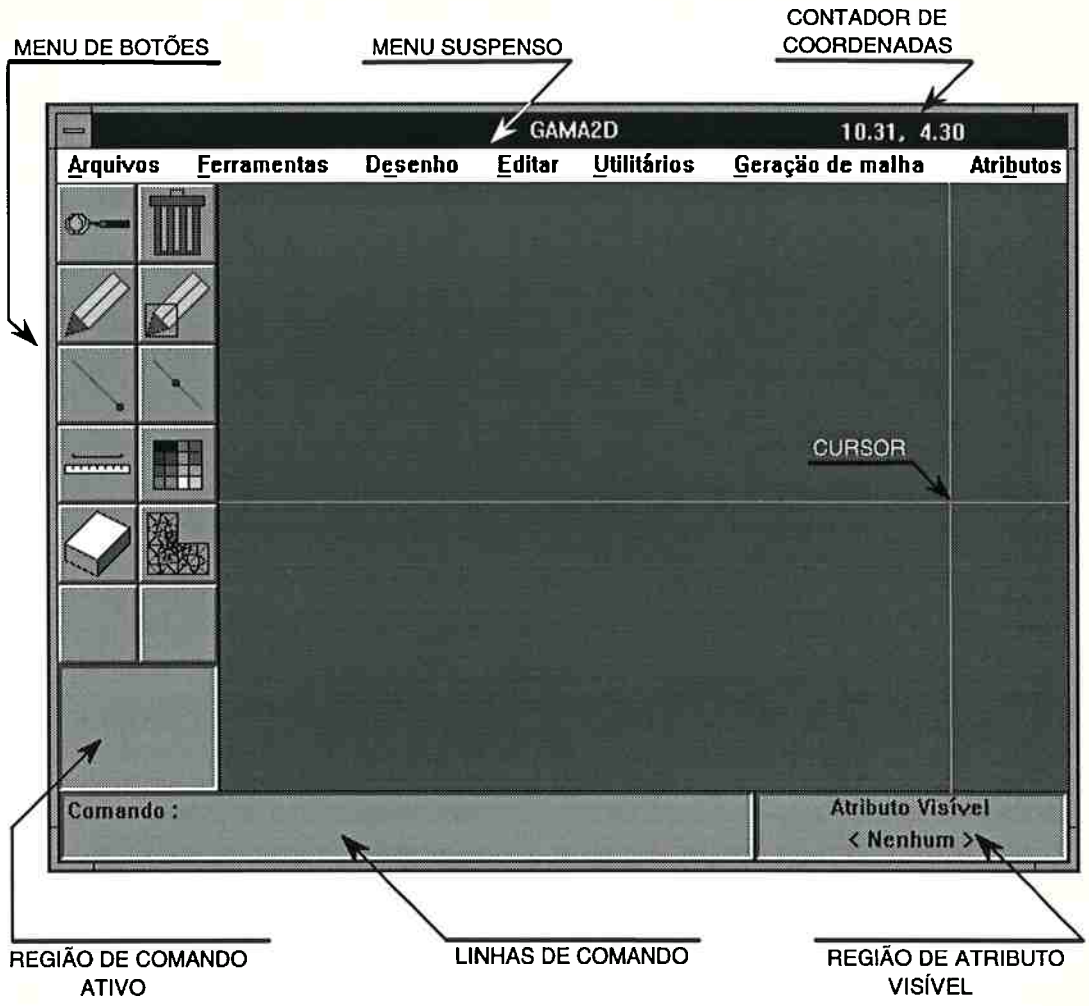
TUTORIAL

Apresentaremos a seguir um exemplo completo de desenho e geração de malha utilizando-se o GAMA2D. O desenho é construído passo a passo e todas as interações com o usuário são apresentadas, incluindo as telas mostrando a configuração a cada comando.

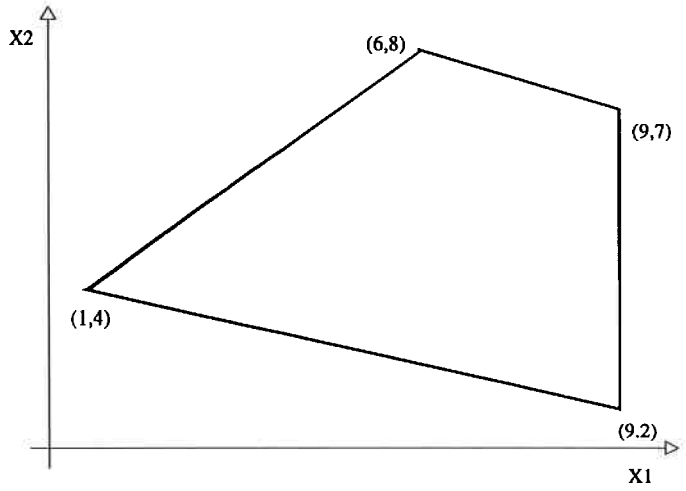
Após a execução completa deste desenho você terá aprendido os seguintes comandos:

- de desenho: Linha (coordenadas retangular e polar) e Face
- auxiliar: PF (ponto final)
- de visualisação: Zoom Total
- de configuração: Passo do Cursor
- de geração de malha: Controle de Malha, Quadtree, Um Nível, Contorno, Interior

Para que o texto seja melhor compreendido devemos conhecer as denominações utilizadas para definir as regiões da tela. A próxima figura mostra as 8 regiões em que a tela foi subdividida.


MENU DE BOTÕES
MENU SUSPENSO
CONTADOR DE COORDENADAS
GAMA2D
10.31, 4.30
Arquivos
Ferramentas
Desenho
Editar
Utilitários
Geração de malha
Atributos
CURSOR
Comando :
Atributo Visível
< Nenhum >
REGIÃO DE COMANDO ATIVO
LINHAS DE COMANDO
REGIÃO DE ATRIBUTO VISÍVEL

Apresentamos a seguir, todas as etapas de construção do exemplo padrão que vem sendo usado durante todo o texto. Na figura a seguir apresenta-se a geometria do exemplo para servir de referência para a construção.



Entrando no GAMA2D

Ligue o computador.

Digite : GAMA2D ↵Enter

Aparecerá a tela principal do GAMA2D, indicando se o mouse está ou não instalado. Estamos supondo que o usuário estará dispondo de um mouse apesar dele não ser imprescindível.

Mova o cursor até o botão de OK da janela de apresentação e clique o botão esquerdo do mouse. As seleções com o mouse sempre se fazem movendo o cursor até a opção desejada e clicando o seu botão esquerdo.

Definindo o modelo geométrico

Para definir o modelo geométrico devemos inicialmente definir as arestas e vértices que formam o contorno e, em seguida, definir os ciclos que constituem as faces. No exemplo básico iremos construir quatro arestas e definir uma face.

Definindo as arestas e os vértices

As arestas e os vértices podem ser construídos simultaneamente utilizando-se o comando Linha. Há três maneiras de acessar o comando LINHA: pelo menu de botões, pelo menu suspenso ou pela linha de comando.

Utilizando a linha de comando todas as funções do GAMA2D podem ser ativadas digitando-se o nome do comando por extenso ou um mnemônico formado pelos seus três primeiros caracteres. O comando para desenhar linhas (retas) é reconhecido, então, por Linha ou LIN.

Vamos começar desenhando a reta formado pelos pontos de coordenadas (1,4) e (9,2).

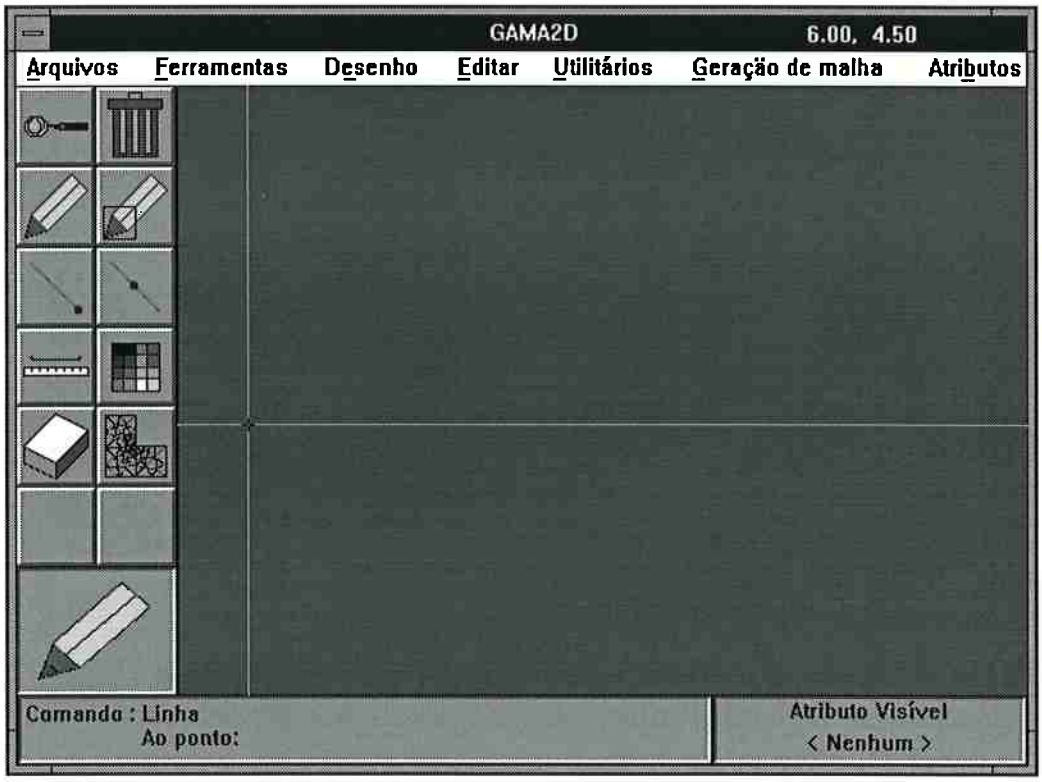
Digite: LIN ↵Enter

Aparecerá o ícone com o lápis na Região de Comando Ativo indicando que o comando Linha foi ativado. Na linha de comando será apresentada a seguinte mensagem:

Do ponto:

Digite: 1,4 ↵Enter

Será desenhado o ponto com coordenadas (1, 4) na área gráfica.

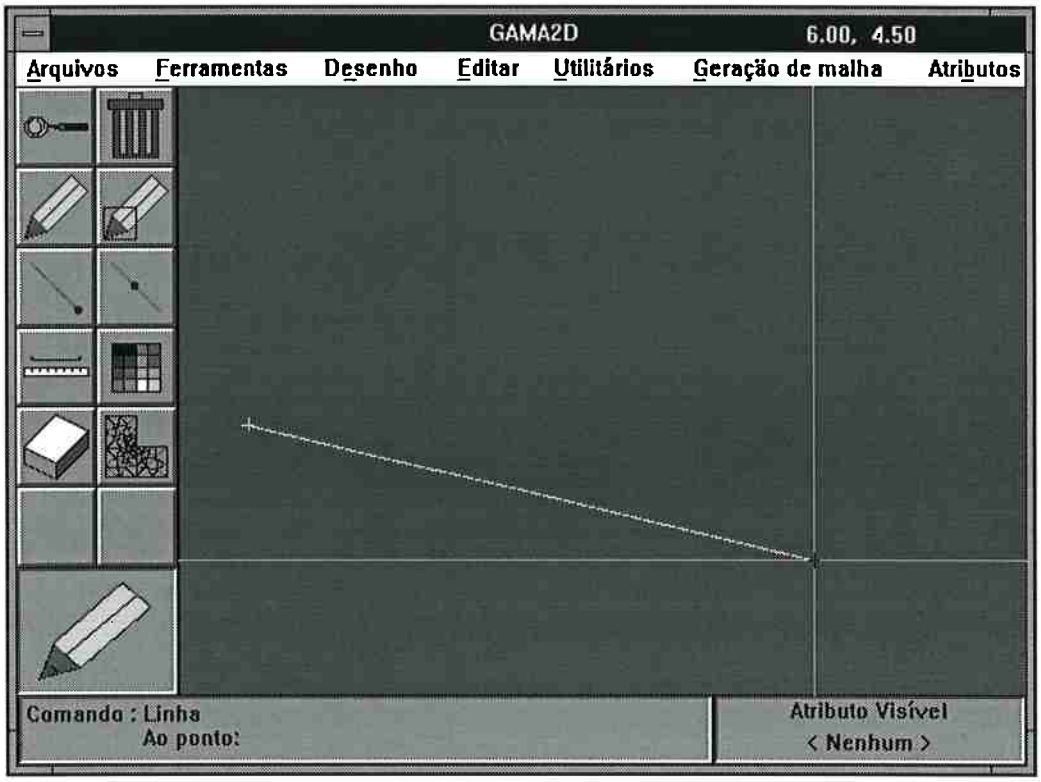


Note que a linha de comando apresenta a seguinte mensagem:

Ao ponto:

Digite: 9,2 ↵Enter

A linha definida pelos pontos de coordenadas (1, 4) e (9, 2) será apresentada na tela.

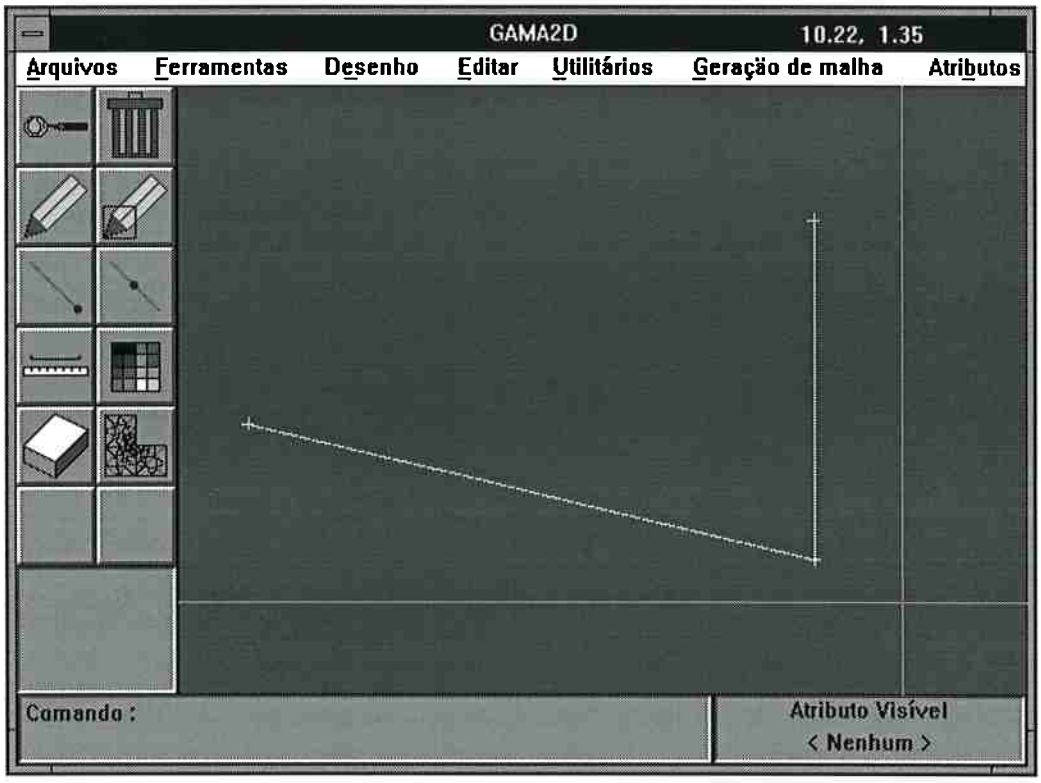


Mova o mouse aleatoriamente. Perceba o efeito de arrasto, isto é, o vértice inicial da nova aresta a ser desenhada é fixado na posição do último ponto fornecido e o vértice final pode ser arrastado para a posição desejada. Para introduzirmos a próxima aresta basta, então, digitarmos o seu vértice final. Vamos utilizar uma forma relativa de introduzir o vértice final através de coordenadas polares.

As coordenadas polares são reconhecidas através do caráter @ colocado no segundo campo, logo após a vírgula. O primeiro campo especifica o comprimento da aresta e o segundo, a sua inclinação em relação aos eixos ordenados.

Digite : 5,@90 ↵Enter

A aresta definida pelos vértices de coordenadas (9, 2) e (9, 3) aparecerá na tela.



Vamos introduzir a próxima aresta definindo as coordenadas através do posicionamento do cursor pelo mouse. Temos inicialmente que ajustar o passo do cursor coordenado pelo mouse. Para efetuar essa operação vamos, então, sair do comando Linha.

Tecle: ESC

Note que a Região de Comando Ativo ficou vazia.

O ajuste do passo do mouse encontra-se no menu Utilitários, no item Configurar Sistema.

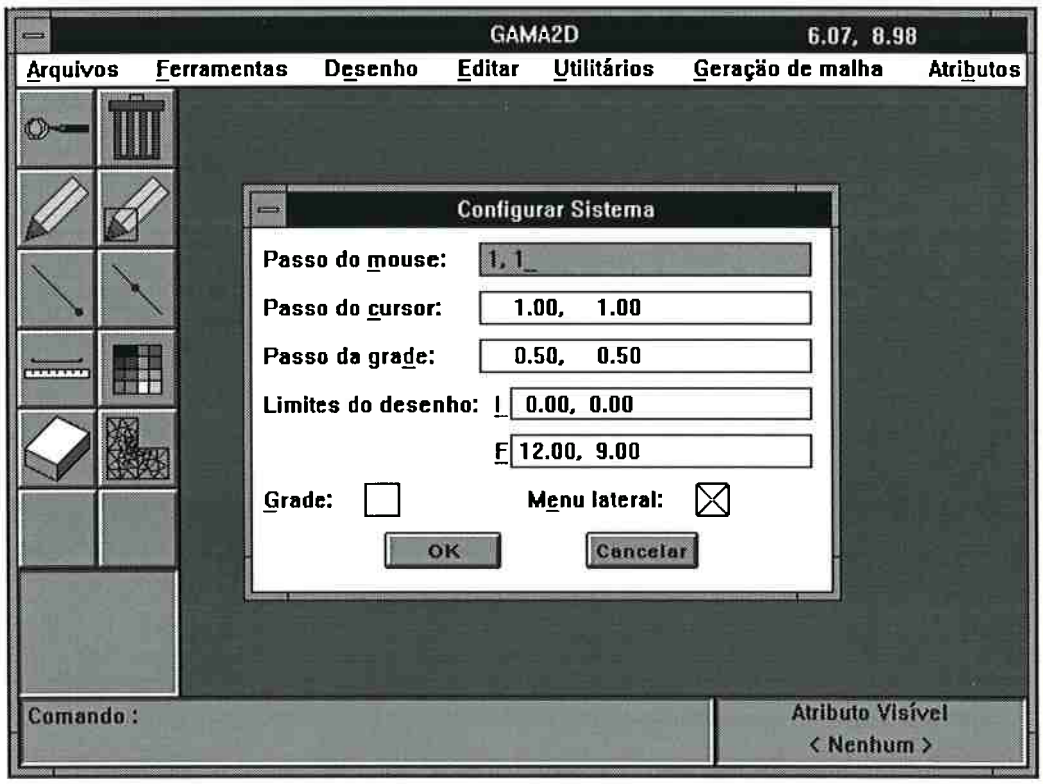
Vá com o mouse até o menu superior e selecione o item Utilitários. Aparecerá um submenu.

Selecione Configurar Sistema.

Aparecerá a janela de configuração do sistema.

Selecione o retângulo localizado à frente de Passo do Mouse.

Digite: 1,1 ↵Enter



Selecione OK. O passo do mouse será configurado para se deslocar em intervalos de uma unidade na direção do eixo x e uma unidade na direção do eixo y.

Vamos retomar o desenho da fronteira do modelo geométrico introduzindo a aresta definida pelos vértices de coordenadas (1, 4) e (6, 8). Uma maneira rápida de acessar o comando Linha é através do menu de botões.

Vá com o mouse até o menu de botões e selecione o ícone com o lápis.

A região de comando ativado apresentará o ícone com o lápis indicando que o comando Linha foi ativado e a linha de comando mostrará a mensagem:

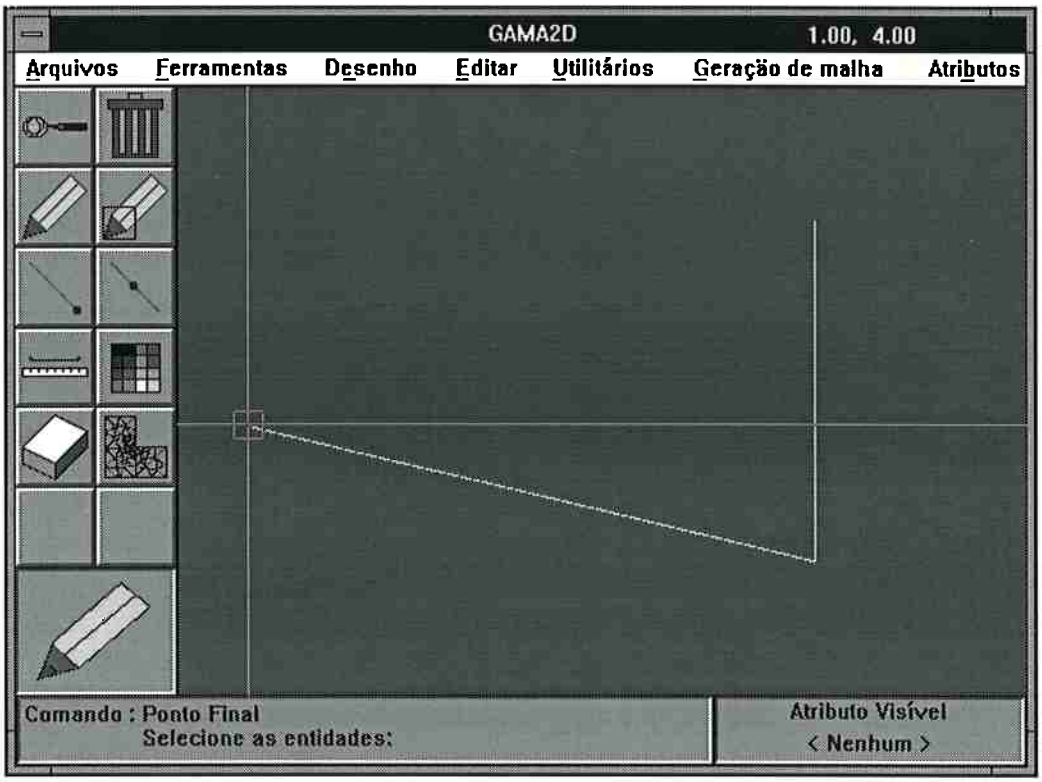
Do ponto:

A aresta que desejamos construir possui um vértice (1, 4) que já foi definido. Podemos, então, realizar essa construção informando que as coordenadas do ponto que estão sendo requisitadas são as mesmas do ponto final de determinada aresta. O comando que realiza essa operação é o Ponto Final.

Selecione no menu de botões o ícone que apresenta uma reta com o seu último ponto em evidência.

Note que o desenho do cursor apresentará a forma de uma mira.

Posicione o cursor na reta com vértices (1, 4) e (9, 2), próximo do vértice (1, 4).



Clique o botão esquerdo.

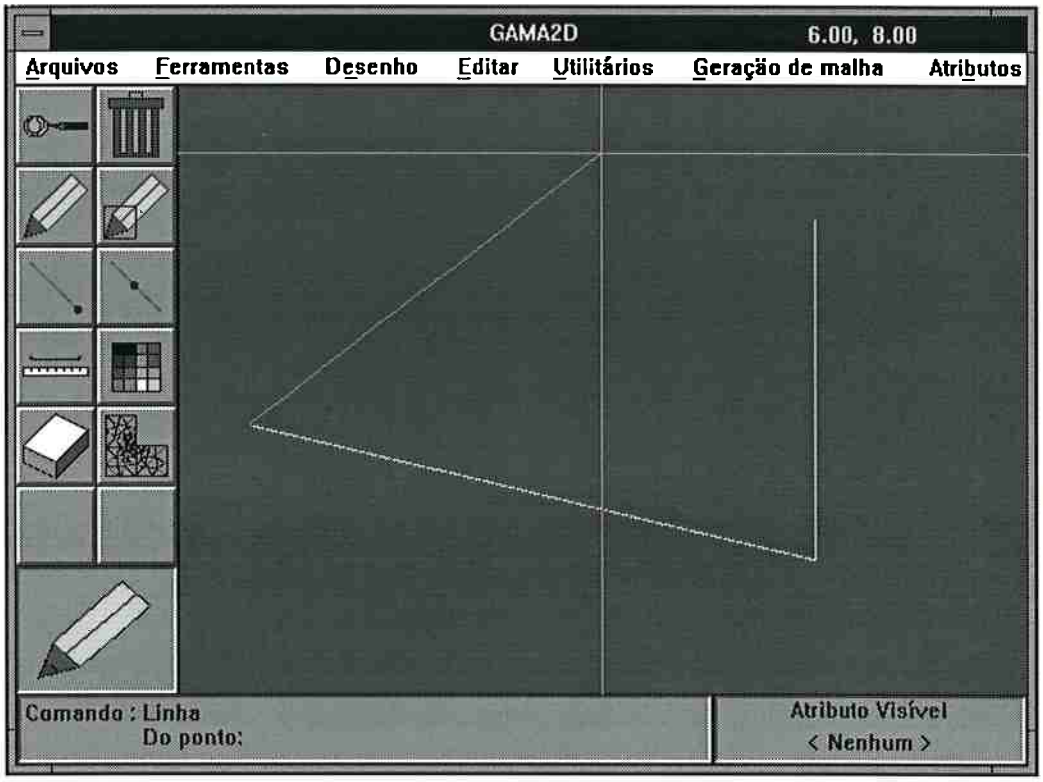
Aparecerá na linha de comando a mensagem:

Ao ponto:

Note que o desenho do cursor voltou a ter a forma de cruz.

Movimente o mouse aleatoriamente. Observe que a medida que o cursor se movimenta na tela de desenho as suas coordenadas podem ser acompanhadas na região do contador de coordenadas.

Vá com o cursor (através do mouse) até a coordenada (6, 8) (acompanhe as coordenadas pelo contador de coordenadas), clique o botão esquerdo, marcando esse ponto na tela.



Na linha de comando aparecerá a mensagem:

Ao ponto:

Observe que para fechar o desenho do contorno basta apenas a aresta definida pelos vértices (6, 8) e (9, 3). Como o vértice (9, 3) já foi definido, vamos utilizar o comando Ponto Final. Ele pode ser ativado, além do menu de botões, através do menu suspenso.

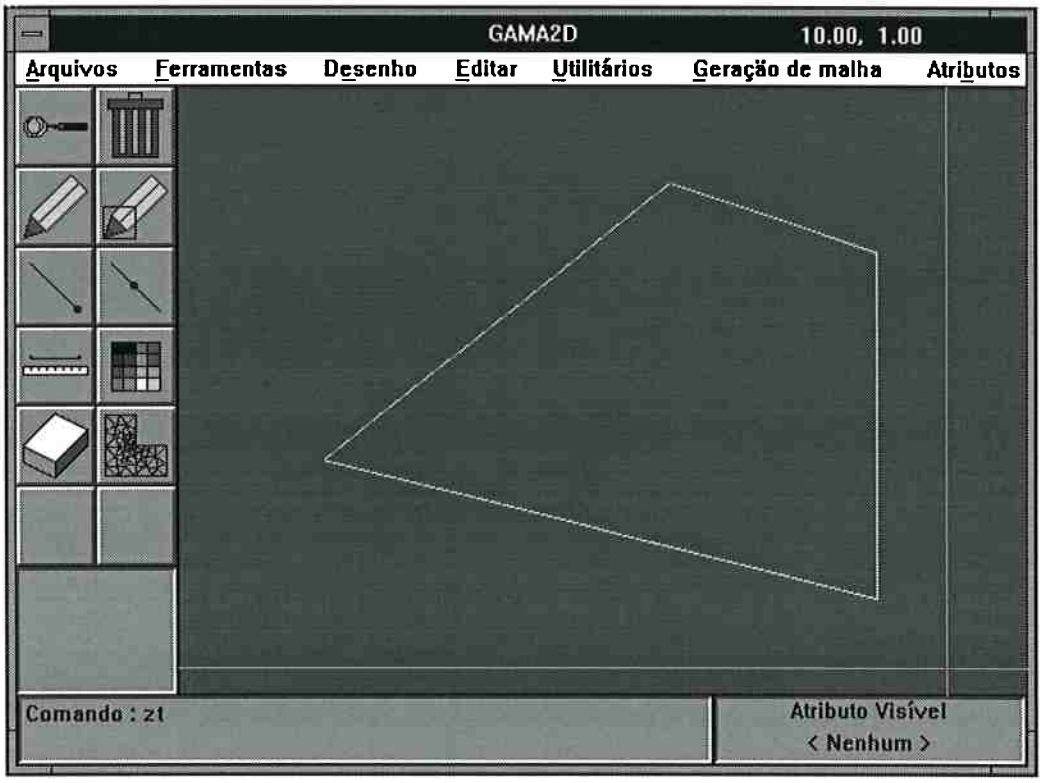
Selecione Ferramentas. Um novo menu se abrirá.

Selecione Ponto Final.

Posicione o cursor em forma de mira sobre a aresta que possui o vértice (9, 7), próximo dele. Clique o botão esquerdo.

Acabamos de concluir o contorno da geometria.

Tecle o botão direito do mouse para desativar o comando Linha. Com isso a região do comando ativado ficará vazia.



Para permitir uma melhor visualização do desenho podemos recorrer ao comando de enquadramento de desenho na tela o Zoom Total.

Digite : ZT ↵Enter

Numerando Arestas e Vértices

A identificação das entidades arestas e vértices pode ser feita atribuindo-se um número para cada uma delas. Essa identificação servirá como referência para todos os relatórios que venham a ser exibidos. A identificação tanto das arestas como dos vértices pode ser feita de forma automática ou manual. Na automática é o próprio programa quem se encarrega de atribuir valores para cada uma das entidades. Na manual o usuário é quem se encarrega de estabelecer a identificação das entidades.

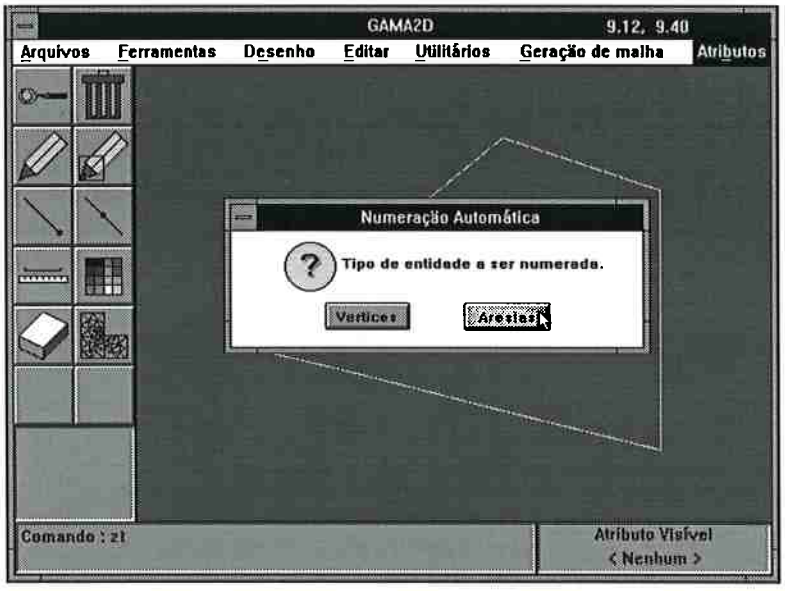
Numerando arestas

Vamos iniciar com a identificação das arestas a qual faremos de maneira automática.

Selecione Atributos no menu suspenso. Um submenu se abrirá

Selecione o item Numeração Automática. Abrirá uma janela contendo dois botões: arestas e vértices.

Selecione o botão Arestas.



O programa irá numerar as arestas tomando como referência a sua ordem de construção, isto é, a primeira aresta a ser construída receberá número 1, a segunda 2 e assim por diante.

Para visualizar os números de identificação das arestas na tela devemos proceder como descrito a seguir.

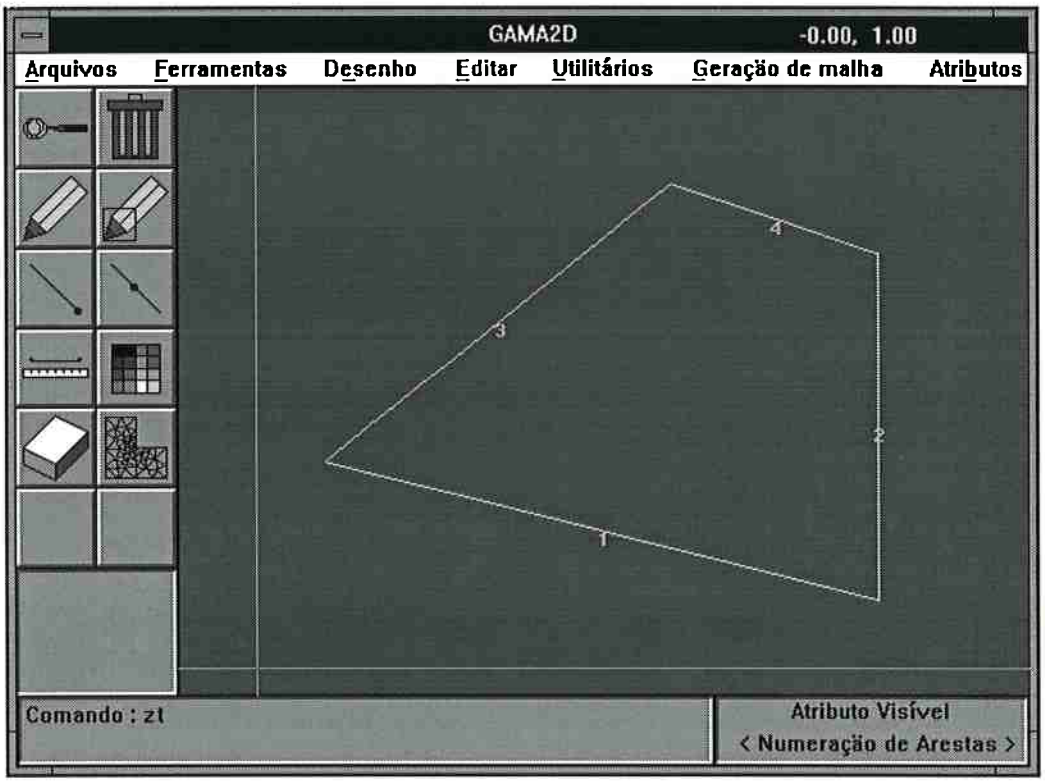
Selecione Utilitários no menu suspenso. Um submenu se abrirá.

Selecione o item Setar Atributos. Abrirá uma janela com vários itens cuja visualização na tela pode ser ligada ou desligada.

Selecione Numeração das Arestas no modo Ligado.

Na tela gráfica será apresentado o modelo geométrico com os números de identificação das arestas próximos de cada uma delas.

Selecione OK.



Numerando os vértices

Utilizando um procedimento análogo ao que acabamos de adotar, vamos numerar agora os vertices.

Selecione Atributos no menu suspenso. Um submenu se abrirá.

Selecione o item Numeração Automática. Abrirá uma janela contendo dois botões: arestas e vértices.

Selecione o botão Vértices.

Do mesmo modo que as arestas, o programa irá numerar os vértices na seqüência cronológica de construção.

Para visualizar a numeração que identifica os vértices.

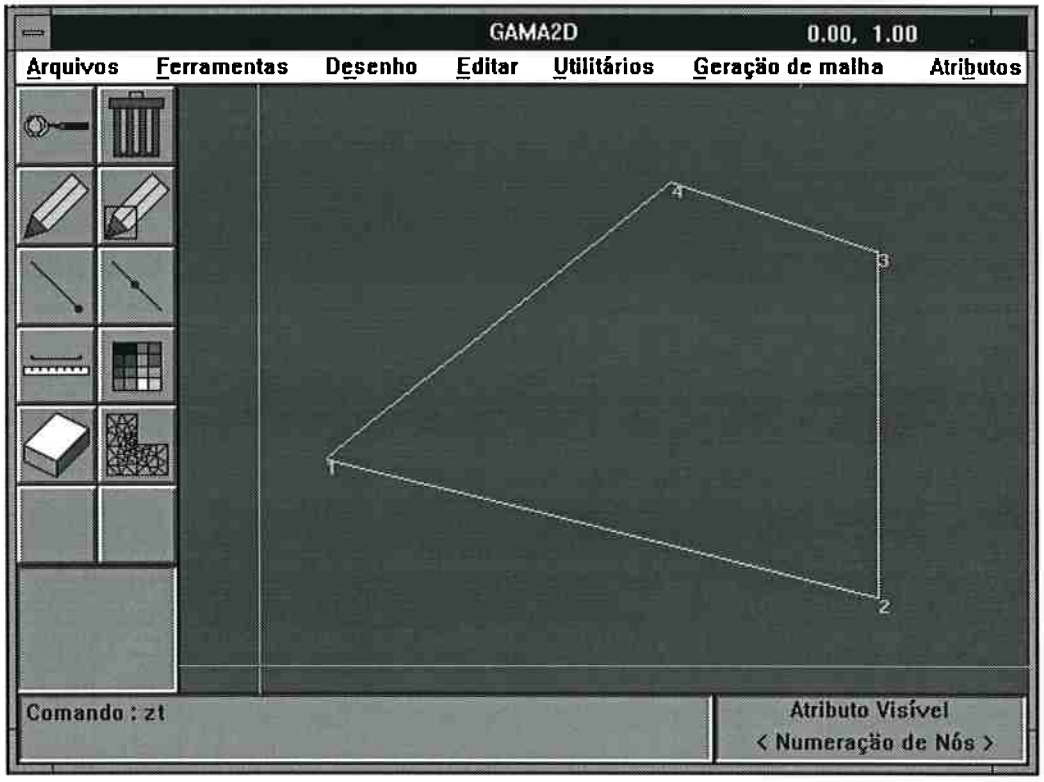
Selecione Utilitários no menu suspenso. Um submenu se abrirá.

Selecione Setar Atributos. Abrirá uma janela com vários itens cuja visualização na tela pode ser ligada ou desligada.

Selecione Numeração dos Vértices no modo Ligado.

Selecione OK.

Na tela gráfica será apresentado o modelo geométrico com os números de identificação dos vértices próximos de cada um deles. Note que somente um tipo de atributo é apresentado na tela num dado momento.



Definindo face

Como veremos a seguir, a ordem de seleção das arestas é fundamental para a definição do domínio. Vamos ativar a numeração das arestas que nos permite visualizar o código atribuído a cada uma delas.

Selecione Utilitários no menu suspenso. Um submenu se abrirá.

Selecione o ítem Setar Atributos. Uma janela se abrirá.

Selecione Numeração das Arestas no modo Ligado.

Para facilitar o selecionamento das arestas devemos refinar o passo do mouse. O ajuste do passo do mouse encontra-se no menu Utilitários, no item Configurar Sistema.

Vá com o mouse até o menu superior e selecione o item Utilitários.

Selecione Configurar Sistema. Aparecerá a janela de configuração do sistema.

Selecione o retângulo localizado à frente de Passo do Mouse.

Digite : 0.01 , 0.01 ↵Enter

Selecione OK. O passo do mouse será configurado.

Selecione no menu superior o item Desenho. Um submenu se abrirá.

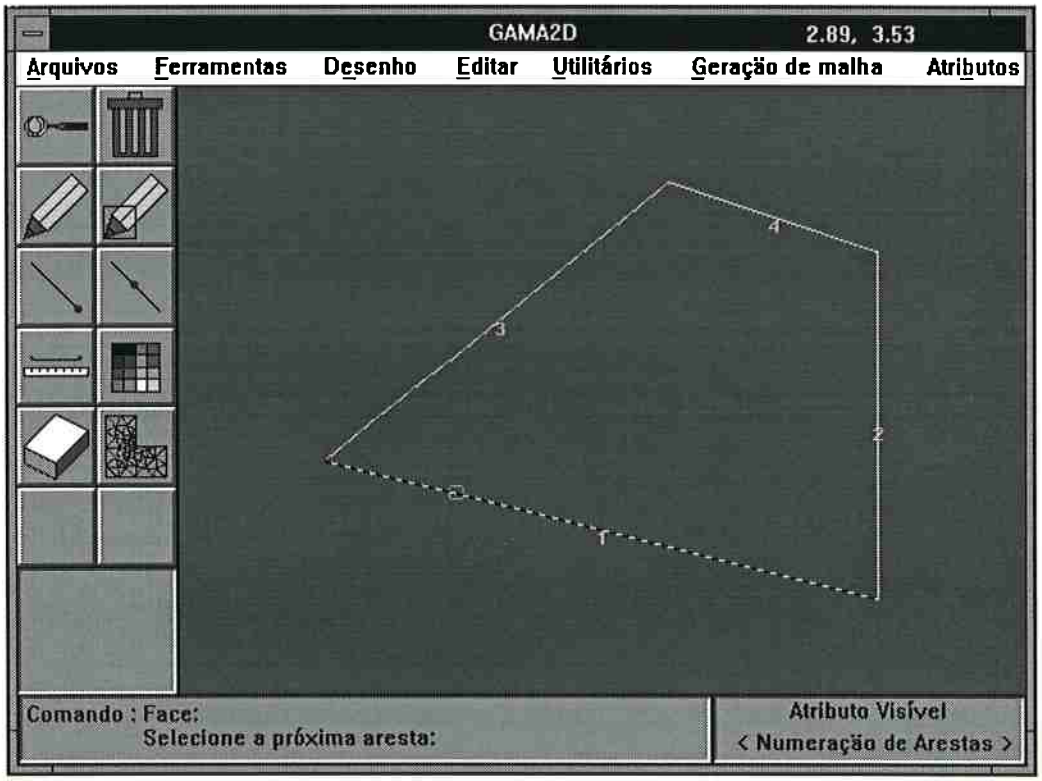
Selecione Definir Face.

Para construir uma face temos inicialmente que definir os ciclos que a constituem. Um ciclo é definido como uma seqüência de arestas sem fronteira. O ciclo tem a ele associado um sentido. Convenciona-se que a região definida pelo ciclo é aquela que está à direita do sentido de caminhamento. A ordem de seleção das arestas que compõe o ciclo é, portanto, extremamente importante. Para o nosso exemplo a ordem das arestas selecionadas poderia ser a seguinte: 1, 3, 4, 2. Qualquer

sequência de arestas, que tomada no sentido horário, definisse um ciclo poderia ser adotada.

Posicione o cursor em forma de quadrado sobre a aresta 1.

Clique o botão esquerdo. A aresta 1 se tornará tracejada indicando que foi selecionada.

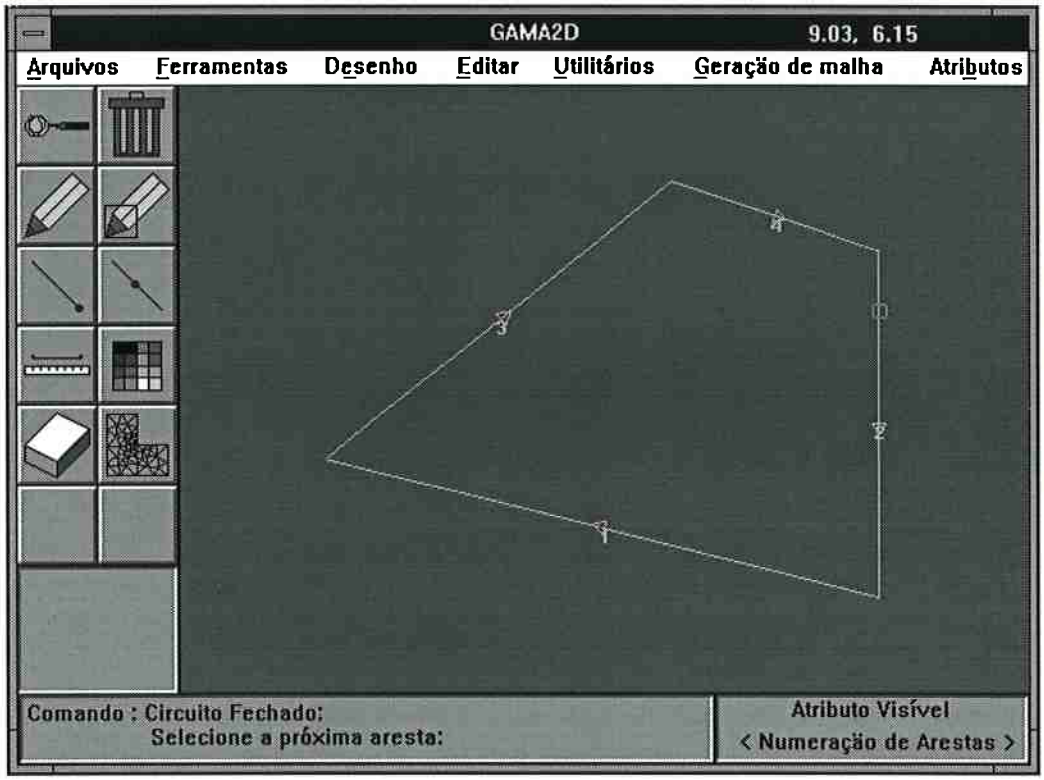


Selecione a aresta 3. A aresta 3 ficará tracejada.

Selecione a aresta 4. A aresta 4 ficará tracejada.

Selecione a aresta 2. A aresta 2 ficará tracejada e, logo a seguir, o ciclo será apresentado em linha contínua sendo que o seu sentido é apresentado através de setas desenhadas sobre as arestas. A linha de comando apresentará a seguinte mensagem:

Circuito fechado:
Selecione a proxima aresta.



Como a face do nosso desenho é composta por apenas um ciclo. Devemos informar que a definição de ciclos já está encerrada.

Tecle a barra de espaço

Aparecerá a seguinte mensagem:

Face definida ...

O modelo da geometria foi concluído sendo que a visualização da face pode ser ativada da maneira descrita a seguir.

Selecione Utilitários no menu suspenso. Um sub-menu será apresentado.

Selecione Setar Atributos. Uma janela se abrirá apresentando os atributos e a possibilidade de ligar ou desligar a visualização de cada um deles. Note que apenas um atributo pode estar ativado num dado momento.

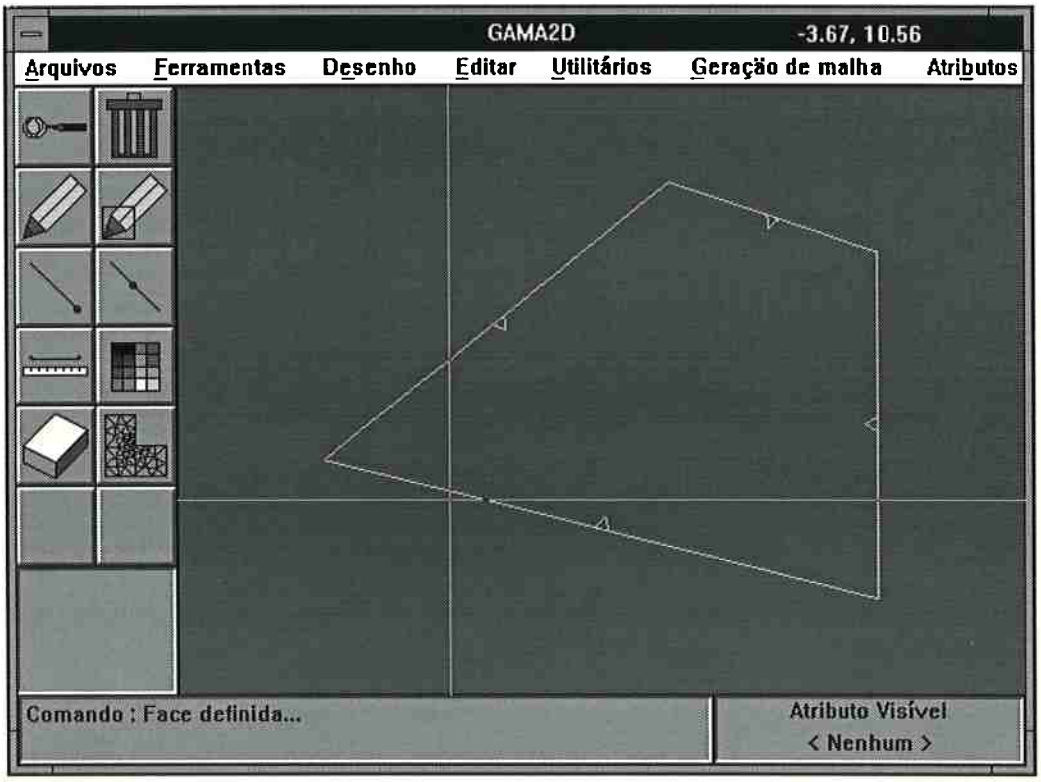
Selecione o quadrado que representa o modo ligado de Setas de Indicação de Face.

Para que a numeração das arestas não se sobreponha no desenho devemos desativa-la.

Selecione Numeração das Arestas no modo Desligado.

Selecione OK.

Aparecerão setas indicando a região que foi definida.

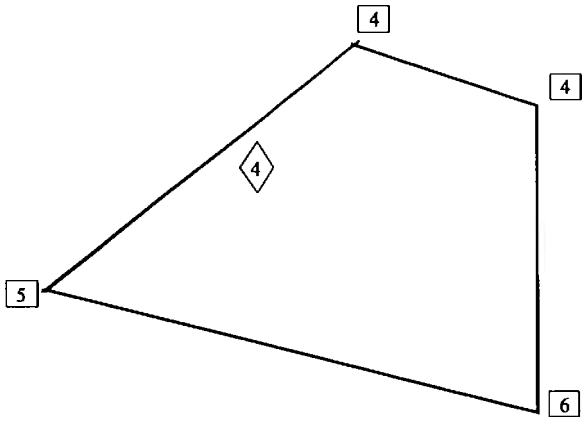


Criando a malha

A malha pode ser gerada passo a passo, através do menu suspenso ou diretamente pelo menu de botões. Em qualquer um dos casos o primeiro passo é a definição dos parâmetros de controle de malha. Através do menu suspenso pode-se controlar cada etapa do processo de geração de malha. O ícone com um modelo discretizado localizado no menu de botões segue automaticamente todos os passos até a geração da malha.

Definindo os parâmetros de contole de malha

O primeiro passo para a criação da malha é a definição dos parâmetros de controle de malha dos vértices e arestas. São eles que definem o grau de discretização da malha. A figura apresentada a seguir mostra os parâmetros de controle de malha do nosso exemplo.



Vamos inicialmente introduzir os parâmetros de controle de malha dos vértices.

Selecione no menu suspenso Geração de malha. Um sub-menu se abrirá.

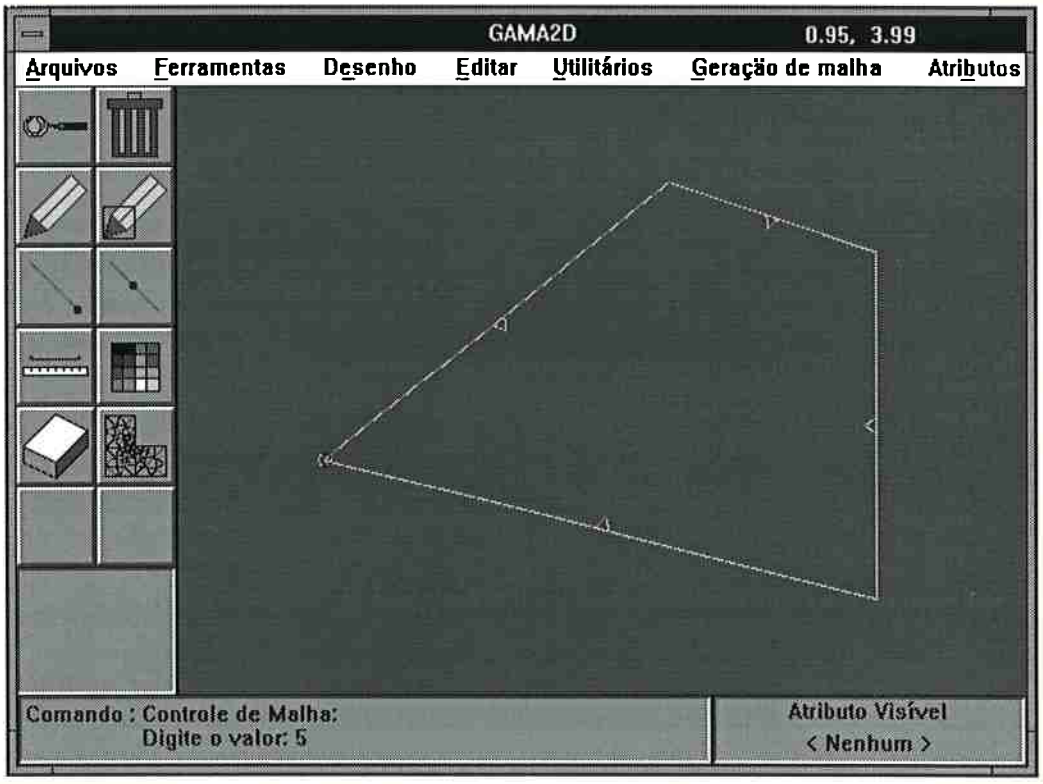
Selecione o item Controle de malha. Uma janela se abrirá apresentando as opções vértices e arestas.

Selecione o botão de vértices.

Perceba que o cursor se tornou quadrado indicando que se está no modo de seleção. Aparecerá a seguinte mensagem na linha de comando.

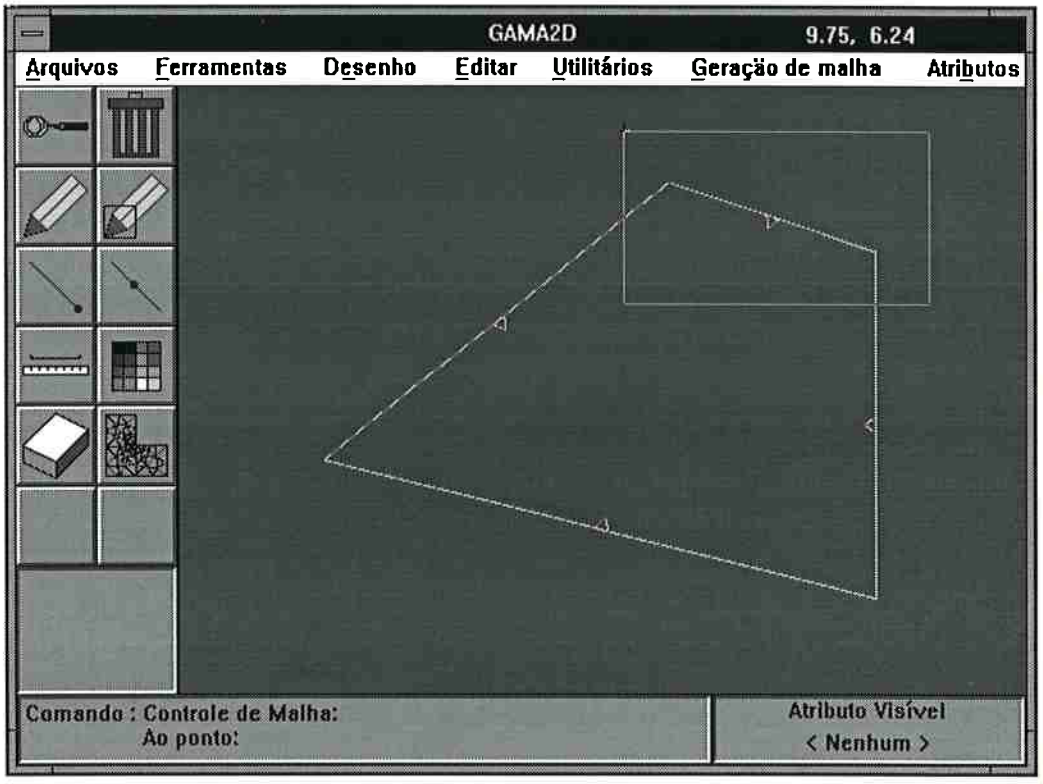
Controle de Malha:
Selecione as entidades:

Posicione o cursor sobre o vértice que deve ter parâmetro igual a 5. Clique o botão esquerdo.



Aparecerá um círculo circunscrevendo o vértice, indicando que ele foi selecionado e, a linha de comando apresentará a seguinte mensagem:

Digite o valor:
Digite: 5 ↵Enter



Selecione o vértice que deve ter parâmetro 6.

Aparecerá um círculo indicando que o vértice foi selecionado e a mensagem:

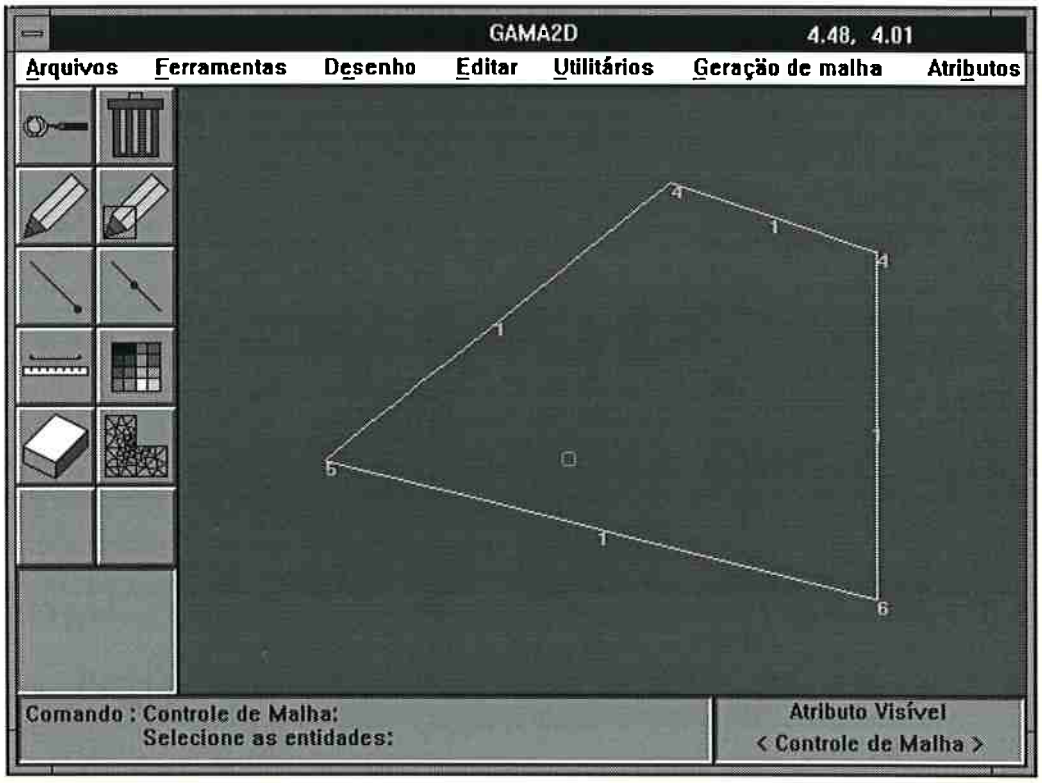
Digite o valor:

Digite: 6 ↵Enter

Como os outros dois vértices possuem o mesmo parâmetro de controle de malha, podemos introduzir seus valores simultaneamente utilizando a seleção por janela.

Digite: j ↵Enter

Defina uma janela que contenha os dois vértices que devem ter parâmetro 4.



Cada um dos dois vértices irá aparecer demarcado por um círculo indicando que estão selecionados.

Na linha de comando aparecerá a seguinte mensagem

Digite o valor:
Digite: 4 ↵Enter

Para podermos conferir os valores introduzidos, temos que ativar a visualização dos parâmetros de controle de malha.

Selecione Utilitários. Um sub-menu se abrirá.

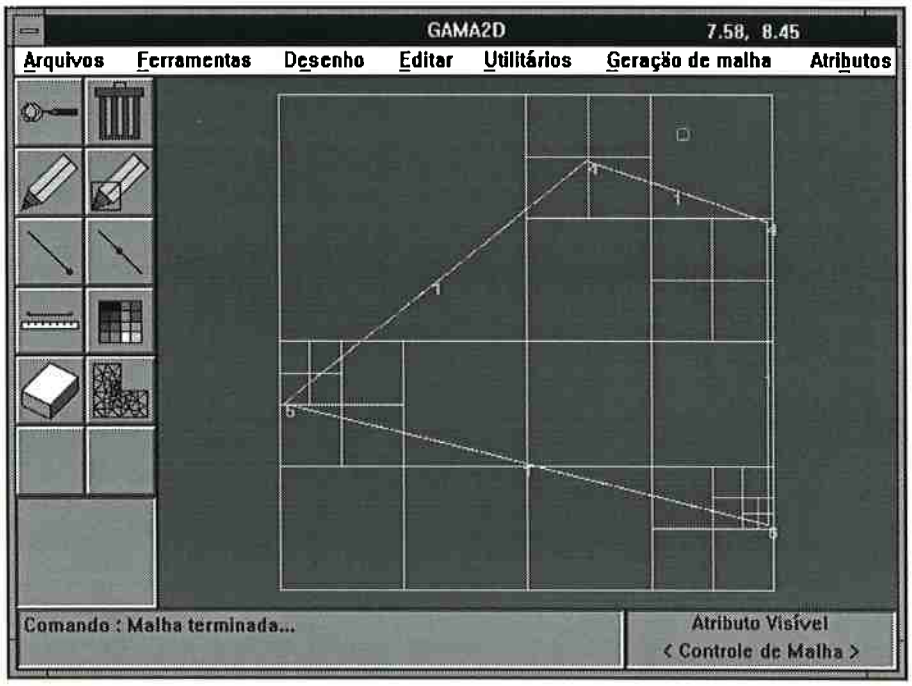
Selecione Setar Atributos. Uma janela se abrirá.

Selecione o quadrado identificando a linha Controle de Malha e a coluna Ligado.

Selecione Setas de Indicação de Faces no modo Desligado.

Tecle OK.

A região de Atributo Ativo passsará a apresentar a identificação: <Controle de Malha> e os valores dos parâmetros aparecerão próximos às entidades que representam.



Estando todos os valores dos parâmetros de controle de malha corretos podemos proceder a geração da malha. Vamos optar pela sua construção através do menu suspenso que nos dá a oportunidade de entender cada etapa do processo.

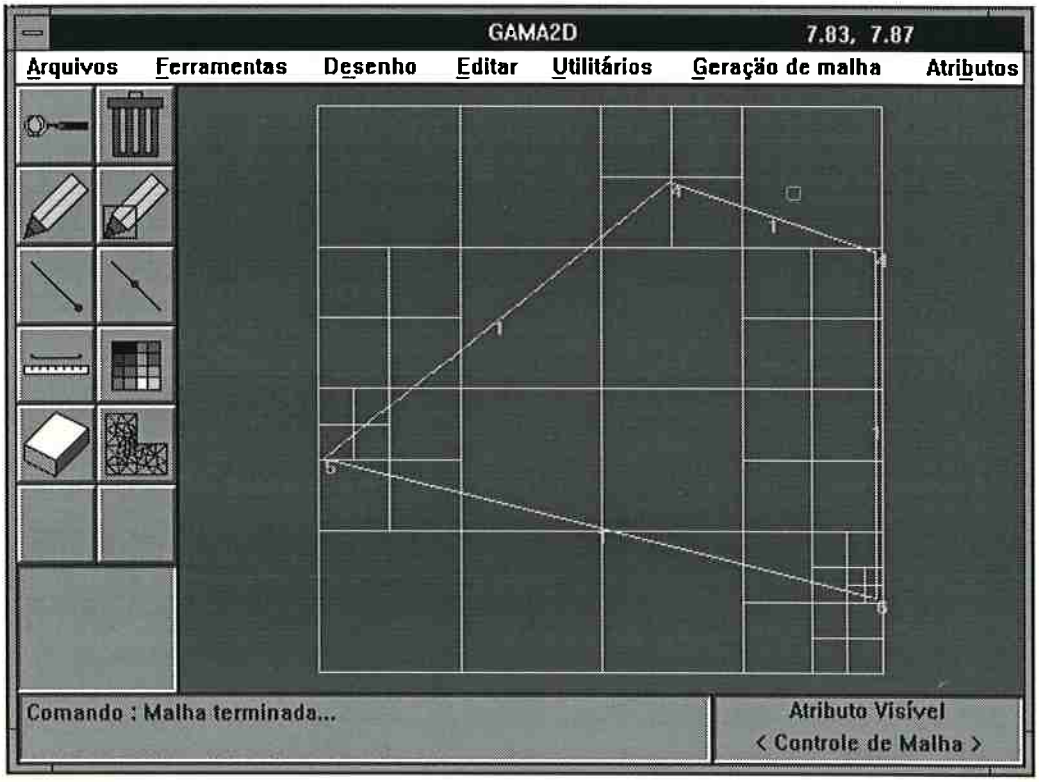
Gerando o quadtree

A primeira etapa trata da geração do quadtree obedecendo todos os níveis de refinamento impostos pelos parâmetros de controle de malha.

Selecione Geração de malha / Controle de Malha. Um sub-menu se abrirá.

Selecione Quadtree.

O quadtree será gerado conforme apresentado a seguir.



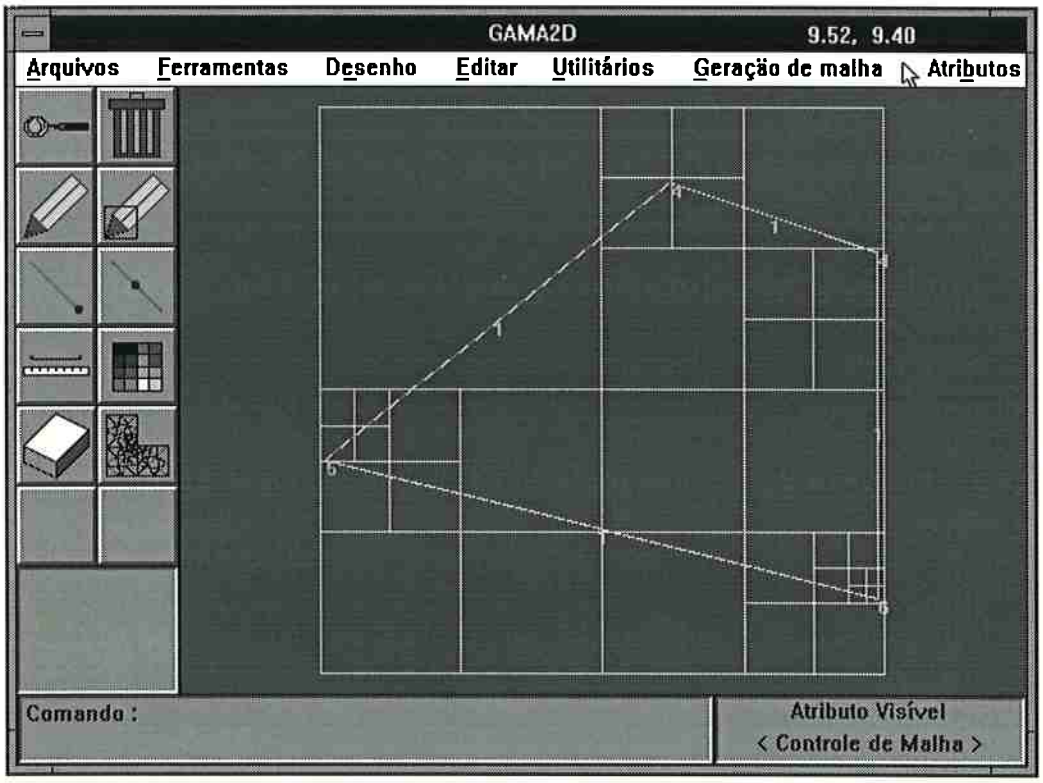
Impondo um nível de diferença entre quadrantes vizinhos

Para não gerar elementos distorcidos é necessário que quadrantes vizinhos não possuam mais do que um nível de diferença.

Selecione Geração de malha / Controle de Malha. Um sub-menu se abrirá.

Selecione Um nível.

Perceba que alguns quadrantes foram subdivididos.

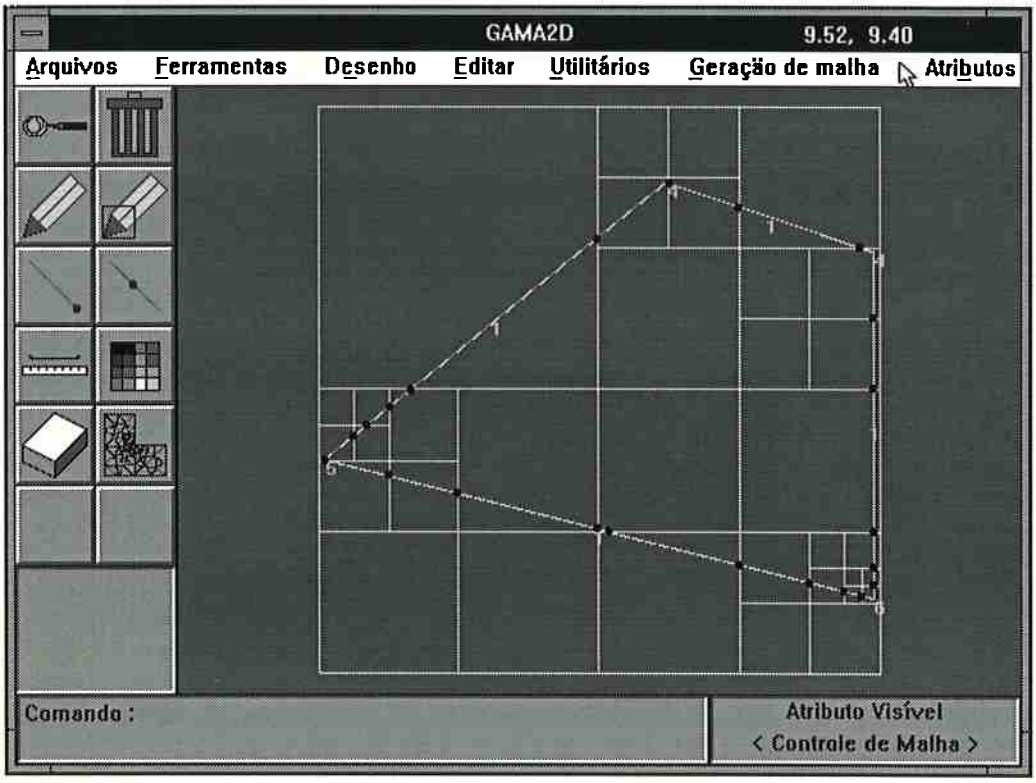


Discretizando o contorno

A seguir a fronteira da face deve ser discretizada através da intersecção da sua geometria com os quadrantes, determinando quem são os quadrantes da fronteira e qual a sua geometria.

Selecione Controle de Malha. Um sub-menu se abrirá.

Selecione Contorno.



Definindo o interior do domínio

O próximo passo é determinar quais os quadrantes que não fazem parte do domínio, isto é, são quadrantes vazios, e quais aqueles que estão inteiramente contidos no interior do domínio.