BRepOO: Concepção e Implementação dum Núcleo de Modelação de Sólidos segundo uma Metodologia de Programação Orientada para Objectos

Carlos Martins, Abel Gomes e José Teixeira
Dept. de Matemática - Fac. de Ciências e Tecnologia da Univ. de Coimbra

CENTRO DE COMPUTAÇÃO GRÁFICA - CCG/ZGDV
Rua de Moçambique, 17, R/C Esq, 3000 COIMBRA
Tel.: 039 - 70 26 46; Fax: 039 - 70 26 47

Sumário

Neste artigo é desenvolvido um núcleo de modelação de poliedros sólidos com base num esquema de representação fronteira, designado por BRepOO (*Object-Oriented Boundary Representation*). A concepção e a implementação do modelo computacional foram realizadas sobre um modelo matemático bem definido, que se distingue dos modelos convencionais por utilizar os conceitos da teoria dos complexos celulares, e não da teoria das superfícies fechadas. Este modelo computacional, ao invés de tantos outros, reflecte completamente o modelo matemático, para o que contribui significativamente a utilização duma linguagem de programação orientada para/por objectos.

1. Introdução e Objectivos

O modelo de representação fronteira (*Boundary Representation*) é um dos modelos de sólidos mais importantes, e também mais usados, em modelação geométrica.

As principais vantagens oferecidas por este modelo de sólidos, encontram-se ao nível da interacção gráfica directa com o utilizador, o que permite realizar operações locais sobre o modelo de um sólido; por exemplo, a chanfradura duma aresta pode ser feita por selecção da referida aresta através dum dispositivo de entrada como o rato. Isto significa que existe uma relação de um-para-um entre o modelo gráfico ("aquilo que se vê") no ecrã e o modelo computacional de sólidos para o qual existe uma estrutura de dados específica, usualmente designada por BRep. Isto é, existe uma correspondência biunívoca entre as entidades geométrico-topológicas, isto é, os vértices, as arestas e as faces dum sólido ao nível da BRep, e sua representação gráfica no ecrã dum computador.

As linhas mestras que conduziram à implementação da BRepOO *(Object-Oriented Boundary Representation)*, foram basicamente as seguintes:

1. Construir um modelador BRep de acordo com um modelo algébrico bem definido, que é, grosso modo, uma álgebra de poliedros [7], de forma a encontrar um modelo computacional estável, o que, rigorosamente, não acontece noutros sistemas de modelação.

2. Usar uma arquitectura vincadamente modular na implementação do modelo computacional que facilitasse não só a manutenção e a legibilidade, mas também uma utilização de alto-nível do núcleo de modelação, o que também não acontece, em geral, noutros sistemas BRep.

2. Fundamentos Matemáticos

Os sólidos são, a par das superfícies de forma livre, os objectos de estudo em modelação geométrica. Genérica e abstractamente, um sólido é um espaço topológico que goza de um conjunto de propriedades interessantes, sendo a mais importante aquela que diz respeito à decomposição em entidades topológicas (vértices, arestas, faces e volumes) elementares, designadas por células. Todo o espaço topológico que admite uma decomposição celular é designado por *poliedro*; neste sentido todo o sólido é um poliedro, embora o inverso não seja verdade (veja-se [1] para uma discussão mais profunda sobre este tópico). Além disso, em termos intuitivos, um sólido apresenta uma extensão finita e delimitada no espaço físico-real, o que matematicamente é definido através da propriedade de compacidade [2]. Teoricamente, podemos, então, considerar que um sólido é um poliedro compacto e homogeneamente 3-dimensional em $\mathbb{R}^3$. Portanto, todo o poliedro sólido admite uma decomposição em células de dimensão 0 (vértices), 1 (arestas), 2 (faces) e 3 (volumes). Ao conjunto de todas as células associadas a um poliedro é usual chamar-se *complexo celular*.

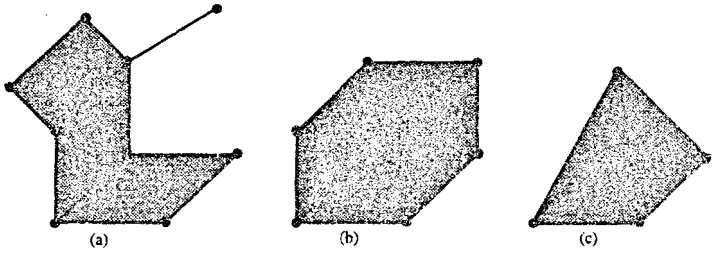

Figura 1: Exemplos de complexos celulares.

Na Figura 1 são apresentados três exemplos de complexos celulares, o primeiro dos quais não representa um poliedro sólido visto que a propriedade de homogeneidade 3-dimensional (que matematicamente descreve a nossa noção intuitiva de solidez) não é satisfeita; no entanto, os complexos celulares (b) e (c) representam poliedros sólidos, o último dos quais apresenta a seguinte decomposição celular: 5 vértices, 8 arestas, 5 faces e 1 volume. Note-se que todas as células são, por definição, abertas (excepto os vértices, que são abertos e fechados simultaneamente). Isto quer dizer que a única célula volúmica existente em (c) não é mais do que o interior do poliedro correspondente, ao passo que a sua fronteira é constituída por todas as células de dimensão inferior ou igual a dois (vértices, arestas e faces) que o delimitam. Do mesmo modo, a fronteira duma face é dada por todas as células 0-dimensionais e 1-dimensionais que a delimitam, enquanto que a fronteira duma aresta é fornecida por dois vértices.

Ora, num esquema de representação BRep, as células volúmicas dos poliedros sólidos (ou, equivalentemente, o interior dos poliedros sólidos) não são explicitamente representadas, mas tão somente as suas células fronteira. Em termos práticos, num esquema de representação fronteira, os poliedros sólidos são representados pela sua fronteira (vértices, arestas e faces), a qual é também, e por definição, um poliedro, mais concretamente um poliedro homogeneamente 2-dimensional. Realce-se que a validade da representação está *a priori* assegurada dado existir uma correspondência um-para-um entre um poliedro sólido e a sua fronteira [2], desde que a decomposição celular desta seja miníma, isto é, desde que a geometria duma célula seja distinta de todas as células adjacentes da mesma dimensão. Em todo o caso, um poliedro homogeneamente 2-dimensional que é fronteira dum poliedro sólido identifica um único poliedro sólido, qualquer que seja a sua decomposição celular.

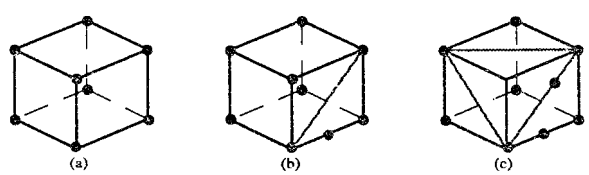

Figura 2: Poliedro fronteira com diferentes decomposições celulares.

Na Figura 2, (a), (b) e (c) representam diferentes decomposições celulares dum mesmo poliedro fronteira, o qual, por sua vez, identifica um único poliedro sólido; em (a) temos uma decomposição miníma, ao passo que em (b) temos uma sobredecomposição

de uma face e de uma aresta, e, por último, em (c) temos sobredecomposições celulares de três faces e de duas arestas.

Naturalmente que, no contexto da modelação geométrica de sólidos, só estamos *a priori* interessados em decomposições celulares minímas, ou, se se quiser, em complexos celulares minímos, visto que qualquer sobredecomposição celular revela sempre a existência de redundância na representação computacional, o que, em termos de memória, se pode mostrar crítico; basta que para tanto o processo de decomposição celular não seja controlado correctamente, e entre em ciclo.

Registe-se ainda que o controlo efectivo da decomposição celular é regulado pela fórmula de Euler-Poincaré dos complexos celulares e respectivos poliedros, $v - e + (f - fh) = 2\ (s - g)$, em que v, e, f, fh, s e g representam o número de vértices, arestas, faces, furos em faces, superfícies fronteira exteriores e interiores (no caso de existirem cavidades ôcas nos sólidos), e furos passantes ou ansas, respectivamente; o primeiro membro da fórmula diz respeito ao complexo celular, ou seja, às células, ao passo que o segundo membro denota as propriedades globais do poliedro ou espaço topológico subjacente [7].

A fórmula de Euler-Poincaré é um invariante algébrico a partir do qual é possível estabelecer um método (computacional) de decomposição da superfície fronteira de um sólido nas diversas entidades referidas (vértices, arestas, bordos (loops), faces e superfícies fechadas). A decomposição da superfície fronteira de um sólido é feita incrementalmente pela aplicação de operadores unários, chamados *Operadores de Euler*, ao respectivo poliedro (ou, alternativamente, ao complexo celular). Deste modo, obtemos um modelo matemático de sólidos que é essencialmente uma álgebra de poliedros, a qual consiste numa classe de poliedros (ou de complexos celulares) conjuntamente com uma colecção finita de operadores de Euler; estes operadores efectuam, no essencial, as subdivisões elementares da superfície fronteira dum poliedro sólido.

3. Modelo Computacional de Poliedros

A representação da superfície fronteira dos poliedros sólidos é conseguida conjuntamente pelas representações da sua topologia e da sua geometria. A topologia da superfície fronteira dum poliedro sólido é fornecida pelo complexo celular que lhe está associado, ou seja, os seus vértices, as suas arestas e as suas faces; além destas entidades topológicas básicas, existem outras que são compostas, tais como os invólucros (*shells* interiores e exteriores) e os bordos (*loops*). Um invólucro é uma colecção de faces,

arestas e vértices que delimitam uma região no espaço Euclidiano $\mathbb{R}^3$, enquanto que um bordo é uma colecção alternada de arestas e vértices que delimitam uma face dum poliedro. Estas entidades topológicas compostas são úteis no modelo computacional porque permitem uma representação mais flexível dos poliedros.

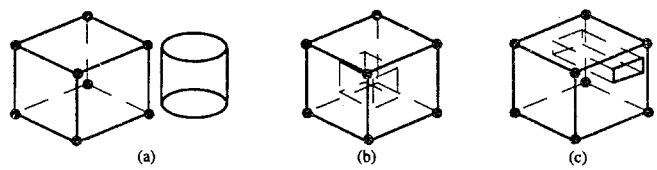

Figura 3: Poliedro fronteira com diferentes decomposições celulares.

Por exemplo, na Figura 3, o poliedro (a) é desconexo, i.e., tem duas componentes que são representadas por dois invólucros (ou *shells*) exteriores ao nível computacional; por sua vez, o poliedro (b) tem uma única componente, cuja representação computacional é um invólucro exterior, e ainda uma cavidade ôca que é representada por um invólucro interior; por último, o poliedro (c) tem também uma única componente que é representada por um invólucro exterior, mas duas das suas faces não são simplesmente conexas, ou seja, duas das suas faces contêm dois furos (um furo por cada face), os quais são devidos à existência dum furo passante que atravessa o poliedro dum lado ao outro. Isto significa a fronteira de cada uma destas duas faces deve ser representada por dois loops, um exterior e outro interior (este referente ao furo na referida face).

3.1 Classes de Entidades Topológicas

De acordo com as entidades topológicas, simples e compostas, existentes no complexo celular dum poliedro, e assumindo a concepção clássica de sistemas orientados para/por objectos de que a cada conceito corresponde a uma classe, ou é representadopor uma classe, tem-se as seguintes classes de entidades para o sistema BRepOO:

1. Classe **Entity**: é uma classe abstracta que contém todas as características comuns às entidades topológicas existentes no sistema; é, pois, a classe base de todas as outras classes existentes no sistema:
2. classe **Vertex**: é a classe que permite gerar vértices;
3. classe **Edge**: é a classe geradora de arestas;
4. classe **Face**: todas as faces dum poliedro são geradas a partir desta classe;
5. classe **Loop**: permite criar objectos que são *loops*;

6. classe **Shell**: permite a geração de objectos que são *shells*;

7. classe **Polyhedron**: serve para criar objectos poliédricos.

Note-se que as classes Vertex, Edge e Face dizem respeito às entidades topológicas elementares, enquanto que as classes Loop, Shell e Polyhedron são referentes às entidades topológicas compostas.

A relação hierárquica entre as classes é uma relação *isA*; consequentemente a hierarquia de classes é clássica, como se pode ver na Figura 4.

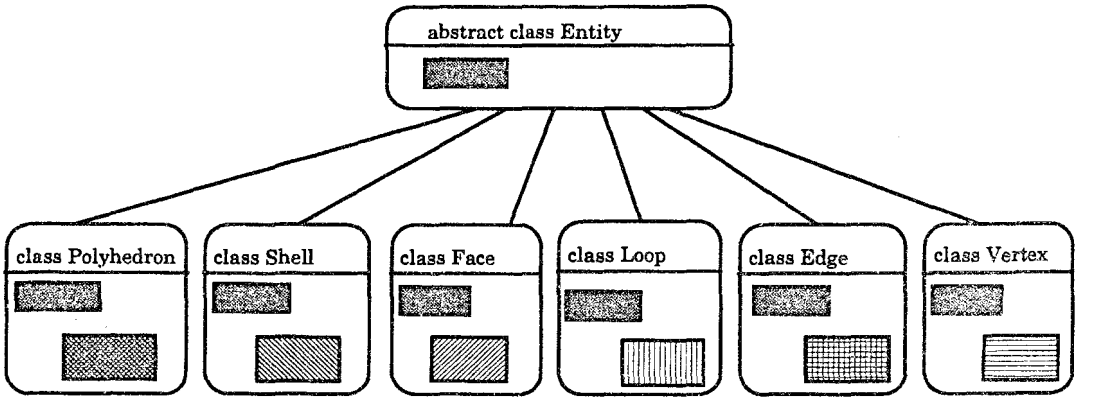

Figura 4: Relações hierárquicas entre as classes constituintes do BRepOO.

Assim, por exemplo, o facto de todo o poliedro ser uma entidade topológica expressa uma relação *isA* que é representada pela derivação da classe Polyhedron a partir da classe Entity; o mesmo se passa com as restantes classes relativamente à classe Entity.

3.2 Operadores de Euler

Os operadores de Euler permitem a construção incremental de poliedros por decomposição ou subdivisão das células do respectivo complexo celular. A decomposição das células é regulada pela fórmula de Euler-Poincaré, e resulta da aplicação de um ou mais operadores de Euler a um poliedro. Deste modo, obtem-se uma álgebra de poliedros, que é o modelo matemático que suporta o modelo computacional.

Em termos de representação computacional, esta álgebra de poliedros está expressa na classe Polyhedron, na qual os operadores de Euler não são mais do que funções ou métodos de instância (ver anexo no fim do artigo). Isto permite fazer uma utilização de alto-nível do núcleo de modelação, visto que para criar e manipular poliedros basta dominar as regras de decomposição celular expressas nos operadores de Euler. De facto, como se pode ver em anexo, os operadores de Euler (de alto-nível) são métodos de

instância da zona pública da classe Polyhedron; por outro lado, a estrutura de dados do núcleo BRepOO está oculta na zona privada, a qual comporta também os métodos de Euler (de baixo-nível) que manipulam a estrutura de dados, por chamada desencadeada pelos operadores de Euler de alto-nível. Desta forma, libertam-se eventuais utilizadores-programadores do núcleo BRepOO de toda a complexidade associada à estrutura de dados. Na verdade, no referido anexo é de notar que os operadores de alto nível "trabalham" apenas com os identificadores (numeros associado a cada entidade) das entidades topológicas, enquanto os de baixo nível operam sobre as representações internas dessas entidades topológicas.

Os operadores de Euler de baixo-nível obedecem a uma fórmula de Euler-Poincaré modificada e adaptada à estrutura de dados, de molde que, por exemplo, um furo numa face é representado indirectamente por um *loop*. Esta fórmula é dada por $v - e + 2f - l = 2.(s - g)$, em que $l = fh + f$ representa o número total de loops (loops interiores referentes aos furos nas faces, e, os loops exteriores das faces) respeitantes no modelo.

Operadores de Euler	$v - e + 2f - l = 2\,(s - g)$
construtivos	(v, e, f, l, s, g)
make_shell_face_loop_vertex	(1, 0, 1, 1, 1, 0)
make_edge_vertex	(1, 1, 0, 0, 0, 0)
make_edge_face_loop	(0, 1, 1, 1, 0, 0)
kill_edge_make_loop	(0, -1, 0, 1, 0, 0)
kill_face_make_loop_hole 1) Junção de duas superfícies 2) Criação de um furo passante	 (0, 0, -1, 0, -1, 0) (0, 0, -1, 0, 0, 1)
destrutivos	(v, e, f, l, s, g)
kill_shell_face_loop_vertex	(-nv, -ne, -nf, -nl, -1, 0)(*)
kill_edge_vertex	(-1, -1, 0, 0, 0, 0)
kill_edge_face_loop	(0, -1, -1, -1, 0, 0)
make_edge_kill_loop	(0, 1, 0, -1, 0, 0)
make_face_kill_loop_hole 1) Criação de uma superfícies 2) Eliminação de um furo	 (0, 0, 1, 0, 1, 0) (0, 0, 1, 0, 0, -1)

(*) por n_ entende-se o nº de _ entidades; por exemplo, nv significa n vértices.

Tabela 1: Descrição dos operadores de Euler.

Os operadores de Euler de baixo-nível são apresentados na seguinte Tabela 1, onde se pode observar a sua subordinância à fórmula de Euler-Poincaré modificada. Na parte direita da Tabela 1 podemos constatar as variações incrementais do número de entidades de cada classe, resultantes da aplicação dos operadores. Por exemplo, o operador make_edge_vertex ao actuar sobre um poliedro adiciona-lhe um vértice e uma aresta; logo $(v, e, f, l, s, g)=(1, 1, 0, 0, 0, 0)$.

Para uma discussão mais alargada, embora mais clássica, sobre os operadores de Euler deve consultar-se os artigos de Baumgart [3], de Braid *et al.* [4,5], de Eastman e Weiler [6] e de Mäntyla e Sulonen [8,10]. A validade dos operadores de Euler é discutida por Mäntyia em [9].

4. Estrutura de Dados

A implementação dos operadores de Euler requer a existência duma estrutura de dados. Esta estrutura de dados deve ser capaz de representar, de um modo directo ou indirecto, todas as relações existentes entre as várias entidades topológicas do modelo de representação fronteira, Figura 5.

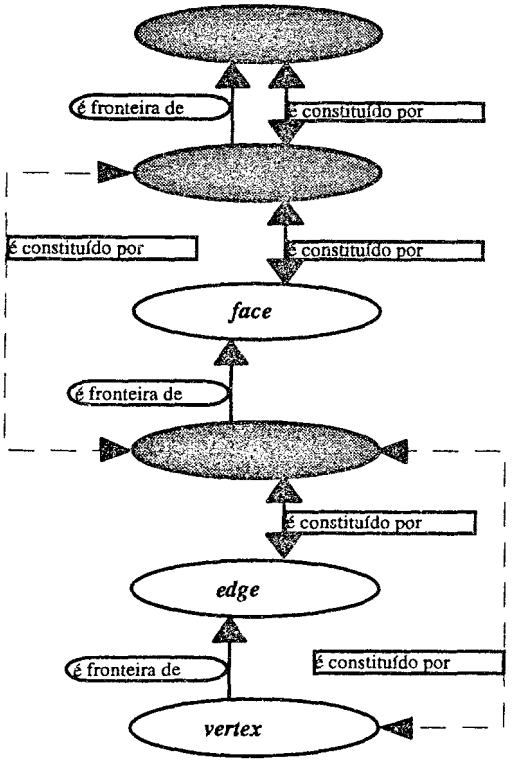

Figura 5: Estrutura de dados do núcleo BRepOO.

Como se pode ver na Figura 5, na estrutura de dados estão representadas várias relações entre as entidades topológicas, nomeadamente:

1. relação "*é constituído por*" (ou, inversamente, "*é parte de*");

2. relação "*é fronteira de*".

A primeira relação, *"é constituído por"*, indica que toda a entidade topológica composta é constituída por entidades topológicas elementares. Por exemplo, um *loop* é constituído por arestas e vértices; contudo, a relação entre um *loop* e os seus vértices não é directa, embora seja inferível pelo cálculo da fronteira das suas arestas, o que justifica o aparecimento da linha tracejado que identifica a relação entre *loops* e vértices. Por outro lado, uma *shell* é constituída por uma colecção de faces, incluindo as respectivas fronteiras ou *loops*. A relação entre uma *shell* e as fronteiras das faces é feita também indirectamente pelas próprias faces.

A segunda relação, "A *é fronteira de* B", é uma relação em que a entidade A delimita a entidade B. Por exemplo, uma *shell* é fronteira de um poliedro sólido, embora um poliedro possa ser constituído por mais do que uma *shell*; do mesmo modo, uma *loop* é fronteira duma face, mas uma face pode ser delimitada por várias *loops*.

Internamente, estas relações entre entidades topológicas são representadas ao nível das classes por variáveis de instância que contêm ponteiros (ou apontadores) para entidades ou objectos de outras classes.

5. Relações de Adjacência

Além das relações "*é constituído por*" e "*é fronteira de*", há outro tipo de relações que não aparecem expressas no diagrama da estrutura de dados da Figura 5, mas que pela sua importância não devem ser esquecidas: relações de adjacência entre entidades topológicas. O número máximo de relações de adjacência que é possível representar na estrutura de dados é igual ao número de entidades topológicas elementares levantado ao quadrado, a saber: v(V), v(E), v(F), e(V), e(E), e(F), f(V), f(E) e f(F). Por exemplo, a relação de adjacência v(E) denota a colecção E de arestas que são adjacentes ao vértice v. De acordo com Weiler [11,12], uma estrutura de dados é suficiente se é capaz de representar explicita ou implicitamente, ou, se se quiser, directa ou indirectamente, todas as relações de adjacência; aquelas que não estão representadas directamente são, ou devem ser, susceptíveis de ser inferidas a partir da relações explícitas de adjacência.

No caso da estrutura de dados utilizada pelo núcleo BRepOO é utilizada uma representação baseada na Semi-Aresta (*Half-Edge*) [8], a qual é uma versão modificada da estrutura da Aresta-Alada (*Winged-Edge*) [3]. A representação W-E (*Winged-Edge*) é essencialmente descrita pelas adjacências E(V)-E((E)(E))-E(F), ou seja, as adjacências de vértices V, arestas E e de faces F em torno de arestas E. Isto significa que o elemento topológico de referência é a aresta. Por exemplo, como se pode ver Figura 6(a), as adjacências da aresta e_{ref} são as seguintes: e_{ref} (V)={v_1,v_2}, e_{ref} ((E)(E))={ccw-e_1,cw-e_2,cw-e_1,ccw-e_2}, e_{ref}(F)={f_1,f_2}.

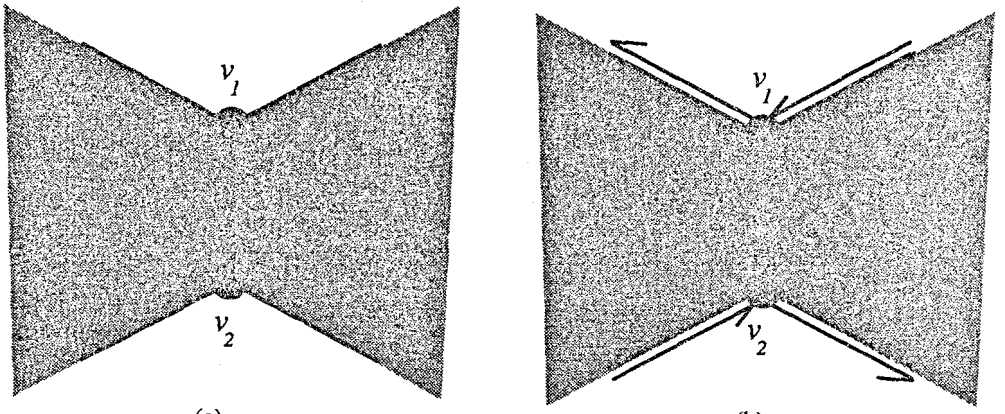

Figura 6: (a) representação da Aresta-Alada; (b) representação da Semi-Aresta

Na representação da Semi-Aresta, Figura 6(b), as arestas são orientáveis, de tal modo que internamente qualquer aresta é decomposta em duas semi-arestas orientadas simetricamente. Na representação da Aresta-Alada, a orientabilidade das arestas é implicitamente representada pelos campos cw (*clockwise*) e ccw (*counter clockwise*) das arestas adjacentes.

6. Exemplo de Aplicação

Nesta secção procede-se à apresentação de um exemplo prático da utilização do núcleo de modelação *BRepOO*. O exemplo diz respeito à construção de um *cubo*, no qual se pretende ilustrar a utilização dos operadores de Euler na construção e manipulação de poliedros.

```
Polyhedron cubo = Polyhedron();
```

Criado o objecto *cubo* (o qual não contém nenhuma célula, como faces, arestas ou vértices) torna-se necessário aplicar-lhe sucessivamente vários operadores de Euler de molde a adicionar-lhe a estrutura topológica (faces, arestas ou vértices).

```
m_sflv (NULL,&s1,&f1,&l1,&v1,*p1);          // make_shell_face_loop_vertex
for ( i = 1 ; i <= 4 ; i ++ ){              // nº de lados
  if (i < 4)
     m_ev ( eh1, eh1, i, &eh1, &v, *pto); // make_edge_vertex
  else
     m_efl (2*n-2, 1, n, 1, &f, &l, &eh); //make_edge_face_loop e base do cubo
for ( i = 1; i <= 4; i++ ){                 // constroi arestas laterais
  if (i<4)
     m_ev ( e, e, i, &eh1, &v2, *p1);       // make_edge_vertex
  else
     m_ev (2*n, 2*n, i, &eh1, &v2, *p1);
};
for ( i = 1; i <= 4; i++ ){                 // constrói faces laterais
  if (i < 4)
     m_efl (e,e+2,i+n,i+n+1,&eh1,&v2,&f1);// make_edge_face_loop
  else
     m_efl ( e, e+2, i+n, n+1, &f1, &v2, &eh3);
```

};// O cubo está construido !!!!

Este exemplo demonstra a facilidade de utilização do núcleo de modelação *BRepOO*, bem como a sua identificação com o modelo matemático apresentado.

7. Conclusões e Desenvolvimentos Futuros

No que respeita às vantagens da arquitectura (concepção e implementação) do sistema, as mais importantes são:

1) o modelo computacional reflecte o modelo matemático, o que lhe empresta uma notável estabilidade;

2) reforço da interface:

i) o código que se obtém é bastante estruturado e legível, o que facilita a sua posterior manutenção;

ii) utilização, construção e manipulação de alto-nível dos objectos poliédricos, com base na zona "public" da classe *Polyhedron*, com os detalhes de baixo-nível da estrutura de dados relegados para a zona "private";

3) gestão de memória relegada para o núcleo do sistema;

4) reforço da modularidade pela utilização do modelo de programação orientada para/por objectos: classes, tratamento uniforme das estruturas de dados a baixo nível *(templates)* e redução do código — a linguagem de programação usada foi a C++.

Em suma, pode-se dizer o desenvolvimento deste núcleo de modelação de sólidos (de representação fronteira), segundo uma metodologia de programação orientada para objectos, permitiu chegar a um modelo computacional estável, em consequência não só dum modelo matemático bem definido, mas também da utilização duma linguagem de programação orientada para objectos, a qual permite representar eficazmente os conceitos e os mecanismos inerentes ao modelo matemático.

Com base neste núcleo de modelação, poderão desenvolver-se futuramente diversos tipos de aplicações, como, por exemplo, as operações Booleanas de poliedros.

Bibliografia

[1] Agoston, M.:*"Algebraic Topology: A First Course"*, Pure and Applied Mathematics Series, Marcel Dekker, Inc., Nova Iorque, 1976.

[2] Aleksandrov, P.:*"Combinatorial Topology"*, Graylock Press, Albany , Nova Iorque, 1957.

[3] Baumgart, B.:*"Winged-Edge Polyhedron Representation"*, Stanford Artificial Intelligence Report nº. CS-320, Outubro 1972.

[4] Braid, I., Hillyard, R. e Stroud, I.:*"Stepwise Construction of Polyhedra in Geometric Modelling"*, CAD Group Document nº. 100, Univ. of Cambridge Computer Laboratory, Outubro 1978.

[5] Braid, I., Hillyard, R. e Stroud, I.:*"Stepwise Construction of Polyhedra in Geometric Modelling"*, em Brodlie, K. (ed.): *Mathematical Methods in Computer Graphics and Design,* Academic Press, 1980.

[6] Eastman, C. e Weiler, K.: *"Geometric Modeling Using The Euler Operators"* Research Report n° 78, Institute of Physical Planning, Carnegie-Mellon University, Fevereiro 1979.

[7] Gomes, A.:*"Modelos Algébricos de Sólidos e Morfologia"*, em Provas de Aptidão Pedagógica e Capacidade Científica, Departamento de Matemática, Universidade de Coimbra, Novembro 1992.

[8] Mäntyla, M. e Sulonen, R.: *"GWB: A Solid Modeler with Euler Operators"*. IEEE Computer Graphics & Applications, Vol. 2, n° 7, Setembro 1982.

[9] Mäntylä, M.:*"A Note on the Modeling Space of Euler Operators"*, IEEE Computer Graphics & Applications, Vol.2, n°7, Stembro 1984.

[10] Mäntylä, M.:*"An Introduction to Solid Modeling"*, Computer Science Press, Maryland, U.S.A., 1988.

[11] Weiler, K.,:*"Edge-Based Data Structures for Solid Modelling in Curved-Surface Environments"* IEEE Computer Graphics & Applications, Vol.5, n°1, 1985.

[12] Weiler, K.:*"Topological Structures for Geometric Modelling"* PhD. Thesis, Renssealaer Polythechnic Institut, Troy, Nova Iorque, Agosto 1986.

```
class Polyhedron: public Entity{
 friend class Shell;
 private:
 // operadores de Euler construtivos (baixo nivel)
 clong make_shell_face_loop_vertex (Shell* shell, Shell **new_shell,
                                    DBLQueueItem<Face> **new_face,
                                    DBLQueueItem<Loop>   **new_loop,
                                    Vertex **new_vertex);
 clong make_edge_vertex(DBLQueueItem<EdgeHalf>* edgeh1,
                        DBLQueueItem<EdgeHalf>* edgeh2, Vertex* vertex,
                        DBLQueueItem<EdgeHalf>** new_edgehalf,
                        Vertex **new_vertex );
 clong make_edge_face_loop(DBLQueueItem<EdgeHalf>* edgeh1,
                        DBLQueueItem<EdgeHalf>*edgeh2,Vertex *vertex1,
                        Vertex *vertex2, DBLQueueItem<Face> **new_face,
                        DBLQueueItem<Loop> **new_loop,
                        DBLQueueItem<EdgeHalf> **new_edgehalf );
 clong kill_edge_make_loop(DBLQueueItem<EdgeHalf> *edgeh1,
                        DBLQueueItem<EdgeHalf> *edgeh2,
                        DBLQueueItem<Loop> **new_loop);
 clong kill_face_make_loop_hole(DBLQueueItem<Face>* face1,
                              DBLQueueItem<Face>* face2 );

 // operadores de Euler destrutivos (baixo nivel)
 clong kill_shell_face_loop_vertex(Shell* shell );
 clong kill_edge_vertex(DBLQueueItem<EdgeHalf>* edgeh, Vertex* vertex);
 clong kill_edge_face_loop(DBLQueueItem<EdgeHalf>*  edgeh );
 clong make_edge_kill_loop(DBLQueueItem<EdgeHalf>* edgeh1,
                        DBLQueueItem<EdgeHalf>* edgeh2,
                        Vertex* vertex1, Vertex* vertex2,
                        DBLQueueItem<EdgeHalf>** new_edgeh);
 clong make_face_kill_loop_hole(DBLQueueItem<Loop>* loop,
                        DBLQueueItem<Face>** new_face,
                        ShellType new_type);
```

```
// funções de inquerito de identidades
protected:
Shell* test_all_shell ( ID s_id );
(...)
public:
Polyhedron() ; // Construtor
~Polyhedron (); // Destrutor

// operadores de Euler construtivos (alto nivel)
clong m_sflv(ID S_id, ID *newS_id, ID *newF_id, ID *newL_id,ID
             *newV_id, const Point& point);
clong m_ev(ID EH1_id, ID EH2_id, ID V_id, ID *newE_id,
             ID *newV_id, const Point& point);
clong m_efl(ID EH1_id, ID EH2_id, ID V1_id, ID V2_id, ID *newF_id,
             ID *newL_id, ID *newE_id);
clong k_eml(ID E_id, ID *newL_id );
clong k_fmlh(ID F1_id, ID F2_id );

// operadores de Euler destrutivos (alto nivel)
clong k_sflv(ID S_id);
clong k_ev(ID EH_id,ID V_id);
clong k_efl(ID EH_id);
clong m_ekl(ID EH1_id, ID EH2_id , ID V1_id ,
             ID V2_id , ID *newEH_id);
clong m_fklh( ID L_id, ID *newF_id, ShellType stype);
};
```