Esquema de Multiresolução para Aplicações Geométricas em Tempo Real

Rui Rodrigues José Morgado
ESTV
Viseu
{rsrodrigues,fmorgado}@di.estv.ipv.pt

Frutuoso Silva Abel Gomes
UBI
Covilhã
{fsilva,agomes}@di.ubi.pt

Resumo

Este artigo descreve um novo esquema multiresolução que manipula a geometria e topologia duma forma separada. A sua estrutura de dados designa-se por estrutura de dados baseada na aresta fantasma*. Este esquema recorre a um critério topológico (não a um critério geométrico, como é usual) para simplificar a malha mais rapidamente, o que deixa em aberto a sua utilização em aplicações em tempo real.*

Palavras-Chave

Multiresolução, Níveis de Detalhe (LODs), Simplificação e Refinamento

1. INTRODUÇÃO

O realismo e o desempenho em tempo real são hoje em dia características importantes em ambientes virtuais, jogos de vídeo, etc. Mas aumentar o realismo dos objectos significa normalmente incrementar o seu detalhe, o que acarreta não só mais memória ocupada mas também menor velocidade de processamento.

Há basicamente duas técnicas para resolver estes problemas: LODs e esquemas multiresolução. Os LODs têm sido a solução mais usada em aplicações em tempo real. A característica fundamental dos LODs é que mantêm várias instâncias do mesmo objecto em memória simultaneamente, embora com resoluções diferentes, o que se traduz por um significativo desperdício de memória. Por outro lado, um esquema multiresolução só guarda uma única instância dum objecto em memória, mas o seu menor desempenho temporal tem limitado a sua utilização em aplicações de tempo real. O seu menor ou maior detalhe é conseguido por aplicação de algoritmos de simplificação e refinamento, respectivamente. No entanto, segundo Garland [Garland 99], a instância corrente em memória deve possuir aproximadamente o mesmo tamanho que a sua instância mais detalhada.

As principais contribuições são, pois, as seguintes:

- uma estrutura de dados de multiresolução integrada onde existe uma separação entre informação topológica e informação geométrica, o que permite a definição de um conjunto de regras de transição independentes da forma;
- um critério puramente topológico —ao invés dos usuais critérios geométricos— que permite acelerar o processamento do processo de simplificação de cenas ou objectos geométricos.

2. TRABALHO RELACIONADO

Há duas classes de modelos de multiresolução: *modelos de subdivisão recursiva* e *modelos de subdivisão não-recursiva*. Os primeiros foram desenvolvidos para representar objectos e superfícies através da subdivisão recursiva das células duma malha poligonal [Sabin 02] [Floriani 02]. Por exemplo, as *quadtrees* representam um tipo de subdivisão recursiva. Os segundos são modelos que resultam da aplicação de algoritmos de refinamento e de simplificação que permitem criar malhas mais ou menos detalhadas, respectivamente. Refira-se que o esquema de malhas de multiresolução aqui proposto é não-recursivo.

A natureza dos elementos que constituem uma malha tem um impacto significativo no desempenho das estruturas de dados das malhas multiresolução [Silva 05]. Por exemplo, assuma-se que uma malha triangular está codificada como um conjunto de triângulos, e cada um deles é codificado por um tuplo de três vértices. Neste caso, encontrar as faces incidentes num determinado vértice pode ser uma operação que consome muito tempo porque pode ser necessário analisar todas as faces na estrutura de dados.

Existem dois modos para aumentar o desempenho destes algoritmos de pesquisa e atravessamento:

- *Orientabilidade*. Estruturas de dados topologicamente orientadas é a solução que muitos investigadores encontraram para aumentar a velocidade dos respectivos algoritmos, *Half-Edge* [Mäntyla 88] e *Radial Edge* [Weiler 88]. Existem também estruturas de dados não orientadas, ainda que orientáveis, como por

exemplo, as estruturas PSC (*Progressive Simplicial Complexes*) [Popovic 97] e AIF (*Adjacency and Incidence Framework*) [Silva 05].

- *Esquema de incidência*. Numa relação de incidência relacionam-se vértices, arestas e faces de um objecto geométrico. No entanto, o esquema de incidência pode ter ou não as relações de incidência explicitamente representadas na estrutura de dados. Com base no trabalho de Weiler [Weiler 88], Ni e Bloor [Ni 94] concluiram que a estrutura de dados simétrica C_4^9 é óptima para malhas bi-dimensionais, a qual, como se verá, será adoptada neste artigo.

Além disso, como se refere em [Floriani 02], as estruturas de dados para malhas multiresolução não recursivas dependem do tipo de elemento da malha que é modificado, o que conduz à seguinte classificação:

- *Estruturas de dados baseadas em operações sobre arestas*. Associadas a operações de contracção da aresta (*edge collapse*) e de divisão do vértice (*vertex split*). Estas modificações podem ser armazenadas na estrutura de dados como ***dois vectores que descrevem a diferença entre a posição do vértice original e a posição de cada um dos novos vértices*** [Hoppe 96]. Mas esta informação não é suficiente para andar para a frente e para trás na malha multiresolução durante o processo de refinamento e simplificação. Por isso, são necessárias estruturas de dados suplementares para codificar relações de dependência numa malha multiresolução [ES99] [Hoppe 97].

- *Estruturas de dados baseadas em operações sobre vértices*. Estas estão associadas operações de inserção e remoção de vértices. Nas estruturas baseadas em operações sobre vértices temos dois tipos de operações: remoção de células (*cell decimation*) e confluência de vértices (*vertex clustering* [Danovaro 01].

Neste artigo, o algoritmo de simplificação proposto baseia-se na operação de contracção da aresta. Os algoritmos desta família distinguem-se uns dos outros pela forma como seleccionam a aresta a contrair. Em *Mesh Optimization* [Hoppe 93] e nas *Progressive Meshes* [Hoppe 96] a selecção da aresta a contrair é feita com base na minimização da chamada função de energia. Na *Surface Simplification Using Quadric Error Metrics* [Garland 97], o critério baseia-se na minimização do erro associado a cada vértice. Já em *Normal-based Simplification Algorithm*(NSA) [Silva 04], a escolha da aresta a contrair é feita com base na variação das normais às faces em redor da aresta alvo. A grande diferença entre estes critérios geométricos de simplificação e aquele aqui proposto é que o nosso critério é puramente topológico, pois só usa informação de incidência entre células. É, pois, expectável que o desempenho global do esquema de multiresolução por nós proposto seja significativamente maior.

3. ESTRUTURA DE DADOS

3.1. Descrição

A *estrutura de dados baseada na célula fantasma* (EDCF) é uma estrutura de dados especificamente desenvolvida para malhas de multiresolução. O seu núcleo não-multiresolução é o mesmo que a da estrutura de dados AIF (*Adjacency and Incidence Framework*) proposta por Silva e Gomes [Silva 05]. A camada multiresolução é completamente distinta.

À semelhança da AIF, é uma estrutura de dados C_4^9 óptima que basicamente codifica as células fronteira da malha. Por isso, suporta não só a representação de malhas triangulares, mas também malhas poligonais. Além disso, é bastante concisa, pois não faz uso de células topologicamente orientadas. Segue-se a sua codificação em C++:

```
class Vertex {
  Identifier *vid; //id. do vértice
  List<Edge> loe;  //arestas incidentes
  Point3D *pt;     // x, y, z coordenadas
  BinTree <VertexGene> *bog;
}
class Edge {
  int eid;         //id da aresta
  Vertex *v1,*v2; // vért. adjacentes
  List<Face> lof; //faces incidentes
  Stack<Identifier,Identifier> *soii;
}
class Face {
 int fid;         //id face
 List<Edge> loe; // arestas adjacentes
 Vector3D *v3d;  // normal à face
}
class Mesh {
  int mid;          //id malha
  int ln;           // nível de detalhe
  List<Vertex> lov; //vertices
  List<Vertex> logv;//reciclagem vert.
  List<Edge> loe;   //arestas
  List<Edge> loge;  //arestas fantasma
  List<Face> lof;   //faces
  List<Face> logf;  //faces fantasma
}
```

Portanto, a diferença principal entre a codificação da AIF e a da EDCF está na existência de duas genealogias, uma para os vértices outra para as arestas.

3.2. Genealogia do vértice

A genealogia dos vértices, codificada na *class Vertex* como `BinTree <VertexGene> *bog`, é uma árvore binária onde é guardada informação sobre os antecessores dos vértices. Esta técnica é similar à floresta de vértices proposto por El-Sana, ou seja, guarda a história familiar do vértice [ES99]. Mas, ao contrário daquela, a genealogia do vértice não fica com informação suplementar de incidências durante o processo de simplificação/refinamento.

A codificação foi feita da seguinte forma:

```
class VertexGene{
```

```
  Identifier *leftAncestorID;
  Identifier *rightAncestorID;
  Vector3D *v3d;  //  vector de Hoppe
}

class Identifier {
  int id;     // número do vértice
  int level;  // nível multiresolução
}

template<class T> class BinTree {
  T data;
  BinTree<T> *left, *right;
}
```

Numa malha ainda no nível 0 de simplificação, procede-se à contracção duma aresta limitada pelos vértices P e Q, num novo vértice N (ponto médio de P e Q). É também, necessário proceder ao armazenamento de informação que permita mais tarde reconstruir os vértices P e Q a partir do vértice N.

Essa informação, que designamos **genealogia do vértice**, é armazenada na variável `data` da árvore binária que é do tipo `VertexGene` que guarda a identidade dos dois antecessores P, Q, nas variáveis `leftAncestorID` e `rightAncestorID`, respectivamente. É também armazenada a informação relativa ao vector $\vec{v} = \frac{Q-P}{2}$ no campo `v3d` da classe `VertexGene` que vai permitir recuperar a geometria dos vértices antecessores P, Q durante o processo de refinamento. Para optimizar o processo de simplificação optou-se por aproveitar a estrutura do vértice P e actualizar a informação com os dados do novo vértice N. Enquanto o vértice Q é colocado na lista de vértices para reciclagem da malha ($logv$), que não é mais que uma lista onde ficam os vértices eliminados, que poderão ser reutilizados posteriormente durante o refinamento.

A evolução da árvore genealógica, no que diz respeito aos identificadores da malha de um cubo (com 8 vértices) durante o seu processo de simplificação é mostrado na Figura 1. Notar que após a primeira simplificação, tem-se uma malha M_1 com 4 novos vértices, V_8, V_9, V_{10} e V_{11}, em que cada um tem uma árvore genealógica de identificadores dos seus vértices antecessores. Em M_3, obtemos um só vértice V_{14} com uma única árvore genealógica de identificadores. O processo de simplificação implica pois, a união dos vértices e as suas árvores genealógicas.

3.3. Genealogia das arestas

A *genealogia das arestas* é codificada como uma pilha de pares de identificadores de vértices:

```
Stack<Identifier,Identifier> *soii
```

Ao contrário do que acontece com os vértices, as arestas contraídas não são removidas da memória. É guardada a identificação dos vértices que a delimitam `v1` e `v2`, criando assim a genealogia da aresta, que está implementada sob a forma de uma pilha genealógica. Segue-se a contracção da aresta propriamente de modo que `v1` e `v2` passam a ser o mesmo vértice. Este facto inibe a

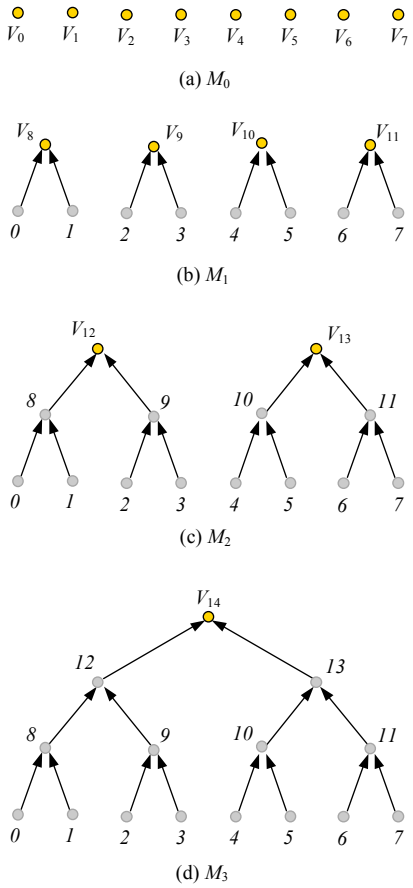


Figura 1. Árvore genealógica dos vértices da malha de um cubo.

visualização gráfica da aresta contraída (fica reduzida a um ponto); daí a designação de *aresta fantasma*.

A genealogia da aresta é codificada do seguinte modo:

```
class Stack {
  array< Identifier, Identifier> a;
  int top; // index do topo da pilha
}
```

Na Figura 2 pode observar-se as pilhas das arestas do cubo após cada uma das quatro etapas de simplificação, M_0, M_1, M_2 e M_3. Note-se que todas as 12 arestas são sempre mantida em memória. As arestas preenchidas a cinzento (V_8, V_8), (V_9, V_9), (V_{10}, V_{10}) e (V_{11}, V_{11}) em M_1 são as arestas que resultam da contracção das arestas (V_0, V_1), (V_2, V_3), (V_7, V_4) e (V_5, V_6) de M_0, respectivamente.

Além da aresta contraída, outras arestas também foram actualizadas. É este processo de *reetiquetagem*[1] dos identificadores dos vértices que delimitam as arestas que permite construir a pilha genealógica de cada aresta. Esta reetiquetagem das arestas tem duas etapas. Na primeira, efectua-se a *contracção da aresta*, i.e., ao contrair uma aresta delimitada por dois vértices V_i e V_j num novo vértice V_k torna-

[1] Os ponteiros v1 e/ou v2 passam a apontar para outros vértices

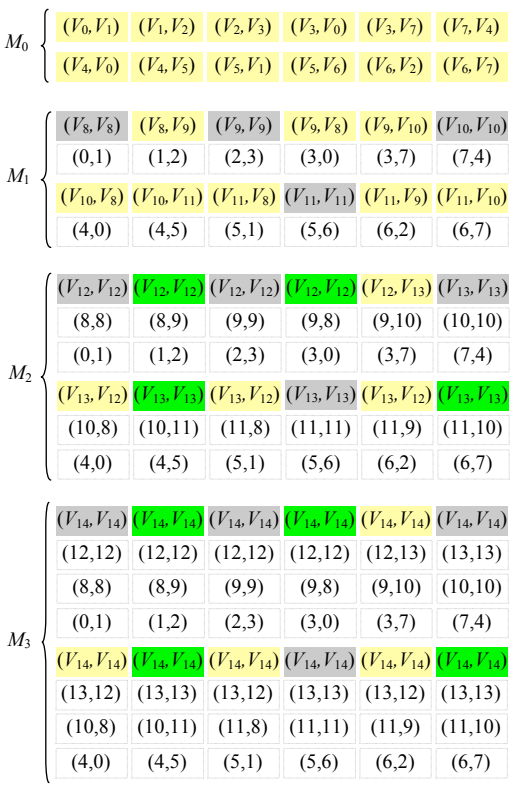


Figura 2. Genealogia das arestas do cubo em diferentes níveis de simplificação.

se obrigatório reetiquetar V_i e V_j como V_k e V_k, respectivamente. Na segunda executa-se a chamada *cadeia de reetequitagem* devido ao facto dos vértices V_i e V_j deixarem de existir na lista de vértices, é então necessário encontrar todas as arestas que são delimitadas por V_i ou V_j, reetiquetando-os depois como V_k e V_k, respectivamente.

3.4. Tratamento das faces

Uma vez que as faces triangulares são delimitadas por arestas, e estas não são eliminadas da memória mesmo quando são contraídas (comprimento nulo), acontece que qualquer face com alguma aresta contraída reduz-se graficamente a uma aresta, ou seja, é automaticamente transferida para a lista $logf$ de faces invisíveis.

4. ALGORITMOS MULTIRESOLUÇÃO

4.1. Algoritmo de Simplificação

O principal objectivo de um algoritmo de simplificação é reduzir o número de células (i.e. vértices, arestas e faces) de uma malha, mas mantendo a forma global do objecto.

Uma aresta é contraída se verificar simultaneamente duas condições. A primeira é que *os vértices que a delimitam sejam diferentes*, o que é equivalente a dizer que uma aresta contraída não pode ser contraída outra vez no actual nível de simplificação. A segunda é que *não pode pertencer à asa de uma aresta contraída durante o actual nível de simplificação*. Por exemplo, no nível multiresolução M_0 do cubo da Figura 2, a aresta (V_0, V_1), vai ser contraída. Logo, as suas arestas aladas (V_3, V_0), (V_4, V_0), (V_1, V_2), e (V_5, V_1) não podem ser contraídas neste nível.

Este critério de simplificação é puramente topológico porque usa só as relações de incidência para permitir, ou não, a contracção de uma aresta. Apesar disso, a forma do objecto é preservada. Este algoritmo tem vantagem que é muito mais rápido do que aqueles que usam critérios geométricos, pois não é necessário fazer cálculos geométricos intensivos. Por exemplo, o critério geométrico dos algoritmo NSA requer o cálculo da variação das normais às faces em redor da aresta. A aresta só é contraída se a variação das normais for pequena e abaixo de um dado limiar.

O algoritmo percorre a lista de arestas e se uma aresta (V_i, V_j) verificar o critério de contracção, então esta é contraída do seguinte modo:

1. Calcula o seu ponto médio $P(x, y, z)$ da aresta (V_i, V_j).
2. Determina o vector de Hoppe $\vec{v}_H = V_i - P$.
3. Redefine V_i como o novo vértice V_{n+1} com coordenadas dadas pelo ponto P, onde n representa o identificador máximo dos vértices existentes.
4. Guarda a informação genealógica de V_{n+1}, assim como o seu $\vec{v}_H$. É criado o nó com o vector de Hoppe e a informação genealógica dos vértices que deram origem ao vértice V_{n+1}.
5. Define as sub-árvores genealógicas de V_{n+1}, i.e. se existir informação genealógica associada a V_i e V_j, esta informação passa a ser as subárvores genealógicas esquerda e direita, respectivamente, do novo vértice V_{n+1}.
6. Actualiza a lista de arestas da malha.
7. Actualiza a lista de arestas incidentes em V_{n+1} acrescentando a lista de arestas incidentes em V_j. No entanto, há que evitar possíveis repetições de arestas.
8. Insere V_j na lista de reciclagem de vértices da malha.

4.2. Algoritmo de Refinamento

O principal propósito de um algoritmo de refinamento é reconstruir uma malha mais refinada a partir de uma mais simplificada. Para refinar uma malha são utilizados os dados armazenados *na árvore genealógica de cada vértice visível*, i.e., usam-se os identificadores dos antecessores e o vector de Hoppe para reconstruir a geometria dos seus vértices antecessores. É também utilizada *a pilha genealógica de cada aresta* de modo a reconstruir as relações topológicas entre as células da malha.

O algoritmo de refinamento vai operar sobre a lista de vértices visíveis da malha M_i que foram criados no nível i. Para cada um destes vértices V_k, o refinamento decorre assim:

1. Transferir o primeiro vértice V_j da lista de reciclagem de vértices para a lista de vértices visíveis.

2. Restaurar a identidade de V_j a partir do identificador armazenado no nó direito do `VertexGene` da raiz da árvore binária genealógica de V_k.

3. Restaurar o identificador de V_k como V_i a partir do identificador armazenado no nó esquerdo do `VertexGene` que está na raiz da árvore genealógica.

4. Restaurar a geometria de V_i e V_j a partir do vector de Hoppe armazenado no antigo V_k (i.e. o actual V_i).

5. Restaurar as árvores genealógicas de V_i e V_j a partir das sub-árvores esquerda e direita de V_k (o actual V_i).

6. Actualizar arestas que eram delimitadas por V_k, com identificadores dos novos vértices V_i e V_j, fazendo uso da pilha genealógica das arestas.

7. Actualizar as listas de arestas incidentes em V_i e V_j, respectivamente, pela divisão da estrela[2] da aresta do antigo V_k (o actual V_i).

5. RESULTADOS EXPERIMENTAIS

5.1. Custos de Armazenamento

De acordo com a fórmula de Euler, uma malha triangular com n vértices tem cerca de m faces ($m = 2n$) e a arestas ($3m = 2a$). Assumindo que um real e um ponteiro ocupam 4 bytes e um inteiro 2 bytes, a memória necessária para armazenar uma malha na nossa estrutura de dados é:

- $(4+3\times4+4\times k+4)n = (20+4k)n = (10+2k)m$ bytes para os vértices, o que inclui 1 real para o identificador, 3 reais para as coordenadas, k ponteiros para as arestas incidentes (k é número de arestas incidentes) e 1 ponteiro para a árvore genealógica do vértice;

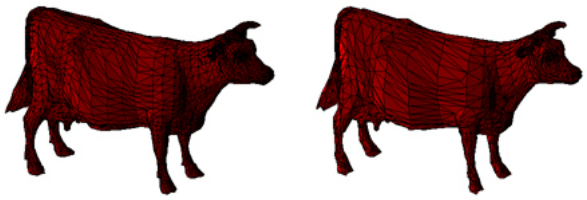
- $(2 + 2 \times 4 + 2 \times 4 + 4)a = (22)a = 33m$ bytes para as arestas, que vem de 1 inteiro para o id da aresta, 2 ponteiros para os vértices adjacentes, 2 ponteiros para as faces incidentes e finalmente 1 ponteiro para genealogia da aresta);

- $(2 + 3 \times 4 + 3 \times 4)m = 26m$ bytes para as faces, de 1 inteiro para o id da face, 3 reais para as arestas adjacentes e 3 reais para as componentes da normal.

No total, o espaço de memória necessário para armazenar uma malha triangular é dado por $(69 + 2k)m$. Com base nesta formula, apresenta-se a seguir uma comparação entre a utilização de LODs e um esquema de multiresolução usando para efeito a malha da vaca da Figura 3 com 5804 faces, 8706 arestas e 2904 vértices.

Em termos matemáticos dois pontos podem ser calculados a partir de um ponto médio e um vector, pode implementar-se então um método para simplificar/refinar um conjunto

[2]A estrela duma aresta é o conjunto das suas arestas aladas



(a) Nível 0 (b) Nível 1

Figura 3. Dois níveis de detalhe consecutivos de uma vaca .

de pontos reduzindo-o a um único ponto, ou então, a partir do ponto reconstruir os pontos originais, como se pode ver na Figura 4. No processo de simplificação de dois pontos, por exemplo, V_0 e V_1 num novo ponto V_8, calcula-se:

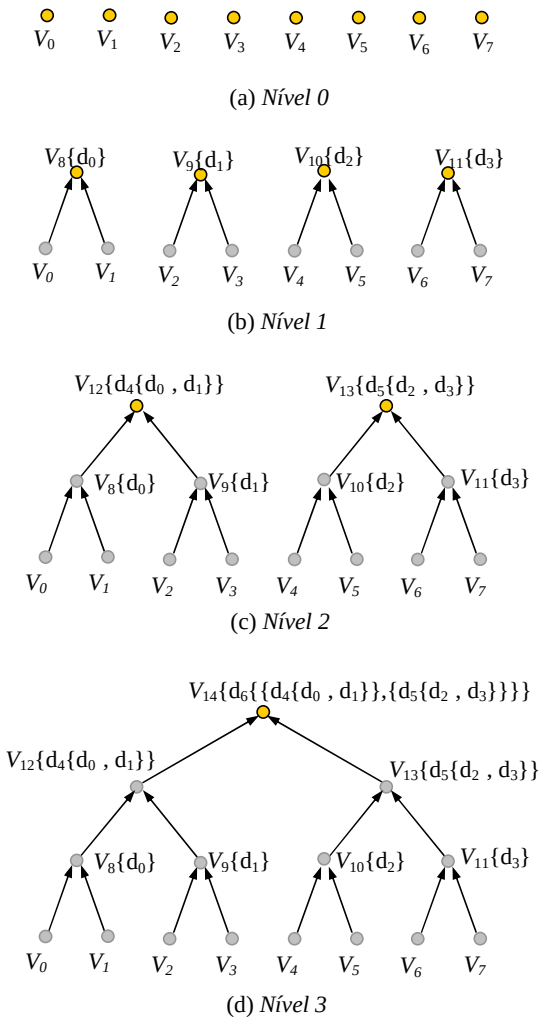
- O ponto médio dos pontos. Sendo portanto $V_8 = (V_0 + V_1)/2$
- O vector $\vec{v} = \vec{d_0}$. Definido por $(V_1 - V_0)/2$.

Partindo do princípio de que *o espaço de armazenamento ocupado por um ponto e um vector é o mesmo, i.e., ambos podem ser representados pela estrutura* `Point3D` e de que, como mostra o esquema de simplificação apresentado na Figura 4, em qualquer nível de simplificação i temos informação que permite construir qualquer um dos outros níveis de detalhe, $i + 1$ ou $i - 1$, pode-se concluir que o espaço necessário para armazenar a informação dos vértices num esquema multiresolução é constante qualquer que seja o nível de detalhe. No nível 0 da Figura 4, existem 8 pontos, i.e, 8 `Point3D`, no nível 1 também se tem 8 `Point3D` de 4 pontos e 4 vectores, e assim por diante até ao nível 3 onde se continua a ter 8 `Point3D` de 1 ponto e 7 vectores.

Na malha original, da Figura 3, com $k = 6$, tem-se um consumo de memória igual a 81×5804, ou seja, 470124 bytes. Esta mesma malha num esquema multiresolução com um nível de simplificação consome $470124 + 8706 \times 4 = 504948$ bytes (8706 é o número de arestas). O valor 8706×4 refere-se ao *overhead* da genealogia das arestas que permanecem em memória.

Agora, num esquema de LODs através da nossa estrutura de dados de multiresolução, 2 níveis de detalhe (original e simplificada com 3008 faces visíveis) resultam num consumo de $470124 + 81 \times 3008 = 713772$ bytes. O mesmo esquema de LODs, mas com a nossa estrutura de dados sem os campos multiresolução, consome $72 \times 5804 + 72 \times 3008 = 634464$ bytes.

A análise dos resultados indica que em qualquer uma das situações os LODs têm um aumento de aproximadamente 50% e 35% do espaço de memória em comparação com a malha original, enquanto na malha multiresolução o aumento do espaço usado é de cerca 7,5%. Ou seja, como era expectável, os resultados das malhas multiresolução são consideravelmente melhores que os dos LODs.



(a) *Nível 0*

(b) *Nível 1*

(c) *Nível 2*

(d) *Nível 3*

Figura 4. Simplificação de um conjunto de pontos.

Há outras estruturas de dados que consomem menos de memória, como por exemplo, as Progressive Meshes (36 bytes por face), as PSC (72 bytes por face), mas tais estruturas de dados foram desenhadas para a visualização de malhas, não para processamento multiresolução. Quando é necessário proceder à manipulação dessas malhas através de operações que vão para além da simples visualização, estas tornam-se muito lentas pois as relações de adjacência/incidência não estão convenientemente definidas. Além disso, a nossa estrutura de dados já integra campos para armazenar a informação auxiliar ao esquema multiresolução, ao passo que outras são omissas em relação a esta matéria, não sendo possível quantificar os valores exactos.

5.2. Desempenho temporal

Recorde-se desde já que em [Silva 04], Silva e Gomes apresentam um conjunto de testes onde se mostra que o algoritmo NSA tem melhores resultados em termos de desempenho temporal do que outros; em particular, tem melhor desempenho do que o algoritmo Qslim [Garland 97], na altura considerado o mais rápido. Portanto, na comparação que a seguir se leva a cabo, só se considera o algoritmo NSA como referência. Nos testes dos algoritmos foi usado um Windows XP PC equipado com um processador Pentium 4 a 3.0GHz, 1GB de RAM, uma placa gráfica Intel 82865G com 96MB.

Na Tabela 1, são apresentados os tempos dos algoritmos NSA e GCSA (*ghost-cell based simplification algorithm*) na simplificação e refinamento da malha do coelho (69473 faces). Os resultados apresentados mostram que o algoritmo GCSA é em média 3 vezes mais rápido do que o algoritmo NSA. Para isso contribuem: *a inexistência de alocação e desalocação de células em/da memória*, o que se deve à existência de listas de reciclagem; *a estrutura de dados ser C_4^9 óptima*, o que facilita o desenvolvimento de algoritmos rápidos de atravessamento de malhas; a existência de *duas listas para cada tipo de célula*, desta maneira aumenta o processamento da geometria e saída gráfica das células visíveis em cada nível de multiresolução; finalmente a utilização de um *critério topológico para simplificação*, característica que acaba por ter um papel importante nos resultados alcançados.

	NSA		GCSA	
Faces	Simp.	Ref.	Simp.	Ref.
69473	0,450	0,347	0,235	0,188
36767	0,252	0,201	0,156	0,093
19967	0,150	0,107	0,094	0,047
11229	0,093	0,054	0,031	0,015
6629	0,056	0,033	0,015	0,010
4099	0,036	0,032	0,005	0,005
2785				

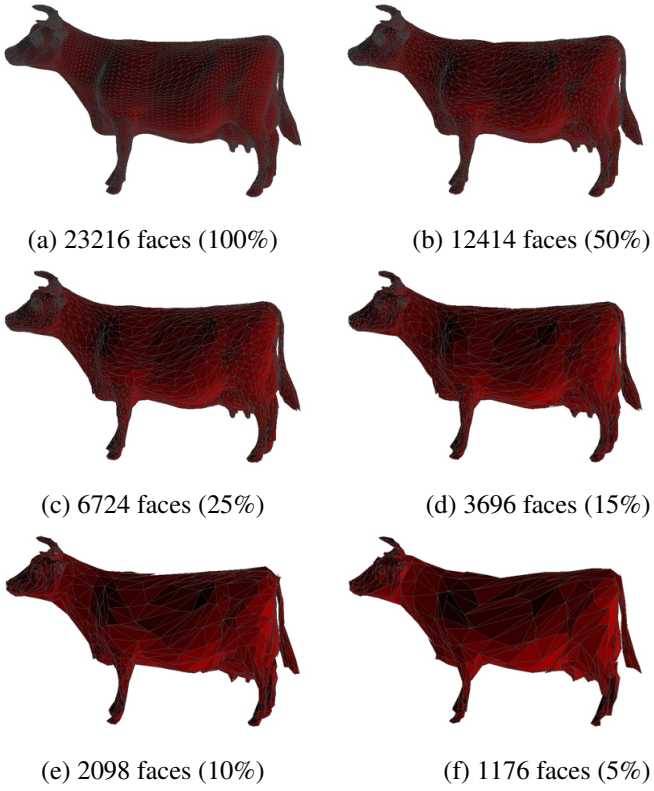
Tabela 1. Tempos de simplificação e refinamento dos algoritmos GCSA e NSA.

5.3. Simplificação e preservação da forma

Como já foi referido, o algoritmo de simplificação GCSA usa um critério topológico e não um critério geométrico, o que pode levantar algumas dúvidas em relação à preservação da forma. A seguir são apresentados alguns dos resultados em termos de preservação da forma e taxa de simplificação.

Os resultados em relação à preservação da forma são visualmente mostrados nas Figuras 5 e 6. Como se pode observar, a forma dos objectos está garantida desde que cada modelo tenha acima de 15% do número inicial de faces.

Na Figura 5 (f) e Figura 6 (f) pode ver-se que a forma dos objectos ainda é mantida na malha com 5% do número de faces originais.



(a) 23216 faces (100%) (b) 12414 faces (50%)

(c) 6724 faces (25%) (d) 3696 faces (15%)

(e) 2098 faces (10%) (f) 1176 faces (5%)

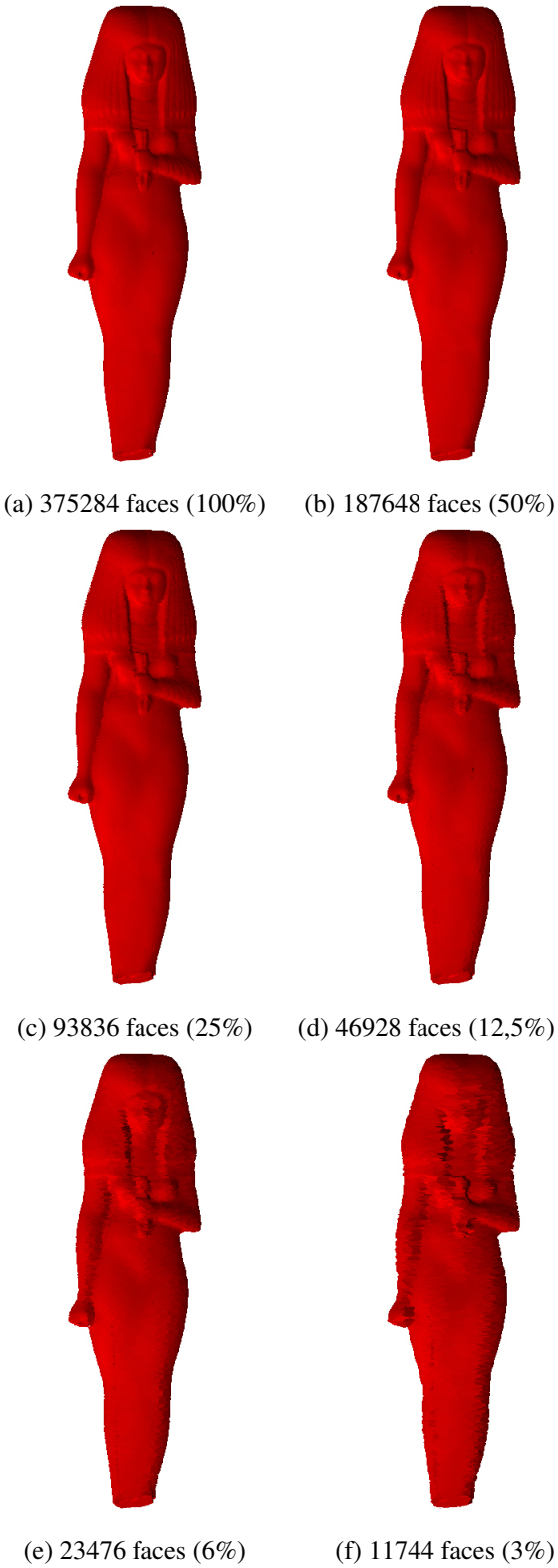
Figura 5. Preservação da forma para seis níveis de detalhe de uma vaca.

Na Figura 7 são apresentados os resultados do algoritmo de simplificação em termos da redução do número de células dos objectos, ou seja, da taxa de simplificação. Para um conjunto diferenciado de objectos em que o número de faces varia consideravelmente, o número de células visíveis diminui aproximadamente para metade de um nível multiresolução para o seguinte durante o processo de simplificação.

A principal razão para que as malhas não percam qualidade nas primeiras iterações está no critério restritivo em relação às células que são passíveis de serem contraídas em cada execução do algoritmo de simplificação.

6. CONCLUSÕES E TRABALHO FUTURO

Um novo esquema de multiresolução acaba de ser proposto e descrito neste artigo. Ao invés doutras estruturas de dados, a sua estrutura de dados de multiresolução é integrada e permite a restauração separada da geometria e da topologia. O seu elevado desempenho deve-se a não usar um critério geométrico, mas sim um critério topológico de simplificação, o que deixa em aberto a sua utilização, em vez do tradicional esquema de LODs, nas aplicações geométricas em tempo real.



(a) 375284 faces (100%) (b) 187648 faces (50%)

(c) 93836 faces (25%) (d) 46928 faces (12,5%)

(e) 23476 faces (6%) (f) 11744 faces (3%)

Figura 6. Preservação da forma para seis níveis de detalhe da estátua de Isis.

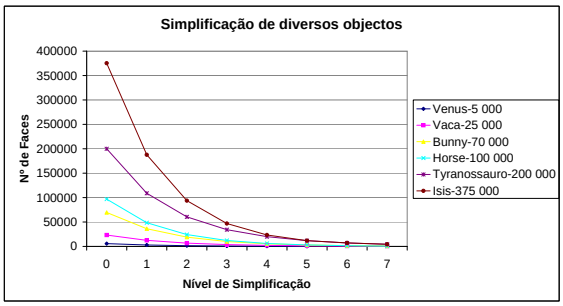


Figura 7. Evolução da simplificação de algumas malhas.

Referências

[Danovaro 01] E. Danovaro, L. De Floriani, P. Magillo, e E. Puppo. Representing vertex-based simplicial multi-complexes. Em G. Bertrand, A. Imiya, e R. Klette, editores, *Digital and Image Geometry*, volume 2243, de *Lecture Notes in Computer Science*, páginas 128–147. Springer-Verlag, 2001.

[ES99] J. El-Sana e A. Varshney. Generalized view-dependent simplification. *Computer Graphics Forum*, 18(3):83–94, 1999.

[Floriani 02] L. Floriani e P. Magillo. Multiresolution mesh representation: Models and data structures. Em A. Iske, E. Quak, e M. Floater, editores, *Tutorials on Multiresolution in Geometric Modelling*, páginas 363–418. Springer-Verlag, 2002.

[Garland 97] M. Garland e P. Heckbert. Surface simplification using quadric error metrics. Em *Proceedings of the SIGGRAPH'97*, volume 31, páginas 209–216. ACM Press, 1997.

[Garland 99] M. Garland. Multiresolution modeling: Survey and future opportunities. Em *Eurographics'99 State-of-Art Report*. Eurographics Association, 1999.

[Hoppe 93] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, e W. Stuetzle. Mesh optimization. Em *SIGGRAPH '93: Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, páginas 19–26, New York, USA, 1993. ACM Press.

[Hoppe 96] H. Hoppe. Progressive meshes. Em *Proceedings of the ACM SIGGRAPH'96*, volume 30, páginas 99–108. ACM Press, 1996.

[Hoppe 97] H. Hoppe. View-dependent refinement of progressive meshes. Em *Proceedings of the ACM SIGGRAPH'97*, volume 31, páginas 189–198. ACM Press, 1997.

[Mäntyla 88] M. Mäntyla. *An Introduction to Solid Modeling*. Computer Science Press, 1988.

[Ni 94] X. Ni e M. Bloor. Performance evaluation of boundary data structures. *IEEE Computer Graphics and Applications*, 14(6):66–77, 1994.

[Popovic 97] J. Popovic e H. Hope. Progressive simplicial complexes. Em *Proceedings of the ACM SIGGRAPH'97*, volume 31, páginas 217–224. ACM Press, 1997.

[Sabin 02] M. Sabin. Subdivision of box-splines. Em A. Iske, E. Quak, e M. Floater, editores, *Tutorials on Multiresolution in Geometric Modelling*, páginas 3–24. Springer-Verlag, 2002.

[Silva 04] F. Silva e A. Gomes. Normal-based simplification algorithmfor meshes. Em *Proceedings of Theory and Practice of Computer Graphics 2004 Conference (TPCG'04)*, páginas 211–218. IEEE Computer Society Press, 2004.

[Silva 05] F. Silva. *Estruturas Poligonais Deformáveis com Resolução Variável*. Tese de Doutoramento, Universidade da Beira Interior, Portugal, 2005.

[Weiler 88] K. Weiler. The radial edge structure: a topological representation for non-manifold geometric boundary modeling. Em M. Wozny, H. McLaughlin, e J. Encarnação, editores, *Geometric Modeling for CAD Applications*, páginas 3–36. Elsevier Science Publishers B.V. (North-Holland), May 1988.