Protótipo Evolutivo de um Sistema de Modelagem de Sólidos

Luis Paulo Barbour Scott

ORIENTADORA: Dra. Maria Cristina Ferreira de Oliveira.

**Dissertação apresentada ao Instituto de Ciências Matemáticas
de São Carlos - USP, como parte dos requisitos para obtenção
do Título de Mestre em "Ciências de Computação e Matemática
Computacional".**

São Carlos

1993

RESUMO

Muitas vezes, a construção de modelos de objetos ou processos é não apenas conveniente mas também necessária para que se possa analisar e manipular processos ou objetos reais de maneira mais fácil e barata. Existem vários tipos de modelos, entre os quais os modelos computacionais. Estes são, geralmente, criados utilizando sistemas de modelagem interativos, como os sistemas de modelagem de sólidos, intensamente utilizados nas áreas de projeto e construção de peças, aviões, automóveis, etc... Esses modeladores podem utilizar várias técnicas de modelagem geométrica para descrever um objeto, como os modelos por fronteira ("Boundary Representation", B-Rep) e os modelos CSG ("Constructive Solid Geometry").

Este trabalho trata da técnica de modelagem por fronteira e relata a construção de um primeiro protótipo do Sistema de Modelagem de Sólidos Tridimensionais (SMS), que está sendo desenvolvido no ICMSC-USP, São Carlos. Esse protótipo é constituído de um subsistema gerenciador de representações, responsável pelo acesso e atualização da representação B-Rep (interna), e uma interface simples contendo algumas operações de descrição de objetos.

O sistema teve seu desenvolvimento baseado no "Paradigma da Orientação a Objetos". Uma discussão sobre modelagem de sólidos e modeladores de sólidos é apresentada, juntamente com a descrição do projeto orientado a objetos e a implementação do protótipo.

ABSTRACT

The creation of object or process models is not only convenient but necessary to analyze and manipulate real processes and objects at a low cost in a simpler way. Among the several types of models, used by professionals in different areas are the computer models. These models are usually created using interactive modeling systems such as the solids modeling systems extensive by used in the design of airplanes, mechanical parts, automobiles, etc... These modelers make use of a variety of geometric modeling techniques to represent objects, such as both the B-Rep (Boundary Representation) and the CSG (Constructive Solid Geometry) models.

This work discusses the B-Rep solid modeling technique and reports the construction of the first **3D Solid Modeling System (SMS) prototype**. It is being developed at the ICMSC-USP, in São Carlos. The prototype is composed by both a **B-Rep Representation Manager Subsystem**, responsible for accessing and updating the (internal) B-Rep representation, and a **simple interface with "sweeping-based" operations for describing the objects**. Its design followed the object-oriented approach. A general discussion about solid modeling and solid modelers are also presented. It is also presented a description of the object-oriented project for the prototype and its implementation.

Aos meus pais Eleakin e Rachel, e aos
meus irmãos Fabio e Elisa, aos quais
devo tudo que sou e tenho nessa vida. E
a Constantino Barbour pelo grande avô e
amigo que foi.

AGRADECIMENTOS

À Professora Dra. Maria Cristina Ferreira de Oliveira pela excelente orientação, apoio e paciência no decorrer do desenvolvimento deste Trabalho.

Ao Professor Dr. Paulo César Masiero pela colaboração na especificação do sistema.

À FAPESP pelo apoio financeiro concedido.

À Bethe, Laura, Paulinho, Silvana e Marquinho pela atenção e incentivo constantes.

Aos amigos João, Omar, Oswaldo, Guilherme, Marcão, Cozin, Marcus Vinicius, Hasegawa, Delamaro, Tchelo, Renata (Titi), Cybelle, Pastel e Kate pelo companherismo, apoio e incentivo.

Ao professor Dr. Odelar Leite Linhares por todo incentivo e atenção nesses 6 anos.

À avó Leonice M. Barbour pelo incentivo, apoio e carinho, e aos tios Valmiro, Maria José, Fulvia, Pedro, Sarita e Laerte pelo incentivo.

Aos professores Chahine, Márcio e José Ruggiero pelo incentivo.

A todos os professores, alunos e funcionários do Instituto de Ciências Matemáticas de São Carlos que, de uma forma ou de outra, colaboraram para o desenvolvimento deste Trabalho.

CONTEÚDO

Capítulo 1: Introdução	-01-
1.1 Considerações Gerais	-01-
1.2 Motivação	-03-
1.3 Descrição do Trabalho	-04-
1.4 Organização da Dissertação	-05-
Capítulo 2: Modelagem de Sólidos	-07-
2.1 Considerações Iniciais	-07-
2.2 Modelagem por Decomposição	-08-
2.3 Modelagem Construtiva	-10-
2.4 Modelagem por Fronteira (B-Rep: Boundary Representation)	-13-
2.5 Considerações Finais	-22-
Capítulo 3: Modeladores de Sólidos	-23-
3.1 Considerações Iniciais	-23-
3.2 Alternativas de Arquitetura	-24-
3.3 Possíveis Estruturas de Representação	-30-
3.4 Procedimentos Topológicos e Geométricos	-45-
3.5 Observações sobre a Interface	-48-

3.6 Exemplos de Modeladores de Sólidos.....	-49-
3.7 Considerações Finais.....	-55-

Capítulo 4: Projeto do Protótipo

do SMS.....	-57-
4.1 Considerações Iniciais.....	-57-
4.2 Conceitos Básicos do Paradigma da Orientação a Objetos.....	-58-
4.3 Abordagem para o Projeto Orientado a Objetos.....	-61-
4.4 Análise dos Requisitos e Funcionalidade do SMS.....	-64-
4.5 Hierarquia de Classes.....	-74-
4.6 Dependência entre Classes.....	-77-
4.7 Listagem de Métodos.....	-81-
4.8 Fluxo de Mensagens.....	-86-
4.9 Considerações Finais.....	-88-

Capítulo 5: Construção do Protótipo do SMS.....-89-

5.1 Considerações Iniciais.....	-89-
5.2 Simulação da Orientação a Objetos.....	-90-
5.3 Subistema de Gerenciamento de Representações.....	-100-
5.4 Interface do SMS.....	-105-
5.5 Operações de Modelagem de Alto Nível.....	-116-

5.7 Testes.....	-118-
5.8 Considerações Finais.....	-119-

Capítulo 6: Conclusões.....-120-

6.1 Considerações Iniciais.....	-120-
---------------------------------	-------

6.2 Contribuições do Trabalho.....	-121-
------------------------------------	-------

6.3 Sugestões para Futuros Trabalhos.....	-122-
---	-------

Referências Bibliográficas.....-123-

Bibliografia Consultada.....-129-

LISTA DE FIGURAS.

Figura 1.1:-Modelo Fio de Arame e três possíveis interpretações para o objeto representado.....	-03-
Figura 2.1:-Exemplo de modelo por enumeração exaustiva.....	-09-
Figura 2.2:-Exemplo de modelo por decomposição celular.....	-10-
Figura 2.3:-Representação de um sólido através de uma árvore CSG.....	-12-
Figura 2.4:-Representação B-Rep de um cubo.....	-14-
Figura 2.5:-Exemplo de superfícies que não correspondem a variedades de dimensão 2.....	-17-
Figura 2.6:-Exemplo de um modelo planar.....	-18-
Figura 3.1:-Sistemas Geométricos e seus componentes.....	-25-

Figura 3.2:-Arquitetura ideal de um modelador de sólidos.....	-28-
Figura 3.3:-Arquitetura primariamente B-Rep.....	-29-
Figura 3.4:-Estrutura em camadas do GWB.....	-30-
Figura 3.5:-Visão hierárquica da estrutura Aresta-Alada.....	-33-
Figura 3.6:-Informações armazenadas pela Aresta-Alada para representar um cubo.....	-33-
Figura 3.7:-Visão hierárquica da estrutura Aresta-Alada Completa.....	-35-
Figura 3.8:-Informações armazenadas pela estrutura Aresta-Alada Completa para representar um cubo.....	-35-
Figura 3.9:-Decomposição da aresta "e" em duas semi-arestas h1 e h2.....	-37-
Figura 3.10:-Visão hierárquica da estrutura Semi-Aresta (HalfEdge).....	-37-

Figura 3.11:-Visão hierárquica da estrutura Vértice-Aresta.....	-39-
Figura3.12:-Objetocomcavidades.....	-41-
Figura 3.13:-Visão hierárquica da estrutura Aresta-Híbrida.....	-42-
Figura 3.14:-Estrutura Aresta-Híbrida.....	-43-
Figura 3.15:-Estrutura Alça-Aresta.....	-44-
Figura 3.16:-Construção de um modelo planar, usando Operadores de Euler.....	-47-
Figura3.17:-Operaçãode "tweaking"	-51-
Figura3.18:-Operaçãodechanframento.....	-52-
Figura3.19:-Arquitetura do modelador GeneSys.....	-54-

Figura 3.20:-Visão hierárquica da estrutura de dados B-Rep.....	
do GeneSys.....	-55-
Figura 4.1:-Componentes do SMS.....	-66-
Figura 4.2:-DFD geral.....	-70-
Figura 4.3:-Expansão do DFD geral.....	-71-
Figura 4.4:-Expansão do processo 2.....	-72-
Figura 4.5:-Expansão do processo 4.....	-74-
Figura 4.6:-Diagrama de hierarquia de classes do protótipo.....	-75-
Figura 4.7:-Estrutura Implementada (modificação da semi-aresta).....	-78-
Figura 4.8:-Diagrama de dependência entre classes do protótipo.....	-79-
Figura 4.7:-Tabela dos métodos ativados pelas classes do sistema.....	-87-

Figura 5.1:-Estrutura de dados B-Rep implementada para o protótipo do SMS.....	-101-
Figura 5.2:-Tela inicial do protótipo.....	-106-
Figura 5.3:-Tela para edição de uma nova cena.....	-107-
Figura 5.4:-Menu com as operações sobre cena.....	-108-
Figura 5.5:-Menu com as operações sobre sólido.....	-108-
Figura 5.6:-Menu com as operações sobre curva.....	-109-
Figura 5.7:-Tela para a recuperação de uma cena e sua edição	-110-
Figura 5.8:-Operação rotacionar sólido.....	-111-
Figura 5.9:-Operação juntar (colar) sólidos.....	-111-
Figura 5.10:-Exemplo de mensagem de erro.....	-112-

Figura 5.11:-Cena criada como exemplo, contendo um cilindro,

uma pirâmide e um bloco.....-113-

Figura 5.12:-Cena criada como exemplo, contendo um sólido criado pela

operação de varredura rotacional de uma curva.....-114-

CAPÍTULO 1

INTRODUÇÃO

1.1 - Considerações Gerais.

Muitas vezes, a construção de modelos de objetos ou processos é não apenas conveniente mas também necessária para que possamos manipular e analisar os processos ou objetos reais. Construir um modelo de um objeto significa substituir o objeto por uma representação que possa ser analisada de maneira mais fácil e barata. Existem vários tipos de modelos, entre os quais pode-se citar os modelos moleculares, os modelos matemáticos, os desenhos de engenharia e os modelos computacionais [MAN_88].

Os modelos computacionais consistem de dados armazenados em "arquivos" que podem ser manipulados e analisados. A quantidade de informações que um modelo computacional deve conter depende do conjunto de perguntas que ele pretende responder, e portanto depende da aplicação. A principal vantagem dos modelos computacionais sobre os outros tipos de modelos mencionados é que eles são de propósito geral e por isso podem ser usados em uma grande variedade de aplicações. Em geral, modelos computacionais são criados usando sistemas de modelagem interativos, entre os quais encontram-se os sistemas de modelagem de sólidos [MIL_89], que são intensamente utilizados nas áreas de projeto e construção de peças, aviões, automóveis, etc...

A modelagem geométrica é um conjunto de técnicas ou metodologias utilizadas para descrever e analisar a forma e outras características geométricas de um objeto [MAN_88], ou seja, concentra-se na criação, modificação e análise dos modelos geométricos. Ela faz uso de

conceitos de várias áreas como geometria espacial, geometria analítica, cálculo, topologia e análise numérica.

O termo **modelagem geométrica** começou a ser usado nos primeiros anos da década de 70, com o crescimento rápido da computação gráfica e da tecnologia de CAD/CAM ("Computer Aided Design" / "Computer Aided Manufacturing") [MOR_85]. A **modelagem de sólidos** é um ramo da modelagem geométrica que surgiu para cobrir as limitações de outras técnicas de modelagem, como "wire frame" (fio de arame) e modelagem por superfícies, na representação de objetos sólidos tridimensionais [MOR_85, GOM_90, THA_85, CAS_85]. Os modelos fio-de-arame ("wire-frame") caracterizam-se por serem ambíguos e incompletos (figura 1.1) e os modelos de superfícies tradicionais possuem como principal limitação a ausência de informações sobre a topologia do objeto. Dentro da modelagem de sólidos existem várias abordagens que usam conceitos diferentes para descrever os objetos, sendo que as mais comuns são denominadas **modelagem por decomposição, modelagem construtiva e modelagem por fronteira** (por bordo ou B-Rep) [GOM_90, MAN_88, MOR_85, VAR_84].

Figura 1.1:-Modelo fio-de-arame e três possíveis interpretações para o objeto representado.

1.2 - Motivação

O avanço tecnológico ocorrido nas últimas décadas permitiu a utilização da **Computação Gráfica** em inúmeras áreas, como industrial, cinematográfica e médica. Na área industrial, os sistemas CAD/CAM estão sendo cada vez mais utilizados em diversas aplicações, tornando-se mais poderosos e sofisticados. Esses sistemas (e outros como, sistemas para visualização de moléculas, reconstrução de imagens médicas e de satélites, ...) requerem mecanismos de modelagem precisos e abrangentes, o que tem incentivado as pesquisas nas áreas de modelagem

geométrica e modelagem de sólidos, e torna essas áreas campos de bastante interesse dentro da Computação Gráfica.

Existe um projeto no ICMSC-USP São Carlos, que visa a construção de um **Sistema de Modelagem de Sólidos Tridimensionais**, voltado para aplicações científicas e de engenharia. Um sistema de modelagem de sólidos (SMS) é um sistema computacional que permite a criação, edição e acesso a representações de objetos sólidos. Os processos que acessam essas representações de sólidos são programas de visualização que podem ser considerados parte do SMS e programas de aplicação externos ao SMS [REQ_82].

O sistema a ser desenvolvido deverá suportar diferentes técnicas convencionais de modelagem de sólidos (como Geometria Construtiva e Representação por Fronteira) associadas às técnicas de representação de superfícies. Desta forma, pretende-se obter um **modelador de sólidos híbrido** capaz de aproveitar as vantagens das diferentes representações.

1.3 - Descrição do Trabalho.

O projeto no qual está inserido este trabalho tem por objetivo a construção da arquitetura básica de um modelador de sólidos B-Rep, que possa ser estendido posteriormente para acomodar outros tipos de representações de sólidos, como CSG. O trabalho, descrito nessa dissertação relata portanto o desenvolvimento e implementação de um primeiro protótipo do SMS, que consiste de um subsistema gerenciador de representações B-Rep e uma interface inicial para a descrição de objetos.

O subsistema gerenciador de representações é responsável pela manutenção e gerenciamento do conjunto de representações internas do modelador. Este subsistema é composto por dois níveis de "software": o primeiro é formado pelas estruturas de dados para a representação dos modelos e um conjunto de primitivas básicas de acesso a essas estruturas, e o segundo é formado pelos Operadores (Topológicos) de Eulers. A nível de interface, foram implementados procedimentos para a descrição de modelos, de forma a permitir a verificação da funcionalidade e correção do subsistema de representação. O protótipo teve seu desenvolvimento baseado no "Paradigma da Orientação a Objetos", permitindo um estudo dos problemas e vantagens do uso desta técnica no projeto e na implementação de um modelador de sólidos.

1.4 - Organização do Dissertação.

O capítulo 2 contém os conceitos básicos da modelagem de sólidos, e uma breve introdução às três principais técnicas de modelagem de sólidos entre muitas que são discutidas na literatura. A modelagem por fronteira é discutida em detalhes, uma vez que foi a técnica implementada nesse primeiro protótipo do SMS. No capítulo 3, são discutidos os Modeladores de Sólidos em geral. Primeiramente, são apresentadas arquiteturas de software para modeladores de sólidos. A seguir são apresentadas várias estruturas de dados B-Rep, bem como os procedimentos geométricos e topológicos responsáveis pela manipulação dessas estruturas. Também são feitas algumas observações sobre a interface dos modeladores. Por último, apresenta-se uma breve descrição de alguns modeladores de sólidos encontrados na literatura e suas principais características.

No capítulo 4, são discutidos os principais conceitos do **paradigma da orientação a objetos** e a abordagem para projeto orientado a objetos adotada para o desenvolvimento do sistema. A seguir, é descrita a análise dos requisitos e funcionalidade do SMS, também são apresentados os resultados do projeto orientado a objetos para o protótipo desenvolvido.

O capítulo 5 trata da implementação. Inicialmente, é discutida a simulação da orientação a objetos utilizada na implementação do protótipo. A seguir, a estrutura de dados implementada é apresentada, juntamente com os métodos básicos de manipulação associados. A interface do modelador, e as operações de modelagem de alto nível implementadas também são discutidas. Também são abordados alguns aspectos de teste do protótipo. O capítulo 6 apresenta a conclusão do trabalho, apresentando as suas contribuições, seguidas de sugestões para futuras pesquisas.

Juntamente com a dissertação foram criados dois documentos contendo detalhes do projeto e da implementação do protótipo [SCO_93a, SCO_93b], que servem como referências adicionais à dissertação. Encontra-se em preparação o manual do usuário.

CAPÍTULO 2

MODELAGEM DE SÓLIDOS

2.1 - Considerações Iniciais.

A modelagem de sólidos possui como principal objetivo a representação completa e não ambígua de objetos sólidos. Esse objetivo traz alguns problemas para os modelos de sólidos, entre eles a complexidade e a garantia de integridade [MAN_88], tornando a construção de modeladores de sólidos uma tarefa não trivial.

A modelagem de sólidos surgiu na década de 60 com pouco suporte teórico. Uma teoria adequada e matematicamente rigorosa apareceu somente no final dos anos 70 com o trabalho de A. Requicha [REQ_80, REQ_82]. As técnicas de representação mais comuns dentro da modelagem de sólidos são a modelagem por decomposição, modelagem construtiva e modelagem por fronteira.

A partir das propriedades de um sólido válido (**rigidez, finitude, homogeneidade tridimensional, determinismo de fronteira, ...**) definidas por Requicha [REQ_80], pode-se estabelecer uma série de propriedades que são desejáveis nos esquemas de representação [REQ_82, MAN_88], entre as quais incluem-se:

- não ambigüidade:** cada representação admitida pelo esquema corresponde a um único objeto.
- unicidade:** cada objeto possui uma única descrição em um dado esquema.
- potência descritiva:** o conjunto de objetos possíveis de serem representados deve ser o mais amplo possível.

-potência descritiva: o conjunto de objetos possíveis de serem representados deve ser o mais amplo possível.

-concisão: o esquema deve ser capaz de representar os sólidos da maneira mais concisa possível, isto é, armazenando o mínimo de informações.

-facilidade descritiva: o esquema deve permitir ao usuário descrever os objetos de maneira simples.

Apesar de serem propriedades desejáveis, nem todas são encontradas nos esquemas de representação de sólidos, com exceção da primeira.

Neste capítulo são introduzidas as principais técnicas de modelagem de sólidos, a saber, a modelagem por decomposição (seção 2.2), modelagem construtiva (seção 2.3) e modelagem por fronteira (seção 2.4). Esta última técnica será discutida em detalhes, uma vez que no trabalho desenvolvido utilizou-se desse tipo de representação. Mais informações sobre estes esquemas podem ser encontrados em [MAN_88, MIL_89, MOR_85, REQ_82, HOF_89].

2.2 - Modelagem por Decomposição.

Os modelos por decomposição descrevem os sólidos através da **combinação (união)** de **blocos básicos de construção**. O tipo de bloco usado e a maneira como a combinação dos blocos é armazenada dão origem a diferentes modelos por decomposição. Têm-se por exemplo modelos gerados por **enumeração exaustiva** e **modelos de decomposição celular**, entre outros. Estes modelos são introduzidos a seguir.

2.2.1 - Enumeração Exhaustiva.

Nos modelos por enumeração exhaustiva, os blocos básicos são cubos de tamanhos iguais. Assim, estes modelos representam os sólidos através de cubos contidos (total ou parcialmente) no sólido. Pressupõe-se que os cubos não se sobrepõem e possuem tamanho e orientação uniformes, formando uma subdivisão regular do espaço. A enumeração exhaustiva fornece uma representação aproximada do objeto que, em geral, não é concisa. Um exemplo desse tipo de representação pode ser visto na figura 2.1.

Figura 2.1:- Exemplo de modelo por enumeração exhaustiva [MAN_88].

2.2.2 - Decomposição Celular.

A decomposição celular é semelhante à enumeração exhaustiva, porém os blocos básicos não precisam ser cubos e não precisam ter tamanho uniforme, podendo inclusive ser células (blocos) curvas.

As células são semidisjuntas, isso é, tocam-se apenas nas superfícies que as limitam, sendo que uma célula não pode conter pontos internos a outra célula. Um exemplo de modelo por decomposição celular é mostrado na figura 2.2.

Figura 2.2:-Exemplo de modelo por decomposição celular [MAN_88].

2.3 - Modelagem Construtiva.

Os modelos construtivos representam os sólidos como uma combinação de primitivas pré-definidas. Todo modelo construtivo considera um sólido como um conjunto de pontos do espaço euclidiano tridimensional (E^3), e a idéia é combinar conjuntos de pontos simples, que possam ser representados de maneira direta, para modelar outros conjuntos de pontos mais complexos.

Existem dois tipos de modelos construtivos, os modelos "half-space" (semi-espaços) puros e os modelos CSG ("Constructive Solid Geometry"), sendo que estes últimos são os mais comuns [MAN_88, MIL_86]. Os "half-spaces" são conjuntos infinitos de pontos, resultantes da divisão do espaço euclidiano por um plano em dois sub-conjuntos infinitos. Os modelos "half-space" puros são formados pela combinação desses conjuntos através de operações booleanas (união, intersecção e diferença); de forma a se obter regiões limitadas do espaço que descrevem um sólido.

Já a técnica CSG usa o conceito de "half-space" (semi-espaço) para definir as suas primitivas sólidas básicas, mas permite que o usuário trabalhe apenas com as primitivas pré-definidas (finitas). Modelos CSG são formados por primitivas básicas combinadas através de operações booleanas regularizadas [MAN_88, MIL_86, MOR_85, GOM_90] e movimentos rígidos (rotação e translação). A maneira mais natural de representar um modelo CSG é através da chamada árvore CSG, cuja definição é apresentada na "caixa" a seguir (box 2.1).

```
<árvore CSG>:=  
    <primitiva>;  
    <árvore CSG> <operações de conjunto> <árvore CSG>;  
    <árvore CSG> <movimento rígido>.
```

Box 2.1: -Definição de árvore CSG.

Onde <primitiva> é a instância de uma primitiva sólida, representada por um identificador e um conjunto de parâmetros que definem a sua dimensão; <movimento rígido> é uma translação ou rotação e <operação de conjunto> é uma operação booleana (ou operação

de conjunto). As operações de conjunto realizadas sobre as primitivas devem ser operações "regularizadas" para garantir a construção de sólidos válidos [MAN_88, JAR_85, GOM_90].

Da definição de árvore CSG, percebe-se que as primitivas estarão nas folhas da árvore, e as operações de conjunto e os movimentos rígidos formarão os nós internos. A figura 2.3 ilustra uma árvore CSG e o sólido que ela descreve.

Figura 2.3: -Representação de um sólido através de uma árvore CSG [MAN_88].

O poder de expressividade dos modelos CSG, isto é, o conjunto de sólidos que a técnica consegue representar, depende do conjunto de primitivas e operações disponíveis, uma vez que os objetos modelados são combinações das primitivas. A validade de um modelo CSG é fácil de ser garantida, pois toda árvore CSG é topologicamente válida [MAN_88]. As duas principais

vantagens dos modelos CSG é que eles tendem a ser bastante concisos e podem ser descritos facilmente por uma pessoa.

2.4 - Modelagem por Fronteira (B-Rep: "Boundary Representation").

Os modelos por fronteira (B-Rep, ou por Bordo) diferem dos modelos por decomposição (enumeração, decomposição celular,...) e dos modelos construtivos, que descrevem os objetos como conjuntos de pontos do espaço, pois representam os mesmos indiretamente, através das superfícies que os limitam. Estes modelos evoluíram dos modelos "fio de arame" [MAN_88]. Eles dividem a superfície que representa o sólido em faces, de tal forma que estas tenham uma representação matemática compacta. Uma série de critérios topológicos deve ser respeitada durante essa divisão [MAN_88]. As curvas que limitam as superfícies às quais as faces pertencem também são subdivididas em arestas que possuam uma representação conveniente (isso é, pertençam a uma curva que possua uma equação paramétrica). A porção da curva que forma cada aresta é representada em função dos vértices (pontos extremos). A figura 2.4 ilustra essa decomposição para um cubo.

Figura 2.4:-Representação B-Rep de um cubo [MAN_88].

Dessa maneira, um modelo B-Rep possui três elementos básicos: as faces, arestas e vértices que compõem um sólido, além de informações geométricas (equações de curvas e superfícies que definem as arestas e faces) e informações topológicas (dados sobre as relações de adjacência, ou conectividade, entre os elementos).

2.4.1 - Conceitos Básicos.

Os modelos por fronteira são também chamados de modelos explícitos, porque trabalham explicitamente com informações topológicas sobre os objetos. Modelos B-Rep que possuem apenas faces planares são denominados modelos poliedrais. Nesse caso, as arestas são sempre segmentos de reta, o que simplifica bastante a estrutura de dados que deve ser mantida por um modelador para armazenar a descrição dos sólidos [TUR_88]; e também a implementação de operações de modelagem.

Sempre que um modelo de um objeto é criado, é preciso validá-lo. Os critérios de validade para um modelo B-Rep analisam considerações geométricas e topológicas, e incluem as seguintes condições:

- 1- O conjunto de faces de um modelo por fronteira forma uma casca fechada e completa do sólido.
- 2- Faces do modelo não interceptam outras faces exceto em vértices e arestas comuns a elas.
- 3- As bordas da face são polígonos simples, que não se interceptam entre si.

A primeira condição está relacionada com a integridade topológica dos sólidos modelados e é fácil de ser imposta através da estrutura de dados. Infelizmente, a integridade geométrica definida pelas duas últimas condições não pode ser "forçada" facilmente. Assim, informações geométricas inadequadas podem ser associadas a modelos B-Rep, criando dessa maneira modelos inválidos [JAR_85, MAN_88].

Já vimos que os modelos B-Rep usam a superfície que limita o sólido para descevé-lo, e possuem como principais elementos as Faces, Arestas e Vértices que os compõem, além de informações geométricas e topológicas sobre o sólido modelado. Alguns conceitos importantes para a compreensão dos modelos B-Rep, entre os quais os conceitos de Modelos Planares e de Operadores de Euler, são introduzidos a seguir [MAN_88, MIL_86, BOY_82] .

2.4.2 - Modelos Planares.

Tem-se que o espaço Euclidiano E^3 é uma idealização apropriada do espaço real onde encontram-se os objetos sólidos que queremos modelar. Dessa maneira a abstração matemática mais geral de um objeto sólido é um sub-conjunto de E^3 , isto é, um conjunto de pontos do espaço Euclidiano.

Assim, podemos representar um sólido matematicamente como um conjunto fechado, limitado e regular, que possua a propriedade de rigidez sobre operações de translação e rotação [MAN_88]. Porém, o conceito de conjunto contínuo de pontos não possui significado no computador, e assim é necessário manter uma representação indireta do sólido. Dessa maneira, utiliza-se a superfície do sólido como uma representação indireta para ele.

A caracterização do sólido baseada em superfícies considera a fronteira (borda) do objeto: A borda é formada por um conjunto de faces "grudadas" que formam uma "casca" completa ao redor do objeto. Intuitivamente uma superfície pode ser considerada um sub-conjunto de E^3 , que é essencialmente "bidimensional", isso é, todo ponto da superfície está rodeado por uma região "bidimensional" de pontos pertencentes à superfície.

Essa **bidimensionalidade** inerente de uma superfície indica que ela pode ser estudada através de um **modelo bidimensional**, denominado **modelo planar**. Porém, antes é necessário definir uma noção mais abstrata de uma superfície que representa um sólido válido em E^3 , para então construir um **modelo bidimensional simples** para representar estas superfícies. Desta maneira, usa-se o conceito de "**Variedade de Dimensão 2**" ("**2-Manifold**"). A definição formal é dada a seguir na caixa 2.2.

Definição: Uma variedade de dimensão 2 (um "2-Manifold") M é um espaço topológico onde todo ponto possui uma vizinhança topologicamente equivalente a um disco de E^2 (espaço euclidiano bidimensional).

Caixa 2.2:- Definição matemática formal de uma variedade de dimensão 2.

Para tornar mais fácil a compreensão do que é uma variedade de dimensão 2, pode-se usar uma noção intuitiva. Assim, intuitivamente, um inseto vivendo em uma M vê uma região simples e contínua da superfície ao seu redor; esta visão pode ser aplicada a nós e a Terra, ou seja, a esfera é uma variedade 2. Superfícies que não são deste tipo ("variedades de dimensão 2") são ilustradas na figura a seguir (figura 2.5).

Figura 2.5:- Exemplo de superfícies que não correspondem a variedades de dimensão 2 [MAN_88].

Uma restrição das técnicas de representação B-Rep mais convencionais é que os modelos planares, criados para representar a "casca" (superfície) de um sólido, conseguem representar apenas variedades de dimensão 2. Com isso, os modelos planares correspondem necessariamente a representações apenas de sólidos manufaturáveis.

Portanto, um modelo planar é a representação da superfície de um objeto sólido, decomposta em faces e fornece uma abstração matemática para os modelos de fronteira. As propriedades topológicas da superfície de um objeto podem ser obtidas a partir do seu modelo planar.

Um modelo planar é um grafo planar dirigido $\{N, A, R\}$, com um número finito de vértices $N = \{n_1, n_2, n_3, \dots\}$, arestas $A = \{a_1, a_2, a_3, \dots\}$ e polígonos $R = \{r_1, r_2, r_3, \dots\}$ limitados pelas arestas e vértices. Cada polígono possui uma certa orientação com suas arestas e vértices. O modelo planar de uma pirâmide é ilustrado na figura 2.6.

Figura 2.6:-Exemplo de um modelo planar para uma pirâmide [MAN_88].

2.4.3 - Operadores de Euler.

É possível mostrar, através de um desenvolvimento matemático (v. Mäntylä [MAN_88]), que um conjunto pequeno de operações (e suas inversas) é suficiente para descrever e manipular

todos os modelos planares que correspondem a sólidos válidos. Pois garante-se que todo modelo planar construído por esse operadores obedece a fórmula de Euler-Poincaré: $v - e + f = 2(s+h)$. Onde "v" é o número de vértices, "f" o número de faces, "e" o número de arestas, "s" o número de "shell" (cascass) e "h" o número de buracos do sólido representado pelo modelo planar.

Esta teoria pode ser estendida para os modelos de fronteira. As operações de manipulação de modelos planares correspondem aos operadores de Euler, que atuam na topologia da estrutura de dados de um modelo de fronteira.

A teoria matemática dos modelos planares, que está por trás dos Operadores de Euler, garante que, através deles, apenas sólidos topologicamente válidos são construídos. Assim, esses operadores possuem duas propriedades muito importantes para a técnica B-Rep, que são:

1-"Toda estrutura de dados B-Rep (que descreve sólidos) válida pode ser criada por uma seqüência de operadores de Euler".

2-"Os operadores de Euler não podem criar estruturas de dados B-Rep (sólidos) topologicamente inválidas".

Os operadores de Euler podem ser agrupados em **operadores globais (que podem mudar propriedades topológicas globais dos objetos)** e **locais**. Isso significa que os operadores globais podem dividir um sólido em dois componentes, ou criar buracos nas faces de um sólido [MAN_88]; enquanto os operadores locais apenas criam arestas, vértices ou dividem as faces de um sólido.

2.4.4 - Propriedades e Validade.

Entre as propriedades dos modelos B-Rep pode-se mencionar:

1-São difíceis de serem descritos diretamente pelo usuário.

Esse problema pode ser resolvido usando estratégias que auxiliem na descrição dos modelos e que garantam a validade dos modelos construídos, ou pelo menos dificultem a construção de modelos inválidos. Entre as estratégias que podem ser empregadas a nível de interface com o usuário estão: fornecer ao usuário uma linguagem de descrição baseada na técnica CSG, e posterior conversão automática (exata ou aproximada) dos modelos CSG para B-Rep; a construção de sobre primitivas bidimensionais e a realização de operações de varredura ("sweeping") dessas primitivas para gerar um objeto tridimensional (a operação de varredura corresponde a transladar ou rotacionar, em torno de um eixo, uma superfície bidimensional no espaço tridimensional); e fornecer suporte para a execução de modificações locais e de operações booleanas sobre os modelos.

2- Facilitam os algoritmos de visualização, devido a disponibilidade de informações explícitas sobre a topologia do sólido (conectividade entre faces, arestas e vértices). Todas as técnicas de visualização podem ser diretamente aplicadas aos modelos B-Rep. Além disso, se o modelo B-Rep for poliedral, técnicas bem conhecidas de remoção de linhas e superfícies escondidas podem ser aplicadas.

3- São não ambíguos, ou seja, um modelo válido corresponde a um único objeto sólido.

4- Não são únicos, pois um mesmo objeto pode ser decomposto de várias maneiras diferentes. Por isso, um mesmo objeto pode ser representado por diferentes modelos B-Rep válidos.

5-Sua validade é difícil de ser estabelecida. Existe uma vasta teoria matemática baseada em modelos planares e conceitos de validade topológica, que fornece a base para a construção de modelos B-Rep válidos [MAN_88, JAR_85].

6-Por armazenarem as informações topológicas de forma explícita, os modelos por fronteira podem se tornar muito extensos. Existem alguns métodos para tornar o modelo mais conciso, sendo que uma possível estratégia é descrita por Mäntylä em [MAN_88].

2.4.5 - Conjunto de Superfícies e Curvas.

O conjunto de curvas e superfícies admitidas por um modelador B-Rep é influenciado pela estrutura de dados e pelos procedimentos geométricos disponíveis do modelador [GOL_87, CAS_85]. O conjunto de superfícies e curvas que o sistema admite, por sua vez, influencia o conjunto de objetos que podem ser modelados pelo sistema e também o desempenho do modelador. Atualmente, existe intensa pesquisa sobre a incorporação de superfícies de forma-livre¹ a modeladores de sólidos, pois isso aumenta significativamente o conjunto de objetos que podem ser modelados. Ainda existem vários problemas sérios relacionados à incorporação de superfícies de forma-livre aos modeladores de sólidos. Pois complicam bastante os algoritmos

¹-Superfícies cuja forma não pode ser classificada em uma família como a das quádricas.

geométricos (responsáveis por garantir a validade geométrica dos sólidos) e alguns algoritmos de modelagem como as operações booleanas de conjunto (ver discussões em [MIL_89, GOL_87, CAS_87, LIC_87]).

2.5 - Considerações Finais.

Existem várias técnicas de modelagem de sólidos e cada qual possui vantagens e desvantagens em contextos específicos. Duas técnicas bastante utilizadas atualmente são CSG e B-Rep. Os modelos B-Rep são muito empregados para modelar objetos sólidos, sendo sustentados e validados por uma forte base matemática que envolve conceitos de topologia e geometria; e possuem importantes vantagens em relação aos outros para a visualização dos modelos e cálculos de propriedades, além de outras. Mas é importante enfatizar que nenhum esquema de representação é perfeito, e um esquema em particular será mais ou menos adequado dependendo do tipo de uso a ser feito da representação.

Neste capítulo, introduziu-se a modelagem de sólidos e seus principais esquemas de representação, para a seguir apresentar, no capítulo 3, um estudo sobre modeladores de sólidos em geral.

CAPÍTULO 3

MODELADORES DE SÓLIDOS

3.1 - Considerações Iniciais.

Todos os modeladores de sólidos possuem uma arquitetura de software que depende das técnicas de modelagem presentes no mesmo, uma vez que estas influenciam os componentes necessários ao modelador. Por sua vez, a precisão e poder de descrição estão diretamente ligados à arquitetura dos modeladores. Baseado nos conceitos sobre as técnicas de modelagem de sólidos, vistos no Capítulo 2, é apresentada neste capítulo uma revisão bibliográfica sobre alguns tópicos que são importantes para a compreensão da especificação e do desenvolvimento do protótipo desenvolvido nesse trabalho, descritos nos Capítulos 4 e 5.

Na seção 3.2 são feitas algumas considerações gerais sobre a arquitetura de software de um modelador de sólidos, e os componentes de um modelador B-Rep. O estudo dos componentes foi feito principalmente a partir da descrição do modelador GWB ("Geometric Workbench") proposto por Mäntylä [MAN_88], que dá um tratamento bastante detalhado do processo de construção. A seguir, na seção 3.3 é apresentado um estudo sobre várias estruturas de dados B-Rep descritas na literatura. Na seção 3.4 são discutidos os operadores topológicos e geométricos essenciais a um modelador B-Rep, e a na seção 3.5 são feitas algumas observações sobre a interface de um modelador.

Os componentes básicos que definem o gerenciador de representações do modelador, isto é, a estrutura de representação interna e as operações para sua manipulação (operações topológicas) são descritas em detalhes por serem mais relevantes para o trabalho desenvolvido, que concentrou-se na implementação desse gerenciador. Finalmente, a seção 3.6 apresenta brevemente alguns modeladores de sólidos descritos na literatura.

3.2 - Alternativas de Arquitetura.

Um conceito fundamental da **Modelagem Geométrica** é a separação funcional entre o processo e as técnicas de modelagem e à aplicação. É importante que as técnicas de modelagem sejam relativamente independentes dos objetos a serem modelados e do uso que se pretende a fazer dos modelos. Dessa forma, é possível distinguir funcionalmente entre o software que manipula os modelos e fornece respostas às questões geométricas e a aplicação propriamente dita. A figura 3.1 ilustra os componentes funcionais de um modelador de sólidos e o papel do modelador em um sistema geométrico.

Figura 3.1 :- Sistema Geométrico e seus componentes.

O sistema da figura 3.1 assume que os objetos são descritos em termos de uma linguagem de descrição, baseada nos conceitos de modelagem disponíveis no modelador. O usuário pode fornecer a descrição em forma textual ou através de uma interface gráfica. As descrições dos objetos são então traduzidas para a representação interna e armazenadas pelo modelador. A relação entre a linguagem de descrição e a representação interna não precisa ser direta: as representações internas podem empregar conceitos de modelagem diferentes daqueles utilizados na descrição original. Além disso, o modelador de sólidos pode incluir várias linguagens de descrição, voltadas para diferentes tipos de usuários e aplicações.

O modelador também deve fornecer interfaces para a comunicação com outros sistemas (algoritmos, ou outros modeladores) e deve também incluir facilidades para armazenar descrições de objetos e outros dados em bases de dados permanentes.

Quando leva-se em conta a escolha das técnicas de modelagem, existem várias arquiteturas possíveis para um modelador de sólidos. Um fator importante a ser considerado é que um modelador pode admitir múltiplas representações internas coexistindo (arquitetura híbrida). Dessa forma, um sistema de modelagem por fronteira pode ser puro (isto é, usar apenas a representação B-Rep), ou ser híbrido (usar outras formas de representação além da B-Rep).

A necessidade de se ter um sistema híbrido é uma questão de eficiência, pois, como já foi mencionado, cada esquema de representação (modelagem por decomposição, modelagem por fronteira e modelagem construtiva) possui vantagens e desvantagens em situações específicas [MAN_88, THA_85, MIL_86]. Os modelos construtivos, por exemplo, são concisos e fáceis de serem construídos, mas não muito apropriados como fonte de dados para algoritmos de visualização (responsáveis pela geração das imagens correspondentes aos modelos). Já os modelos por fronteira são extensos e difíceis de serem descritos pelo usuário, mas são adequados para a utilização direta por algoritmos de visualização eficientes.

Dessa maneira, o ideal é que o sistema de modelagem possua uma arquitetura híbrida, que possa utilizar a representação mais apropriada para cada tarefa. Isso não é fácil, pois os sistemas híbridos apresentam vários problemas [MIL_86]. O principal deles está em como manter consistentes as representações que coexistem no sistema. Ao modificar uma representação é necessário atualizar as outras, o que é possível utilizando algoritmos de conversão que possam traduzir (de maneira exata ou aproximada) dados de uma representação para outra.

Os algoritmos de conversão criam um modelo do objeto em uma determinada representação, a partir do modelo desse objeto em outra representação. Porém nem todas

conversões entre representações são possíveis (e muitas vezes só é possível fazer uma conversão aproximada). Um típico algoritmo de conversão muito usado e conhecido é o que converte representações CSG para representações B-Rep, denominado genericamente **algoritmo de avaliação de borda** [MIL_86, MAN_88, CAS_85]. Uma arquitetura híbrida ideal é ilustrada na figura 3.2. Apesar de ideal, essa arquitetura não é factível devido, entre outros problemas, à ausência de algoritmos de conversão da representação B-Rep para a CSG (v. [MIL_86]).

Figura 3.2:- Arquitetura ideal de um modelador de sólidos.

A maioria dos modeladores de sólidos discutidos na literatura apresenta uma arquitetura híbrida CSG-B-Rep como alternativa à arquitetura ideal, onde a técnica CSG é usada principalmente para auxiliar a descrição inicial dos objetos pelos usuários do sistema e a B-Rep é empregada para a representação interna sobre a qual serão posteriormente executadas operações de edição e manipulação. Essa arquitetura, denominada primariamente B-Rep, é ilustrada na figura 3.3

Figura 3.3:- Arquitetura primariamente B-Rep.

A figura 3.4, a seguir, apresenta os principais componentes do modelador **GWB** [MAN_88]. Muitos desses componentes são encontrados em todos os modeladores B-Rep, e alguns são indispensáveis, como a estrutura de representação interna e os procedimentos para manipulação dessa estrutura, os operadores topológicos (chamados Operadores de Euler) e os operadores geométricos.

O **GWB** é organizado em camadas de software, sendo que as principais são: uma camada formada pela estrutura de dados, uma camada formada pelos procedimentos que

implementam os operadores de Euler e os procedimentos geométricos básicos, e outra camada formada pelos componentes que definem a interface (linguagem de descrição, primitivas pré-definidas, técnicas de modelagem como operações de modificação local e global (operações booleanas), operações de varredura ("sweeping") e mecanismos para correção de erros). A organização em camadas garante um sistema modular e os procedimentos em uma camada específica acessam apenas os procedimentos da camada anterior.

Figura 3.4:- Estrutura em camadas do GWB [MAN_88].

3.3 - Possíveis Estruturas de Representação

A escolha da estrutura de dados é importante porque influencia o conjunto de superfícies e curvas que o sistema admite (certas estruturas admitem faces descritas por vários tipos de superfícies, inclusive superfícies de forma livre, enquanto que outras só permitem faces planares), bem como o conjunto de objetos que pode ser representado.

Existem várias estruturas de dados para representar a geometria e a topologia de um modelo B-Rep, a maioria derivada da estrutura Aresta-Alada ("Winged-Edge"), proposta por

Baumgart em 1975 [BAU_75]. Algumas delas são descritas a seguir. Elas têm complexidade variável, dependendo da quantidade de informações sobre relações de adjacência que armazenam, e cada uma possui vantagens e desvantagens em termos de tempo de acesso, simplicidade dos algoritmos de manipulação e conjunto de sólidos que podem ser representados.

Em [WOO_85] pode-se encontrar uma análise da eficiência de várias estruturas B-Rep em termos do espaço de armazenamento exigido e tempo de acesso à entidades topológicas (faces, arestas, vértices). O estudo é baseado em análise e topologia combinatorial, com o objetivo de definir uma técnica para o projeto de estruturas de dados que simplifique os algoritmos geométricos. O termo algoritmo geométrico é usado de maneira geral, referindo-se a qualquer algoritmo que use a estrutura de dados.

Neste estudo, Woo identifica um conjunto de nove funções básicas de atualização e acesso a estruturas B-Rep que são comuns à maioria dos algoritmos geométricos. Também classifica as estruturas em oito classes, de acordo com o número de relações de adjacência entre os elementos topológicos que elas armazenam. Usando fórmulas de contagem e análise combinatorial, o autor calcula a complexidade de armazenamento e estabelece limites superiores e inferiores para o tempo de acesso à cada estrutura de dados. Apresenta também uma nova estrutura de dados (que denomina "simétrica"), que afirma ser mais eficiente em termos de espaço de armazenamento e tempo de acesso que a **Aresta-Alada**, devido ao tipo de relações de adjacência que a estrutura armazena.

3.3.1 - Aresta-Alada ("Winged-Edge").

A **Aresta-Alada** é a estrutura de dados B-Rep mais tradicional. Ela (assim como as outras que serão discutidas no texto) é uma estrutura baseada em arestas, ou seja, representa as

faces de um objeto em termos da sequência fechada de arestas denominada "**Ciclo de arestas**", ou "**Loop de arestas**" ou simplesmente "**Ciclo**" ou "**Loop**" que as compõem, usando a aresta como elemento de referência a partir do qual a adjacência com outros elementos é determinada. Os vértices que compõem a face são obtidos indiretamente a partir das arestas.

A **Aresta-Alada** possui nós explícitos associados a cada um dos três elementos básicos que compõem o objeto (vértices, faces e arestas) (figuras 3.5 e 3.6). Cada aresta possui uma orientação que pode ser negativa ou positiva, dependendo do sentido em que ela aparece no ciclo que delimita a face. Como cada aresta ocorre exatamente em duas faces, devido a critérios topológicos de validade, em uma face ela possui sentido positivo e na outra face possui sentido negativo.

Para determinar o sentido das arestas em cada face, basta olhar o objeto de fora para dentro, verificando se a aresta ocorre no sentido horário ou anti-horário no ciclo que delimita a face. Se a aresta ocorre no sentido horário, define-se que possui orientação positiva; caso contrário possui orientação negativa.

Aos nós vértice são associadas apenas as suas coordenadas. Ao nós aresta são associadas quatro informações: o vértice inicial, o vértice final, a próxima aresta na face em que ela ocorre no sentido positivo, e a próxima aresta da face em que ela ocorre no sentido negativo. Finalmente, são associadas aos nós face duas informações: a identificação da aresta inicial do ciclo que define a face e o sentido em que ela ocorre na face.

Figura 3.5:- Visão hierárquica da estrutura Aresta-Alada.

Figura 3.6:- Informações armazenadas pela estrutura Aresta-Alada para representar um cubo.

A estrutura Aresta-Alada possui a vantagem de facilitar a implementação de algoritmos de remoção de superfícies escondidas e algoritmos de sombreado, por armazenar

explicitamente informações sobre faces vizinhas. Entretanto esta estrutura só representa objetos sólidos cujas faces não possuam buracos (isto é, cujas faces possuam uma única borda), pois ela armazena uma única sequência fechada de arestas para cada face.

3.1.2 - Aresta Alada Completa ("Full Winged Edge")

A Aresta-Alada Completa é uma derivada da Aresta-Alada que também não é capaz de representar faces delimitadas por mais de uma borda. A diferença entre ambas está no conjunto de dados associados aos nós faces e arestas, e na disponibilidade de informações de adjacência.

Para cada nó Face, a Aresta-Alada completa armazena uma aresta pertencente à face. Para cada nó Vértice são armazenadas as coordenadas e o identificador de uma aresta adjacente. A cada nó Aresta são associadas 8 informações: o vértice inicial, o vértice final, a próxima aresta da face em que ela ocorre no sentido positivo, a próxima aresta da face em que ela ocorre no sentido negativo, os identificadores das faces vizinhas (isto é, as duas faces nas quais a aresta ocorre), o identificador da aresta anterior da face na qual ela ocorre no sentido positivo e o identificador da aresta anterior da face na qual ela ocorre no sentido negativo.

A figura 3.7 ilustra a visão hierárquica da estrutura Aresta-Alada-Completa, e a figura 3.8 exemplifica as informações armazenadas para um cubo.

Figura 3.7:- Visão hierárquica da estrutura Aresta-Alada Completa.

Figura 3.8:-Informações armazenadas na estrutura Aresta-Alada Completa para representar um cubo.

3.3.3 - Semi-Aresta ("HalfEdge").

Essa estrutura de dados também é derivada da Aresta-Alada, tendo sido proposta por Mäntylä em [MAN_88] e utilizada no modelador GWB. A estrutura básica proposta suporta apenas modelos poliedrais (ver capítulo 2), mas pode ser estendida, segundo o autor, para acomodar diferentes tipos de curvas e superfícies quádricas.

A semi-aresta é baseada na definição dos modelos planares como grafos dirigidos $\{N, A, R\}$, onde N é o conjunto de vértices, A é o conjunto de arestas e R o conjunto de polígonos. Para representar o grafo $\{N, A, R\}$, é utilizada uma estrutura com cinco níveis hierárquicos constituída de nós do tipo Sólido, Face, Ciclo, Semi Aresta e Vértice.

Essa estrutura usa o conceito de "semi- aresta" ("halfedge"), que interpreta cada aresta como composta de duas "metades" que correspondem às suas ocorrências nos sentidos positivo e negativo nas faces adjacentes. Assim cada aresta é decomposta em dois componentes (duas semi-arestas), que são segmentos de reta em um ciclo. A figura 3.9 ilustra a decomposição da aresta (e) em duas metades ($h1$ e $h2$), e a visão hierárquica da estrutura de dados semi-aresta é mostrada na figura 3.10.

Figura 3.9:-Decomposição da aresta "e" em duas semi-arestas h1 e h2 [MAN_88].

Figura 3.10:-Visão hierárquica da estrutura de dados Semi-Aresta ("halfedge") [MAN_88].

Uma das vantagens dessa estrutura sobre a Aresta-Alada é a de permitir que uma face possua várias bordas, o que é possível porque a cada face é associada uma lista de ciclos. Isso

permite que um número maior de objetos possa ser modelado, quando uma face possui mais de uma borda, uma é considerada externa e as outras são consideradas como anéis (isto é, ciclos internos que delimitam buracos na face).

Segundo Mäntylä [MAN_88], a estrutura **semi-aresta** pode ser estendida para incluir todos os tipos de superfície encontrados na forma CSG (quádricas naturais e tori) e também superfícies paramétricas (de forma livre). Mäntylä fornece também uma implementação, em "C", dessa estrutura.

3.3.4 - Vértice-Aresta ("Vertex-Edge") e Face-Aresta ("Face-Edge").

Weiler, em [WEI_85], apresenta três novas estruturas de dados baseadas em arestas, denominadas **"aresta-alada modificada"**, **"Vertex-Edge"** (ou vértice aresta) (V-E) e **"Face-Edge"** (ou face aresta) (F-E). O propósito das estruturas discutidas no artigo é a representação das informações topológicas com generalidade suficiente de maneira que uma grande variedade de superfícies e curvas geométricas possa ser utilizada.

A estrutura **aresta-alada modificada** mantém as informações sobre a aresta em um único nó, enquanto as outras duas dividem as informações relacionadas a cada aresta em duas partes, de forma semelhante à **semi-aresta**. A **aresta-alada modificada** é semelhante à **Aresta-Alada Completa** (descrita no item 3.3.1.4), diferenciando-se apenas pelo fato de armazenar as informações de adjacência com um campo a mais que as identifica com cada um dos vértices

finais da aresta (isto é, representa explicitamente os elementos adjacentes para cada uma das extremidades da aresta).

As estruturas Face-Aresta (F-E) e Vértice-Aresta (V-A) são idênticas no seu formato de armazenamento, mas diferem na interpretação semântica [WEI_85]. Também pode-se observar que a F-E e V-E armazenam um conjunto de informações muito semelhante ao da semi-aresta, sendo que a estrutura Vértice-Aresta é a mais semelhante à "Semi-Aresta" em termos de interpretação semântica.

A V-E divide as informações de adjacência das arestas em dois componentes (elementos do tipo "semi-aresta"), cada um relacionado a um dos vértices finais da aresta. A implementação desta estrutura, em Pascal, e sua visão hierárquica são apresentadas na caixa 3.1 e na figura 3.11, respectivamente.

```

Type
  vptr = ^vértice; ehptr = ^semi_aresta; fptr = ^face; sptr = ^casca;
  lptr = ^ciclo;

casca = record
  sin, sout: sptr;
  lista_face: fptr;
end;

lista_semi_aresta: ehptr;

face = record
  próxima, última: fptr;
  cicl: lptr
end;

ciclo = record
  próximo: lptr;
  case vértice: boolean of
    true: vert: vptr;
                                false:
  end;

semi_aresta = record
  vert: vptr;
  cwe, ccwe: ehptr;
  face: fptr;
  outra_semiarest: ehptr;
end;

```

Caixa 3.1- Detalhes de implementação da estrutura Vértice-Aresta.

Figura 3.11:-Visão hierárquica da Vértice-Aresta [WEI_85].

Pode-se ver que a vértice-aresta possui quatro nós, associados aos elementos casca, face, ciclo e semi-aresta ("edghealf"). O Elemento casca ("shell") representa um conjunto de faces conectadas e limitadas que formam um sólido ou uma cavidade, e permite a representação de objetos como o da figura 3.12.

Figura 3.12:- Objeto com cavidades [WEI_85].

A diferença entre a interpretação semântica da "Vértice-Aresta" e da "Face-Aresta", é que esta última armazena as informações de adjacência das arestas dividindo-as em dois componentes, cada um relacionado a um dos lados da aresta. A implementação e visão hierárquica da F-E é idêntica a da V-E.

As principais vantagens mencionadas para essas estruturas são o fato de poderem suportar uma grande variedade de curvas e superfícies geométricas (inclusive superfícies de forma livre) e de permitirem objetos com cavidades.

3.3.5 - Aresta-Híbrida ("Hybrid Edge").

Essa estrutura de dados também é derivada da Aresta-Alada e pode-se dizer que ela é resultado da união das estruturas Aresta-Alada e Semi-Aresta. Em [KAL_89] encontramos a proposta dessa estrutura, uma análise de suas vantagens e uma implementação.

A principal diferença e (vantagem) desta estrutura em relação as outras descritas é que ela suporta a representação e manipulação de linhas e polígonos, além de sólidos. É importante frisar que, apesar desta característica, a Aresta-Híbrida também não permite descrição de sólidos topologicamente inválidos [KAL_89, MAN_88] (assim como a aresta-alada e suas derivadas).

Uma visão hierárquica dos elementos que compõem a estrutura pode ser vista na figura 3.13, e a estrutura é apresentada na figura 3.14.

Figura 3.13:-Visão hierárquica da estrutura Aresta-Híbrida [KAL_89].

Figura 3.14:-Estrutura Aresta-Híbrida [KAL_89].

3.3.6 - Alça-Aresta ("Handle-Edge").

Ao contrário das estruturas de dados discutidas anteriormente, que são variações da Aresta-Alada, a Alça-Aresta, ou "Handle-Edge", [TAV_92] é uma extensão da estrutura semi-aresta

usada por Mäntylä, sendo manipulada pelos "Operadores de Morse", ao invés dos usuais Operadores de Euler.

Figura 3.15- Estrutura Alça-Aresta [TAV_92].

Os Operadores de Morse desempenham o mesmo papel que os operadores de Euler, manipulando a estrutura de dados B-Rep de maneira que estas continuem topologicamente válidas. Eles são mais simples que os de Euler; e todo Operador de Euler pode ser escrito como uma composição de operadores de Morse. É importante observar que a Alça-Aresta pode representar objetos que não correspondem a sólidos tridimensionais válidos (ou manufaturáveis), assim não se restringe à representação de variedades de dimensão 2. Esta estrutura foi utilizada

para visualizar superfícies matemáticas complexas imersas em dimensão maior que três, que apesar de não apresentarem as propriedades de um sólido em R^3 , podem representar "sólidos válidos" em dimensões maiores que 3 [TAV_92].

3.4 - Procedimentos Topológicos e Geométricos.

Procedimentos que implementam os Operadores (Topológicos) de Euler são indispensáveis para os modeladores B-Rep, porque são os responsáveis pela manipulação direta da estrutura de dados, além de servirem de base para a definição de procedimentos de níveis mais altos, como por exemplo os procedimentos que implementam operações booleanas e operações de modificação local [MAN_84]. No modelador **GWB** [MAN_88], esses procedimentos pertencem à segunda camada mais interna do modelador (logo acima da estrutura de dados B-Rep, figura 3.1) e, juntamente com uma modificação da **semi-aresta** foram implementados no protótipo desenvolvido.

No **GWB**, os procedimentos que implementam os operadores de Euler são agrupados em procedimentos de alto nível e baixo nível. Os procedimentos baixo nível manipulam diretamente os ponteiros da estrutura de dados, enquanto os de alto nível têm seus operandos parametrizados em termos de identificadores de faces, arestas e vértices. Uma descrição dos Operadores de Euler utilizados no **GWB** é apresentada a seguir. Todos os operadores são executados sobre sólidos:

MEV- make edge, vertex. Cria uma aresta e um vértice.

MEF- make edge, face. Cria uma aresta e uma face.

MVFS-make vertex, face, solid. Cria um sólido com um vértice e uma face (ou seja, cria o esqueleto do sólido).

KEMR-kill edge, make ring. Remove uma aresta e cria um anel (ciclo interno a face).

KFMRH-kill face, make ring, hole. Remove uma face e cria um anel e um buraco.

KEV-kill edge, vertex. Remove uma aresta e um vértice.

KEF-kill edge, face. Remove uma aresta e uma face.

KVFS-kill vertex, face, solid. Remove um vértice, uma face, e o sólido.

MEKR-make edge, kill ring. Cria uma aresta e remove um anel.

MFKRH-make face, kill ring, hole. Cria uma face e remove um anel e um buraco.

Um exemplo do funcionamento dos **Operadores de Euler** é mostrado a seguir, onde uma caixa retangular é construída aplicando-se uma seqüência desses operadores. A figura 3.16 mostra o estado do objeto e do modelo planar correspondente, a cada passo.

No primeiro passo aplica-se o operador MVFS, que cria o esqueleto do objeto (figura 3.16 (a)). No segundo passo aplica-se o operador MEV três vezes, resultando na figura 3.16 (b) (foram criadas três arestas e três vértices). No próximo passo aplica-se o operador MEF e é obtida a primeira face da caixa (figura 3.16 (c)). A seguir o operador MEV é aplicado a cada vértice, o que resulta na figura 3.15 (d). Por último aplica-se quatro vezes o operador MEF, formando a caixa retangular (figura 3.16 (e)).

Figura 3.16:-Construção de um modelo planar, usando Operadores de Euler [MAN_88].

Denomina-se procedimentos geométricos, ou algoritmos geométricos [MAN_88], os algoritmos responsáveis pela manipulação das informações geométricas contidas no modelo B-Rep. Esses procedimentos, juntamente com os que implementam os operadores de Euler, formam a segunda camada mais básica do modelador GWB e também são indispensáveis a um modelador B-Rep (figura 3.1).

Os procedimentos geométricos podem ser divididos em algoritmos que detectam se uma entidade (superfície, curva ou ponto) está contida em outra, e algoritmos que calculam a intersecção entre duas entidades geométricas. Exemplos de procedimentos geométricos seriam, encontrar a sobreposição entre duas faces, calcular a intersecção entre uma face e uma aresta e calcular a intersecção entre duas faces. Esses procedimentos não foram implementados no protótipo apresentado nos Capítulos 4 e 5.

3.5- Observações sobre a Interface.

A interface é a camada mais externa do GWB. É importante que a interface de um modelador ofereça bons mecanismos para a construção e edição de modelos, principalmente devido à já mencionada dificuldade de se descrever diretamente modelos por fronteira (válidos) [VAR_84, BOY_82, CHI_83, HIL_82]. Em geral, os sistemas B-Rep devem apresentar, a nível de interface, os seguintes mecanismos:

1- Linguagens de Descrição, que podem por exemplo ser baseadas na técnica CSG. Quando um modelador B-Rep utiliza um mecanismo como esse, é necessário implementar algoritmos de conversão entre representações CSG e B-Rep.

2- Definição de objetos tridimensionais a partir da criação de modelos bidimensionais, seguida de operação de varredura ("sweeping").

3- Definição de objetos tridimensionais, a partir da realização de operações de varredura sobre primitivas tridimensionais pré-definidas.

4- Operações de modificação local, que são operações que modificam a forma do objeto, construídas com base nos operadores de Euler [VAR_84] e operações de modificação global.

5- Mecanismos que desfçam operações errôneas ou indesejadas que tenham sido realizadas sobre os objetos. Esses mecanismos são necessários porque é comum, durante o processo de modelagem, que o usuário realize operações geometricamente inválidas ou indesejadas.

3.6- Exemplos de Modeladores de Sólidos.

O número de modeladores de sólidos desenvolvidos e a pesquisa nesta área tem crescido cada vez mais, devido à necessidade de sistemas de modelagem mais abrangentes e precisos, para a aplicação em diversos campos. A seguir descreve-se alguns dos principais modeladores discutidos na literatura. Além do GWB, são descritos brevemente outros sistemas bastante discutidos na literatura como os modeladores da família BUILD-X . Essa família de modeladores é especialmente importante, pois serviu de base para muitas pesquisas na área de modelagem por fronteira [JAR_85, HIL_82]. Também são apresentados o GMSolid [MOR_85, BOY_82] e o GeneSys, um sistema desenvolvido no Brasil [FIS_91].

3.6.1.GMSolid.

Este modelador foi desenvolvido pela General Motors com o propósito de aumentar o uso do computador no auxílio de projetos ("design"), análise e manufatura de peças para automóveis e ferramentas. O GMSolid é um modelador híbrido B-Rep-CSG, isto é, possui as duas formas de representação [BOY_82].

No GMSolid, a maneira mais básica de descrever um sólido é através da criação de primitivas (blocos, cilindros, cones e esferas) e a combinação destas através de operações

booleanas de conjunto: **união, intersecção e diferença**. Quando uma operação de conjunto é realizada, são criadas duas representações para o sólido resultante a **CSG** e a **B-Rep**. Esta última contém as informações topológicas de maneira explícita. A Representação por fronteira pode ser gerada diretamente pelo usuário, ou indiretamente através da conversão da **CSG** para **B-Rep** por um subsistema chamado **"Avaliador de Fronteira"**.

As superfícies suportadas pelo **GMSolid** são as **quádricas**, que podem ser representadas por equações da forma $F(x, y, z)=0$, onde F é um polinômio de segundo grau em x, y, z . Este modelador, como a maioria dos modeladores de sólidos, possui várias operações de descrição à nível de interface além das operações booleanas; como por exemplo diferentes operações de corte ("cut").

3.6.2.Família **BUILD-X**.

Essa família de modeladores, possui diversas versões (**BUILD-I, BUILD-II, BUILD-III,...**), que foram desenvolvidos por um grupo na Universidade de Cambridge, sendo que o **BUILD-II** foi concebido como uma ferramenta para experimentar técnicas e algoritmos de modelagem geométrica. Vários sistemas **B-Rep** comerciais foram desenvolvidos baseados nesta família de modeladores.

Os **BUILDs** são sistemas de modelagem que utilizam a representação por fronteira e possuem um conjunto de 10 Operadores de Euler para manipular a estrutura de dados **B-Rep**. Esses modeladores apresentam diversas operações de descrição de sólidos, **varredura ("sweeping")** translacional e rotacional de objetos bi-dimensionais e tridimensionais;

operações de modificação local (que alteram a topologia e geometria dos objetos sólidos) e operações booleanas de conjunto.

Após a realização de uma operação de modificação local, o BUILD-III sempre verifica a consistência geométrica do sólido resultante. Entre as operações de modificação local apresentadas por estes modeladores estão, "Tweaking" e "Chamfering" (Chanframento), ilustradas nas figuras abaixo (3.17, 3.18) [JAR_85]. As operações citadas acima são baseadas nos Operadores de Euler.

Figura 3.17:- Operação de "tweaking" [JAR_85].

Figura 3.18:- Operação de chanframento [JAR_85].

3.6.3-"Geometric Workbench" (GWB).

Este modelador já foi comentado no início do capítulo e baseado nele foram descritos os principais componentes de um sistema de modelagem de sólidos [MAN_88]. O GWB é um modelador B-Rep que possui uma arquitetura de software organizada em três camadas principais (v. figura 3.4). Ele utiliza a estrutura de dados semi-aresta ("halfedge") para armazenar a representação dos sólidos e um conjunto de 10 Operadores de Euler para manipulá-la. Esses operadores estão organizados em duas classes: **Operadores de Alto Nível**, onde os parâmetros são identificadores de entidades topológicas (vértice, aresta, sólido,...), e **Operadores de Baixo Nível**, cujos parâmetros são ponteiros para elementos da estrutura de dados.

O modelador proposto por Mäntylä possui diversas operações para descrição de objetos: **varredura translacional e rotacional**, **operações de conjunto**, **de colagem ("glueing")**, e **de particionamento ("splitting")**, entre outras, todas construídas sobre os operadores de Euler. O modelador possui um mecanismo interessante para desfazer operações errôneas e indesejáveis,

pois é muito fácil o usuário realizar uma operação indesejada e este tipo de mecanismo impede que ele perca a cena ou um sólido complicado e tenha que recomeçar a descrição do objeto novamente.

Por último, observa-se que esse sistema apresenta uma linguagem e um interpretador para o usuário descrever os sólidos na forma de "arquivos". O GWB foi desenvolvido em "C". Uma tabela comparativa dos modeladores descritos pode ser encontrada em [MAG_93].

3.6.4-GeneSys.

O último modelador a ser comentado, o GeneSys, foi desenvolvido na Pontifícia Universidade Católica do Rio de Janeiro [FIS_91]. É um sistema recente e possui uma arquitetura híbrida contendo várias formas de representação de maneira integrada, sendo que as principais são a B-Rep e uma modificação da CSG [FIS_91]. O desenvolvimento desse modelador foi iniciado em 1988, e teve como ponto de partida a implementação de um protótipo que demonstrou a viabilidade da estrutura Aresta-Alada proposta por Baumgart e também que os Operadores de Euler são indispensáveis para manipular de forma consistente essa estrutura.

O núcleo de representação do GeneSys é constituído basicamente de uma representação B-Rep e a CSG modificada, chamada CSG_{hor} ("High-Level Constructive Solid Operation Geometry"). Existem outras formas de representação no GeneSys, porém estas duas são as principais. A representação B-Rep do sistema possui uma alteração em relação às representações tradicionais como a "Aresta-Alada", que consiste na inclusão três entidades topológicas

("World", "Object" e "Solid"). A representação CSG_{HOR} caracteriza-se por permitir em uma estrutura a multiplicidade de versões de um mesmo sólido, e também permitir que os nós da árvores CSG sejam primitivas B-Rep.

O GeneSys possui uma arquitetura de software baseada em dois princípios: **Modularidade e Abstração** [FIS_91], sendo composto de três subsistemas e uma base de dados (figura 3.19).

Figura 3.19:- Arquitetura do modelador GeneSys [FIS_91].

Os dois primeiros subsistemas "UDI" (User Interface & Application) e "MIA" (Modeling Interface & Application) são responsáveis pela interface gráfica, captação de dados e aplicações. A camada mais inferior é composta pelo subsistema "GTM" ("Geometry e Topology Modeling") e pela base de dados. O GTM possui um papel importante no GeneSys, sendo responsável por oferecer métodos para a criação e alteração consistente das representações contidas na base de dados. Todos os subsistemas são decompostos em módulos com funcionalidades independentes.

A representação B-Rep deste sistema nunca é alterada diretamente pelo usuário, que faz as alterações por intermédio da CSG_{HOP}. A visão hierárquica da estrutura B-Rep utilizada é ilustrada a seguir (figura 3.20)

Figura 3.20:- Visão hierárquica da estrutura de dados B-Rep do GeneSys [FIS_91].

O sistema GeneSys possui várias características interessantes: a **arquitetura híbrida**, uma preocupação com modularidade e encapsulamento de informações e uma representação CSG modificada onde as primitivas da árvore CSG não são representadas apenas por semi-espacos ("half-spaces"), mas podem ser instâncias consistentes de uma representação B-Rep.

3.7-Considerações Finais.

Existem várias arquiteturas de software possíveis para um modelador de sólidos. O ideal é que o modelador seja híbrido, ou seja, admita mais de uma representação. Entretanto, a construção de um sistema desse tipo ainda apresenta problemas sérios e não resolvidos [SAK_90], entre eles o de manter consistentes as diferentes representações que coexistem no sistema. No caso de um modelador que utilize a técnica B-Rep, a decisão sobre que estrutura de dados usar é importante, pois influencia o conjunto de superfícies que o modelador admite, e portanto o conjunto de sólidos que pode ser modelado.

Deve-se lembrar que os modelos B-Rep são difíceis de serem construídos diretamente, de forma que os modeladores devem apresentar recursos para a definição de objetos a nível de interface com o usuário [KIM_84]. Neste capítulo, foram discutidas as alternativas de arquitetura dos modeladores de sólidos, possíveis estruturas de representação B-Rep, os operadores topológicos e geométricos e a alguns aspectos sobre a interface. Também foi apresentada uma breve descrição de alguns modeladores encontrados na literatura, cujo estudo permite extrair os requisitos e a funcionalidade para o SMS.

O protótipo apresentado nos Capítulos 4 e 5 concentrou-se na implementação de uma estrutura de representação, dos operadores topológicos que constituem o Gerenciador de Representações do modelador. Foi implementada também uma interface simples, baseada na técnica de varredura ("sweeping"), e algumas operações de alto nível, como colagem ("glueing").

CAPÍTULO 4

PROJETO DO PROTÓTIPO DO SMS.

4.1-Considerações Iniciais.

Neste capítulo, são apresentados inicialmente os conceitos básicos do **Paradigma da Orientação a Objetos** (seção 4.2), juntamente com a abordagem para projeto orientado a objetos utilizada no desenvolvimento do protótipo do SMS (seção 4.3). Na seção 4.4 é apresentada uma análise dos requisitos e funcionalidade do SMS. A seguir, nas seções 4.5 a 4.8 são apresentados os resultados do projeto orientado a objetos feito para o protótipo. Algumas considerações finais são feitas na seção 4.9.

4.2-Conceitos Básicos do Paradigma da Orientação a Objetos.

A engenharia de software, durante esses anos, evoluiu em busca de uma filosofia que eliminasse os problemas encontrados no desenvolvimento e manutenção de software, dando origem a várias técnicas e métodos. Como um dos mais recentes candidatos a solução apareceu a **orientação a objetos**. Este termo tem sido utilizado em conexão com "**ambientes de software**", "**arquiteturas**", **interfaces com usuários**, **métodos para especificação**, **análise e projeto de sistemas computacionais e linguagens de programação**.

A idéia básica por de trás da orientação a objetos é simples, e está relacionada com a nossa percepção do mundo como **uma coleção de objetos que interagem entre si**. O paradigma

da orientação a objetos permite que objetos do mundo real sejam mapeados em objetos no computador, assim os sistemas podem ser representados como um conjunto de objetos interagindo. Portanto, desenvolver um sistema orientado a objetos é mais que uma escolha de estilo de programação.

É importante distinguir entre projeto ("Design") e programação orientada a objetos e ter em mente que são dois conceitos relacionados mas não necessariamente dependentes. Um projeto orientado a objetos pode ser implementado (programado) sem usar a programação orientada a objetos. O ideal, é claro, é que a orientação a objetos seja aplicada tanto ao projeto quanto à programação.

O paradigma da orientação a objetos possui alguns conceitos básicos que são descritos a seguir. No estudo desses conceitos, encontrou-se alguma dificuldade inicial devido à falta de uma nomenclatura uniforme e à existência de definições algumas vezes conflitantes [WEG_90, GIB_90]. Essa dificuldade foi superada, pela adoção da nomenclatura e conceitos definidos por [BOO_91, WEG_90, HEN_92] que mostraram-se muito semelhantes e fáceis de serem compreendidos.

O conceito de **objeto** é o primeiro e o mais básico. Um objeto, no mundo físico, é tipicamente um produtor ou consumidor de itens de informações [PRE_87]. Portanto pode ser uma máquina, comandos, arquivos, sinais, "strings" , pessoas, etc...

Quando mapeia-se um objeto do mundo físico em software, este é constituído de uma estrutura privada e de processos chamados métodos. Assim, um objeto em software pode ser definido como na caixa a seguir [HEN_92].

Objeto = Dados + Funcionalidade + Encapsulamento.

Onde encapsulamento é um conceito relacionado com modularização de programas e troca de dados através de interfaces (Abstração), e os métodos definem a funcionalidade.

Caixa 4.1- Definição de objeto.

Objetos interagem entre si através da troca de mensagens que ativam os métodos associados a eles. Todo objeto pode possuir métodos e atributos, que são conceitos que aparecem tanto na programação como no projeto. Muitos objetos do mundo físico possuem características similares e podem ser agrupados em um conjunto chamado **Classe de Objetos**. Portanto define-se uma classe de objetos como sendo um conjunto de objetos que possuem atributos e/ou métodos semelhantes.

Métodos são procedimentos definidos e declarados, que atuam sobre um objeto ou uma **classe de objetos**. **Atributos** são variáveis que definem o estado de um objeto, ou seja, são entidades que caracterizam os objetos. Cada objeto possui seu próprio conjunto de atributos.

Juntamente com os conceitos de classe, objeto, atributos e métodos, os conceitos de herança e polimorfismo são essenciais para o paradigma da orientação a objetos. **Herança** é a habilidade de um objeto derivar seus atributos (dados) e métodos (funcionalidade) automaticamente de outro objeto. Tanto objetos como classes possuem esta habilidade. **Polimorfismo** é a possibilidade de se ter diferentes tipos de objetos respondendo de maneira

diferente a uma mesma mensagem. Na prática, corresponde à existência de diferentes métodos com o mesmo nome.

Existem atualmente várias linguagens orientadas a objetos, como Smalltalk, C++, Objective C, Objective Pascal. Na literatura há uma discussão polêmica a respeito do que é necessário para que uma linguagem seja considerada orientada a objetos [TAK_90, GIB_90]. Como exemplo é apresentada a classificação proposta por Peter Wegner [WEG_90], que define três classes de linguagens (caixa 4.2).

-Linguagens Baseadas em Objetos: São as que suportam o conceito de objeto como um aspecto da linguagem. Ex: ADA.

-Linguagens Baseadas em Classes: São linguagens baseadas em objetos onde todo objeto deve pertencer a uma classe. Ex: CLU.

-Linguagens Orientadas a Objetos: Linguagens baseadas em classes que possuem mecanismos de herança que permitem a definição de hierarquias de classes. Ex: C++, Smalltalk.

Caixa 4.2- Classificação das linguagens segundo Wegner.

Com a evolução e a disseminação das linguagens Smalltalk e ADA nos anos 80, o interesse por métodos de projeto usando a visão de objetos cresceu muito, e hoje diferentes métodos de projeto e análise orientados a objetos [YOU_90, RUM_88, BOO_91] são empregados nas mais diversas áreas da computação.

Métodos de projeto orientado a objetos, assim como outros métodos orientados à informação, criam uma representação do mundo real e mapeiam essa representação em software. O objetivo, é construir um sistema que possua três importantes propriedades: abstração, encapsulamento de informações e modularidade [PRE_87, TAK_90]. Todo método de projeto

esforça-se para que o software apresente essas propriedades, mas segundo seus defensores, os métodos de projeto orientado a objetos atingem esses objetivos de maneira simples e correta.

4.2.1 - Aplicações em Computação Gráfica.

A visão orientada a objetos está sendo utilizada em quase todas as áreas da computação, em aplicações que vão desde **Banco de Dados à Computação Gráfica**. Na computação gráfica, Wisskirchen [WIS_90] discute gráficos orientados a objetos, com ênfase em interfaces gráficas. Em [BLA_90] encontra-se uma coleção de artigos descrevendo interfaces e sistemas gráficos orientados a objetos.

Já existem modeladores de sólidos projetados utilizando a orientação a objetos, como o modelador descrito em [GRA_86]. Apesar disto, existe ainda muito a ser estudado quanto à aplicação desse paradigma na computação gráfica, principalmente na área de modelagem de sólidos.

4.3 - Abordagem para o Projeto Orientado a Objetos.

Neste item é apresentado um resumo da metodologia definida por Booch [BOO_91], que foi estudada e adotada (parcialmente) para desenvolver o projeto orientado a objetos do primeiro protótipo do SMS. Booch, em [BOO_91], estende e modifica seu método descrito em [BOO_86], tornando-o mais completo e abrangente. Assim, em sua nova versão o método passa a explorar

os conceitos de classes e relações de herança, não explorados na versão anterior, e também são acrescentados vários tipos de relações possíveis entre as classes e os objetos que compõem um sistema.

O método de Booch é complexo e possui vários **diagramas e gabaritos** para descrever um projeto orientado a objetos, como os **diagramas de classes**, que descrevem as classes existentes no sistema e as relações entre elas; os **diagramas de objetos** que descrevem os objetos e as relações entre eles (indicando a troca de mensagens entre objetos); os **diagramas de módulos** que mostram como as classes estão agrupadas em módulos e como estes se relacionam entre si; os **diagramas de temporização** que descrevem a relação entre os métodos acionados, objetos e tempo (podem mostrar o tempo de vida de um objeto dentro do sistema).

Durante o desenvolvimento do modelo orientado a objetos para o Gerenciador de Representações do SMS foram utilizados, por motivo de simplicidade e objetividade, apenas os principais diagramas, que seriam os diagramas de objetos e de classes que compõem o sistema, juntamente com os gabaritos que descrevem os métodos e atributos associados a eles.

Durante a fase de análise, devido à dificuldade inerente para a definição das classes, objetos, atributos e métodos relevantes para o sistema, Booch propõe como alternativa, entre outras, o uso da análise estruturada em uma primeira etapa, para então derivar os objetos, classes, e métodos a partir dos Diagramas de Fluxos de Dados (DFDs) [GAN_86] obtidos.

Como a notação de Booch mostrou-se complexa e de difícil compreensão, na última fase do projeto ela foi substituída pela proposta por Mullin [MUL_89], que mostrou-se mais prática e compreensível. Esta notação é descrita na caixa 4.3.

.Diagrama de Hierarquia da Classes- Ilustra quais as classes existentes no sistema e as relações de herança entre elas.

.Diagrama de Dependência de Objetos- Em um sistema, a definição de um objeto pode utilizar outro objeto referenciado através de um campo. Essas relações de dependência nas definições de objetos são ilustradas neste diagrama.

.Diagrama de fluxo de entrada e saída de mensagens- Indica para quais objetos um determinado objeto envia mensagens, e de quais recebe. Os objetos são representados por círculos e o envio de mensagens por setas.

.Lista de Métodos- Ilustra todos os métodos definidos no sistema, divididos por classe, e descreve seus parâmetros, tipo e funcionalidade.

Obs- Por motivos de clareza e praticidade, os diagramas de fluxo de entrada e saída de mensagens sugerido por Mullin foram substituídos por uma tabela, descrita na seção 4.7.

Caixa 4.3-Notação utilizada no projeto orientado a objetos do protótipo do SMS.

Na análise e especificação do protótipo do SMS, foram seguidos os passos descritos por Booch [BOO_91] para se decompor um sistema em classes e objetos com atributos e métodos associados. O primeiro passo foi fazer uma análise detalhada da funcionalidade e componentes do sistema desejado. O resultado dessa análise é apresentado na seção 4.4 a seguir.

Em um segundo passo, foram encontradas as classes, atributos e métodos apenas do primeiro protótipo do SMS, a ser desenvolvido nesse projeto, que engloba o gerenciador de representações B-Rep e uma interface simples. Esse passo foi dividido em quatro: (i) identificação da hierarquia das classes do sistema, (ii) identificação das relações de

dependência entre as classes, (iii) a listagem dos métodos associados à cada classe, (iv) definição do fluxo de mensagens entre as classes de objetos do sistema. Desta maneira, na primeira fase (especificação dos requisitos e funcionalidade) foi considerado o SMS como um todo, mas o projeto orientado a objetos restringiu-se à funcionalidade desejada para o protótipo a ser desenvolvido. É importante observar que apesar de alguns métodos e trocas de mensagens terem sido definidos ainda na fase de projeto, a lista final dos métodos e o fluxo de mensagens só pôde ser definido completamente durante a implementação do sistema.

4.4 - Análise dos Requisitos e Funcionalidade do SMS.

A especificação dos requisitos e funcionalidade do SMS, primeiro passo da fase de análise, foi desenvolvida em duas etapas: descrição informal e análise estruturada do SMS. Antes de descrever estas fases, os componentes do SMS são apresentados para situar o protótipo que foi desenvolvido com relação ao sistema de modelagem de sólidos como um todo, e auxiliar o estudo dos requisitos e funcionalidade do modelador.

Todos os sistemas de modelagem de sólidos apresentam alguns componentes básicos, como uma estrutura de dados para armazenar os esquemas de representação, um conjunto de algoritmos básicos para manipulação dessa estrutura e mecanismos de interface com o usuário que implementam diferentes técnicas de modelagem. Com base no estudo dos requisitos e funcionalidade apresentados pelos modeladores híbridos (B-Rep-CSG), foram definidos os componentes do SMS, e foi feita uma descrição informal que refletisse a funcionalidade e os

principais requisitos desejados. Os componentes e a descrição informal são apresentados a seguir.

4.4.1 - Componentes.

O SMS deverá ser um modelador híbrido B-Rep-CSG, que possui a representação B-Rep como principal esquema de representação interna. Os componentes do modelador são ilustrados na figura 4.1.

Pode-se ver que o sistema de modelagem é composto por três subsistemas:

Um **Subsistema de Interface**, que fornece aos usuários uma (ou mais) linguagens de descrição de objetos baseadas nos conceitos de modelagem disponíveis no modelador, bem como um conjunto de operações que possam ser executadas sobre os objetos e cenas.

Um **Subsistema de Modelagem**, que traduz as descrições de objetos para a representação interna. Deve fornecer interfaces para comunicação com outros sistemas (algoritmos, aplicações, outros modeladores), bem como incluir funções para armazenagem de cenas e objetos em base de dados permanentes. Deve também admitir múltiplas representações internas de objetos fornecendo os procedimentos de conversão necessários.

Um **Subsistema de Representação**, responsável pela manutenção e gerenciamento do conjunto de representações internas (estruturas de dados) admitidas pelo sistema. O trabalho apresentado concentrou-se no desenvolvimento do subsistema de gerenciamento de representações B-Rep. Porém, algumas operações de modelagem de alto nível para descrição de objetos, a nível de interface, foram implementadas para que o gerenciador de representações pudesse ser testado. Dessa maneira, foi desenvolvido um **primeiro protótipo do SMS**.

Figura 4.1:- Componentes do SMS.

4.4.2 - Descrição Informal.

O SMS permite a criação, edição e acesso a representações de objetos sólidos e cenas compostas por um ou mais objetos. Além dos processos ativados pelos usuários durante a fase de modelagem interativa, outros processos que acessam essas representações são programas de visualização (que podem ser considerados parte do modelador) e programas aplicativos (externos

ao modelador), que as utilizam para responder a questões práticas referentes aos objetos modelados (como cálculo de volume, área, etc.).

O SMS é composto de três subsistemas que colaboram entre si, mas operam de forma independente, conforme ilustrado nas figuras 4.1 e 3.1 (Capítulo 3).

Os usuários interagem com o sistema para descrever e editar objetos através de uma interface gráfica interativa que constitui o subsistema de interface do modelador. Os comandos recebidos pela interface são traduzidos em solicitações para o subsistema de modelagem, que implementa as técnicas de modelagem disponíveis no sistema e é responsável pela execução das operações de modelagem. Para tal, esse subsistema interage com o subsistema de representação, que acessa e altera quando necessário as representações de objetos mantidas pelo sistema. Os programas aplicativos interagem diretamente com o subsistema de modelagem, emitindo solicitações através de uma linguagem de comandos.

Um usuário pode, a nível de interface, descrever objetos criando instâncias de primitivas sólidas pré-definidas (blocos, pirâmides, esferas, cilindros, cones, tori, ...) e combinando-as através de operações booleanas (intersecção, união e diferença) e transformações geométricas (rotação, translação e escala). Durante essa fase, o processo de modelagem pode ocorrer sobre a representação CSG ou B-Rep dos objetos. A interface permite ainda a descrição de objetos através da execução de operações de varredura ("sweeping") rotacional ou translacional sobre primitivas bi- ou tri-dimensionais.

Uma vez obtida uma descrição inicial para o objeto desejado, o modelo CSG ou de varredura ("sweep") é convertido para um modelo B-Rep, e um segundo conjunto de operações básicas estão disponíveis para efetuar operações de edição (globais e locais) sobre tais modelos.

Entre estas estão incluídas operações booleanas, operações de modificação local (como extrusão de uma face, chanframento, ...) e de seccionamento. Tais operações, ativadas pela interface, serão executadas pelo módulo B-Rep do subsistema de modelagem do SMS.

O subsistema de interface deve também fornecer uma (ou mais) visões do objeto sendo modelado, e para tal interage com o módulo de visualização, que faz parte do subsistema de modelagem.

Todas as operações são ativadas pela seleção de menus e ícones através do "mouse" e, quando conveniente, por linhas de comando textuais. Dessa maneira, o subsistema de interface pode incluir um analisador léxico simples para checar a sintaxe dos comandos. A interface deve ainda, sempre que possível, fornecer mecanismos para desfazer operações errôneas, bem como impedir a execução de operações incorretas (esse último caso requer interação com o subsistema de modelagem). Uma vez analisado o comando, e julgado correto, a interface traduz o comando ou seleção em solicitações ao subsistema de modelagem.

O subsistema de modelagem é responsável pelo disparo das operações solicitadas pelo usuário através da interface. Para tal, este subsistema requer um interpretador de comandos ("command parser") para analisar a operação, checar sua validade (semântica) e ativar os procedimentos de modelagem associados - que poderão atuar nas representações CSG ou B-Rep, dependendo das operações e do contexto (uma vez executada uma operação de edição na B-Rep, a CSG original deve ser marcada de alguma forma, uma vez que a modificação não poderá ser refletida na CSG).

Além disso, fica a cargo deste subsistema fornecer uma interface para comunicação com outros modeladores e aplicações externas, fornecer procedimentos para a visualização dos objetos

sendo modelados, providenciar mecanismos para armazenar descrições dos sólidos em uma base de dados permanente e fornecer procedimentos para efetuar a conversão de representações CSG para B-Rep (algoritmos de avaliação de fronteira).

O subsistema gerenciador de representações é responsável pelo acesso direto às estruturas de dados CSG e B-Rep: qualquer acesso às representações de objetos exigido pelo subsistema de modelagem é feito através do gerenciador. A B-Rep constitui a principal forma de representação interna do sistema (já que "sweep" é temporário, mantida apenas durante a fase inicial de descrição dos objetos, e a CSG é secundária) e armazena, de forma separada, as informações topológicas e geométricas sobre os objetos (além de atributos adicionais, como material, etc.), ou seja, dados sobre como as faces, ciclos de arestas, vértices e buracos que compõem um sólido estão conectados; bem como as coordenadas dos vértices.

A representação só pode ser manipulada através de algoritmos básicos de acesso que garantam a integridade topológica dos objetos. Estes algoritmos implementam os Operadores de Euler e alguns procedimentos geométricos básicos. Os operadores de Euler garantem a integridade topológica da representação B-Rep, e fornecem a base para a implementação dos mecanismos para desfazer operações errôneas ou indesejadas: a estrutura de dados armazena a sequência de operadores de Euler utilizados e seus parâmetros, e para desfazer uma operação basta percorrer a estrutura de dados ativando os operadores inversos.

Os procedimentos geométricos percorrem a estrutura de dados acessando e manipulando apenas as informações geométricas, e são usados para responder questões geométricas e garantir a validade dos objetos depois da execução de uma operação local. Estes algoritmos podem ser agrupados em duas classes: os que detectam se duas entidades topológicas (face, aresta ou

Figura 4.3:- Expansão do DFD geral

A figura 4.3 ilustra os principais componentes e funções do SMS. Entre os componentes (depósitos de dados) encontram-se cena, representação (ou sólido) B-Rep e CSG e "LOG" (arquivo que armazena todas as operações realizadas sobre os sólidos, para que uma operação indesejada possa ser desfeita). Como algumas das principais funções tem-se: mostrar a cena que está sendo editada; validar geometricamente uma cena e realizar operações de descrição de sólidos e manipulação de sua representação. Neste diagrama, já é possível notar

candidatos a objetos, como cena, sólido B-Rep e sólido CSG. Além disso, deve-se notar que o fluxo de dados "dados sobre cena" consiste do nome dos sólidos que compõem a cena e dados sobre os componentes desses sólidos (faces, arestas, curvas,...).

Figura 4.4:-Expansão do processo 2.

O DFD na figura 4.4, apresenta a expansão do processo "mostrar cena", composto de três processos principais que podem ainda se desdobrar em vários sub-passos, podendo dar origem a um subsistema de visualização. A expansão do processo 2 apresenta um fluxo chamado **cena iluminada** que consiste de outro candidato a objeto do sistema, sendo uma especialização de cena. Observa-se que os fluxos de dados estão dando origem aos candidatos a objetos, como Booch descreve [BOO_91].

O diagrama da figura 4.5 apresenta a expansão do processo 4 "manipular cenas", que consiste do processo principal do modelador e engloba o subsistema de representações e parte dos subsistemas de modelagem e de interface.

Eles são responsáveis por permitir ao usuário manipular os modelos (objetos) através de operações de alto nível, que são decompostas em operações de manipulação das estruturas de dados existentes. Essas operações, por sua vez, buscam informações na estrutura de dados ou alteram as informações nela contida, e são candidatas a métodos (como por exemplo buscar uma face, criar uma aresta, remover um vértice, ...). Ainda nesse diagrama os candidatos a objetos sólido, sólido B-Rep, faces, superfícies, curvas, arestas, semi-arestas e vértices aparecem implicitamente no fluxo de dados "dados sobre cena".

Figura 4.5:- Expansão do processo 4

Dessa maneira, especificou-se a funcionalidade do SMS, e obteve-se sérios candidatos à classes, objeto e métodos. O projeto lógico completo, contendo as expansões de todos os principais processos, pode ser encontrado em [SCO_93a].

4.5- Hierarquia de Classes.

Neste primeiro sub-passo foram definidas 16 classes para o protótipo do SMS, a partir dos candidatos a objetos identificados durante a análise estruturada e também durante a fase de implementação do gerenciador de representações. A figura 4.6 ilustra o diagrama de hierarquia de classes do sistema.

Figura 4.6:- Diagrama de hierarquia de classes do protótipo.

Na figura, nota-se que a herança não foi intensamente utilizada nesta fase do projeto, consequência da abordagem ascendente ("bottom-up") adotada para o desenvolvimento do modelador; ao contrário da abordagem descendente ("top-down") geralmente seguida nos projetos orientados a objetos. Assim, o protótipo foi desenvolvido a partir da camada mais interna do modelador: a estrutura de dados, no topo da qual serão construídas as outras, mais externas (procedimentos de modelagem, interface,...).

No desenvolvimento desse primeiro protótipo, o que se verificou é que foram exploradas mais as relações de uso entre as classes (ver seção 4.6) do que a relação de herança. Acredita-se que a herança deverá ser utilizada mais extensivamente a medida que o modelador seja estendido para suportar outros tipos de representação de sólidos, que seriam especializações da classe

genérica sólido. Isso é indicado pelo fato de um dos poucos usos identificados de herança ocorre na definição das classes "sólido B-Rep" e "sólido CSG", que são ambas especializações da classe "sólido".

Muitas classes de objetos, como cena, sólido e sólido B-Rep, foram identificadas a partir do estudo de requisitos e funcionalidade do modelador desenvolvido na seção anterior. Entretanto, a definição das classes, objetos e métodos sofreu vários refinamentos durante a fase de projeto e implementação do protótipo. O modelo orientado a objetos inicial foi modificado várias vezes, com a especificação de novos objetos, métodos e atributos. Assim, pode-se dizer que o projeto do sistema evoluiu juntamente com a sua implementação. Essa flexibilidade e facilidade para modificação é uma das principais características do paradigma da orientação a objetos, que o torna particularmente interessante para o desenvolvimento de protótipos.

Existe uma certa subjetividade no processo de identificação de classes e objetos em um sistema orientado a objetos, e o conjunto de classes e métodos encontrado para o protótipo poderia ser diferente se a abordagem utilizada no processo fosse outra.

A abordagem utilizada para a identificação das classes resultou em classes do SMS que agrupam objetos com atributos semelhantes o que aparentemente é a solução mais "natural". Em [GRA_86] entretanto, os autores descrevem o desenvolvimento de um sistema de modelagem e utilizam a abordagem de agrupar em classes os objetos que possuem métodos semelhantes, o que poderia resultar em um projetos com um número maior de classes mas um número menor de métodos.

4.6-Dependência entre Classes.

Após a definição da hierarquia de classes, foram especificadas as relações de dependência entre objetos dentro do protótipo, isto é, quais classes de objetos são usadas (referenciadas) por outras em sua definição. As dependências entre as classes de objetos são facilmente identificadas a partir da estrutura de dados B-Rep implementada (modificação da estrutura semi-aresta ou "halfedge" (v. Capítulo 3). Essa estrutura é apresentada na figura 4.7, e está construída principalmente sobre listas duplamente encadeadas de objetos. Assim, pode-se ter uma lista de cenas, sendo que cada cena consiste de uma lista de sólidos. Sólidos, por sua vez são compostos por listas de faces, e assim por diante.

Figura 4.7:- Estrutura implementada (modificação da semi-aresta).

Figura 4.8:- Diagrama de dependências entre classes do protótipo.

O diagrama de dependências é apresentado na figura 4.8. Os círculos na figura 4.8 indicam os objetos, enquanto os retângulos arredondados indicam os elementos (campos de dados) que servem como conexão entre pares de objetos. Por exemplo, os objetos na classe **aresta** usam objetos da classe **semi-aresta** através dos campos "**semi_esq**" e "**semi_dir**" (v. figura 4.8). Isso porque uma aresta contém ponteiros para as duas semi-arestas que a compõem, conforme indicado na figura 4.7. Os objetos da classe **face**, por sua vez, usam objetos da classe **ciclo**, porque cada face possui um ponteiro para uma lista de ciclos, conforme pode ser visto na figura 4.7.

Os objetos das classes **sólido**, **cena**, **sólido B-Rep**, **face**, **aresta**, **semi-aresta**, **ciclo**, **curva**, **superfície** e **vértice** possuem uma relação de dependência com os objetos da classe **lista** e **nó** (não ilustrada de maneira explícita no diagrama). Isso ocorre porque a estrutura de dados B-Rep implementada é totalmente baseada em listas, como pode ser visto na figura 4.7. Isso faz com que esses objetos sejam sempre manipulados como nós ou elementos de listas. O diagrama, portanto, reflete a estrutura de dados utilizada.

Novas relações de dependência deverão ser estabelecidas, principalmente para as classes genéricas como **cena**, **sólido** e **vértice**, quando o modelador for estendido e novos usos forem estabelecidos para essas classes. O aparecimento de novas relações de adjacência provavelmente não alterará significativamente as dependências já estabelecidas entre as classes mais especializadas (específicas da estrutura implementada), como **face**, **aresta** e **semi-aresta**.

4.7 - Listagem de Métodos.

Foram definidos 269 métodos associados às classes, que são listados em tabelas no relatório de documentação do sistema [SCO_93a]. Nesse relatório, são fornecidas as tabelas de métodos para cada classe. As tabelas apresentam o nome de cada método, uma breve descrição de sua funcionalidade, seu tipo (Privado ou Público) e os parâmetros de entrada e saída, além de um número de identificação. Esse número pode ser utilizado para localizar a implementação do método no código do SMS. Os métodos públicos são aqueles que podem ser requisitados por objetos de outra classe, e os privados apenas por objetos da classe à qual pertence (para a qual está definido).

Todas as classes de objetos poderão ter no futuro novos métodos associados, tais como os que implementam operações de modificação local e global sobre sólidos, bem como operações geométricas e topológicas (auxiliares) para suportar esses métodos. Portanto, a definição dos métodos associados às classes é um processo contínuo durante o desenvolvimento do modelador. Acredita-se que o suporte fornecido a esse processo de desenvolvimento contínuo é uma das vantagens oferecidas pelo método de projeto e desenvolvimento orientado a objetos adotado. A distinção entre os métodos públicos e privados é mantida para auxiliar os "desenvolvedores" do sistema. A título de exemplo, as tabelas de métodos para as classes **Lista** e **Semi-Aresta** são ilustradas a seguir.

OBJETO: LISTA			
METODO	PARÂMETROS ENTRADA	PARÂMETROS SAÍDA	NUMERAÇÃO
Criar (Público) .Cria objetos do tipo LISTA	.tipo: ponteiro para um TIPO, que contém a informação dizendo de que tipo deve ser a lista a ser criada.	.ponteiro para a lista criada	1
Apagar (Público) .Destroi objetos do tipo LISTA	.li: ponteiro para a lista a ser destruída	.nenhum	2
Remover (Público) .Remove um nó de um objeto do tipo LISTA	.li: ponteiro para a lista da qual será removido o objeto .element: ponteiro para um NODE, contendo os dados do objeto a ser removido	.nenhum	3
Retirar (Público) .Retira um nó de um objeto do tipo LISTA, sem desalocar este nó, retornando um ponteiro para ele	.li: ponteiro para a LISTA da qual será removido o NO .element: ponteiro para um NODE, contendo os dados do elemento a ser retirado	.ponteiro para um NO (elemento retirado da lista)	6
Adicionar (Público) .Adiciona um nó a um objeto LISTA	.li: ponteiro para a LISTA na qual o elemento será inserido .novo_element: ponteiro para um NODE, contendo os dados do elemento a ser inserido .onde: ponteiro para um NO, indicando a posição onde o novo_element deve ser inserido. Caso seja NULL, novo_element é inserido no final da lista; caso contrário, é inserido antes de onde	.ponteiro para um NO, que contém o elemento inserido na lista	5
Procurar_lista (Público) .Procura um determinado NO de um objeto LISTA	.li: ponteiro para a LISTA na qual o elemento será pesquisado .element: ponteiro para um NODE, contendo os dados do elemento a ser pesquisado	.ponteiro para um NODE que contém os dados do elemento a ser pesquisado	4
Primeiro_elemento (Público) .Retorna o primeiro elemento de uma lista.	.aux_lista: ponteiro para uma LISTA, a lista da qual deseja-se o primeiro elemento	.ponteiro para um NO, o primeiro elemento de aux_lista	7

OBJETO: SEMI_ARESTA			
METODO	PARÂMETROS ENTRADA	PARÂMETROS SAÍDA	NUMERAÇÃO
Gerar_identificação (Privado) .Gera um número que serve de identificador único do objeto SEMI-ARESTA	.nenhum	.um ponteiro para um CODIGO	27
Criar (Público) .Cria objeto SEMI-ARESTA	.loop_pai: ponteiro para NO, que assume a estrutura de um loop (LOOP do qual a SEMI-ARESTA a ser criada fará parte) .vertf: ponteiro para NO que assume a estrutura de VERTICE (o VERTICE para o qual a aresta irá apontar) .solido_pai: ponteiro para um NO que assume a estrutura de SOLIDO-BREP (sólido ao qual a semi-aresta irá pertencer)	.ponteiro para um objeto NO (SEMI-ARESTA criada)	28
Remover (Público) .Destroi um determinado objeto SEMI-ARESTA	.lop: ponteiro para um NO que assume a estrutura de um LOOP (LOOP pai da SEMI-ARESTA a ser removida) .element: ponteiro para um NO que assume a estrutura de uma SEMI-ARESTA (a que deve ser removida)	.nenhum	29
Achar_loop (Público) .Encontra o LOOP pai ao qual uma SEMI-ARESTA pertence	.element: ponteiro para um NO que assume a estrutura de uma SEMI-ARESTA	.ponteiro para um NO (LOOP ao qual a SEMI-ARESTA pertence)	30
Aresta_pai (Público) .Encontra a ARESTA a qual um objeto SEMI-ARESTA pertence	.element: ponteiro para um NO que assume a estrutura de uma SEMI-ARESTA	.ponteiro para um NO (a ARESTA a qual a SEMI-ARESTA element pertence)	31
Identificador (Público) .Retorna o identificador da SEMI-ARESTA	aux_semiaresta: ponteiro para um NO, que assume a estrutura de uma SEMI-ARESTA	.ponteiro para um CODIGO (identificador de aux_semiaresta)	38

Achar_vertice (Público) .Encontra o vértice da SEMI-ARESTA	.element: ponteiro para um NO que assume a estrutura de uma SEMI-ARESTA	.ponteiro para um NO (VERTICE para o qual a SEMI-ARESTA (element) pertence)	43
Mudar_vertice (Público) .Muda o vértice da SEMI-ARESTA	.element: ponteiro para um NO que assume a estrutura de uma SEMI-ARESTA .vert: ponteiro para um NO que assume a estrutura de um VERTICE	.nenhum	49
Ligar_semiaresta_aresta (Público) .Liga a SEMI-ARESTA a sua ARESTA pai	.element: ponteiro para um NO que assume a estrutura de uma ARESTA (a ARESTA à qual a SEMI-ARESTA será associada) .semi_arestas: ponteiro para um NO que assume a estrutura de uma SEMI-ARESTA	.nenhum	33
Mudar_semiaresta_ligada_vertice (Público) .Muda o VERTICE da SEMI-ARESTA que está ligada a um determinado vértice	.aux_vert: ponteiro para um NO que assume a estrutura de um VERTICE .loop: ponteiro para um NO que assume a estrutura de um LOOP	.nenhum	44
Mudar_loop (Público) .Muda o LOOP ao qual a SEMI-ARESTA pertence	.aux_semiaresta: ponteiro para um NO que assume a estrutura de uma SEMI-ARESTA .aux_loop: ponteiro para um NO que assume a estrutura de um objeto LOOP	.nenhum	34
Proxima_semiaresta (Público) .Encontra a próxima SEMI-ARESTA de um determinada SEMI-ARESTA	.aux_semiaresta: ponteiro para um NO que assume a estrutura de uma SEMI-ARESTA	.ponteiro para um NO que assume a estrutura de uma SEMI-ARESTA (próxima SEMI-ARESTA de aux_semiaresta)	36
Retirar (Público) .Retira um determinado objeto SEMI-ARESTA de um LOOP sem destruí-lo	.lop: ponteiro para um NO que assume a estrutura de um LOOP (loop pai da semi-aresta a ser retirada) .element: ponteiro para NO que assume a estrutura de SEMI-ARESTA (a SEMI-ARESTA a ser retirada)	.ponteiro para a SEMI-ARESTA retirada	32

Semiaresta_anterior (Público) Encontra a SEMI-ARESTA anterior a uma determinada SEMI-ARESTA, na lista de SEMI-ARESTAS	.aux_semiaresta: ponteiro para um NO que assume a estrutura de uma SEMI-ARESTA	.ponteiro para um NO que assume a estrutura de uma SEMI-ARESTA (SEMI-ARESTA anterior de aux_semiaresta)	37
Mate (Público) Encontra a SEMI-ARESTA irmã de um determinada SEMI-ARESTA	.aux_semiaresta: ponteiro para um NO que assume a estrutura de uma SEMI-ARESTA	.ponteiro para um NO que assume a estrutura de um objeto SEMI-ARESTA (SEMI-ARESTA irmã de aux_semiaresta)	40
Mudar_proxima_semiaresta (Público) Muda a seguinte SEMI-ARESTA de uma determinada SEMI-ARESTA	.aux_semiaresta: ponteiro para um NO que assume a estrutura de uma SEMI-ARESTA .prox_semiaresta: ponteiro para um NO que assume a estrutura de uma SEMI-ARESTA (a nova próxima SEMI-ARESTA de aux_semiaresta)	.nenhum	41
Mudar_semiaresta_anterior (Público) Muda a SEMI-ARESTA anterior a uma determinada SEMI-ARESTA	.aux_semiaresta: ponteiro para um NO que assume a estrutura de uma SEMI-ARESTA .prox_semiaresta: ponteiro para um NO que assume a estrutura de uma SEMI-ARESTA (a nova SEMI-ARESTA anterior)	.nenhum	42
Rec_ident_semiaresta (Público) Recupera o identificador de uma SEMI-ARESTA entrado pelo usuário	.parâmetros do "Xview" event item	.nenhum	45
Verificar_existência (Público) Verifica a existência de uma SEMI-ARESTA	.aux_face: ponteiro para um NO que assume a estrutura de uma FACE (a qual pertence a SEMI-ARESTA) .element1 e element2: ponteiros CODIGO (código dos vértices)	.ponteiro para a SEMI-ARESTA, caso exista, ou um ponteiro nulo caso não exista	46
Buscar (Público) Busca uma SEMI-ARESTA, dado um identificador e o SOLIDO pai	.aux_solido: ponteiro para um NO que assume a estrutura de um SOLIDO-BREP (que deve conter o vértice procurado) .element: ponteiro para um CODIGO, o identificador da SEMI-ARESTA procurada	.ponteiro para a SEMI-ARESTA procurada; caso não encontre, um ponteiro nulo	39

4.8 - Fluxo de Mensagens.

A quantidade de fluxo de mensagens entre os objetos do protótipo é bastante grande. A tabela a seguir ilustra as trocas de mensagens entre os objetos das diferentes classes do sistema (figura 4.9). Nas linhas da tabela estão os objetos origem (de onde partem as mensagens), e nas colunas encontram-se os objetos que recebem as mensagens. As mensagens são referenciadas através de um número que pode ser usado para localizá-los na lista de métodos das classes [SCO_93a], e também no código do sistema.

Classe Quilômetro	Classe Ponto	Lista	Vértice	Semi aresta	Aresta	Curva	Ciclo	Face	Super- fície	Sólido B-Rep	Sólido	Cena
Vértice	4,1,5,3 6,7	8								143,146		
Semi Aresta	3,5,7,8	16, 17		27,49,49,56			101, 103	116,123		142		
Aresta	3,5,6,7	17		43	26	66				144		
Curva	7,5,3 6	20, 19, 11 26		43,36,31 37,40		66,73,66,65 65,66,67,67, 66, 61	102	123,117 113		149,142,143 153,152,153 142,150,159 173,154,161 226,226,227, 230	201,192	246,240 237 246
Ciclo	7,1,5,6 3	16		43,36			99, 106	118,117 122		142		
Super- fície	3,5,7,8								130			
Face	7,1,3,5,6	14		30			103	113	131	142		
Sólido B-Rep	1,3,5,7,8 2	8,15,10,21, 19		28,31,30,46 47, 46, 44 39,49,28,44, 36,35,40,41 32,45,40	51,54,52,57 53,54		100,111,103 112,101 107,102,106 104	121,126 116,116 124,127 122,126		136,139,160 132, 133 146, 144	166,166,164 192,166,160 202,227,229 229	237, 240 246 246
Sólido	7,5,3,1,6	10,11,12,18 19,20		28	52,55	74,75	102, 101	116,116 116,122 124,123		146,144,142 140,162,146 166,166	166,229 225,226,227 230,222,223 160,166,167 207,166	240,237 246
Cena	7,1,5,3	20,11,14,18		38,31,43,38 40	60,54,53,59	74, 66, 70, 66,66	109,109,104 102	123,117, 122,127 114	136,137 134, 135,136	149,141,143 144, 142	204,166, 166,166,206 202,194,196	232,269,224 242,241,239 243,251

Figura 4.9- Tabela dos métodos ativados pelas classes do sistema.

4.9-Considerações Finais.

Nesse capítulo, foram introduzidos os conceitos básicos do paradigma da orientação a objetos bem como abordagem utilizada para o projeto orientado a objetos do SMS. A seguir, foram apresentados a especificação detalhada dos requisitos e funcionalidade do modelador, bem como os resultados do projeto orientado a objetos do protótipo do SMS, no qual foi baseada a implementação descrita no Capítulo 5.

O projeto orientado a objetos do protótipo, foi desenvolvido utilizando uma linguagem de programação não orientada a objetos ("C"), devido à necessidade de compatibilidade da linguagem com a versão do sistema "Xview" disponível no ICMSC, que foi utilizado para implementar a interface do SMS. Para não comprometer o projeto e desenvolvimento orientado a objetos, foram simuladas algumas características da orientação a objetos. Essa simulação é descrita no Capítulo 5, juntamente com os detalhes de implementação do gerenciador de representações, da interface e de algumas operações de modelagem de alto nível que constituem o protótipo desenvolvido.

CAPÍTULO 5

CONSTRUÇÃO DO PROTÓTIPO DO SMS.

5.1-Considerações Iniciais.

O protótipo do SMS foi desenvolvido em "estações de trabalho" SUNs, utilizando a linguagem "C" e o sistema "Xview" [HEL_90], e atualmente totaliza aproximadamente 9.500 linhas de código. Apesar de não ser a solução ideal, a utilização de uma linguagem não orientada a objetos deveu-se à necessidade de compatibilidade da linguagem com a versão do sistema Xview disponível nas estações do ICMSC.

Neste capítulo, são vistos os aspectos de implementação do primeiro protótipo do SMS, cujo projeto foi descrito no capítulo anterior. Inicialmente, descreve-se como é realizada a simulação da orientação a objetos em "C" (seção 5.2). A seguir, é discutida a implementação do sub-sistema de gerenciamento de representações (seção 5.3), da interface (seção 5.4) e de algumas operações de modelagem de alto nível (seção 5.5). Na seção 5.6 são abordados alguns aspectos de teste do protótipo.

5.2- Simulação da Programação Orientada a Objetos.

Foram implementados mecanismos e estruturas de dados básicas para simular os conceitos de classe, polimorfismo, herança, abstração e encapsulamento de dados em "C". Antes da descrição dessas funções e estruturas, é apresentada uma breve descrição do funcionamento da simulação.

Todas as classes de objetos básicos do sistema (variáveis globais) são definidas no arquivo "objetos_básicos.h". Entre os objetos definidos como básicos as classes CADEIA (uma string) e CODIGO (long int). Além disso, toda classe de objetos possui sua definição (atributos) em um arquivo "<objeto>.h" (por exemplo, sólido.h, aresta.h,...) e seus métodos de manipulação e acesso encontram-se em um arquivo "<operações_objeto.h>". Os objetos devem ser acessados ou manipulados através de seus métodos apenas, reforçando os conceitos de abstração de dados e encapsulamento de informações [DUF_90].

A nível de notação, foram definidas algumas diretrizes de programação durante implementação do protótipo:

- . O nome das classes de objetos (tipo de dados) criadas é composto de letras maiúsculas.
- . O nome de todas as instâncias de objetos (variáveis) declaradas é composto de letras minúsculas.
- . O nome de toda função que implementa um método é iniciado com letra maiúscula.

Toda classe de objetos possui uma tabela (dinâmica) contendo seus métodos, sendo que cada método é associado à mensagem que o ativa e à função que o implementa. Para que um

objeto ative um método pertencente a sua própria classe ou à outra classe qualquer, ele deve chamar uma função denominada **Enviamentagem** (ver seção 5.2.2) com o nome da mensagem como primeiro parâmetro e um objeto da classe destino como segundo (este faz o papel do objeto que recebe a mensagem). Esta função verifica a que classe pertence o objeto que recebeu a mensagem e busca na tabela de métodos dessa classe o endereço da função que implementa o método correspondente. As tabelas de métodos são dinâmicas, existindo apenas em tempo de execução, sendo inicializadas toda vez que o protótipo é executado.

Se o método não for encontrado na classe do objeto que recebe a mensagem, a função **Enviamentagem** verifica se a classe possui uma classe pai. Caso possua, a tabela de métodos da classe pai é percorrida. Portanto pode-se ter uma mensagem (por exemplo criar,...) implementada por diferentes funções (o que equivale ao conceito de polimorfismo) e pode-se ter também classes herdando métodos de outras classes (herança, ainda que em um único nível).

Além disso, no código do protótipo, um objeto de uma classe A não deve fazer acesso direto aos atributos de um objeto de outra classe, acessando-os apenas através da ativação de métodos da outra classe, através da função **Enviamentagem**.

Uma classe importante do sistema é a classe NO, cujos objetos compõem a classe LISTA. Objetos NO podem assumir o papel de qualquer objeto definido no sistema (sólido, aresta, vértice, ...), e ao objeto NO que é associado o e a tabela de métodos da classe que define o tipo de NO. Isto porque todos os objetos do sistema são listas ou nós de listas, devido à natureza da estrutura de dados B-Rep utilizada, que é baseada em listas (ver Capítulo 4).

A simulação da orientação a objetos seguiu as sugestões apresentadas em [DUF_90]. Para possibilitar a simulação, foram criadas algumas estruturas de dados e funções básicas que são descritas a seguir.

5.2.1 - Estruturas e Tipos de Dados.

O tipo de dado que implementa o conceito de **método**, descrito abaixo, consiste de uma estrutura composta de dois campos: o nome da mensagem e um ponteiro para a função que implementa o método, a qual retorna sempre um ponteiro do tipo void que pode, através da operação de "casting" do C, apontar para qualquer estrutura definida.

```
Typedef struct METD{  
    METODOS mensagem;  
    void *(( *metfunc) ());  
}METODO_MAP;
```

O tipo de dado que implementa o conceito de **classe** é uma estrutura composta por três campos: o nome da classe, um ponteiro para a classe pai, da qual a classe pode herdar métodos e um ponteiro para um vetor contendo os métodos da classe.

```
Typedef struct CLASSES{  
    METODO_MAP (*(metodos) [34]);  
    CADEIA nome_classe;  
    struct CLASSES *ancestor;  
} CLASSE;
```

A definição de uma tabela contendo um número máximo de 34 métodos não possui nenhuma razão especial: era preciso estabelecer um limite máximo, que foi fixado em 34. Esse valor pode ser aumentado se preciso.

A estrutura que contém os métodos associados a uma classe possui um único campo: um ponteiro para um vetor de métodos (METODO_MAP). Cada classe possui em sua definição uma estrutura desse tipo, declarada como sendo do tipo MET.

```
typedef struct METS{
    METODO_MAP *metodos [34];
}MET;
```

Abaixo são descritas as estruturas que contém os métodos associados a cada classe do protótipo do SMS e a seguir, a estrutura que contém as mensagens definidas no sistema.

Cena:
MET *metodos_cena;

Sólido:
MET *metodos_solido

Sólido B-Rep.
MET *metodos_sólido_brep;

Face
MET *metodos_face;

Loop (Ciclo)
MET *metodos_loop;

Superfície
MET *metodos_superficie;

Aresta
MET *metodos_aresta;

Curva
MET *metodos_curva;

Semi-Aresta
MET *metodos_semiaresta;

Vértice
MET *metodos_vertice;

Lista
MET *metodos_lista;

typedef enum METHODS

```
{GERAR_IDENTIFICACAO, CRIAR, REMOVER, ACHAR_FACE, ACHAR_ARESTA VERTICES,
LIGAR_VERTICE_SOLIDO,MVFS, KVFS, SEMI_ARESTA_DIREITA, SEMI_ARESTA_ESQUERDA,
LOOP_EXTERNO, ACHAR_LOOP, FACE_PAI, SOLIDO_PAI, ACHAR_SEMI_ARESTA, ARESTA_PAI,
ACHAR_VERTICE, LIGAR_SEMIARESTA_ARESTA, COORDENADAS_VERTICE, SEMI_ARESTAS,
LIGAR_SEMIARESTA_VERTICE, APAGAR, PROCURAR, ADICIONAR, MUDAR_VERTICE,
ACHAR_SEMIARESTA_LIGADA_VERTICE, COMPARA_VERTICE, LIGAR_ARESTA_SEMIARESTA,
```

```

MUDAR_SEMIARESTA_LIGADA_VERTICE, MUDAR_LOOP_EXTERNO, MUDAR_LOOP_SEMIARESTA,
SEMIARESTA_ANTERIOR, PROXIMA_SEMIARESTA, MUDAR_SEMIARESTA_ANTERIOR,
MUDAR_PROXIMA_SEMIARESTA, MUDAR_SEMIARESTA_LOOP, BUSCAR,
VERIFICAR_EXISTENCIA, KEV, MEV, KEF, MEF, VERIFICAR_VERTICE, PROXIMA_ARESTA,
A_KEV, A_MEV, A_MEF, A_KEF, VERIFICAR_MESMA_FACE, MATE, ULTIMA_SEMI_ARESTA,
LOOP_DIFERENTES, MEKR, A_MEKR, A_KEMR, KEMR, KFMRH, A_KFMRH, MUDAR_FACE,
MFKRH, A_MFKRH, MUDAR_LOOP_FACE, PROXIMO_VERTICE, ULTIMO_VERTICE,
FACE_EXTERNA, VERTICE_ANTERIOR, SWEEPING, CRIA_CIRCULO, CRIA_ARCO,
ACHAR_SOLIDOS, ACHAR_CENA, CRIA_RETANGULO, CRIAR_QUADRADO, CRIAR_POLIGONO,
CRIA_SPLINE, IDENTIFICADOR, MOSTRAR, DESENHA, PROXIMO, PROXIMA,
ASSOCIAR_DESCRICAO, ARMAZENAR, NOME, TIPO_OBJ, PRIMEIRO_ELEMENTO, DESCRICAO,
ACHAR_SUPERFICIE, CALCULAR_SPLINE, ROTACIONAR, TRANSLADAR,
MUDAR_COORDENADAS, PROC_ROTACIONAR, PROC_TRANSLADAR, CRIAR_BLOCO,
CRIAR_CILINDRO, CRIAR_PRISMA, CRIAR_PIRAMEDA, UNIR_SOLIDOS, RETIRAR, NUMERO_FACES,
RECUPERAR, ESCREVER, LISTAR, TROCAR
} METODOS;

```

A estrutura acima simula o conceito de mensagem, sendo que toda vez que um método é criado a mensagem deve ser inserida nesse conjunto, caso ela ainda não exista. Cada elemento da estrutura METODS (uma mensagem) está associado a um ou vários métodos implementados.

Para saber qual método implementa uma certa mensagem enviada a um objeto de uma classe, basta percorrer a função "Inicializa_metodos" da classe, (ver a seção a seguir), e relacionar o nome atribuído ao campo mensagem da estrutura metodos_<objeto> ao nome da função armazenado no campo metfunc. Para saber quais métodos uma mensagem ativa, seria necessário realizar o mesmo processo, percorrendo a função inicializa_metodos<objeto> de todas as classes definidas no sistema.

5.2.2 - Funções.

A função **Enviarmensagem**, descrita abaixo, é responsável por verificar a qual classe uma certa mensagem está associada, e acionar a função que implementa o método correspondente. Caso o método não exista, uma mensagem de erro é retornada.

```
void EnviaMensagem(mensagem, receiver, argumento1, argumento2, argumento3, argumento4, ...,
argumento9)
```

```
long int mensagem;
    struct NO *receiver; (Objeto ao qual pertence o método)
    void argumento1;
    void argumento2;
    void argumento3;
    void argumento4; (Parâmetros para o método)
    void argumento5;
    void argumento6;
    void argumento7;
    void argumento8;
    void argumento9;
    {
    int índice;
        CLASSE *aux, *aux3;
        METODO_MAP *map, *aux2;
        aux = receiver->classe;
        do
        {
            índice = 0
            do
            {
                map=NULL;
                free(map);
                map = (struct METD *) malloc(sizeof(METDO_MAP));
                aux2 = map;
                map = ((*aux->métodos) [índice]);
                if (map == NULL)
                {
                    operação_erro(17);
                    return NULL;
                }
                if (mensagem == map->mensagem)
                {
                    free(aux2);
                    return((map->metfunc) (argumento1, argumento2, argumento3, argumento4, argumento5,
argumento6, argumento7, argumento8, argumento9));
                }
                índice++;
            }while(map != NULL);
            aux = aux->ancestor;
        }while(aux != NULL);
        operação_erro (17);
        return NULL;
    }
}
```

A função `Enviarmensagem` identifica no seu primeiro parâmetro a mensagem, no segundo parâmetro o objeto ao qual esta foi enviada, e nos seguintes, os parâmetros necessários para executar o método. Assim, em uma primeira fase a função identifica a classe do receptor ("receiver"), localiza a tabela de métodos dessa classe e verifica se o método associado à mensagem está definido na tabela. Caso não o encontre, busca na tabela da classe pai do receptor, e se o método não for encontrado uma mensagem de erro é enviada. Caso o método seja encontrado, `Enviarmensagem` ativa a função que o implementa. O número de parâmetros para o método foi estabelecido em 9 porque era necessário ser fixado e nenhum método criado exigiu mais de 9 parâmetros. Esse número pode ser aumentado, se preciso.

A função `Inicializa_métodos_<classe>` é responsável pela inicialização dos métodos associados a uma classe, sendo que cada classe possui uma função como esta. Toda vez que o protótipo do SMS é inicializado, o primeiro passo consiste em chamar essas funções, inicializando os métodos de todas as classes. A seguir, é apresentada a função de inicialização associada à classe `Lista` como exemplo.

```
void Inicializa_métodos_lista()
{
    métodos_lista = (struct METS *) malloc(sizeof(METS));
    métodos_lista[0] = (struct METD *) malloc(sizeof(METODO_MAP));
    métodos_lista[0]->mensagem = CRIAR;
    métodos_lista[0]->metfunc = Criar_lista;

    métodos_lista[1] = (struct METD *) malloc(sizeof(METODO_MAP));
    métodos_lista[1]->mensagem = DESTRUIR;
    métodos_lista[1]->metfunc = Destruir_lista;

    métodos_lista[2] = (struct METD *) malloc(sizeof(METODO_MAP));
    métodos_lista[2]->mensagem = REMOVER;
    métodos_lista[2]->metfunc = Remover_lista;

    métodos_lista[3] = (struct METD *) malloc(sizeof(METODO_MAP));
    métodos_lista[3]->mensagem = ADICIONAR;
    métodos_lista[3]->metfunc = Adicionar_lista;

    métodos_lista[4] = (struct METD *) malloc(sizeof(METODO_MAP));
    métodos_lista[4]->mensagem = PRIMEIRO_ELEMENTO;
```

```

métodos_lista[4]->metfunc = Primeiro_elemento;

métodos_lista[5] =(struct METD *) malloc(sizeof(METODO_MAP));
métodos_lista[5]->mensagem = PROCURAR;
métodos_lista[5]->metfunc = Procurar_lista;

métodos_lista[6] =(struct METD *) malloc(sizeof(METODO_MAP));
métodos_lista[6]->mensagem = RETIRAR;
métodos_lista[6]->metfunc = Retirar_lista;

```

Foi também criada uma função chamada `Inicializa_objeto`, ilustrada a seguir, que cria um objeto de uma classe qualquer que não possui nenhum atributo além do ponteiro para a tabela de métodos da classe à qual o objeto criado pertence. Esta classe é definida na chamada de `Inicializa_objeto`, sendo o único parâmetro dessa função.

```

NO *Inicializa_objeto(tipo)
    TIPO tipo;

{
    NO *element;

    /*Alocando espaço*/

    element =(struct NOS*)malloc(sizeof(NO));

    element->classe =(struct CLASSES*)malloc(sizeof(CLASSE));
    /*Associando as Classes e os metodos correspondentes*/
    switch(tipo){
        case FAC: {
            strcpy(element->classe->nome_classe, "FACES");
            element->classe->metods =métodos_face;
            break;
        }
        case LOP: {
            strcpy(element->classe->nome_classe, "LOOPS");
            element->classe->metods =métodos_loop;
            break;
        }
        case AREST: {
            strcpy(element->classe->nome_classe, "ARESTAS");
            element->classe->metods =métodos_aresta;
            break;
        }
        case VERT: {
            strcpy(element->classe->nome_classe, "VERTICES");
            element->classe->metods =metodos_vertice;
            break;
        }
        case SEMIAREST: {
            strcpy(element->classe->nome_classe, "SEMI-ARESTAS");
            element->classe->metods =metodos_semiaresta;
            break;
        }
    }
}

```

```
    }  
    case SOLIDBREP: {  
        strcpy(element->classe->nome_classe, "SOLIDOBREP");  
        element->classe->metods = metodos_solidos_brep;  
        break;  
    }  
    case LIST:  
    {  
        strcpy(element->classe->nome_classe, "LISTAS");  
        element->classe->metods = metodos_lista;  
        break;  
    }  
    case CURV:  
    {  
        strcpy(element->classe->nome_classe, "CURVA");  
        element->classe->metods = metodos_curva;  
        break;  
    }  
    case SUPERFIC:  
    {  
        strcpy(element->classe->nome_classe, "SUPERFICIE");  
        element->classe->metods = metodos_superficie;  
        break;  
    }  
  
    case CEN:  
  
    {  
        strcpy(element->classe->nome_classe, "CENA");  
        element->classe->metods = metodos_cena;  
        break;  
    }  
  
    case SOLID:  
    {  
        strcpy(element->classe->nome_classe, "SOLIDO");  
        element->classe->metods = metodos_solidos;  
        break;  
    }  
    }  
    return element;  
}
```

As funções descritas são responsáveis pela simulação de polimorfismo, envio de mensagens e herança de métodos. Os métodos do protótipo são implementados através de funções e "procedimentos", que internamente podem chamar dois tipos de funções apenas: **Inicializa_objeto** (responsável por criar um objeto temporário de uma determinada classe que possui apenas o ponteiro para a tabela de métodos da classe, e nenhum valor para os atributos)

e **Enviamentagem** (através da qual os métodos são acionados). Portanto os endereços reais das funções e procedimentos que implementam os métodos não são conhecidos em tempo de compilação, mas apenas em tempo de execução.

Para adicionar um novo método associado a uma classe, é necessário:

1-Criar a função que o implementa no arquivo de operações da classe (**operações_<classe>.h**).

2-Inserir o novo método na tabela de métodos da classe, o que é feito alterando a função **Inicializa_métodos** da classe.

3-Também é preciso definir a mensagem associada ao método na estrutura **METHODS** (descrita anteriormente), caso ela ainda não exista.

4-Por último, é verificado se o número de parâmetros do método excede 9. Caso exceda, é necessário alterar a função **Enviamentagem**.

É interessante comparar genericamente o funcionamento de um sistema realmente orientado a objetos, com a simulação implementada. Em um sistema orientado a objetos define-se uma classe descrevendo seus atributos e os métodos associados. Os objetos podem ser criados e destruídos dinamicamente. Os atributos de um objeto só podem ser acessados pelos métodos da sua classe. Na simulação pode-se forçar uma disciplina de programação na qual os atributos de um objeto são acessados por seus métodos apenas, mas não é possível impedir eventuais quebras nessa disciplina. Nos sistemas orientados a objetos, se uma classe A for definida como uma especialização de outra, A herdará automaticamente os atributos da outra, além de novos atributos. Esta característica é perdida na simulação.

Em sistemas orientados a objetos, a classe é uma característica intrínseca a todos os elementos de dados e não está necessariamente associada a um ou a vários tipos de dados criados. Isso também não acontece na simulação descrita aqui, onde toda classe está associada ao elemento **NO**. Mais detalhes sobre a simulação da orientação a objetos e um exemplo do funcionamento podem ser encontrados em [SCO_93b].

Para desenvolver o protótipo do SMS foram implementadas 12 classes de objetos às quais foram associados 269 métodos (ver caixa 5.1)

CLASSE:	MÉTODOS ASSOCIADOS
Cena	36
Sólido	45
Sólido B-rep	51
Face	17
Ciclo	14
Semi-aresta	25
Aresta	15
Vértice	19
Curva	34
Superfície	08
Lista	07

obs- Um objeto NÓ também foi criado e utilizado na definição do objeto lista. Esse objeto é considerado intrínseco ao sistema e não possui métodos próprios.

Caixa 5.1-Lista das classes implementadas e total de métodos associados à cada uma.

5.3- Subsistema de Gerenciamento de Representações

O subsistema de gerenciamento de representações implementado é composto por duas camadas de software: a primeira formada pela estrutura de dados B-Rep e os algoritmos básicos de manipulação da estrutura; e a segunda que implementa os Operadores (Topológicos) de Euler.

A estrutura de dados B-Rep implementada, responsável por armazenar a descrição dos objetos, consiste de uma ligeira modificação da Semi-Aresta proposta por Mäntylä [MAN_88] e é composta por 10 elementos hierárquicos: Cena, Sólido, Sólido B-Rep, Face, Ciclo, Aresta, Semi Arestas, Vértice, Curva e Superfície. A visão hierárquica desta estrutura é apresentada na figura 5.1

Figura 5.1- Estrutura de dados B-Rep implementada.

O elemento "Sólido", presente na estrutura, permitirá que objetos descritos pelo SMS, no futuro, possam ser formados por mais de um sólido e que estes possuam mais do que um tipo de representação. A estrutura interna dos objetos (aresta, face, ...) que compõem a estrutura de dados da figura 5.1 são descritas em [SCO_93b]. Juntamente com a estrutura de dados acima, foram implementados 136 métodos básicos de manipulação, responsáveis por buscar,

armazenar ou alterar informações sobre sólidos B-Rep na estrutura. A lista dos métodos associados à classe de objetos a que pertencem é ilustrada na caixa 5.2.

-Classe Cena: Gerar_Identificador, Criar, Sólidos, Identificador, Próxima, Buscar, Remover, Lista_curvas e Lista_superfícies.

-Classe Sólido: Gerar_Identificador, Criar, Cena, Nome, Associa_Descrição, Tipo, Descrição, Retirar, Remover, Buscar e Buscar2.

-Classe Sólido B-Rep: Gerar_Identificador, Criar, Remover, Faces, Arestas, Vértices, Ligar_vértice_sólido, Buscar, Mudar_faces, Nome, Sólido, Próximo, Retirar e Último_vértice, Número_faces.

-Classe Superfície: Criar, Gerar_identificador, Remover, Retirar, Nome, Classe, Próxima e Indetificador.

-Classe Face: Gerar_Identificador, Criar, Remover, Loop_externo, Sólido_pai, Buscar, Mudar_loop, Verificar_mesma_face, Face_externa, Identificador, Próxima, Achar_loop, Retirar, Mudar_loop_externo e Superfície.

-Classe Loop: Gerar_identificador, Criar, Remover, Semiarestas, Face_pai, Semi_aresta_ligada_vértice, Mudar_semi_arestas, Última_semi_aresta, Verificar_loops, Mudar_face, Próximo, Identificador, Buscar e Retirar.

-Classe Semi-aresta: Gerar_Identificador, Criar, Remover, Loop_pai, Aresta_pai, Ligar_semiaresta_aresta, Mudar_vértice, Mudar_loop, Retirar, Próxima_semiaresta, Semiaresta_anterior, Identificador, "Mate", Mudar_próxima_semiaresta, Mudar_semiaresta_anterior, Mudar_semiaresta_ligada_vértice, Buscar, Mudar_aresta.

-Classe Curva: Gerar_identificador, Remover, Buscar, Criar, Retirar, Próxima, Nome, Classe, Buscar2 e Identificador.

-Classe Aresta: Gerar_Identificador, Criar, Remover, Retirar, Semi_aresta_esquerda, Semi_aresta_direita, Próxima, Verificar_existência, Identificador, Ligar_semiaresta_aresta, e Buscar.

-Classe Vértice: Gerar_Identificador, Criar, Remover, Coordenadas, Semi_aresta, Ligar_semi_aresta, Verificar_vértices, Retirar, Identificador e Verificar_existência.

-Classe Lista: Criar, Remover, Retirar, Adicionar, Apagar, Procurar_lista e Primeiro_elemento

Caixa 5.2-Lista do métodos que compõem o gerenciador de representações.

Na segunda camada do gerenciador de representações, implementou-se os 10 Operadores de Euler propostos por Mäntylä [MAN_84, MAN_88], como métodos associados

à classe Sólido B-Rep. No SMS, os Operadores de Euler também são divididos em operadores de Alto Nível e Baixo Nível. Nos operadores de baixo nível os parâmetros são ponteiros para os elementos da estrutura de dados; e nos de alto nível os parâmetros são identificadores e nomes de entidades topológicas (sólidos, sólidos B-Rep, faces, loops, arestas, semi-arestas, vértices, curvas e superfícies). A lista de métodos que implementam os Operadores de Euler totaliza 18 métodos, e é ilustrada a seguir, na caixa 5.3.

```

Mvfs - "Make vertex face and solid".
Kvfs - "Kill vertex face and solid".
Mev - "Make edge and vertex".
Kev - "Kill edge and vertex".
Mef - "Make edge and face".
Kef - "Kill edge and face".
Mekr - "Make edge and kill a ring".
Kear - "Kill edge and make a ring".
Mfkrh - "Make a face and kill a ring and role".
Kfarh - "Kill face and make a ring and role".
A_Mev - Operador Mev (alto nível) (verifica a existência de elemento
topológicos e ativa o Mev).
A_Kev - Operador Kev (alto nível) (verifica a existência de elementos
topológicos e ativa o Kev).
A_Mef - Operador Mef (alto nível) ....
A_Kef - Operador Kef (alto nível)
A_Mekr - Operador Mekr (alto nível)
A_Kear - Operador Kear (alto nível)
A_Mfkrh - Operador Mfkrh (alto nível)
A_Kfarh - Operador Kfarh (alto nível)

```

Caixa 5.3-Lista dos métodos que implementam os Operadores de Euler.

A implementação dos Operadores de Euler baseou-se na descrição dada por Mäntylä em [MAN_88]. As funções que implementam esses operadores utilizam intensamente os métodos de acesso básicos, mencionados na caixa 5.2. Formando assim, uma estrutura em camadas semelhante à apresentada no Capítulo 3. No modelador GWB [MAN_88] os Operadores de Euler compõem, juntamente com os procedimentos básicos de acesso, a camada que protege a estrutura de dados B-Rep. Porém, no SMS a estrutura de dados é totalmente protegida pelos

procedimentos básicos (caixa 5.2). Nem mesmo os Operadores de Euler possuem acesso direto à estrutura, que é manipulada através dos métodos de acesso e manipulação básicos (Caixa 5.3). Esta camada de proteção à estrutura de dados, que impede o acesso direto dos operadores topológicos não existe no GWB e em outros modeladores discutidos na literatura.

5.4- Interface do SMS.

A interface do sistema foi desenvolvida utilizando as bibliotecas do "toolkit "Xview" desenvolvido para o protocolo XWindow [HEL_90, SCH_90], e é composta de três telas principais: inicial; para edição de uma cena nova; e para edição de uma cena recuperada.

A tela inicial fica aguardando que o usuário ative um menu pressionando um dos botões do "mouse". O menu apresentado possui três opções: **editar nova cena**, **sair do sistema** e **recuperar cena**. O usuário pode retornar à tela inicial sempre que precisar recuperar ou editar outra cena, podendo assim editar várias cenas "concorrentemente", já que o usuário pode "chavear" de uma cena para outra. Entretanto, o usuário pode, em um determinado momento, estar manipulando uma única cena, uma vez que só existe uma janela de edição. Essa característica pode ser melhor aproveitada se o sistema for modificado para trabalhar com várias janelas de edição. Esta tela é ilustrada na figura 5.2 (abaixo).

Figura 5.2:-Tela inicial do protótipo.

A tela para editar uma nova cena é composta por quatro "buttons" que acionam um conjunto de menus contendo as operações de edição para os objetos: cena, sólido, curva e SMS (este último possui apenas duas operações: sair do sistema e retornar à tela inicial, que são as duas opções que o usuário possui caso desista de editar a nova cena). Antes de dar o nome para a nova cena a ser editada e confirmar a operação de edição de uma cena nova, apenas o "Button" relacionado com o objeto SMS está habilitado. Para cada operação sobre cena, sólido e curva, uma pequena tela é apresentada para o usuário entrar com os dados. A tela para edição de uma nova cena é apresentada na figura 5.3. A seguir, são ilustrados os menus contendo as operações possíveis para cada objeto (cena, sólido e curva), nas figuras 5.4 a 5.6.

Figura 5.3:- Tela para edição de uma nova cena.

Figura 5.5:- Menu contendo as possíveis operações sobre cena.

Figura 5.4:-Menu com as possíveis operações sobre o objeto sólido.

Figura 5.6:- Menu contendo as possíveis operações sobre o objeto curva.

No protótipo, curvas que não são segmentos de retas são permitidas apenas temporariamente para realizar a operação de varredura. Associadas às arestas existe apenas segmentos de reta. Portanto o sistema aproxima todas as arestas e faces por segmentos de reta e planos, respectivamente (modelo poliedral).

A tela para a edição de uma cena recuperada, figura 5.7, é semelhante à tela de edição de uma nova cena. Ao invés do sistema pedir o nome de uma nova cena a ser editada, ele pede o nome da cena a ser recuperada do disco. Os "buttons" contendo os menus, e os próprios menus são idênticos aos da tela descrita anteriormente, pois as operações de modelagem disponíveis são as mesmas. Cada uma delas apresenta uma janela para que o usuário possa entrar com os dados necessários, como já mencionado. Dois exemplos dessas janelas são ilustrados nas figuras 5.8 e 5.9.

Figura 5.7:-Tela para a recuperação de uma cena e sua edição.

Figura 5.8:-Operação rotacionar sólido.

Figura 5.9:-Operação juntar sólidos.

A operação "listar cenas" mostra ao usuário a lista de cenas armazenadas na memória e a lista dos objetos que as compõem, porém não relaciona os nomes com os sólidos apresentados. Toda operação requer que o usuário selecione os objetos através de nomes de

sólidos, curvas e identificadores de elementos topológicos, obrigando o usuário conhecer o nome de cada objeto. Esta é uma das limitações apresentadas pela interface do protótipo, que pode ser tratada permitindo que o usuário selecione os objetos, que participam das operações de modelagem (sólido, face, aresta,...), através do "mouse" e não através dos nomes e identificadores desses objetos.

Figura 5.10:- Exemplo mensagem de erro.

As mensagens de erros são enviadas através de "notices" e impedem que o usuário realize qualquer operação antes que pressione o "Button" continue presente no "notice". A figura 5.10 ilustra uma mensagem de erro.

Figura 5.11:- Exemplo de uma cena criada contendo, um bloco, uma pirâmide e um cilindro.

Figurura 5.12:-Cena criada como exemplo, contendo um sólido criado pela operação de varredura rotacional de uma curva.

Uma das limitações da interface, é que ela permite varredura rotacional (figura 5.11) apenas sobre um dos eixos principais e não sobre qualquer eixo especificado. Também realiza operação de varredura sobre curvas abertas apenas, e não sobre sólidos.

É importante observar que a interface é um dos elementos principais de um modelador e deve ser amigável e bem projetada. Assim a interface que foi apresentada é apenas um protótipo e precisa sofrer mudanças. Como mencionado, ela possui várias limitações, sendo que a principal é não permitir que o usuário selecione os objetos diretamente através do "mouse". Isso dificulta muito a edição de cenas e objetos, pois o usuário precisa conhecer cada "detalhe" da cena (isto é, o nome associado a cada objeto e os identificadores de seus elemento topológicos). Outra limitação existente é com respeito visualização da cena, já que o protótipo apresenta ao usuário apenas o modelo "fio de arame" dos objetos, sem tratamento de curvas e superfícies escondidas. Além disso, não permite o usuário alterar os parâmetros de visualização como distância e posição do observador, tipo de projeção, etc...

As limitações existentes na interface são consequência desta não ser o objetivo principal, do trabalho, mas apenas um protótipo criado para testar o gerenciador de representações e algumas operações de modelagem de alto nível.

5.6 Operações de Modelagem de Alto Nível.

Implementou-se algumas operações de modelagem de alto nível para descrição de sólidos, com o propósito inicial de realizar alguns testes. Na caixa 5.3 são apresentadas todas as operações de modelagem disponíveis. As operações de armazenamento e recuperação de umas, apesar de não serem operações de modelagem, fazem parte do **subsistema de modelagem** e são de grande importância, pois permitem que o usuário não perca as cenas editadas e que possa alterar uma cena sempre que desejar.

classe Cena:

Armazenar: armazena a cena em um arquivo.

Trocar: Troca a cena que está sendo editada e mostrada

Apagar: Apaga a cena desenhada na tela.

Remover: Remove a cena da memória.

Listar: Lista as cenas presentes na memória junto com os objetos que as compõem.

classe: Sólido (Sólido B-Rep): **Remover:** Remove um sólido de uma cena.

Rotacionar: rotaciona o sólido de um grau sobre um eixo escolhido.

Translação: Translada um sólido.

Primitivas: Cria primitivas B-Rep: cilindros, blocos, prismas e pirâmides.

Operadores de Euler: realizam operações sobre os elementos topológicos que compõem um sólido B-Rep (arestas, faces,).

Juntar: junta dois sólidos através de suas faces.

classe: Curva : **Varredura ("sweeping") Translacional:** desloca uma curva fechada através de um vetor, criando um sólido.

Varredura ("sweeping") Rotacional: Rotaciona uma curva aberta sobre um eixo (x, y ou z).

Remover: remove uma dada curva.

Criar : cria cinco tipos de curvas: polígonos (abertos ou fechados), splines, circunferências, retângulos e quadrados.

5.5 - Testes

O protótipo do SMS foi testado em duas etapas diferentes. Na primeira etapa, as operações foram testadas sem que os objetos resultantes fossem desenhados. O sistema gerava um arquivo texto descrevendo a estrutura de dados B-rep resultante das operações. Assim, foi possível verificar manualmente por, inspeção, se a estrutura resultante estava representando corretamente os objetos, e portanto se as operações estavam corretas.

Em uma segunda etapa, com o protótipo da interface concluída, as operações foram testadas com o esqueleto dos objetos (modelos fio-de-arame) resultantes sendo mostrados na tela, permitindo assim uma visualização simples das cenas e seus objetos.

Casos especiais de aplicação dos operadores de Euler, como a criação de um ciclo com apenas uma aresta e condições de erro foram testadas. Entre as condições de erros testadas têm-se: a aplicação de uma operação a um sólido inexistente ou a aplicação uma operação sobre uma entidade topológica inexistente. O sistema de gerenciamento de representações foi testado através das operações de descrição de objetos implementadas e da aplicação dos Operadores de Euler de Alto Nível individualmente. Além disso, três usuários que não participaram do desenvolvimento testaram o protótipo e encontraram alguns erros, falhas de interface e dificuldades de uso não detectados pelos testes realizados pelo "desenvolvedor", e estão sendo resolvidos.

5.8 - Considerações Finais.

Neste capítulo foram apresentados os detalhes de implementação do primeiro protótipo do SMS. As estruturas e funções criadas para permitir a simulação da orientação a objetos foram descritas. Também foi discutida a implementação do **gerenciador de representações, da interface e de algumas operações de modelagem**, e finalmente, foram abordados aspectos de teste. Acredita-se que a extensão e manutenção do modelador será bastante facilitada com o projeto orientado a objetos , apesar da dificuldade adicional introduzida pela simulação da orientação a objetos à nível de programação. É importante lembrar que a interface desempenha um papel importante nos modeladores e que a interface desenvolvida precisa ser aperfeiçoada, assim como as operações de edição e de descrição de sólidos a nível de interface. Esta é uma tarefa complicada, já que o desenvolvimento de uma interface simples e de algumas operações de modelagem de alto nível acarretou bastante trabalho e consumiu um tempo considerável.

CAPÍTULO 6

CONCLUSÕES.

6.1-Considerações Iniciais.

A necessidade de modeladores de sólidos mais abrangentes e precisos têm feito crescer a pesquisa por sistemas que possuam uma arquitetura mais o próximo possível da ideal [MIL_89, FIS_91, MIL_86], ou seja, poderosa o bastante para manipular o maior número possível de objetos da forma mais eficiente possível. Esta busca por arquiteturas híbridas e mais abrangentes têm originado sistemas de modelagem complexos [CRO_91, NIE_92] e com as mais variadas combinações de representações possíveis, como o GeneSys, desenvolvido no Brasil [FIS_91].

Seguindo esta filosofia, pretende-se desenvolver, no ICMSC-USP, um modelador de sólidos híbrido B-Rep-CSG que possa, no futuro, incorporar superfícies de forma livre, e ainda, que seja preciso, abrangente, fácil que de ser estendido, com possibilidades de possuir uma interface poderosa e amigável. Assim, o projeto de Mestrado constituiu-se do desenvolvimento de um primeiro protótipo desse Sistema de Modelagem de Sólidos, o qual foi desenvolvido seguindo o paradigma da orientação a objetos. O protótipo é constituído de um gerenciador de representações B-Rep e uma interface simples que fornece algumas operações de descrição de sólidos (varredura rotacional e translacional de curvas, "glueing" e criação de primitivas pré-definidas).

A utilização da orientação a objetos tanto no projeto, quanto, na implementação do modelador, teve o propósito de facilitar a extensão do protótipo para o futuro modelador híbrido.

Deve-se enfatizar que o sistema desenvolvido representa o ponto de partida, que serviu de base para o estudo da modelagem de sólidos por fronteira e também para adquirir experiência no desenvolvimento de sistemas orientados a objetos.

6.2-Contribuições do Trabalho.

O desenvolvimento do protótipo do SMS teve como principal objetivo dar início ao projeto que visa a construção de um modelador de sólido híbrido B-Rep-CSG que suporte, futuramente, superfícies de forma livre. Portanto, como contribuições deste trabalho pode-se mencionar:

- * O desenvolvimento do gerenciador de representações orientado a objetos (o núcleo básico do modelador), e de algumas operações de descrição de sólido a nível de interface.**
- * A aquisição de experiência no desenvolvimento de sistemas de modelagem B-Rep e sistemas orientados a objetos, criando uma base para futuros trabalhos e estudos nesta área, no ICMSC-USP.**
- * O estudo de algumas notações e abordagens para projeto orientado a objetos.**
- * O desenvolvimento de dois documentos, sobre o projeto e sobre a implementação do protótipo, que deverão ser bastante úteis para futuras extensões do protótipo.**

Apesar de existirem diversos modeladores de sólidos, a maioria deles foi desenvolvida no exterior. Sistemas de modelagem de sólidos desenvolvidos no Brasil são poucos, e os grupos trabalhando nesta área também são poucos e na maioria recentes. Portanto, esta é uma

área de pesquisa no Brasil que necessita crescer, já que pode trazer inúmeras contribuições para a medicina, automação de projetos e análises de peças, automóveis, aviões, navios, etc...

6.3-Sugestões para Futuros Trabalhos.

Existem diversos trabalhos que podem ser realizados com base no protótipo criado, entre estes:

- .O desenvolvimento de várias operações de modelagem de alto nível sobre sólidos (como operações de modificação local e global)**

- . Implementação de um mecanismo que permita desfazer operações errôneas ou indesejáveis.**

- .O desenvolvimento de uma interface mais amigável e poderosa.**

- .A inclusão de outras formas de representação, como a CSG, juntamente com algoritmos de conversão de representações**

- .A incorporação de superfícies de forma livre no sistema.**

- . O desenvolvimento de um sistema de visualização que possua algoritmos de iluminação e remoção de superfícies escondidas.**

REFERÊNCIAS BIBLIOGRÁFICAS.

[BAU_75] Baumgart, B., *A Polyhedron Representation for Computer Vision*, In National Computer Conference, AFIPS Conf. Proc., 1975, pp. 589-596.

[BLA_90] BlaKe, H. E. and WissKirchen, P., (Eds): *Advances in Object-Oriented Graphics I*, Springer Verlag, 1990.

[BOY_82] Boyse, W. J. and Gilchris, *GmSolid: Interactive Modeling for the Design and Analysis of Solids*, IEEE CG&A, march 1982, pp. 27-40.

[BOO_86] Booch, G., *Software Engineering with ADA*, Benjaming/Cummings Publishing Company Inc., 1986.

[BOO_91] Booch, G., *Object Oriented Design with Applications*, The Benjamin/Cummings Publishing Company, Inc., 1991.

[CAS_85] Casale, S. M. and Stanton, L. E., *An Overview of Analytic Solid Modeling*, IEEE CG&A, february 1985, pp. 45-56.

[CAS_87] Casale, S.M., *Free-Form Solid Modeling with Trimmed Surface Patches*, IEEE CG&A, January 1987, pp. 33-42.

[CHI_83] Chiyokura, H. & Kimura, F., *Design of Solids with Free-Form Surfaces*, Computer Graphics, vol 17, number 3, 1983, pp. 289-298.

- [CRO_91] Crocker, G. and Reinke, W. F., *An Editable Nonmanifolds Boundary Representation*, IEEE Computer Graphics & Applications, March 1991, pp. 39-51.
- [DUF_90] Duff, C. and Howard, B., *Migration Patterns*, Byte, October 1990, pp. 223-232.
- [FIS_91] Fisher, R., *GeneSys Sistema Híbrido para Modelagem de Sólidos*, Dissertação de Mestrado, PUC-Rio, 1991.
- [GAN_86] Gane, S. and Sarson, T., *Análise Estruturada de Sistemas*, Livros Técnicos e Científicos Editora LTDA., 1986.
- [GIB_90] Gibson, E., *Objects Born and Brend*, Byte, October 1990, pp 245-254.
- [GOL_87] Goldman, N. R., *The Role of Surfaces in Solid Modeling*, Geometric Modeling: Algorithms and New Trends", SIAM, 1987.
- [GOM_90]-Gomes, M. J. e Velho, C. L., *Conceitos Básicos de Computação Gráfica*, VII Escola de Computação, 1990.
- [GRA_86] Grant, E., Amburn, P. and Whitted, T., *Exploiting Classes in Modeling and Display Software*, IEEE CG&A, november 1986, pp. 13-20.
- [HEL_90] Hellen, D., *Xview Progemming Manual for Version 11 of the Window System*, O'Reilly & Associates, Inc, 1990.
- [HEN_92] Henderson. S. B., *Object-oriented information system: An introduction tutorial*, The Australian Computer Journal, vol. 24, n.1, February 1992.

[HIL_82] Hillyard, R., *The Build Group of Solid Modelers*, IEEE CG&A, March 1982, pp. 43-52.

[HOF_89] Hoffmann, M. C., *Geometric and Solid Modeling*, KAUFMANN, 1989.

[JAR_85] Jared, G., *Boundary Representations: Data Structures, Computations, Operations and Applications*, SIGGRAPH'85 tutorial notes, pp 1-25.

[KAL_89] Kalay, E. Y., *The Hybrid-Edge: a Topological Data Structure for Vertically Integrated Geometric Modeling*, Computer-Aided Design, vol 21, n.3, 1989, pp. 130-140.

[KIM_84] Kimura, F., *Geomap-III: Designing Solids with Free-Form Surfaces*, IEEE CG&A, june 1984, pp.58-72.

[LIC_87] Lichten, L. and Samek, M., *Integrating Sculptured Surfaces into a Polyedral Solid Modeling System*, Geometric Modeling: Algorithms and New Trends, SIAM, 1987, pp. 109-122.

[MAG_93] , Magalhães C. C. L. A., *Estudo de Operações de Manipulação de Forma em Modeladores de Sólidos por Fronteira*, ICMSC-1993 (mini-dissertação de exame de qualificação).

[MAN_84] Mäntylä, M., *A Note on the Modeling Space of Euler Operators*, Computer Vision, Graphics, and Image Processing, vol 26, 1984, pp 45-61.

[MAN_88] Mäntylä, M., *An Introduction to Solid Modeling*, Computer Science Press, 1988.

[MIL_86] Miller, R. J., *Sculptured Surfaces in Solid Models: Issues and Alternatives Approaches*, IEEE CG&A, December 1986, pp. 72-87.

[MIL_89] Miller, R. J., *Architectural Issues in Solid Modelers*, IEEE CG&A, september 1989, pp. 72-86.

[MOR_85] Mortenson, E. M., *Geometric Modeling*, J. Wiley & Sons, 1985.

[MUL_89] Mullin, M., *Object Oriented Program Design with Examples in C++*, Addison-Wesley, 1989.

[NIE_92] Nielson, M. G., *Guest Editor's Introduction: Computer-Aided Design*, IEEE Computer Graphics and Applications, September 1992, pp. 29-44.

[PRE_87] Pressman, R. S., *Software Engineering: A Practitioner's Approach*, McGraw-Hill International Editions, second edition, 1987.

[REQ_80] Requicha, G. A. A., *Representation for Rigid Solids: Theory, Methods and Systems*, Computing Surveys, December 1980, pp 437-464.

[REQ_82] Requicha, G. A. A. and Voelcker, B. H., *Solid Modeling: A Historical Summary and Contemporary Assessment*, IEEE CG&A, March 1982, pp. 9-24.

[RUM_88] Rumbaugh, J. E., *Object-Oriented Modeling and Design*, Prentice-Hall, 1988.

[SAK_90] Sakurai, H. and Gossard, C. D., *Recognizing Shape Feature in Solid Models*, IEEE CG&A, September 1990, pp. 22-32.

[SCH_90] Scheifer, R. W. & Gettys, J., *X Window System*, IEEE CG&A, MASSS, USA, Digital Press, 1990.

[SCO_93a] Scott, L. P. B. e Oliveira, M. C. F., *Protótipo Evolutivo de um Sistema de Modelagem de Sólidos Orientado a Objetos - Projeto/Documentação*, Relatório Interno, ICMSC-USP, 1993.

[SCO_93b] Scott, L. P. B. e Oliveira, M. C. F., *Protótipo Evolutivo de um Sistema de Modelagem de Sólidos Orientado a Objetos - Implementação*, Relatório Interno, ICMSC-USP, 1993.

[TAK_90] Takadashi T. e Lisenberg E. H., *Programação Orientada a Objetos*, VII Escola de Computação, 1990.

[TAV_92] Tavares G., Castelo A., Lopes H., *Topological Methods in Geometric Modeling*, CNMAC, 1992.

[THA_85] Thalmann N. M., *Image Synthesis Theory and Practice*, Tokyo, Springer 1985.

[TUR_88] Turner U. J., *Accurate Solid Modeling Using Polyedral Aproximations*, IEEE CG&A, May 1988, pp. 14-28.

[VAR_84] Varady T. and Pratt M. J. , *Design Techniques for the Definition of Solid Objects with free-Form Geometry*, Computer-Aided Geometric Design, 1, 1984, pp 207-225.

[WIS_90] WissKirchen P., *Object-Oriented Graphics-from GKS and PHIGS to Object-Oriented System*, Springer-Verlag, 1990.

[WEG_90] Wegner, P, *Concepts Paradigms of Object-Oriented*, OOPSLA'90, June 1990.

[WEI_85] Weiler K., *Edge-Based Data Structures for Solid Modeling in Curved-Surface Environments*, IEEE CG&A, January 1985, pp. 2-39.

[WOO_85] Woo C. Tony, *A Combinatorial Analysis of Boundary Data Structure Schemate*, IEEE CG&A, March 1985, pp. 19-27.

[YOU_90]-Yourdon E., *Auld Lang Syne*, Byte, October 1990, pp. 257-264.

BIBLIOGRAFIA CONSULTADA

Castelo, F. A., *Aproximação Adaptativa de Variedades Implícitas com Aplicações à Modelagem Implícita e Equações Algébrico-Diferenciais*, Tese de Doutorado, PUC-Rio, Agosto, 1992.

Chiyokura, H., *An Extend Rounding Operation for Modeling Solids with Free-Form Surfaces*, IEEE Computer Graphics & Applications, December 1987, pp. 27-35.

Fisher, R., *Formas de Representação para Objetos Tridimensionais*, Dissertação de Projeto Final, PUC-Rio, 1990.

Foley, J. B., and Van Dam, A., *Fundamentals of Interactive Computer Graphics*, Addison-Wesley, 1982.

Floriani, L. and Falcidieno, B., *A Hierarchical Boundary Model*, ACM Transaction on Graphics, vol 7, number 1, January 1985, pp. 43-60.

Forrest, A. R., *Interactive Interpolation and Approximation by Bézier Polynomials*, The Computer Journal, Vol. 15, number 1, pp. 71-79.

Rogers, D. F. and Adams, J. A., *Mathematical Elements for Computer Graphics*, McGraw-Hill Inc., 1990.

Scott, S. W., *Free-Form Deformation of Solid Geometric Models*, SIGGRAPH'86, vol. 20, n. 4, 1986, pp. 245-254.

Tiler, W., *Rational B-Spline for Curve and Surface Representation*, IEEE Computer Graphics and Applications, September 1983, pp. 61-69.

Ward, P. T., *How to Integrate Object Orientation with Structured Analysis and Design*, IEEE Software, March 1989, pp. 74-82.

WilKerson, B. and Wirfs-Broock, R., *Object-Oriented Design: A Responsibility-Driven Approach*, OOPSLA'89 Proceedings, October 1989, pp. 71-76.