4
As Operações Booleanas no MG

Este capítulo visa mostrar a adaptação do algoritmo apresentado para as operações booleanas em domínios *non-manifold* ao ambiente de modelagem do modelador MG. Descrevem-se aqui todos os passos realizados para que o MG passasse a incorporar mais esta facilidade de modelagem, tornando-se um sistema mais completo e eficiente na geração de modelos para análise pelo MEF. São apresentadas as novas classes criadas, seus métodos e sua integração com o resto do código do programa.

Ressalvam-se os seguintes pontos:

- Importância do uso da POO na inserção de uma nova ferramenta de modelagem.
- Pré-processamento do modelo através do uso dos algoritmos de interseção de curvas e superfícies e de detecção de regiões.
- Utilização de facilidades pré-existentes no MG que tornaram a implementação destas operações uma tarefa bem menos árdua.
- Modificações realizadas em algumas funções presentes no código fonte do modelador e da biblioteca CGC para permitir a correta implementação das operações booleanas.
- Criação de uma nova classe responsável pela interface entre a biblioteca CGC e as operações booleanas no MG.
- Detalhes de implementação e artifícios usados para tratamento de casos particulares.
- Limitações do modelador.
- Tempo de execução do algoritmo e robustez e eficiência do mesmo.

4.1.
Modelagem geométrica usando POO

Um sistema de modelagem que se propõe a representar modelos tridimensionais complexos deve priorizar a organização do seu código fonte. Um programador não familiarizado com o ambiente de modelagem que este sistema representa deve ser capaz de compreender seu código fonte com facilidade.

Além disso, o código fonte deve estar implementado de modo que a inserção de uma nova ferramenta de modelagem seja uma tarefa praticamente isolada do resto das facilidades pré-existentes. Esta *modularização* do código fonte permite que se promova grandes modificações em partes do programa sem haver a necessidade de se reimplementar outras partes que não guardam uma relação estreita com aquelas que foram modificadas.

Neste contexto, o uso da POO se faz de grande utilidade, pois permite uma organização de classes na estrutura do modelador com base em graus de hierarquia e níveis de abstração. Um programa desenvolvido numa linguagem orientada a objetos torna-se mais simples de ser interpretado, estendido e modificado. Além disso, o uso de aplicativos que geram informações compactas, organizadas e didáticas a partir do código fonte de um programa, como por exemplo o Doxygen [47], passa a ser extremamente útil quando se trata de um programa implementado usando POO.

No desenvolvimento deste trabalho, a importância da modularização do código fonte tornou-se evidente. As operações booleanas puderam ser implementadas no MG sem que praticamente nenhuma parte do código fonte já existente precisasse ser modificada. As poucas modificações que foram feitas não representaram nenhuma mudança significativa no ambiente de modelagem, e a estrutura de dados do MG permaneceu intacta. Em linhas gerais, o que se fez foi criar novas classes que pudessem representar os novos conceitos de modelagem que estavam sendo introduzidos, como os grupos e as operações booleanas, e classes que gerenciassem a interface entre estas e o modelador MG ou a biblioteca CGC. Como no MG cada entidade topológica é representada por uma classe, como foi visto no Capítulo 2, a manipulação destas entidades pelas operações booleanas pôde ser feita sem muita dificuldade.

4.2. Descrição das novas classes criadas no MG

Para que o algoritmo de operações booleanas pudesse ser implementado no ambiente de modelagem do MG, novas classes foram criadas no intuito de permitir que os objetos das classes que representam as informações topológicas do modelo pudessem ser manipulados da forma como foi descrito no último capítulo. Todas as classes criadas derivam diretamente de uma mesma classe base já existente, *Application*, como pode ser visualizado na Figura 4.1.

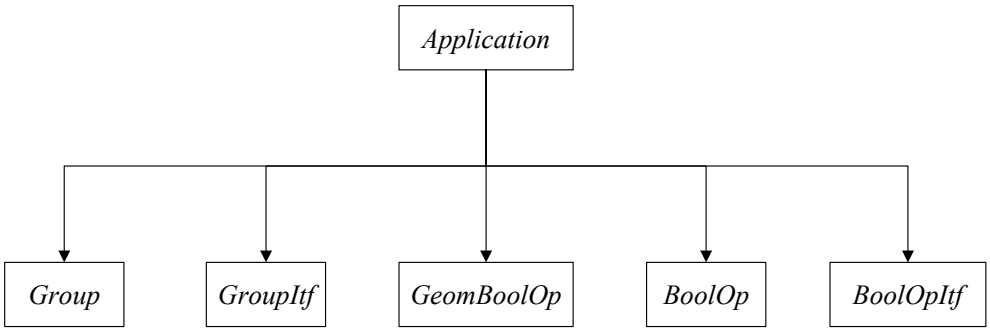


Figura 4.1 – Novas classes criadas no MG.

4.2.1. A classe *Group*

O primeiro novo conceito que precisava ser introduzido era o de *grupo*. No MG, a classe que representa grupos de entidades topológicas não foi criada exclusivamente para que as operações booleanas pudessem ser introduzidas. Outras funcionalidades presentes no modelador também puderam tirar proveito deste novo conceito.

Um objeto da classe *Group* possui simplesmente um identificador do grupo e uma lista de entidades topológicas associadas àquele grupo (Figura 4.2). Os métodos desta classe são responsáveis por operações simples de consulta e manipulação das entidades topológicas pertencentes a um grupo, como a inserção e a remoção de uma entidade, verificação da presença de uma determinada entidade num grupo, consulta ao identificador do grupo e ao número de entidades pertencentes ao mesmo.

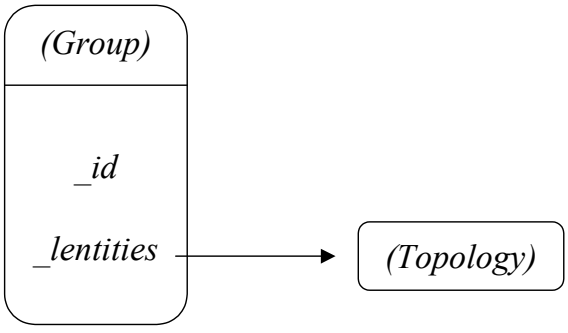


Figura 4.2 – Relações dos objetos da classe *Group*.

4.2.2. A classe *GroupItf*

Para que os grupos pudessem ser manipulados pelo usuário através da interface do MG de forma simples, amigável e eficiente, foi criada uma classe

que gerencia as informações dos grupos e a maneira como elas são representadas e manipuladas na interface do MG. Além disso, a classe *GroupItf* promove a comunicação entre a classe *Group* e a estrutura de dados do modelador.

Um objeto da classe *GroupItf* possui um ponteiro para um elemento de interface que é representado por uma árvore [48]. Esta árvore contém um nó raiz denominado *Groups*, cujos filhos são nós representando os grupos associados ao modelo que está sendo representado. As folhas desta árvore são os nós derivados diretamente dos grupos e representam as entidades topológicas pertencentes a cada grupo (Figura 4.3). Um objeto desta classe possui também uma lista de grupos existentes no modelo e uma outra lista cujos nós são registros contendo uma referência para um grupo e uma referência para uma entidade topológica (Figura 4.4). Este tipo de registro foi criado para auxiliar no gerenciamento das informações de cada grupo contido na lista de grupos.

A classe *GroupItf* possui também dois campos (variáveis) *estáticos*. Um campo *estático* pertencente a uma classe é aquele que é compartilhado por todos os objetos desta classe. Ou seja, existe uma única área de memória que armazena as informações daquele campo para todos os objetos daquela classe. No caso da classe *GroupItf*, um destes campos, denominado *curr_nod*, é um número inteiro representando o nó da árvore que está selecionado na interface (a seleção de um nó é feita pelo usuário através do *mouse*). Apenas um nó pode ser selecionado de cada vez. O outro campo estático da classe *GroupItf* representa um ponteiro para um objeto da própria classe *GroupItf* que sempre apontará para o último objeto desta classe que foi criado. Isto é feito da seguinte forma: na implementação de um *construtor* desta classe (*construtores* são funções automaticamente chamadas quando um objeto de uma classe é criado) faz-se com que este campo estático aponte para o próprio objeto que está sendo criado. Este artifício é utilizado em linguagens orientadas a objetos como uma forma de permitir que funções de *callback* de elementos de interface sejam definidas como métodos privados de alguma classe (estes métodos também devem ser estáticos) [49,50].

O construtor da classe *GroupItf* também registra as *callbacks* de interface, relativas à criação, seleção e remoção de grupos, inclusão de entidades selecionadas na área de *canvas*, remoção de entidades e rotulação (*labeling*) dos grupos criados. As Figuras 2.22 e 2.23 mostram algumas destas funcionalidades na interface do MG.

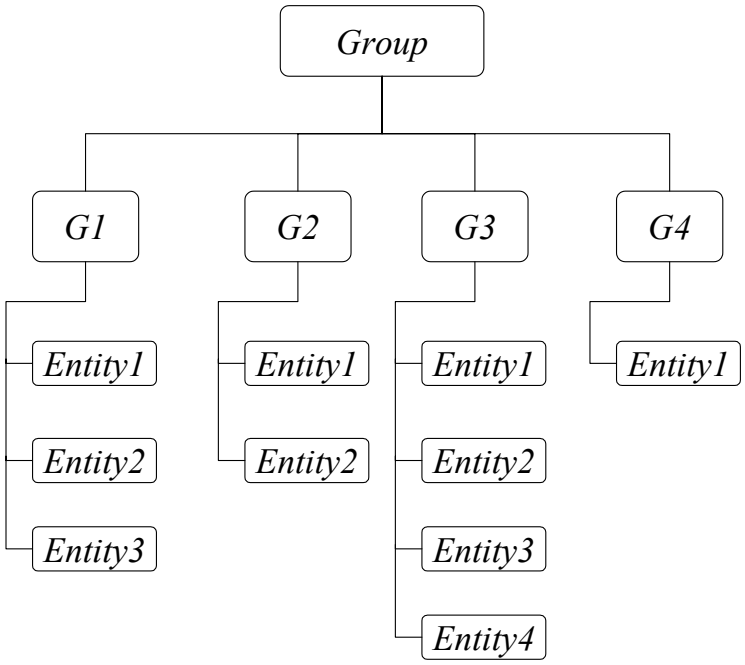


Figura 4.3 – Exemplo de organização da árvore da classe *GroupItf*.

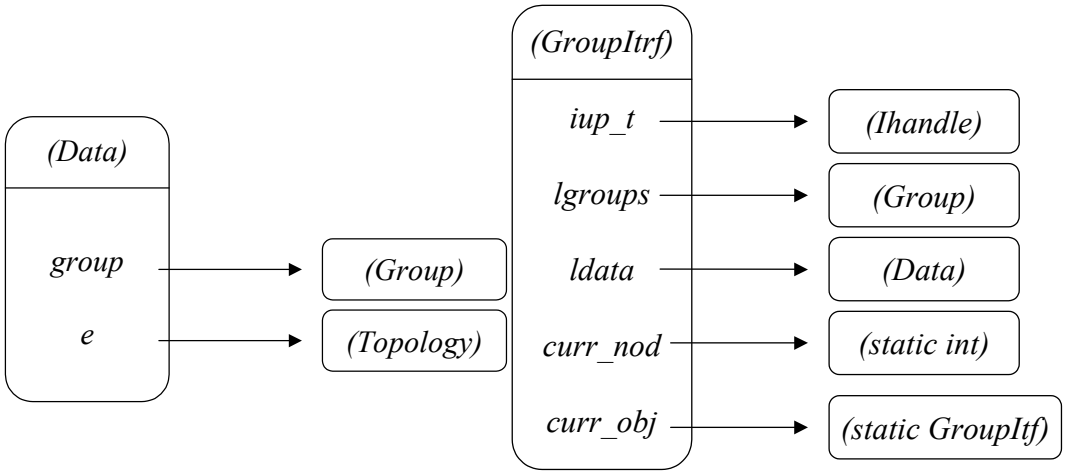


Figura 4.4 – Relações dos objetos da classe *GroupItf*.

4.2.3. A classe *GeomBoolOp*

A classe *GeomBoolOp* está diretamente relacionada com a implementação das operações booleanas no MG. Ela é responsável pelos algoritmos geométricos necessários para que as operações booleanas possam ser realizadas de forma correta.

Um objeto desta classe possui somente um ponteiro para uma lista de listas de referências para objetos da classe *Topology* (Figura 4.5). Os métodos públicos da classe *GeomBoolOp* são:

- Detecção de ponto em região.
- Detecção de ponto sobre retalho de superfície (face).
- Detecção de ponto sobre aresta.
- Detecção de ponto sobre vértice.
- Determinação das coordenadas tridimensionais de um ponto sobre um retalho de superfície a partir das suas coordenadas paramétricas sobre o retalho de superfície.
- Determinação das coordenadas tridimensionais de um ponto sobre uma aresta a partir da sua coordenada paramétrica sobre a aresta.
- Detecção de eventuais regiões formadas a partir de um conjunto de retalhos de superfícies.
- Consulta ao número de regiões formadas a partir de um conjunto de retalhos de superfícies.
- Determinação dos subconjuntos de superfícies que compõem a fronteira de cada região formada a partir de um conjunto de retalhos de superfícies.

Os métodos de detecção de ponto em região e de detecção de eventuais regiões formadas a partir de um conjunto de retalhos de superfície são os únicos que necessitam de uma comunicação entre o modelador MG e a biblioteca CGC, pois estes métodos já se encontram disponíveis nesta biblioteca. Na verdade, na versão original da CGC, apenas retalhos de superfícies poligonais eram considerados. Contudo, durante o seu trabalho, Lira [6] estendeu esta biblioteca de forma que passasse a considerar retalhos de superfície com geometria qualquer (baseada na representação NURBS).

O método de detecção de ponto sobre vértice é implementado de forma bem simples, usando a descrição geométrica do vértice topológico (ponto no espaço tridimensional associado a este vértice) e analisando-se a distância entre os dois pontos.

Os métodos de detecção de ponto sobre aresta e sobre retalho de superfície (face) também são implementados de forma bem simples, pois as classes que representam arestas e retalhos de superfície já disponibilizam

métodos que comparam a distância entre um ponto qualquer e o ponto mais próximo a ele localizado sobre a aresta ou retalho de superfície.

Os métodos de detecção de ponto em região e de detecção de ponto sobre retalho de superfície, aresta ou vértice recebem como parâmetro adicional o valor de uma tolerância, que é utilizada na comparação das distâncias entre pontos. Isto significa que este valor pode ser determinado no momento em que um destes métodos for requisitado, baseando-se em critérios relacionados ao modelo ou ao tipo de operação que está sendo realizada. Métodos baseados na geometria do modelo estão sempre sujeitos a eventuais erros numéricos provocados pelo uso de tolerâncias. A determinação de uma tolerância satisfatória para cada aplicação, modelo ou operação de interface é uma área de estudo extremamente importante em modelagem geométrica. Trabalhos relacionados ao uso de tolerâncias adaptativas aos modelos e ao uso de tolerâncias mínimas suportadas por cada máquina onde as aplicações são executadas podem ser encontrados em [51].

Os métodos de determinação das coordenadas tridimensionais de um ponto sobre uma aresta ou retalho de superfície a partir das coordenadas paramétricas do ponto sobre estas entidades também são muito simples, pois os objetos das classes que representam arestas e retalhos de superfície no MG já incorporam métodos para realizar esta tarefa.

Os objetos da classe *GeomBoolOp* possuem um campo *privado* (campo acessível somente através de métodos públicos disponibilizados pela classe) que é uma lista encadeada de listas encadeadas de entidades topológicas. Esta lista guarda as listas de retalhos de superfície que formam a fronteira de cada região detectada pelo método de detecção de regiões a partir de um conjunto de retalhos de superfície. Logo, o método de consulta ao número de regiões formadas apenas retorna o número de elementos desta lista e o método de determinação dos subconjuntos de superfícies que formam a fronteira de cada região formada apenas retorna cada uma das listas de retalhos de superfície contida nesta lista.

Vale ressaltar que quando a CGC detecta quais são os retalhos de superfície pertencentes à fronteira de cada região formada e retorna esta informação para o MG, este não *cria* efetivamente as regiões (*patches3d*) no modelo. Este é o motivo pelo qual o campo privado citado no parágrafo anterior não foi implementado como uma lista de regiões, mas como uma lista de listas de retalhos de superfície que formam a fronteira de cada região. Este método não tem como objetivo alterar as características do modelo, mas apenas

determinar quais são os retalhos de superfície que contribuem para a formação de regiões, a partir de um conjunto de retalhos de superfície presentes no modelo. Isto porque o método consiste apenas em uma etapa intermediária na aplicação das operações booleanas em um grupo de entidades topológicas.

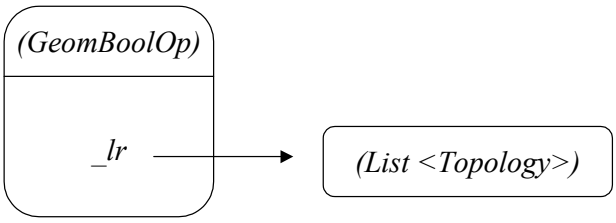


Figura 4.5 – Relações dos objetos da classe *GeomBoolOp*.

4.2.4. A classe *BoolOp*

A classe *BoolOp* é diretamente responsável pelas operações booleanas no MG. Esta é a classe cujos campos e métodos representam as variáveis e procedimentos descritos no algoritmo proposto no capítulo anterior, adaptados ao ambiente de modelagem do MG.

Na implementação desta classe, foi criado um registro auxiliar do tipo enumerável denominado *Type* representando as possíveis posições relativas no espaço de uma entidade topológica de um grupo perante as entidades topológicas do outro grupo: *AoutB*, *BoutA*, *AinB*, *BinA* e *INTERS*.

Além disso, três classes auxiliares também foram criadas: *BoolVtx*, *BoolSegment* e *BoolPatch2d*. Cada uma destas classes contém dois campos privados: um deles é um ponteiro para um objeto do tipo vértice, aresta (segmento) ou face (retalho de superfície ou *patch2d*), respectivamente. O outro é um campo do tipo *Type*, que guarda a informação sobre o grupo que contém aquele elemento topológico e a sua posição relativa (exterior, interior ou compartilhado pelos dois grupos) em relação às entidades do outro grupo.

Um objeto da classe *BoolOp* possui os seguintes campos (Figura 4.6):

- Dois ponteiros para objetos da classe *Group*, que representam os grupos sobre os quais as operações booleanas serão aplicadas (*A* e *B*).
- Duas listas encadeadas de ponteiros para vértices, que armazenam referências para os vértices de cada grupo (*lvtxA* e *lvtxB*).

- Duas listas encadeadas de ponteiros para arestas, que armazenam referências para as arestas de cada grupo (*ledgA* e *ledgB*).
- Duas listas encadeadas de ponteiros para faces, que armazenam referências para as faces de cada grupo (*lfacA* e *lfacB*).
- Duas listas encadeadas de ponteiros para regiões, que armazenam referências para as regiões de cada grupo (*lregA* e *lregB*).
- Uma lista encadeada de objetos da classe *BoolVtx*, contendo referências para todos os vértices dos dois grupos e as suas posições relativas no espaço em relação às entidades do outro grupo (*lallvtx*).
- Uma lista encadeada de objetos da classe *BoolSegment*, contendo referências para todas as arestas dos dois grupos e as suas posições relativas no espaço em relação às entidades do outro grupo (*lallseg*).
- Uma lista encadeada de objetos da classe *BoolPatch2d*, contendo referências para todas as faces dos dois grupos e as suas posições relativas no espaço em relação às entidades do outro grupo (*lallp2d*).
- Duas listas encadeadas de entidades topológicas genéricas (objetos da classe *Topology*), que armazenam referências para as entidades topológicas comuns aos dois grupos e para as entidades topológicas que deverão ser removidas do modelo após a aplicação das operações booleanas sobre os grupos (*lcommon* e *ldelent*).

Alguns detalhes de implementação relativos aos métodos privados que manipulam estas informações serão apresentados posteriormente. Os métodos públicos disponibilizados pela classe *BoolOp* são:

- Método *addgroups*, que recebe como parâmetros referências para os dois grupos e armazena estas referências em *A* e *B*. Este método se encarrega da chamada de métodos privados responsáveis pelo armazenamento das informações provenientes dos grupos nas respectivas listas, e pela classificação das entidades topológicas de um grupo em relação às entidades do outro grupo.
- Métodos *boolUnion*, *boolInters* e *boolDiff*, que são responsáveis pela aplicação das operações booleanas de união, interseção e diferença sobre os grupos *A* e *B*.
- Método *getDelList*, que retorna a lista de entidades topológicas a serem removidas após a aplicação das operações booleanas.

A classe *BoolOp* também disponibiliza um construtor que recebe referências para dois grupos como parâmetros. Este construtor chama o método *addgroups* descrito acima.

Ressalta-se que o método *BoolDiff* executa a operação booleana de diferença entre dois grupos de entidades na ordem em que os grupos foram passados como parâmetros para a função *addgroups*, ou seja, este método faz a operação *A – B*.

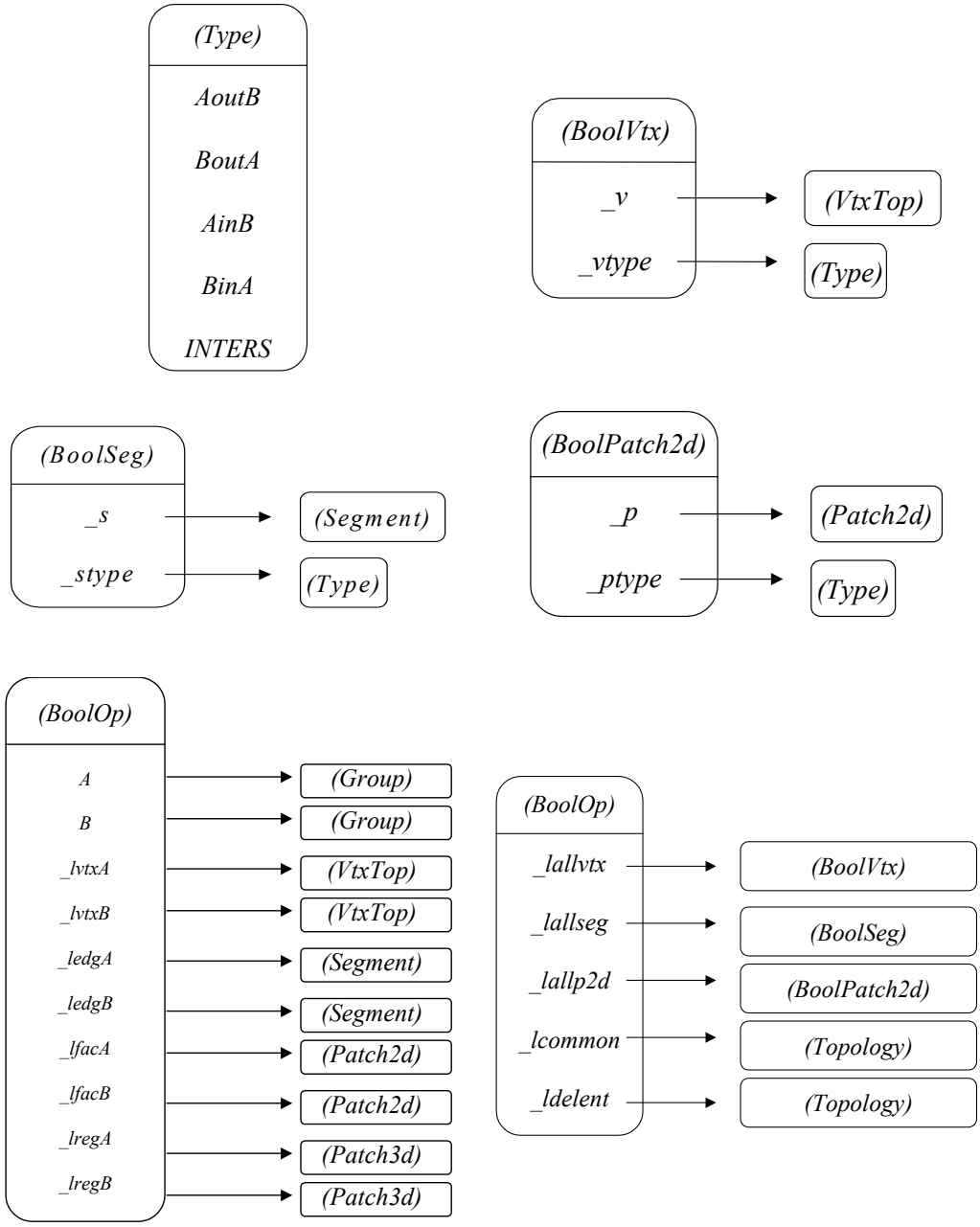


Figura 4.6 – Relações dos objetos da classe *BoolOp*.

4.2.5.
A classe *BoolOpItf*

Analogamente à classe *GroupItf*, a classe *BoolOpItf* gerencia as informações relacionadas às operações booleanas e a maneira como elas são representadas e manipuladas na interface do MG, buscando manter a simplicidade e a eficiência na interação com o usuário. Além disso, a classe *BoolOpItf* promove a comunicação entre a classe *BoolOp* e a estrutura de dados do modelador.

Um objeto da classe *BoolOpItf* possui ponteiros para uma série de elementos de interface: um *label* para orientar o usuário durante a seleção ou a remoção de grupos, uma lista com os identificadores dos grupos existentes, uma lista com os identificadores dos grupos já selecionados pelo usuário e três botões, um para adicionar um grupo à lista de entidades selecionadas, um para remover um grupo desta lista e outro para inverter a ordem de seleção dos grupos. Todos estes elementos de interface fazem parte de um diálogo definido dentro da *callback* de um botão presente na interface do MG responsável pela manipulação dos grupos que serão combinados por meio das operações booleanas (esta *callback* também é um método da classe *BoolOpItf*). Além disso, um objeto desta classe armazena o número de grupos já selecionados, um *flag* para indicar se a regularização é requerida ou não, uma lista encadeada de ponteiros para os grupos selecionados pelo usuário, um ponteiro para um objeto da classe *BoolOp* e um campo estático representado por um ponteiro para um objeto da própria classe *BoolOpItf*, que estará sempre apontando para o último objeto desta classe que foi criado durante a execução do programa (Figura 4.7).

Quatro botões presentes na interface do MG para permitir ao usuário a manipulação dos grupos que serão combinados e das operações booleanas que serão executadas possuem *callbacks* que são métodos privados da classe *BoolOpItf*. Ressalta-se novamente que isto é possível graças ao artifício de utilização de um campo estático da classe que aponte sempre para o último objeto criado. Desta forma, pode-se manipular o objeto dentro de uma *callback* de interface.

O primeiro botão, citado no parágrafo anterior, é responsável pela criação e exibição de um diálogo que contém os elementos de interface tidos como campos privados da classe *BoolOpItf*. Ou seja, este diálogo contém duas listas, uma com grupos existentes no modelo corrente e outra com os grupos já selecionados pelo usuário até o momento de exibição do diálogo. Além disso, os

três botões mencionados permitem ao usuário inserir, remover ou alterar a ordem de seleção dos grupos contidos na lista de grupos selecionados, garantindo contudo que o número de grupos selecionados não seja maior que dois.

Os outros três botões presentes na interface do MG são responsáveis pela chamada das operações booleanas. Ou seja, se o número de grupos selecionados for igual a dois e um destes três botões for acionado, todos os procedimentos descritos no algoritmo proposto neste trabalho são realizados com as entidades presentes em cada grupo. Estes elementos de interface podem ser visualizados na Figura 4.8.

As operações booleanas são executadas chamando-se os métodos apropriados da classe *BoolOp* a partir de um objeto desta classe manipulado pela classe *BoolOpItf*. Ao final de uma operação booleana, o objeto corrente da classe *BoolOpItf* é responsável pela remoção das entidades topológicas presentes na lista de entidades a serem removidas, através de um método desta classe que promove a comunicação das operações booleanas com a estrutura de dados do modelador.

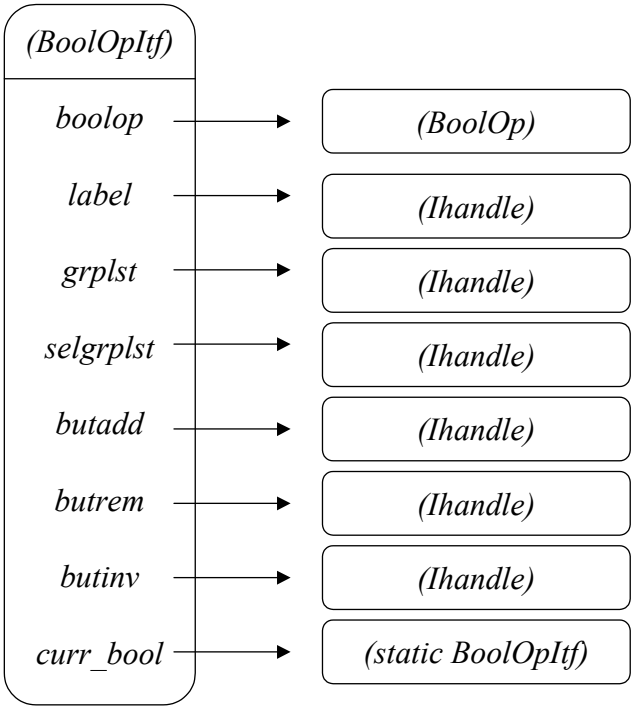
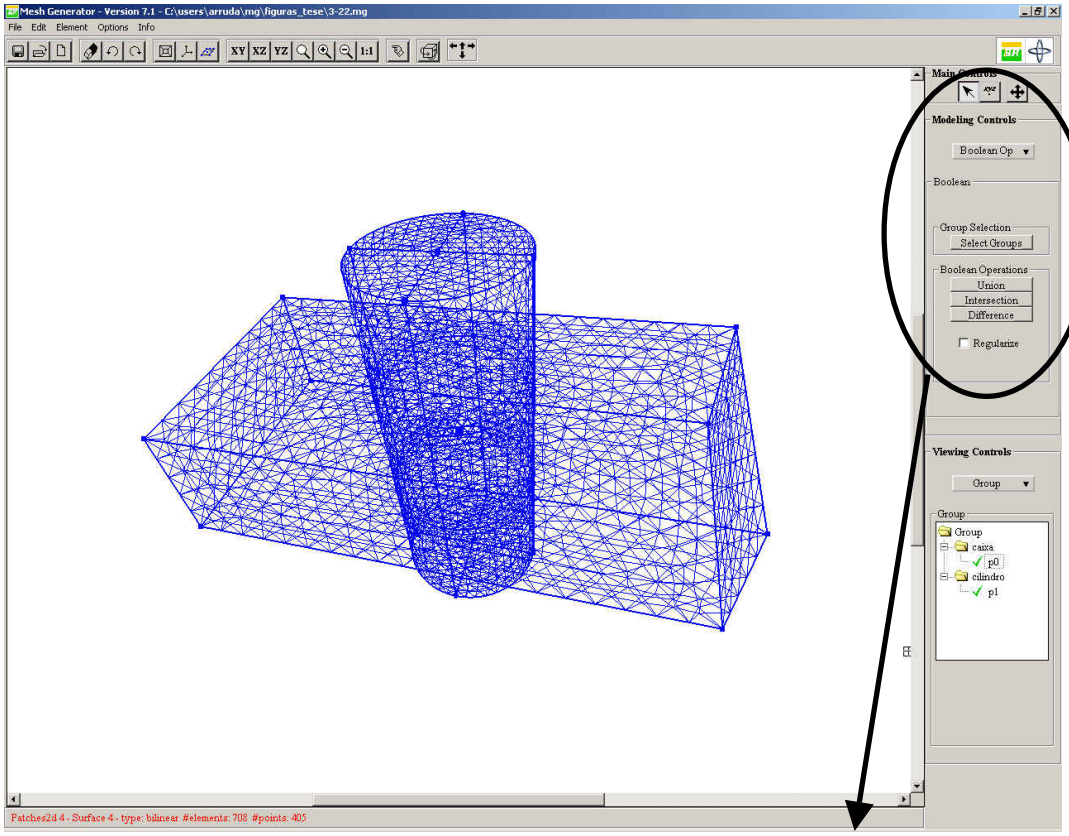


Figura 4.7 – Relações dos objetos da classe *BoolOpItf*.



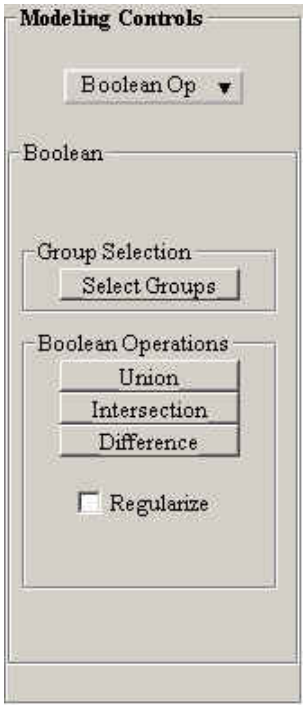
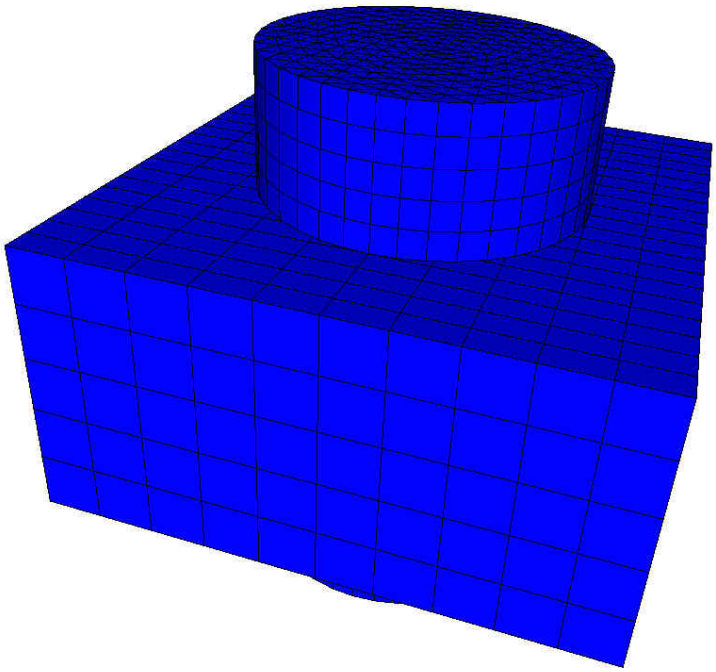


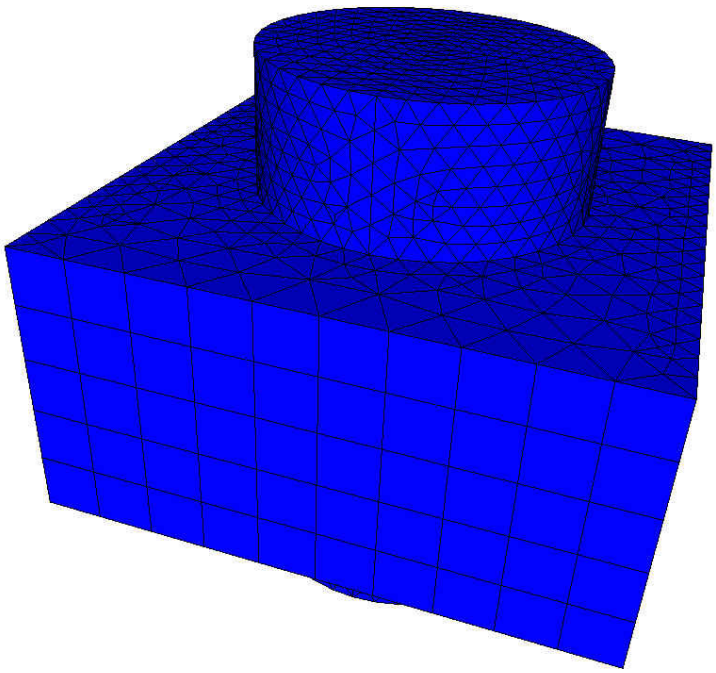
Figura 4.8 – Elementos de interface responsáveis pela chamada do algoritmo de operações booleanas.

4.3.
Pré-processamento dos parâmetros de entrada

No momento em que o usuário seleciona as entidades que irão fazer parte de cada grupo no intuito de realizar operações booleanas entre grupos, ele deve estar ciente de que o algoritmo só funcionará devidamente se forem obedecidas as suas condições de aplicabilidade, como foi visto no capítulo anterior. Isto significa que todas as interseções entre os elementos topológicos que irão participar dos grupos já deverão ter sido previamente calculadas, gerando novos elementos topológicos que apenas se tocam, e que efetivamente deverão ser passados para os grupos. A Figura 4.9 ilustra um modelo gerado no MG antes e depois do cálculo das interseções entre os elementos.



(a)



(b)

Figura 4.9 – Modelo gerado no MG: a) antes da chamada do algoritmo de interseção entre superfícies; b) depois de calculadas as interseções.

Além disso, se o usuário deseja que determinadas regiões façam parte dos grupos, então estas já deverão ter sido reconhecidas, para que se possa efetivamente passar elementos topológicos do tipo *Patch3d* (regiões) para os grupos. Do contrário, se apenas os retalhos de superfícies que formam a fronteira destas regiões forem selecionados, o algoritmo irá encarar estes retalhos de superfícies apenas como faces soltas no espaço, ignorando o fato de que formam um conjunto conexo e fechado de faces. A Figura 3.3 ilustra um exemplo desta diferença conceitual.

Neste ponto é importante se fazer uma ressalva: a interface do MG permite que o usuário escolha entre o reconhecimento automático de regiões ou o reconhecimento de regiões através da seleção explícita pelo usuário dos retalhos de superfície que formam a fronteira de uma região (Figura 4.10). A princípio, a vantagem da utilização de um ou outro procedimento depende do modelo que está sendo analisado e dos objetivos do usuário. Se é necessário que todas as regiões presentes no modelo sejam reconhecidas, ou se a seleção dos retalhos de superfície que formam a fronteira de uma ou mais regiões for muito complicada, então o reconhecimento automático de regiões torna-se bem mais útil e prático. No entanto, se num modelo grande e complexo se deseja efetuar o reconhecimento de regiões somente numa porção pequena do modelo, então a opção pelo reconhecimento manual de regiões é justificada pela economia de tempo no processamento das informações. Além disso, pode acontecer de se desejar tratar um conjunto conexo e fechado de retalhos de superfície apenas como uma casca, como foi citado acima.

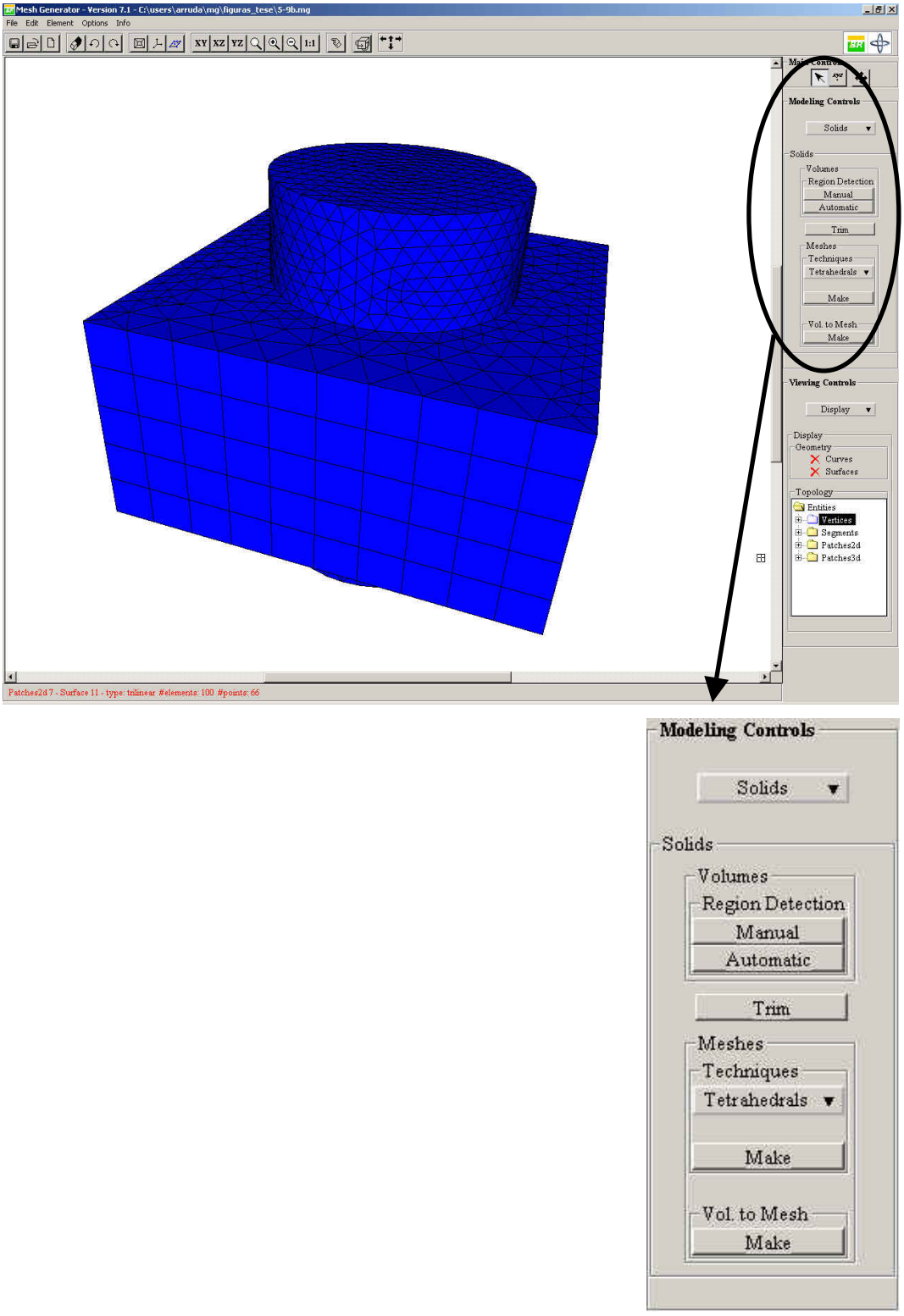


Figura 4.10 – Reconhecimento de multi-regiões: detecção automática ou seleção explícita dos retalhos de superfície que compõem a fronteira de cada região.

Porém, além dessas existe uma outra diferença entre um e outro procedimento, que está relacionada com o tratamento das informações de adjacência dos retalhos de superfície que formam uma ou mais regiões. No

reconhecimento automático de regiões, estes retalhos são passados do MG para a biblioteca CGC, que devolve ao MG as regiões formadas. Já no reconhecimento manual de regiões, o MG se encarrega de consultar as informações de adjacência na sua própria estrutura de dados para determinar a existência de regiões fechadas que utilizem somente os retalhos de superfícies selecionados pelo usuário.

Se um conjunto *A* conexo e fechado de retalhos de superfície formar uma região, mas no interior desta região houver um conjunto *B* formado por um ou mais retalhos de superfície pendentes ou soltos, caso o usuário selecione explicitamente apenas os retalhos de superfície do conjunto *A* e opte pelo reconhecimento manual de regiões, a região formada terá apenas os retalhos de superfície do conjunto *A* como parte da sua fronteira, já que neste procedimento o MG considera somente os retalhos de superfície passados como parâmetro no reconhecimento de regiões. Contudo, se o usuário optar pelo reconhecimento automático de regiões, a biblioteca CGC retornará ao MG a informação de que todos os retalhos de superfície do conjunto *A* e do conjunto *B* compõem a fronteira da região formada. Como os objetos da classe *Patch3d* do MG possuem uma única lista com as referências dos retalhos de superfície que compõem a sua fronteira (identificados pela CGC), sem informações sobre o fato de um determinado retalho de superfície ser pendente ou solto no interior daquela região, então haverá uma inconsistência entre a informação de fronteira desta região e o conceito de fronteira apresentado neste trabalho (seção 3.2), caso esta região venha a fazer parte de algum grupo utilizado na execução das operações booleanas. Para que isto não aconteça, é necessário que se adote o seguinte procedimento: caso uma determinada região seja requisitada para fazer parte de um grupo que servirá de parâmetro de entrada para as operações booleanas, deve-se selecionar explicitamente os retalhos de superfície da sua fronteira e efetuar o reconhecimento manual de regiões, sem que as estruturas internas a esta região, pendentes ou soltas, sejam selecionadas. Se o usuário desejar que as estruturas internas à região façam parte daquele grupo, então ele deve selecioná-las explicitamente e acrescentá-las ao grupo. Obviamente, se as regiões que se deseja reconhecer não possuírem estruturas internas e o usuário quiser optar pelo reconhecimento automático de regiões, isto não acarretará problema algum. Uma das soluções para este problema seria aquela apresentada no capítulo anterior, ou seja, uma modificação na interface entre o MG e a biblioteca CGC de modo que estruturas internas pendentes ou soltas

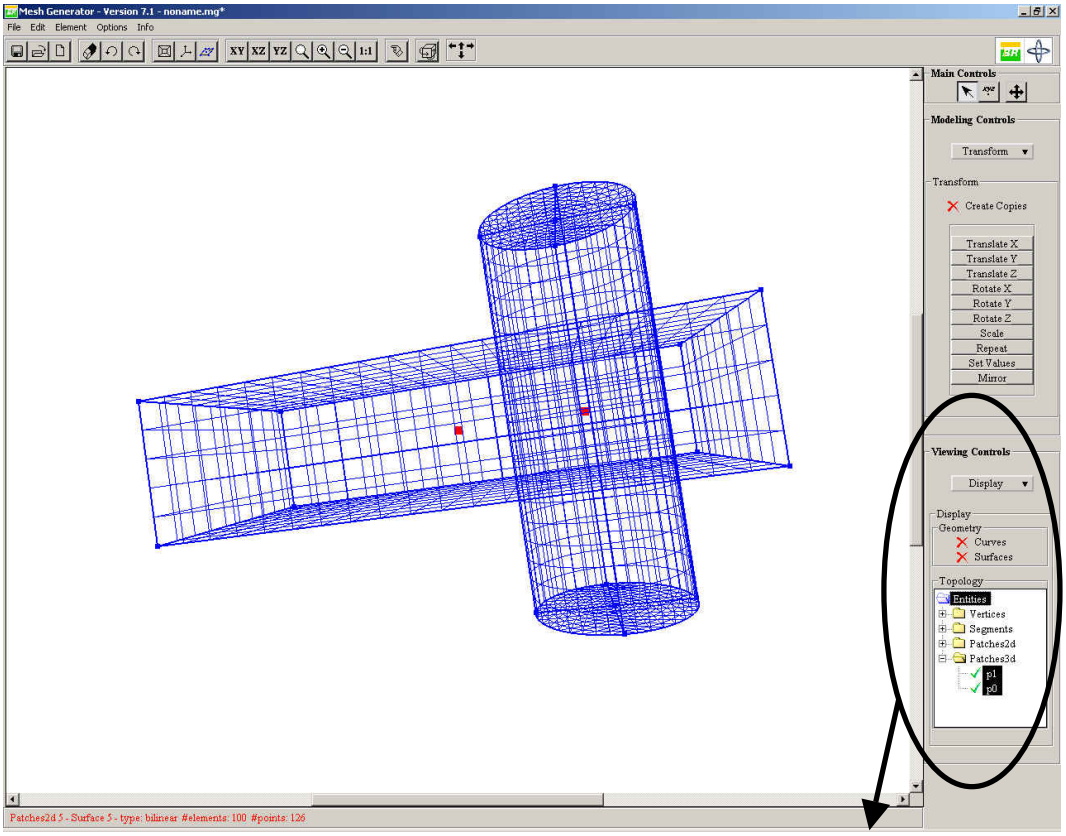
sejam identificadas e diferenciadas das outras. Infelizmente, até a conclusão deste trabalho, esta modificação ainda não havia sido implementada.

Uma modificação realizada no código fonte do MG durante o desenvolvimento deste trabalho diz respeito ao tratamento de regiões que se interceptam quando o algoritmo de interseção é chamado. Originalmente, durante a execução do algoritmo de interseção, o modelador destruía os *patches3d* cujas fronteiras se interceptavam, removendo-os da estrutura de dados. Se após a interseção entre curvas e superfícies se desejasse que as regiões continuassem existindo, o algoritmo de detecção de regiões deveria ser novamente chamado. Para que as novas regiões comportassem exatamente os mesmos volumes que as regiões originais era necessário que o usuário realizasse a detecção manual de regiões, selecionando explicitamente os novos retalhos de superfície que constituíam a fronteira das regiões. Este procedimento podia não ser trivial em alguns casos, devido à presença de retalhos de superfície difíceis de serem selecionados. No entanto, se não houvesse necessidade de que as novas regiões compreendessem os mesmos volumes que as regiões originais, o algoritmo para detecção automática de regiões poderia ser chamado.

Como as operações booleanas normalmente são aplicadas sobre regiões que se interceptam, trechos do código fonte do modelador foram modificados de forma que após o término do algoritmo de interseção os *patches3d* originais não fossem removidos, mas apenas tivessem a sua lista de *patches2d* adjacentes atualizada. Desta forma, ao manipular objetos sólidos no ambiente de modelagem do MG, o usuário passou a contar com duas opções: realizar o reconhecimento de regiões *antes* de calcular as interseções entre os retalhos de superfícies e arestas que formam a fronteira das regiões, ou calcular as interseções *antes* do reconhecimento de regiões, podendo em seguida selecionar manualmente os retalhos de superfície da fronteira de cada região ou optar pelo reconhecimento automático de regiões. As Figuras 4.11 e 4.12 ilustram a interface para as diferentes formas de se tratar o problema.

No Capítulo 2, foi comentada uma limitação do MG em relação ao cálculo da interseção entre superfícies. Quando um retalho de superfície se superpõe a outro, sem que as arestas dos seus contornos coincidam, o algoritmo de interseção falha em alguns casos. Este problema pode estar relacionado com o fato de que as próprias malhas das superfícies existentes são usadas como suporte para a definição das curvas de interseção. Quando estas curvas de interseção coincidem com as próprias arestas do contorno de uma das

superfícies, é possível que o algoritmo de interseção encontre problemas na determinação dos pontos de interseção das arestas do retalho de superfície mais interno contra o outro retalho de superfície, já que neste caso existem infinitos pontos de interseção. Assim, a determinação das curvas e regiões de *trimming* poderia ficar comprometida, e conseqüentemente também a reconstrução das malhas.



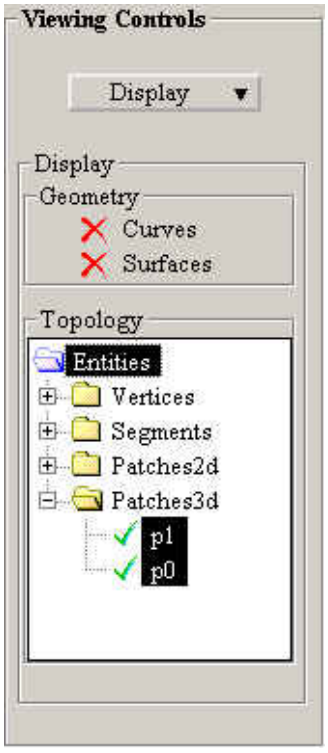


Figura 4.11 – Detecção de regiões antes da interseção.

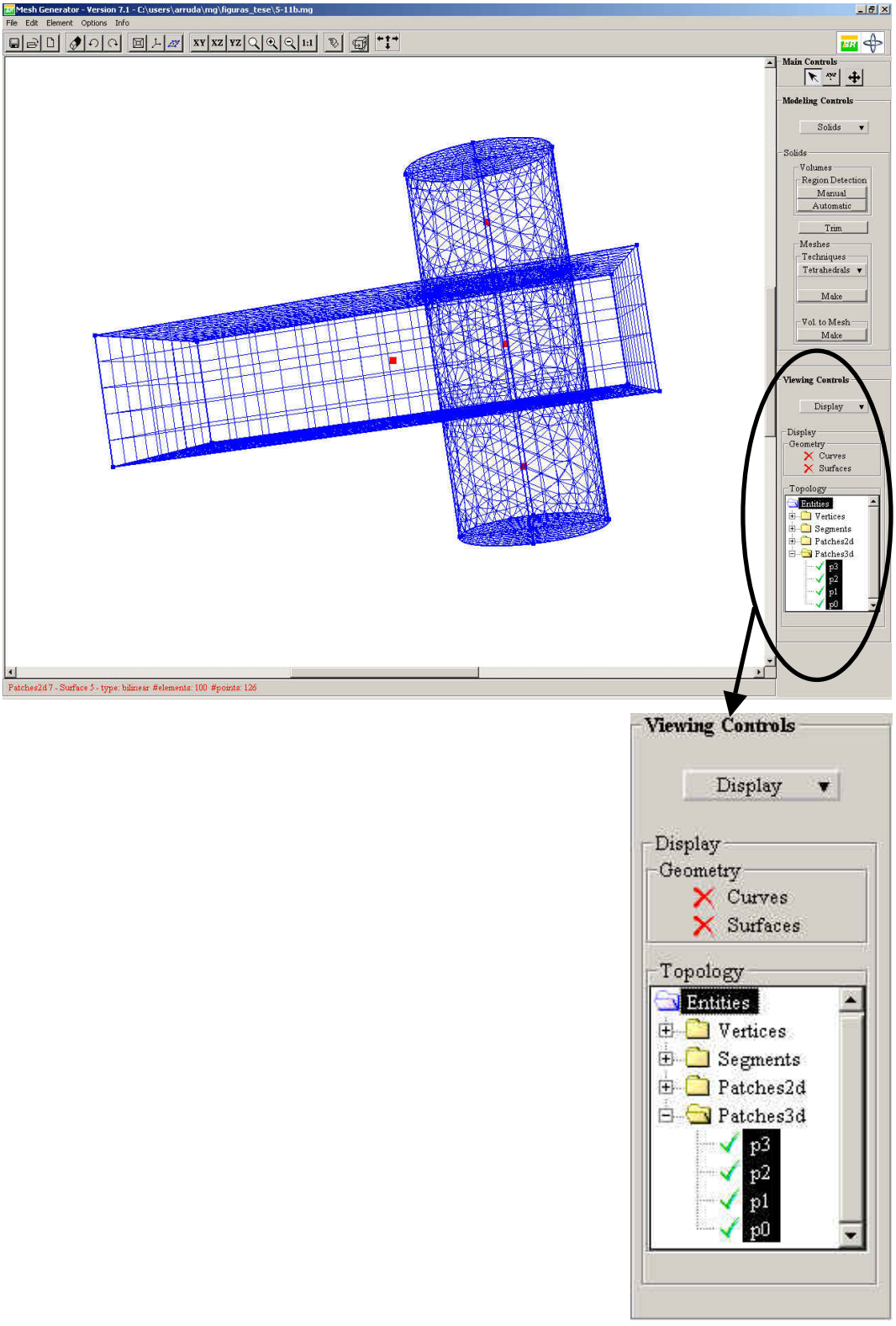


Figura 4.12 – Detecção de regiões depois da interseção.

4.4. Detalhes de implementação das operações booleanas no MG

A forma como o algoritmo de operações booleanas foi implementado no MG necessita de um maior detalhamento. É preciso que se compreenda melhor a maneira como as informações teóricas foram adaptadas para o contexto deste modelador, usufruindo facilidades já existentes que fizeram desta adaptação uma tarefa mais simples. Além disso, a forma como algumas informações de adjacência são armazenadas na estrutura de dados do MG também influenciou bastante na implementação do algoritmo da forma com ele foi descrito. Em alguns casos, houve necessidade de se realizar pequenas modificações em alguns passos do algoritmo original, mas que não comprometeram de forma muito significativa o resultado esperado da aplicação das operações booleanas sobre grupos de entidades topológicas. Algumas limitações do modelador também influenciaram o resultado final das operações booleanas, mas somente para um número reduzido de casos.

4.4.1. Armazenamento das informações dos grupos

Quando uma operação booleana é requisitada pelo usuário através de um dos botões da interface, caso já existam dois grupos de entidades topológicas selecionados, referências para estes são passadas como parâmetros para o método *addgroups* da classe *BoolOp*. O primeiro passo do algoritmo é o armazenamento de referências para todas as entidades topológicas explicitamente contidas nos grupos e para todas as entidades topológicas hierarquicamente inferiores a estas que podem ser obtidas por relações de adjacência.

A princípio, da forma como o algoritmo foi proposto, *loops* internos de faces constituídos por seqüências alternadas de vértices e arestas fechando um ciclo, vértices soltos no interior de faces, arestas soltas ou pendentes no interior de faces ou de regiões, cascas no interior de regiões constituídas por conjuntos conexos e fechados de faces e faces soltas ou pendentes no interior de regiões deveriam ser automaticamente armazenados. Contudo, na prática, a estrutura de dados do MG não permite que se obtenha todas estas entidades apenas por relações de adjacência.

O MG permite que se tenha acesso, através de relações de adjacência, às arestas pendentes ou soltas no interior de faces, bem como às arestas e vértices

pertencentes a um *loop* interno de uma face formando um ciclo fechado, contudo não há ainda suporte para vértices soltos no interior de faces. Se houver interesse na inclusão destes vértices em um dos grupos, eles devem ser explicitamente selecionados. Arestas ou faces pendentes ou soltas no interior de regiões, como já foi dito, também devem ser explicitamente selecionadas para que façam parte de um grupo. Vale lembrar que as regiões que contêm estas estruturas internas devem ser reconhecidas manualmente, por meio da seleção das faces que compõem a sua *fronteira*, da forma como ela foi definida neste trabalho na seção 3.2 (sempre que o termo *fronteira* for mencionado, deve-se ter em mente a definição adotada neste trabalho).

4.4.2. Classificação das entidades topológicas dos grupos

Esta é uma das etapas do algoritmo que teve de ser adaptada para que pudesse ser implementada no ambiente de modelagem do MG sem gerar problemas durante a determinação do resultado de uma operação booleana.

No MG, bem como na maioria dos modeladores que utilizam uma representação de fronteira, a remoção de uma entidade topológica de um modelo pode ocasionar a remoção automática de outras entidades topológicas adjacentes à que foi removida pela estrutura de dados do modelador, para manter a consistência topológica do modelo. Assim, a remoção de um vértice provoca a remoção automática de todas as arestas e faces incidentes naquele vértice. A remoção de uma aresta provoca a remoção automática do conjunto de todas as faces que compartilham a aresta, constituído pelas faces em cuja fronteira esta aresta está localizada e pelas faces que possuem esta aresta pendente ou solta no seu interior. Além disso, a remoção de uma aresta provoca também a remoção automática dos seus vértices extremos, a não ser que existam outras arestas incidentes nestes vértices. A remoção de uma face provoca a remoção automática de todas as regiões em cuja fronteira esta face está localizada.

Exceto pela remoção automática dos vértices extremos de uma aresta quando esta é removida (desde que não haja outras arestas incidindo sobre estes vértices), os outros procedimentos automáticos descritos quando uma entidade topológica é removida não influenciariam os passos do algoritmo, já que foi frisado que a ordem de remoção das entidades topológicas deve ser precisamente aquela que foi descrita, ou seja, primeiramente removem-se as

faces, depois as arestas e depois os vértices. A remoção automática dos vértices extremos de uma aresta removida, contudo, faz com que se tenha de ter cuidado na etapa de inclusão dos vértices a serem removidos na lista *ldelent*. Quando esta lista é passada para o objeto da classe *BoolOpItf* responsável pela interface entre as operações booleanas e o MG, para que ele informe para a estrutura de dados do modelador que as entidades topológicas contidas nesta lista devem ser removidas, é extremamente recomendável que não sejam passadas referências de entidades que já não fazem parte da estrutura de dados, pois isto pode ocasionar problemas, como invasão de memória. Ao mesmo tempo, esta remoção automática dos vértices de uma aresta removida deveria de certa forma simplificar os procedimentos descritos no algoritmo, pois todo vértice que possui arestas incidentes não precisaria ser armazenado nem classificado. Mas na verdade esta simplificação pode comprometer o resultado das operações booleanas em algumas situações, como pode ser visto na Figura 4.13. Como situações como estas são menos comuns de ocorrerem, optou-se por efetuar esta simplificação, classificando-se somente os vértices que não possuem arestas incidentes. Estes são os vértices que poderão ser passados para a lista *ldelent* para serem explicitamente removidos.

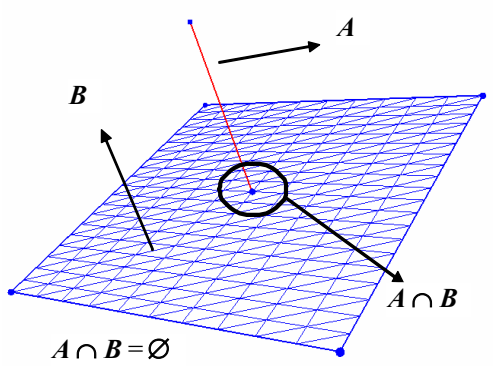
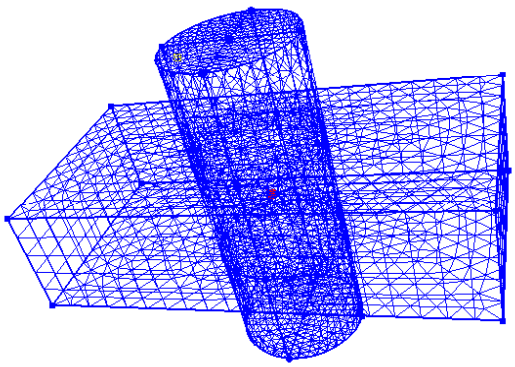


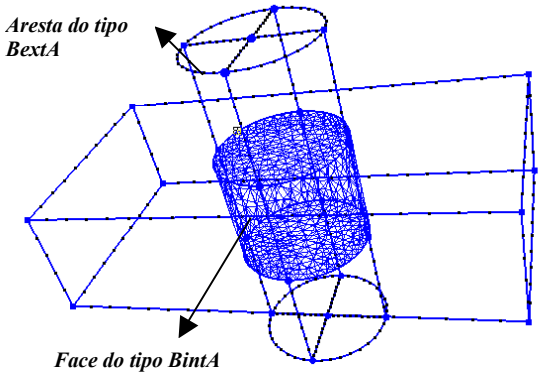
Figura 4.13 – Interseção entre duas entidades topológicas A e B. O resultado da interseção é o vértice comum a ambas as entidades, mas que não pode ser representado devido à remoção automática dos vértices de uma aresta quando ela é removida.

Existe ainda um outro problema relacionado com a remoção das arestas após a aplicação das operações booleanas sobre os grupos. No MG, a descrição geométrica de um retalho de superfície (face) está vinculada com as arestas da sua fronteira, pela própria maneira como ele é gerado. As formas de geração de

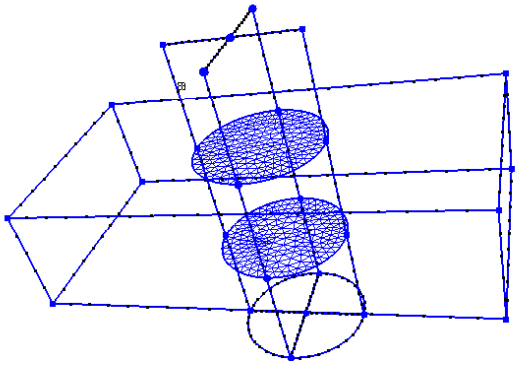
retalhos de superfície no MG foram descritas no Capítulo 2. Dessa forma, quando as interseções entre os retalhos de superfície são calculadas gerando assim novos retalhos de superfície, estes novos retalhos de superfície possuem a mesma descrição geométrica dos retalhos originais que os geraram, ou seja, eles dependem da descrição geométrica das arestas que formavam a fronteira dos retalhos de superfície originais. As arestas das fronteiras destes novos retalhos que foram geradas pela subdivisão das arestas das fronteiras dos retalhos originais contêm as mesmas informações geométricas que elas. Porém, existem arestas da fronteira dos retalhos originais que passam a não fazer parte da fronteira dos novos retalhos. Estas arestas guardam informações necessárias para a descrição geométrica dos novos retalhos de superfícies. Assim, se elas forem removidas, automaticamente os retalhos de superfícies que dependem delas para a completa caracterização da sua geometria serão removidos. Este processo pode ser visualizado na Figura 4.14. O problema maior neste caso é que situações como estas podem ocorrer com bastante freqüência, pois um retalho de superfície pertencente a um grupo pode ser classificado quanto à sua localização espacial em relação às entidades do outro grupo de forma diferente das arestas que compunham a fronteira do retalho de superfície original cuja subdivisão deu origem a este retalho. A Figura 4.14 ilustra um exemplo simples de situação em que isto ocorre.



(a)



(b)



(c)

Figura 4.14 – Influência das curvas geradoras de um retalho de superfície na sua descrição geométrica: a) dois sólidos; b) interseção entre os sólidos; c) remoção de quatro arestas ocasionando remoção automática dos retalhos de superfície.

Tendo em vista este problema, a maneira mais simples de tratá-lo, sem a necessidade de se implementar grandes artifícios para descobrir quais são as arestas responsáveis pela descrição geométrica de cada retalho de superfície, é a manutenção de todas as arestas pertencentes aos dois grupos que fizerem

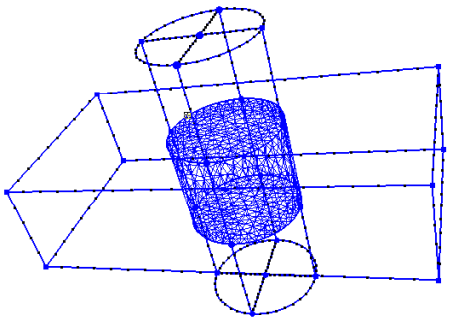
parte da fronteira de alguma face, independentemente do fato desta face ter sido removida ou não. A princípio, isto pode parecer uma simplificação grosseira, mas visualmente esta opção ainda é melhor do que deixar somente as arestas que carregam sozinhas uma parte da descrição geométrica de alguma face que não foi removida e que não fazem parte da fronteira desta face. Uma comparação entre estas duas possibilidades pode ser vista na Figura 4.15. A manutenção de todas as arestas dá uma visão geral do problema, permitindo que o usuário tenha uma noção do contorno dos retalhos de superfície originais antes da aplicação das operações booleanas, em contraste com os retalhos de superfície remanescentes no resultado destas operações.

Com esta simplificação adotada, as únicas arestas que precisam ser classificadas são aquelas que não pertencem à fronteira de nenhuma face existente nos seus respectivos grupos (explicitamente ou implicitamente através da fronteira de uma região). Ou seja, as arestas que precisam ser classificadas são aquelas que se encontram pendentes ou soltas no interior de faces ou regiões, aquelas que se encontram pendentes exteriormente a alguma face ou região, e as que se encontram soltas no espaço tridimensional exteriormente a todas as regiões.

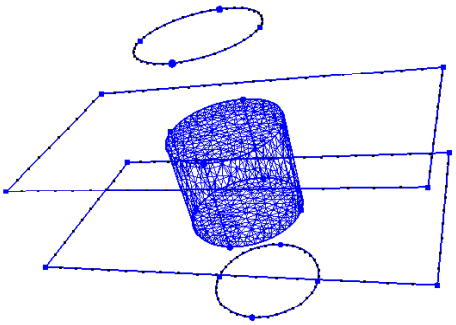
Arestas pendentes ou soltas no interior de faces fazem parte de uma lista encadeada de arames, denominada *lwire*, que faz parte da estrutura de dados de uma face (*Patch2d*). As arestas da fronteira de uma face fazem parte de uma outra lista, *lsegments*. Contudo, as arestas dos *loops* internos a uma face constituídos por ciclos fechados e alternados de arestas e vértices também fazem parte da lista *lwire*, o que não é interessante do ponto-de-vista do algoritmo proposto, já que estas arestas efetivamente fazem parte da fronteira desta face. Como não há como diferenciar as arestas pendentes ou soltas no interior de uma face das arestas de fronteira dessa face que pertencem a um *loop* interno, já que todas elas pertencem à lista *lwire*, as arestas de fronteira que pertencem a um *loop* interno também são classificadas juntamente com as outras, a não ser que elas façam parte da fronteira de uma outra face (Figura 4.16). Isto só ocasionaria algum problema no resultado de uma operação booleana se houvesse alguma situação em que as arestas de fronteira do *loop* interno de uma face tivessem de ser removidas sem que a face tivesse sido removida, pois a remoção destas arestas faria com que a face fosse automaticamente removida. Esta situação poderia ocorrer, por exemplo, numa operação de diferença do tipo *A – B* em que a remoção de uma determinada aresta do tipo *BinA* ou *INTERS* está condicionada ao fato dela fazer parte ou não

da fronteira de alguma face remanescente (Capítulo 3). Neste caso, as arestas de fronteira do *loop* interno de uma face remanescente que fossem classificadas desta forma, por serem consideradas equivalentes a arestas soltas no interior da face, seriam passadas para a lista *ldelent* erradamente, fazendo com que a face fosse automaticamente removida. Uma situação deste tipo é ilustrada na Figura 4.17. No entanto, vale lembrar que o mesmo problema ocorreria no caso de arestas efetivamente pendentes ou soltas no interior da face, já que todas estas arestas estão sendo consideradas equivalentes e recebendo o mesmo tratamento. Logo, o problema maior está no vínculo existente entre arestas localizadas no interior de faces e as faces em si. Uma solução para este problema seria alterar o código fonte do MG de tal forma que quando uma restrição interna a um retalho de superfície fosse removida, o único efeito fosse a reconstrução da malha associada àquele retalho de superfície, sem que este fosse removido.

A classificação das faces de cada grupo não sofre restrição alguma, nem cria a necessidade de se fazer adaptações ao algoritmo proposto para poder ser realizada.



(a)



(b)

Figura 4.15 – Comparação entre duas possibilidade de abordagem: a) manutenção de todas as arestas; b) manutenção somente das arestas que guardam sozinhas informações geométricas sobre os retalhos de superfície.

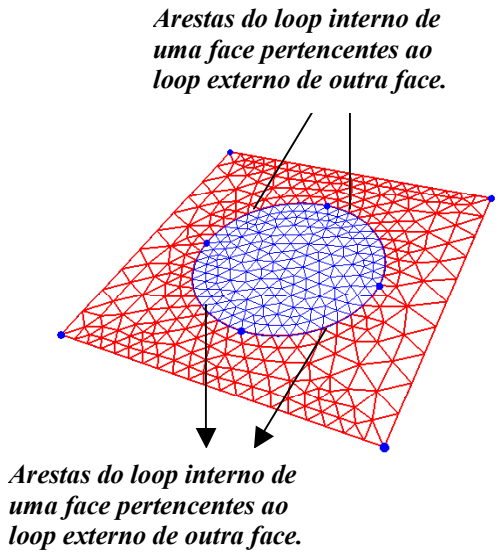


Figura 4.16 – Arestas no interior de uma face formando um *loop* interno fechado.

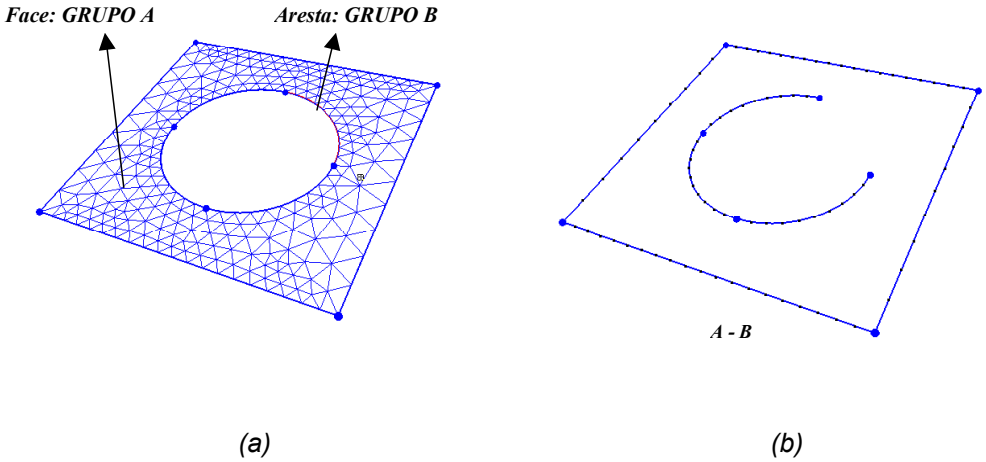


Figura 4.17 – Aresta de fronteira sendo tratada como aresta solta no interior da face.

4.4.3. Operações Booleanas e Regularização do Resultado

Tendo em vista as limitações do modelador expostas nas seções anteriores e as adaptações ao algoritmo para que ele pudesse ser implementado no MG da forma mais fiel possível à sua descrição original, é interessante verificar se os métodos públicos da classe *BoolOp* responsáveis pelas operações booleanas de união, interseção e diferença e o método público desta mesma classe responsável pela regularização do resultado também precisaram ser implementados realizando-se modificações ou aproximações no algoritmo original.

De uma forma geral, esses métodos foram implementados exatamente como foi descrito no algoritmo. O que se deve ter em mente, no entanto, é o conjunto de entidades topológicas que estão sendo tratadas quando as listas *lallvtx* e *lallseg* estão sendo percorridas. Pode-se dizer que, exceto pelas situações particulares já comentadas, em que o resultado da aplicação de alguma operação booleana pode ser diferente do esperado, o único detalhe de implementação que também poderia comprometer o resultado destas operações é o uso do método *regions*, da classe *GeomBoolOp*, que é aquele responsável pela detecção de possíveis regiões formadas a partir de um conjunto de retalhos de superfície passados como parâmetro. Este método é chamado tanto na operação de diferença quanto na regularização do resultado. Na diferença, ele é chamado após o passo de verificação e remoção de faces, para que se possa determinar se eventuais faces do tipo *BinA* que foram temporariamente mantidas como parte do resultado fazem ou não parte das fronteiras das regiões formadas com as faces remanescentes. As faces deste tipo que não fizerem parte da fronteira de nenhuma região formada devem ser removidas. Na regularização, este método é usado para se determinar quais as faces (com qualquer classificação) que fazem parte da fronteira das eventuais regiões formadas com as faces remanescentes de alguma operação booleana. As faces que não fizerem parte da fronteira de nenhuma região devem ser removidas.

O problema, neste caso, também já havia sido comentado anteriormente. Como este método envia as informações necessárias sobre os retalhos de superfície (faces) para a biblioteca CGC, para que esta retorne ao MG a informação sobre os retalhos de superfície que fazem parte da fronteira das regiões formadas, as faces pendentes ou soltas internamente às regiões formadas serão consideradas como parte da fronteira das mesmas, o que vai de encontro à definição de fronteira adotada neste trabalho. Vale notar que neste caso, o problema é mais sério, já que não pode ser resolvido instruindo-se o usuário a agir de uma determinada maneira. Para que estruturas internas não sejam consideradas como parte da fronteira de regiões quando estas regiões estão sendo selecionadas para fazerem parte de um grupo, basta que o usuário faça manualmente o reconhecimento de regiões, desconsiderando as estruturas internas. No presente caso, não há como solucionar o problema de forma análoga, já que o procedimento que está sendo realizado é totalmente automático, não dependendo da interação do usuário.

Contudo, vale ressaltar que para que estruturas pendentes ou soltas internamente a regiões façam parte do resultado de uma operação booleana, é

necessário que elas já sejam estruturas internas a alguma região nas condições iniciais do problema. Ou seja, apenas um modelo inicialmente *non-manifold* poderia recair neste problema, comprometendo o resultado das operações de diferença e regularização.

4.5. Pós-processamento do resultado

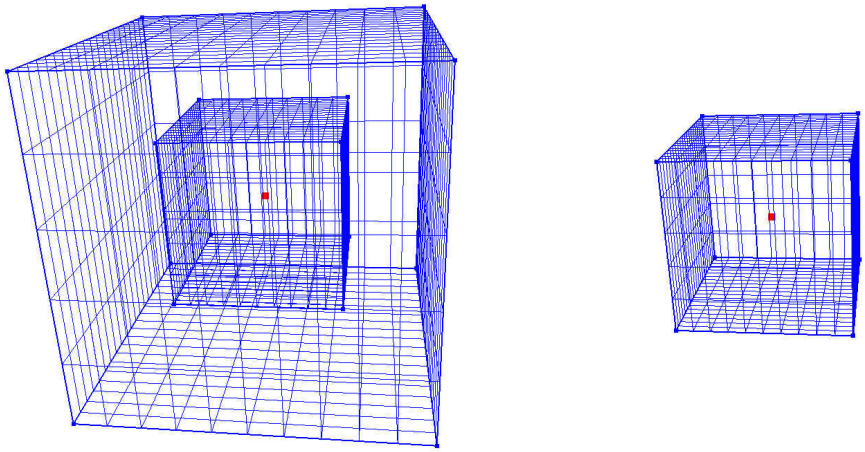
Após a aplicação das operações booleanas sobre os dois grupos e de uma eventual regularização do resultado, pode-se realizar o reconhecimento de regiões com os retalhos de superfície remanescentes. Este procedimento pode ser bastante útil, por exemplo, num processo de criação de um modelo complexo de engenharia, quando se deseja que as regiões resultantes de uma operação booleana sejam combinadas com novas regiões, novamente através de operações booleanas.

O reconhecimento automático de regiões como pós-processamento do resultado de uma operação booleana livra o usuário da tarefa de selecionar explicitamente os retalhos de superfície que formam a fronteira de cada região para efetuar o reconhecimento manual de regiões. Este inclusive pode ser um procedimento demorado e difícil, dependendo da posição relativa dos retalhos de superfície no espaço. Obviamente, o reconhecimento automático de regiões poderia ser efetuado, contudo nem sempre se deseja que todas as regiões do modelo completo sejam reconhecidas, mas apenas aquelas resultantes da aplicação das operações booleanas sobre os grupos de entidades topológicas escolhidos pelo usuário.

O único problema em se realizar o reconhecimento de regiões automaticamente com os retalhos de superfície remanescentes de uma operação booleana é que qualquer conjunto conexo e fechado de retalhos de superfície passa a ser encarado como a fronteira de uma nova região, o que nem sempre se é desejável. A Figura 4.18 mostra um exemplo de interseção entre dois sólidos em que o resultado deve ser tratado apenas como uma casca, ou seja, apenas como o conjunto de retalhos de superfície remanescentes da interseção, sem considerar o volume interno a este conjunto de retalhos de superfície. Um dos sólidos que está sendo combinado é um cubo com um vazio interno representado por uma casca cúbica. O outro sólido é um cubo que ocupa exatamente o volume do vazio interno do primeiro cubo. A interseção entre estes dois sólidos constitui-se apenas das faces da fronteira do segundo cubo. Nesse

caso, o reconhecimento automático de regiões após a interseção resulta em uma região que não pertence ao resultado da operação booleana de interseção realizada.

Como casos como o da Figura 4.18 são bastante particulares e difíceis de serem encontrados em problemas práticos, optou-se por realizar o reconhecimento automático de regiões com os retalhos de superfície remanescentes de uma operação booleana.



(a) *(b)*

Figura 4.18 – Problema com a detecção automática de regiões após a operação booleana: a) sólido com vazio interno ocupado por outro sólido; b) interseção entre os dois sólidos representada apenas por uma casca.

4.6. Eficiência e robustez do algoritmo

Como foi visto, o algoritmo divide-se em três etapas: armazenamento das informações provenientes dos dois grupos, classificação das entidades topológicas (vértices, arestas e faces) e processamento das operações booleanas com estas entidades.

A primeira etapa caracteriza-se por ser bastante rápida. Extrair informações contidas nos grupos e obter novas informações a partir destas pelas relações de adjacência são procedimentos que dependem somente de consultas a listas encadeadas e inserção de novos nós em outras listas encadeadas. O tempo de execução desta etapa varia linearmente com a quantidade de entidades topológicas contidas diretamente ou indiretamente nos dois grupos.

A segunda etapa é determinante na estimação do tempo de execução do algoritmo. Ela é a etapa lenta. Isto porque ela é a única que depende dos

algoritmos geométricos da biblioteca CGC disponibilizados pelos métodos públicos da classe *GeomBoolOp*. A detecção de ponto em retalho de superfície e a detecção de ponto em região são dois procedimentos relativamente lentos, e que são chamados muitas vezes nesta etapa. Mesmo quando os grupos não possuem vértices ou arestas soltos ou pendentes, é necessário se localizar espacialmente os vértices das fronteiras das faces de um grupo em relação às regiões e faces do outro grupo. Pode-se dizer que o tempo de execução desta etapa também depende do número de entidades topológicas presentes em cada grupo, contudo o número de entidades não é tão determinante quanto o número total de vértices não compartilhados pelos dois grupos localizados interiormente à *bounding box* de alguma face ou região do grupo que não o contém. Isto porque estes algoritmos geométricos são chamados apenas no momento em que vértices deste tipo precisam ser classificados. Uma ordem de grandeza do tempo de duração desta etapa será apresentada no próximo capítulo.

A última etapa é menos lenta que a anterior, contudo não tão rápida quanto a primeira. Como nesta etapa as entidades já estão classificadas, o que se faz efetivamente é consultar o tipo de classificação de cada entidade e determinar se ela deve ser removida ou não. Este processo seria bem mais rápido se não fosse por consultas adicionais às listas de regiões de cada grupo e às listas de faces adjacentes a cada região para se determinar quantas regiões de cada grupo compartilham uma mesma face e pela chamada do método *regions* da classe *GeomBoolOp* na operação de diferença, para se determinar se as eventuais faces do tipo *BinA* remanescentes fazem parte da fronteira de alguma região resultante.

A questão da robustez do algoritmo proposto deve ser analisada separadamente para o algoritmo teórico e para a sua adaptação ao ambiente de modelagem do MG. O algoritmo teórico exposto no capítulo anterior busca cobrir todos os casos possíveis em que dois grupos de entidades topológicas em qualquer número e com qualquer dimensão são combinados através das operações booleanas, desde que obedecidas as suas condições de aplicabilidade. Arestas e faces pendentes ou soltas e vértices soltos interiormente ou exteriormente às regiões presentes, regiões de um mesmo grupo ou de grupos distintos com faces, arestas ou vértices comuns em suas fronteiras, regiões com múltiplas cascas, faces com múltiplos *loops* e superposição de faces são exemplos de casos tratados pelo algoritmo proposto. Logo, pode-se dizer que o algoritmo teórico é bastante robusto para os objetivos a que se propõe. Já a versão implementada no MG não possui um nível de

abrangência tão grande, já que vários casos particulares não podem ser tratados apropriadamente devido às limitações atuais do modelador expostas neste capítulo.

Vale ressaltar, no entanto, que a robustez do algoritmo está associada ao tipo de tratamento que se resolveu atribuir às entidades topológicas presentes nos grupos. Isto significa que a maneira como as informações de entrada são manipuladas pelo algoritmo também integra o conjunto de contribuições deste trabalho. Não há na literatura de modelagem geométrica uma maneira formal e padronizada de se tratar entidades topológicas em domínios *non-manifold* quando operações booleanas são aplicadas sobre elas, como há em domínios *manifold*. Logo, este trabalho propõe apenas uma possível forma de abordagem do problema, tendo em vista a sua aplicação no campo da modelagem aplicada à engenharia. Uma maneira mais simplificada de se tratar o problema, por exemplo, seria desconsiderar quaisquer entidades topológicas internas a quaisquer regiões presentes nos dois grupos, mantendo assim o ponto-de-vista de *lugar geométrico* que costuma estar atrelado às operações booleanas. Poderia também ser feita a restrição de não permitir que regiões de um mesmo grupo compartilhem uma mesma face, ou ainda de não poder haver vértices, arestas ou faces completamente soltos no espaço. A superposição de faces também poderia ser proibida.

Todas estas simplificações sem dúvida reduziriam o esforço computacional necessário para o processamento das informações e diminuiriam o número de situações topológicas particulares em que o resultado de uma operação booleana poderia ficar comprometido. Contudo, desejou-se tratar a questão das operações booleanas em domínios *non-manifold* da forma mais abrangente possível.