12º Encontro Português de Computação Gráfica

8 - 10 Outubro 2003

AIF – Uma estrutura de dados B-rep para modelos poligonais

Frutuoso G. M. Silva Abel J. P. Gomes
IT – Grupo de Redes e Multimedia
Dep. Informática, Univ. da Beira Interior
Rua Marquês d'Ávila e Bolama, 6201-001 Covilhã
{fsilva,agomes}@di.ubi.pt

Resumo

Este artigo introduz uma nova estrutura de dados b-rep (boundary representation) designada de AIF (Adjacency and Incidence Framework). A estrutura de dados AIF é concisa e permite o acesso rápido à sua informação topológica. A sua concisão resulta do facto de ser uma estrutura de dados orientável mas não orientada, isto é, a orientação é induzida topologicamente como é necessário em várias aplicações. É uma estrutura de dados óptima na classe C_4^9 *, o que significa que um número mínimo de acessos é necessário para aceder a toda a informação sobre adjacências e incidências. A estrutura de dados AIF suporta a representação de modelos poligonais, não necessariamente triangulares, mesmo em condições de "non-manifold".*

Palavras chave

Estruturas de dados, modelos poligonais, representação de sólidos.

1. INTRODUÇÃO

Na computação gráfica, a complexidade dos modelos tridimensionais tem aumentado notoriamente devido em parte às tecnologias de aquisição de dados tridimensionais como os *scanners* 3D. Este tipo de tecnologias produzem modelos poligonais, normalmente designados por malhas poligonais usadas nas mais variadas aplicações como, por exemplo, multi-resolução, ambientes virtuais ou ainda, visualização de sólidos e superfícies. De facto, muitos trabalhos de investigação tem sido feitos nos últimos anos no âmbito de malhas poligonais, particularmente na análise multi-resolução [Heckbert97, Garland99], morfização de malhas [Lee99], edição interactiva de malhas [Zorin97], e na compressão geométrica e transmissão pela Internet [Taubin98].

Aplicações diferentes têm necessidades diferentes. Algumas necessitam mais de memória, outras necessitam mais de acesso rápido à informação sobre adjacências e incidências. Infelizmente, estas operações de acesso nem sempre são rápidas e eficientes como é necessário nalgumas aplicações e estruturas de dados, em particular em malhas com grande número de células (vértices, arestas e faces). Isto deve-se sobretudo ao *design* das estruturas de dados. Por exemplo, encontrar uma célula na estrutura de dados CTR (Cell-Tuple Representation)[Brisson93] requer processar todos os tuplos, o que é obviamente pouco prático e lento em malhas de grandes dimensões.

Algumas estruturas de dados representam uma malha triangular através dum conjunto de faces, sendo cada face composta pelos vértices [Hoppe98]. Este é também o caso do formato VRML (Virtual Reality Modeling Language) [Carey97]. Contudo, esta estratégia torna difícil a implementação de algoritmos rápidos de acesso à informação topológica. Por exemplo, determinar as faces incidentes num vértice é uma operação lenta porque é necessário percorrer todas as faces da estrutura de dados.

Uma forma de acelerar os algoritmos de procura é a utilização de estruturas de dados orientadas (Winged-Edge [Baumgart72], Half-edge [Mantyla88], ou Radial Edge [Weiler88]). No entanto, estas estruturas de dados necessitam de mais memória para armazenar as células orientadas. Por exemplo, na estrutura de dados Radial Edge, cada face tem associados dois conjuntos de arestas (loops), um para cada lado da face; além disso, se houver três faces incidentes numa aresta, são necessárias seis arestas orientadas.

Lee e Lee [Lee01] propuseram uma nova estrutura de dados orientada, designada por *partial entity structure*, para modelos não *manifold*, que não é mais que uma optimização da estrutura Radial Edge que necessita de menos memória. Esta estrutura contem para além das entidades vértice, aresta, face, *loop* e *shell* outro tipo de entidades, designadas de *partial entity*, para representar os casos não *manifold*. Assim contém as entidades *partial vertex*, *partial edge* e *partial face* para

representar as situações não *manifold* para vértices, arestas e faces, respectivamente.

Kallmann e Thalmann [Kallmann01] propuseram uma nova estrutura de dados, chamada Star-vertices, para representar malhas poligonais. Esta estrutura de dados é baseada na incidência de células à volta de um vértice. É uma estrutura de dados concisa mas lenta no acesso à informação topológica, pois não armazena informação explícita sobre arestas e faces.

Outras estruturas de dados foram desenhadas apenas para malhas triangulares como, por exemplo, a Directed-edges [Campagna98], a Tri-edges [Loop00] e mais genericamente a PSC (Progressive Simplicial Complexes) [Popovic97]. As duas primeiras são estruturas de dados orientadas, enquanto que a PSC é não-orientada. Portanto, a PSC não possui quaisquer células orientadas. Consequentemente, a PSC permite representar objectos orientados e não orientados, mas ao contrário das estruturas de dados orientadas (b-reps), não existe redundância na estrutura de dados PSC.

Este artigo introduz uma nova estrutura de dados b-rep, chamada AIF (Adjacency and Incidence Framework), para malhas genéricas. Estas malhas não necessitam de ser *manifold* nem triangulares. Topologicamente falando, a estrutura de dados AIF é orientável mas não orientada, isto é, não possui células orientadas. O resultado é uma estrutura de dados mais concisa e flexível, sem perda de controlo da informação sobre adjacências e incidências.

Este artigo está organizado em várias secções: A secção 2 descreve a estrutura de dados AIF. A secção 3 apresenta o operador de navegação da estrutura de dados AIF, designado por operador máscara, e que permite um acesso rápido à informação topológica. A secção 4 apresenta a implementação da estrutura de dados AIF. A comparação entre diversas estruturas de dados, incluindo a estrutura de dados AIF, é apresentada na secção 5. Por último, a secção 6 apresenta algumas conclusões e trabalho futuro.

2. ESTRUTURA DE DADOS AIF

Diz-se que uma célula x é adjacente a uma célula y (simbolicamente $x \prec y$) se x está contido na fronteira de y (ou equivalentemente, $fr(y) \cap x \neq \emptyset$) e a dimensão de x é menor que a dimensão de y; equivalentemente, diz-se que y é incidente em x (simbolicamente, $y \succ x$). Por exemplo, um vértice de uma aresta diz-se que lhe é adjacente, mas podem existir várias arestas incidentes no mesmo vértice. A relação de adjacência (incidência) é transitiva, isto é, se $v \prec e$ e $e \prec f$ então $v \prec f$.

Além disso, existem apenas duas relações básicas de adjacência para malhas poligonais, nomeadamente $V \prec E$ e $E \prec F$; as relações inversas $E \succ V$ e $F \succ E$ são as relações de incidência. Esta quatro relações podem ser compostas para formar nove relações de adjacência introduzidas por Weiler [Weiler85]. Por exemplo, as relações $V \rightarrow F$ e $E \rightarrow E$ podem ser obtidas do seguinte modo:

$$V \rightarrow F = (V \rightarrow E) \circ (E \rightarrow F) = (E \succ V) \circ (F \succ E)$$

$$E \rightarrow E = (V \rightarrow E) \circ (V \rightarrow E) = (V \prec E) \circ (E \succ V)$$

Assim, as quatro relações referidas acima constituem uma representação na classe C_4^9 (veja-se Ni e Bloor [Ni94] para mais detalhes). Esta é a melhor representação em termos do número de acessos à informação topológica, isto é, requer um número mínimo de acessos directos e indirectos à estrutura de dados para aceder às quatro relações explícitas e às restantes cinco relações implícitas, respectivamente. Um acesso directo requer uma simples chamada ao operador máscara, enquanto que um acesso indirecto requer duas ou mais chamadas ao operador máscara. O operador máscara é descrito na secção seguinte.

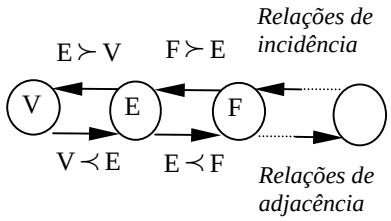

Figura 1: Estrutura de dados AIF .

A estrutura de dados AIF acomoda uma representação óptima na classe C_4^9 para malhas bidimensionais, mas pode ser facilmente generalizada para $C_{2n}^{(n+1)^2}$ quando se pretende representar objectos de dimensão n (Figura 1). A estrutura de dados AIF mantém a informação essencial sobre as adjacências e incidências de células, mas permite inferir informação adicional de modo a atravessar qualquer malha duma forma eficiente.

Uma malha bidimensional é definida em termos de um conjunto $M=\{V,E,F\}$, onde V é um conjunto finito de vértices, E é um conjunto finito de arestas, e F é um conjunto finito de faces simplesmente conexas.

Um vértice $v \in V$ é definido por um conjunto de arestas incidentes em v, isto é, $v=\{e_1,e_2,e_3,\ldots,e_k\}$, onde e_i (i=1…k) é uma aresta incidente em v. Esta é a forma como a relação $E \succ V$ está representada na AIF.

Uma aresta $e \in E$ é um par de conjuntos, o primeiro para os vértices adjacentes à aresta, e o segundo para as faces incidentes na aresta, isto é, $e=\{\{v_1, v_2\},\{f_1, f_2, \ldots, f_k\}\}$, onde v_1 e v_2 são os vértices da aresta, e $f_1, f_2, \ldots, f_k$ são as faces incidentes na aresta. Assim, temos as relações $V \prec E$ e $F \succ E$ também representadas na AIF.

Uma face $f \in F$ é definida em termos de um conjunto de arestas que lhe são adjacentes, ou seja, $f=\{e_1,e_2,e_3,\ldots,e_k\}$, onde cada e_i (i=1…k) é uma aresta adjacente a f. Esta definição inclui a relação $E \prec F$.

Com estas quatro relações básicas de adjacência e incidência na estrutura de dados AIF podemos representar malhas poligonais *manifold* (Figura 2(a)(c)(d)(e)(f)) e não-*manifold* (Figura 2(b)).

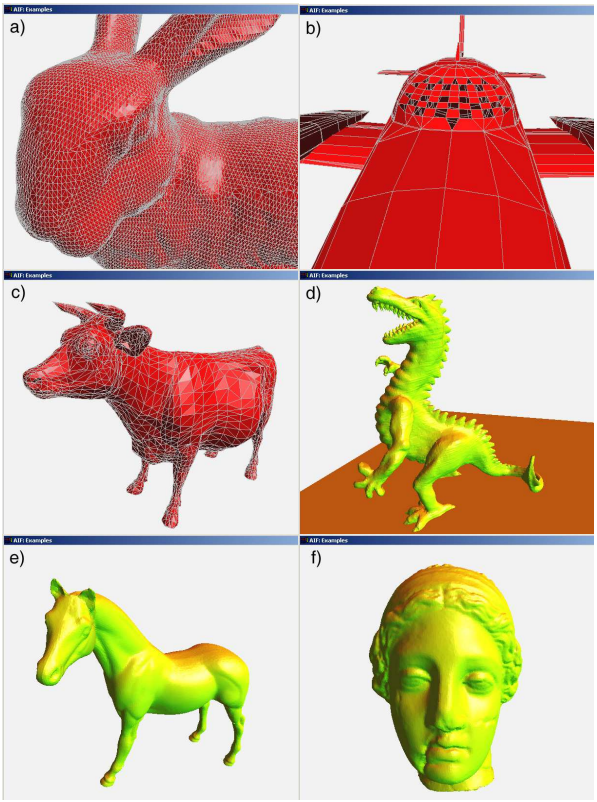

Figura 2: Modelos AIF.

Na Figura 2(b) o modelo do avião é não-*manifold* porque faltam algumas faces na zona do cockpit (zonas mais escuras).

Note-se que uma aresta pode ter um número qualquer de faces incidentes. Uma aresta *e* sem faces incidentes é representada por $e=\{\{v_1,v_2\},\{\}\}$, isto é, uma aresta remanescente. Uma aresta *e* com mais do que duas faces incidentes (Figura 3) é representada através dos seus vértices adjacentes e das faces incidentes, ou seja, $e=\{\{v_1,v_2\},\{f_1,f_2,f_3\}\}$. Assim, a estrutura de dados AIF permite-nos distinguir entre objectos *manifold* e não-*manifold*.

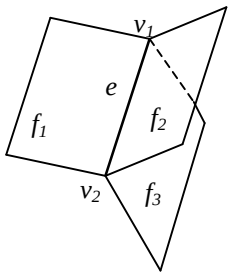

Figura 3: Malha não-*manifold*.

Além disso, a estrutura de dados AIF suporta malhas poligonais genéricas, quer sejam triangulares (Figura 2(a)(c)(d)(e)(f)) ou não (Figura 2(b)).

3. O OPERADOR MÁSCARA

O tempo é um factor crítico nas aplicações de tempo real (e.g. animação e jogos) que fazem uso de malhas poligonais. São necessárias pesquisas e algoritmos rápidos para identificar localmente as células adjacentes ou incidentes noutras células de modo a rapidamente simplificar ou refinar as ditas malhas.

A estrutura de dados AIF usa um único operador, chamado operador máscara, para aceder a toda a informação sobre adjacências e incidências. O operador máscara é definido por: $\bowtie_d: V\times E\times F\rightarrow C$, com $C=V\cup E\cup F$ sendo V o conjunto dos vértices, E o conjunto de arestas, e F o conjunto de faces, tal que $\bowtie_d(v_i,e_j,f_k)=\{c_l^d\}$ tem como resultado um conjunto de células de dimensão *d*. Os argumentos do operador máscara são células do conjunto $V\times E\times F$. A célula NULL como argumento de dimensão $n=d$ significa que todas as células de dimensão *n* que satisfaçam as condições de adjacência/incidência expressas pelos argumentos são devolvidas como resultado. Caso contrário, se $n\neq d$ e a célula de dimensão *n* é também NULL, então nenhuma restrição de adjacência/incidência é imposta ao resultado. No caso de $n=d$ e a célula de dimensão *n* é diferente de NULL, o operador máscara devolve todas as células de dimensão *n* como anteriormente, à excepção da célula argumento de dimensão *n*. Se $n\neq d$ e a célula de dimensão *n* é diferente de NULL, então a célula de dimensão *n* coloca uma restrição adicional de adjacência/incidência ao resultado.

A chamada do operador máscara é feita pelo menos com um argumento não nulo, podendo ter dois ou mesmo os três argumentos não nulos. Qualquer combinação de argumentos é possível. No entanto, no caso da chamada do operador índice zero com apenas o primeiro argumento diferente de NULL, significa que este devolverá os vértices vizinhos do vértice argumento. Assim, se pretendermos saber os vértices do vértice, as arestas da aresta ou as faces da face o operador devolve-nos os vértices, arestas ou faces adjacentes ao argumento respectivamente.

Consideremos a malha da Figura 4 para ilustrar o funcionamento do operador máscara em conjunção com a estrutura de dados AIF.

1. $\bowtie_1(v_1,\text{NULL},\text{NULL})=\{e_1,e_2,e_3\}$, devolve todas as arestas incidentes em v_1.
2. $\bowtie_2(v_1,\text{NULL},\text{NULL})=\{f_1,f_2,f_3\}$, devolve indirectamente todas as faces incidentes em v_1. Requer uma chamada intermédia ao operador máscara $\bowtie_1(v_1,\text{NULL},\text{NULL})$ para devolver todas as arestas e_1, e_2, e_3 incidentes em v_1. Depois o operador máscara $\bowtie_2(v_1,e_i,\text{NULL})$ é chamado para cada aresta e_i (i =1,2,3) de modo a calcular as faces incidentes em e_i e v_1.
3. $\bowtie_0(\text{NULL},e_1,\text{NULL})=\{v_1,v_2\}$, devolve directamente os vértices da aresta e_1.
4. $\bowtie_2(\text{NULL},e_1,\text{NULL})=\{f_2,f_3\}$, devolve directamente as faces incidentes em e_1.
5. $\bowtie_0(\text{NULL},\text{NULL},f_1)=\{v_1,v_3,v_4\}$, devolve indirectamente todas os vértices da face f_1. Requer uma chamada intermédia ao operador máscara

$\bowtie\!\!\blacktriangleleft_1$(NULL,NULL,$f_1$) para primeiro determinar as arestas adjacentes a f_1. Depois o operador máscara é chamado $\bowtie\!\!\blacktriangleleft_0$(NULL,$e_i$,NULL) para cada aresta e_i para determinar os vértices da aresta e_i e f_1.

6. $\bowtie\!\!\blacktriangleleft_1$(NULL,NULL,$f_1$)={$e_2$,$e_3$,$e_5$}, devolve directamente todas as arestas adjacentes à face f_1.
7. $\bowtie\!\!\blacktriangleleft_2$($v_3$,$e_1$,NULL) ={$f_2$}, devolve directamente as faces incidentes em e_1 e v_3.
8. $\bowtie\!\!\blacktriangleleft_1$($v_1$,$e_2$,$f_1$)={$e_3$}, devolve directamente as arestas adjacentes a f_1 incidentes em v_1 e diferentes de e_2.
9. $\bowtie\!\!\blacktriangleleft_0$($v_4$,NULL, f_1)={v_1,v_3}, devolve indirectamente todos os vértices da face f_1 diferentes de v_4. Requer uma chamada intermédia ao operador máscara $\bowtie\!\!\blacktriangleleft_1$(NULL,NULL,$f_1$) para primeiro determinar as arestas adjacentes a f_1. Depois o operador máscara é chamado $\bowtie\!\!\blacktriangleleft_0$($v_4$,$e_i$,NULL) para cada aresta e_i para determinar os vértices de e_i e f_1 diferentes de v_4.
10. $\bowtie\!\!\blacktriangleleft_0$($v_1$,NULL,NULL)={$v_2$,$v_3$,$v_4$}, devolve indirectamente todos os vértices adjacentes a v_1. Requer uma chamada intermédia ao operador máscara $\bowtie\!\!\blacktriangleleft_1$($v_1$,NULL,NULL) para determinar primeiro as arestas incidentes em v_1. Depois o operador máscara é chamado $\bowtie\!\!\blacktriangleleft_0$($v_1$,$e_i$,NULL) para cada aresta e_i para determinar os vértices de cada aresta e_i diferentes de v_1.

Com o operador máscara não existe a necessidade de manipular todas as células da estrutura de dados. Ele permite-nos manipular a malha localmente. Assim, o seu desempenho é independente do tamanho da malha. Isto é muito importante na manipulação de malhas com grande número de células (vértices, arestas e faces), em particular em operações em tempo real.

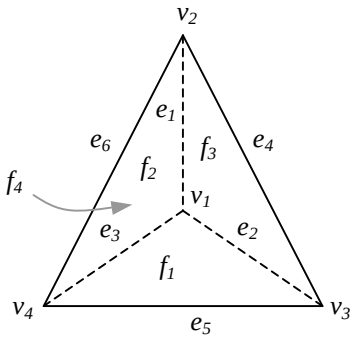

Figura 4: Malha *manifold*.

4. IMPLEMENTAÇÃO DA ESTRUTURA AIF

A estrutura de dados AIF representa malhas que consistem em conjuntos de células, vértices, arestas e faces. Cada tipo de célula foi codificado através de uma classe em C++, mas também se poderia codificar todos os tipos de células com uma única classe, como é necessário para malhas de dimensão *n*. Isto é possível porque qualquer célula de dimensão *n* é vista como um conjunto de células incidentes de dimensão (*n*+1) e um conjunto de células adjacentes de dimensão (*n*-1). A estrutura de dados AIF foi então codificada do seguinte modo:

```
class Vertex {
  evector li;    //arestas incidentes
  Point *pt;     //geometria
}
class Edge {
  Vertex*v1,*v2;//vertices adjacente
  fvector li;    //faces incidentes
}
class Face {
  evector la;    //arestas adjacentes
  Point *nv;     //normal à face
}
class Mesh {
  int ID;        //identificador
  vvector vv;    //vector de vértices
  evector ev;    //vector de arestas
  fvector fv;    //vector de faces
}
```

De notar que a estrutura de dados AIF não é topologicamente orientada, pois não possui células orientadas. Ela é geometricamente orientada pela normal à face na classe `Face`. As normais às faces são determinadas por indução topológica duma orientação consistente nas faces como, por exemplo, a orientação no sentido dos ponteiros do relógio (Figura 5). Isto requer a travessia da fronteira de todas as faces no mesmo sentido. Para isso, aplica-se alternadamente os operadores máscara indexados $\bowtie\!\!\blacktriangleleft_0$ e $\bowtie\!\!\blacktriangleleft_1$ às arestas e vértices adjacentes a cada face. O operador $\bowtie\!\!\blacktriangleleft_0$ retorna o próximo vértice na fronteira da face, e o operador $\bowtie\!\!\blacktriangleleft_1$ devolve a próxima aresta adjacente à face. Isto é, percorrendo a malha com o auxílio do operador máscara é possível induzir uma orientação na malha.

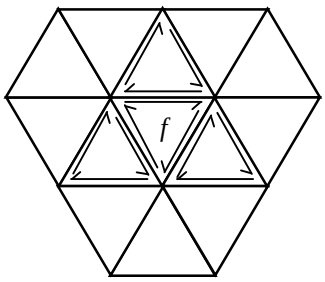

Figura 5: Mecanismo de orientação.

Consideremos a malha da Figura 4 e vejamos como com o auxílio do operador máscara podemos percorrer uma face. Consideremos, por exemplo, a face f_4. O operador $\bowtie\!\!\blacktriangleleft_0$(NULL,NULL,$f_4$) devolve indirectamente todos os vértices da face f_4, ou seja, {v_2,v_3,v_4}. Partindo agora de um vértice qualquer, por exemplo, v_2 podemos chamar o operador máscara $\bowtie\!\!\blacktriangleleft_1$($v_2$,NULL, f_4) que nos devolverá as arestas que contém v_2 e pertence a f_4, ou seja, {e_6,e_4}. Agora escolhendo uma das aresta, por exemplo, e_4 podemos chamar de novo o operador $\bowtie\!\!\blacktriangleleft_0$($v_2$,$e_4$,$f_4$) que nos devolverá o próximo vértice da aresta e_4 diferente de

v_2 na face f_4, ou seja, v_3. Depois o operador $\triangleright\!\!\blacktriangleleft_1(v_3, e_4, f_4)$ devolve-nos a próxima aresta incidente em v_3 diferente de e_4 na face f_4, ou seja, e_5. Continuando este processo podemos percorrer toda face.

5. ESTRUTURA DE DADOS - COMPARAÇÕES

5.1. Memória

A Tabela 1 compara diversas estruturas de dados em termos da memória necessária ao armazenamento de uma malha triangular com *n* vértices. A coluna "Tipo de malha" designa o tipo de malha suportada pela estrutura de dados, ou seja, triangular (Δ) ou poligonal (Pol.), enquanto a coluna "não *manifold*" indica se a estrutura de dados suporta malhas não-*manifold* ou não. A última coluna "Bytes/Δ" designa o número de bytes necessários ao armazenamento de uma face triangular. A variável *k* designa o grau do vértice, isto é, o número de arestas incidentes no vértice.

A estrutura de dados Triangle List é a estrutura mais concisa porque armazena apenas as faces e os vértices. Normalmente, esta estrutura de dados é usada apenas para visualização visto que o hardware gráfico está optimizado para triângulos.

Estruturas de dados	Tipo de malha	Não *manifold*	Bytes/Δ
Triangle list	Δ	sim	18
Star-vertices	Pol.	sim	10+4*k*
Progressive Meshes	Δ	não	33
Tri-edges	Δ	não	35
AIF	Pol.	sim	29+2*k*
PSC	Δ	sim	37+2*k*
Directed-edges	Δ	sim	44
Half-edge	Pol.	não	46
FastMesh	Δ	não	53
Radial edge	Pol.	sim	56
Winged-edge	Pol.	não	60

Tabela 1: Comparação do espaço em memória.

A estrutura de dados Star-vertices apenas armazena vértices e os vértices vizinhos daqueles. A cada vértice vizinho está associado um índice que permite efectuar a navegação à volta dum dado vértice segundo uma ordem pré-definida. É uma estrutura de dados concisa pois apenas necessita de 10+4*k* bytes por triângulo. Além disso, esta estrutura de dados suporta malhas genéricas, mas os acessos à informação sobre adjacências são lentos, pois não inclui arestas e faces explicitamente. A ausência de arestas e faces implica que seis relações de adjacência/incidência não estejam definidas (veja-se na coluna "Não definidas" da Tabela 2).

As estruturas de dados Progressive Meshes [Hoppe98] e Tri-edges [Loop00] são também bastante concisas, visto que necessitam apenas de 33 e 35 bytes por triângulo, respectivamente. No entanto, estas estruturas de dados apenas suportam malhas triangulares *manifold*.

Por outro lado, uma malha triangular com *n* vértices na estrutura de dados AIF requer aproximadamente 35, 37, 39, 41 bytes por triângulo para k=3,4,5 e 6, respectivamente. De facto, de acordo com a fórmula de Euler para uma malha bidimensional triangular, uma malha com *n* vértices tem cerca de *m* faces (m=2n) e *a* arestas (2m=3a). Assim, assumindo que um real e um ponteiro ocupam 4 bytes cada, a memória necessária a uma malha triangular na estrutura de dados AIF é: (3×4+4k)n=(12+4k)n=(6+2k)m bytes para as coordenadas dos vértices, (2×4+2×4)e=16e≈11m bytes para as arestas (referências aos vértices e faces) e (3×4)m=12m bytes para as faces (referências às arestas).

Note-se que a estrutura de dados AIF não inclui células orientadas (e.g. half-edges). Por isso é mais concisa que os b-reps tradicionais (e.g. Winged-edge [Baumgart72]) e as suas variantes orientadas (e.g. Half-edge [Mantyla88], Directed-edges [Campagna98], FastMesh [Pajarola01], e Radial Edge [Weiler88]). Por fim, a estrutura de dados PSC [Popovic97] suporta apenas malhas triangulares e requer mais memória que a estrutura de dados AIF.

5.2. Acessos às Adjacências e Incidências

A Tabela 2 mostra-nos o número de acessos necessários para aceder à informação de adjacência e incidência representada em várias estruturas de dados. Visto que temos três entidades topológicas (vértice, aresta e face), logo existem nove relações de adjacência e incidência [Weiler85].

Na Tabela 2 a coluna "Classe" dá-nos o número de relações de adjacência e incidência representadas explicitamente na estrutura de dados. Por exemplo, uma estrutura de dados C_4^9 tem quatro relações explicitamente representadas em nove relações possíveis. Isto é, temos acesso directo a quatro relações sem nenhum processamento adicional. As restantes cinco relações estão representadas implicitamente, visto que todas as estruturas de dados são completas, ou seja, representam malhas sem ambiguidades. Isto significa que temos de ter mecanismos de inferência para aceder a estas cinco relações de adjacência e incidência. A estes mecanismos de inferência chamamos acessos indirectos. Acontece que os acessos indirectos são mais difíceis porque algumas células topológicas não existem nas estruturas de dados, referidas na coluna "Não definidas". Por exemplo, a estrutura de dados Star-vertices não possui arestas e faces representadas explicitamente; como consequência, seis relações de adjacência/incidência não estão definidas (Tabela 2).

Na Tabela 2 podemos ver que as estruturas de dados AIF e PSC são as mais rápidas em termos do número de acessos. Claro, que a rapidez depende do número de relações de adjacência e incidência explicitamente armazenadas na estrutura de dados, mas quantas mais relações armazenamos mais memória é necessária. Ambas as estruturas de dados AIF e PSC pertencem à classe C_4^9. De acordo com Ni e Bloor [Ni94], podemos dizer que ambas são estruturas de dados óptimas nesta classe, pois um número mínimo de acessos é necessário para aceder a toda informação sobre adjacências e incidências. Contudo, a estrutura de dados PSC apenas

suporta malhas triangulares e necessita de mais memória (ver Tabela 1).

Estrutura de dados	Não definidas	Classe
AIF	0	C_4^9
PSC	0	C_4^9
Winged-edge	0	C_2^9
Radial edge	0	C_2^9
Half-edge	0	C_2^9
FastMesh	3	C_2^9
Directed-edges	3	C_2^9
Triangle List	6	C_2^9
Star-vertices	6	C_2^9
Progressive Meshes	6	C_2^9
Tri-edge	6	C_2^9

Tabela 2: Comparação do número de acessos.

As outras estruturas de dados pertencem à classe C_2^9 (dois acessos directos e sete acessos indirectos para obter toda a informação sobre adjacências e incidências). Por exemplo, a estrutura de dados FastMesh não possui a entidade vértice. Assim, as três relações de adjacência e incidência envolvendo vértices não estão definidas, podendo, no entanto, ser inferidas com um custo adicional em termos de tempo. As restantes seis relações envolvendo arestas e faces já podem ser obtidas mais facilmente mas apenas duas estão explícitas na estrutura de dados.

5.3. Desempenho da estrutura de dados AIF

O desempenho da estrutura de dados AIF foi testado num PC equipado com um processador Pentium 4, com 768MB de memória, uma placa gráfica Nvidia Riva TNT2, e com o sistema operativo Windows 2000. A Tabela 3 mostra-nos alguns resultados para diferentes modelos AIF (Figura 2).

Malha	#V	#F	Ler o ficheiro	Orientar	Criar primitiva	Desenhar primitiva
Vaca	2904	5804	0.180	0.060	0.020	0.040
Avião	3745	3946	0.180	0.040	0.020	0.030
Avião	6795	13546	0.491	0.130	0.030	0.040
Dragão	25418	50761	1.622	0.661	0.240	0.351
Coelho	34835	69473	2.674	1.001	0.311	0.480
Cavalo	48485	96966	3.374	1.372	0.401	0.651
Venus	50002	100000	3.445	1.442	0.441	0.681

Tabela 3: Tempos de inicialização e *rendering* em segundos.

A criação de um modelo AIF a partir de um ficheiro de texto inclui também a orientação da malha através do mecanismo de indução topológica. A visualização de um modelo AIF envolve duas etapas:

i) a criação da primitiva GL (Graphics Library);

ii) desenhar a primitiva no ecrã.

Olhando para a Tabela 3 podemos ver que a visualização de malhas *manifold* (ex. Venus, o cavalo e o coelho da Figura 2) ou malhas poligonais não-*manifold* (e.g. avião da Figura 2) é rápida, mesmo considerando que o *rendering* é efectuado por software e não por hardware.

6. CONCLUSÃO E TRABALHO FUTURO

Este artigo introduz uma nova estrutura de dados, chamada AIF, juntamente com um operador de pesquisa para satisfazer três requisitos fundamentais, nomeadamente:

- *Generalidade*. A estrutura de dados AIF suporta modelos poligonais e não apenas triangulares, *manifold* e não-*manifold*, mesmo para dimensões superiores.
- *Concisão*. Ao contrário das outras estruturas de dados b-rep, a estrutura de dados AIF não é orientada, ou seja, não contém células orientadas. Assim, é mais concisa que os b-rep tradicionais.
- *Rapidez*. A estrutura de dados AIF foi concebida para assegurar pesquisas rápidas através de um único operador, o operador máscara. Este permite aceder rapidamente à informação sobre adjacências e incidências independentemente do tamanho da malha.

Na prática, a estrutura de dados AIF e o operador máscara provaram ser poderosos no acesso à informação sobre adjacências e incidências. Em termos de eficiência no acesso à informação sobre adjacências e incidências apresenta a melhor performance na classe C_4^9 tal como a estrutura de dados PSC. No entanto a estrutura de dados AIF necessita de menos memória e suporta malhas poligonais genéricas ao contrário da estrutura de dados PSC.

No futuro esperamos que a estrutura de dados AIF possa ser utilizada em diferentes tipos de aplicações devido à sua generalidade, desempenho e concisão como, por exemplo, na multi-resolução, nos sistemas de animação e realidade virtual.

AGRADECIMENTOS

Os modelos utilizados são cortesia de:
Cyberware (http://www.cyberware.com) - Cavalo e Venus;
Stanford University (http://graphics.stanford.edu) - Coelho;
Viewpoint Digital Inc. (http://www.viewpoint.com) - Avião;
Avalon archive (http://avalon.viewpoint.com/) - Vaca;
3D Cafe (http://www.3dcafe.com/) - Dragão.

REFERÊNCIAS

[Baumgart72] B. G. Baumgart. Winged-edge polyhedron representation. Technical report, STAN-CS-320, Stanford University, 1972.

[Brisson93] E. Brisson. Representing geometric structures in *d* dimension: topology and order. *Descrete & Computational Geometry*, Vol.9, no.4, pp.387-426, 1993.

[Campagna98] S. Campagna, L. Kobblet, and Hans-Peter Seidel. Directed Edges - A Scalable Representation for

Triangular Meshes. *Journal of Graphics Tools: JGT*, Vol. 3. no.4, pp.1-12, 1998.

[Carey97] R. Carey, G. Bell, and C. Marrin. The Virtual Reality Modeling Language ISO/IEC 14772-1, 1997. *http://www.vrml.org/technicalinfo/specifications/vrml97/index.htm*.

[Garland99] M. Garland. Multiresolution modeling: Survey & future opportunities. *In Eurographics '99 - State of the Art Reports*, pp.111-131, 1999.

[Heckbert97] P. S. Heckbert and M. Garland. Survey of Polygonal Surface Simplification Algorithms. *Siggraph'97 Course Notes 25*, 1997.

[Hoppe96] H. Hoppe. Progressive Meshes. *In Proceedings of Siggraph'96*, pp.99-108, 1996.

[Hoppe98] H. Hoppe. View-Dependent Refinement of Progressive Meshes. *Technical Report MSR-TR-98-02*, Microsoft Research, 1998.

[Kallmann01] M. Kallmann and D. Thalmann. Star-vertices: A Compact Representation for Planar Meshes with Adjacency Information. *JGT*, Vol.6, no.1, 2001.

[Lee99] A. Lee, D. Dobkin, W. Sweldens and P. Schroder. Multiresolution Mesh Morphing. *In Proceedings of Siggraph'99*, pp.343-350, 1999.

[Lee01] S. H. Lee and K. Lee. Partial Entity Structure: A Fast and Compact Non-Manifold Boundary Representation Based on Partial Topological Entities. *In Proceedings of the Sixth ACM Symposium on Solid Modeling and Applications*, pp.159-170, 2001.

[Loop00] C. Loop. Managing Adjacency in Triangular Meshes. *Tech. Report No. MSR-TR-2000-24*, Microsoft Research, January 2000.

[Mantyla88] M. Mäntylä. *An Introduction to Solid Modeling*, Computer Science Press, 1988.

[Ni94] X. Ni, and M. Susan Bloor. Performance Evaluation of Boundary Data Structures. *IEEE Computer Graphics and Applications*, Vol.14, no.6, pp.66-77, 1994.

[Pajarola01] R. Pajarola. FastMesh: Efficient View-dependent Meshing. *In Proceedings of the Pacific Graphics 2001*, pp.22-30, 2001.

[Popovic97] J. Popovic, H. Hoppe. Progressive Simplicial Complexes. *Computer Graphics*, Vol.31, pp.217-224, 1997.

[Taubin98] G. Taubin and J. Rossignac. Geometric compression through topological surgery. *ACM Transactions on Graphics*, Vol.17, no.2, pp.84-115, 1998.

[Wavefront93] Wavefront Technologies, Inc., Wavefront File Formats, Version 4.0 RG-10-004, first ed. Santa Barbara. CA. 1993.

[Weiler88] K. Weiler. The radial edge structure: a topological representation for non-manifold geometric boundary modeling. *In Geometric modelling for CAD applications.* M. Wozny, H. McLaughlin, and J. Encarnacao, (eds.), North-Holland, pp. 3-36, 1988.

[Weiler85] K. Weiler. Edge-Based data structure for solid geometric in Curved-Surface Environments. IEEE CG&A, Vol.5, no.1, pp.21-40, 1985.

[Zorin97] D. Zorin, P. Schröder and W. Sweldens. Interactive Multiresolution Mesh Editing. *In Proceedings of the Siggraph'97*, pp.259-268, 1997.