6
Comparações

Neste capítulo analisaremos o consumo de memória em cada nível da CHE e da CHF. Ainda, complementaremos o estudo comparando o consumo de memória de cada nível com o custo de estruturas clássicas, apresentadas anteriormente no capítulo 2. Dividiremos o capítulo em duas seções. Na primeira, analisaremos o consumo de memória nas estruturas clássicas para superfícies e as compararemos com a CHE. Em seguida estudaremos o consumo das estruturas para 3–variedades compararemos com a CHF.

6.1
Estruturas de Dados para Superfícies em $\mathbb{R}^3$

Handle–Edge: Para representar a topologia de uma superfície triangulada, a estrutura de dados *Handle–Edge* requer 4 ponteiros por face, 13 ponteiros por aresta e 4 ponteiros por vértice. Para estimar o espaço de memória consumido pela com a *Handle–Edge* em função do número de vértices do modelo, podemos utilizar a fórmula de Euler para superfícies: $\mathbf{n}_0 - \mathbf{n}_1 + \mathbf{n}_2 = \chi(\mathsf{S})$, onde $\chi(\mathsf{S})$ é a característica de Euler da superfície S. Se assumirmos que o número de genus de uma superfície é pequeno se comparado à $\mathbf{n}_0$, então $\mathbf{n}_0 - \mathbf{n}_1 + \mathbf{n}_2 \approx 0$. Ainda, supondo que o número de arestas de bordo é pequeno e sabendo que cada aresta de interior é compartilhada por duas faces, então $\mathbf{n}_1 \approx \frac{3}{2}\mathbf{n}_2$. Como a valência de um vértice é em média igual a 6, assumindo um espaço de 4 bytes por referência, o consumo de memória da *Handle–Edge* é aproximadamente $4 \cdot 8\mathbf{n}_0 + 4 \cdot 39\mathbf{n}_0 + 4 \cdot 4\mathbf{n}_0 \approx 204\mathbf{n}_0$ bytes.

Corner–Table: A estrutura de dados *Corner–Table* é bastante concisa. Para armazenar as informações topológicas de uma malha de triângulos, supondo que cada número inteiro ocupa 4 bytes, a estrutura consome $4 \cdot 6\mathbf{n}_2 \approx 24\mathbf{n}_2$ bytes de memória já que, para cada triângulo armazenamos o índice do vértice inicial e o *corner* oposto de cada um de seus 3 *corners*.

Ainda, se adicionarmos a estrutura um *array* que para cada vértice do modelo armazena o índice de um de seus *corners* incidentes, como proposto

por (Vieira 2003), precisamos alocar um espaço extra de memória igual a $4 \cdot \mathbf{n}_0$ bytes. Novamente utilizando a fórmula de Euler podemos expressar o consumo de memória em função no número de vértices de um modelo. Estaremos supondo, novamente, que o número de genus e o número de arestas de bordo são pequenos, e assim teremos $\mathbf{n}_1 \approx \frac{3}{2}\mathbf{n}_2$, logo $\mathbf{n}_2 \approx 2\mathbf{n}_0$. Com isso a quantidade de memória consumida pela *Corner–Table* é aproximadamente $24\mathbf{n}_2 + 4\mathbf{n}_0 \approx 24 \cdot 2\mathbf{n}_0 + 4\mathbf{n}_0 \approx 52\mathbf{n}_0$.

Directed–Edges: Como vimos anteriormente, a *Directed–Edges* é uma estrutura de dados escalonável composta de 3 níveis: *small–sized*, *medium–sized*, e *full–sized*.

No primeiro nível, o *small–sized*, a estrutura armazena a menor quantidade de informações topológicas necessária para que possamos implementar as funções de resposta às informações topológicas de forma eficiente. Cada *directed–edge* armazena uma referência para o seu vértice inicial e para sua *directed–edge* vizinha e assim, supondo que cada referência ocupa 4 bytes, precisamos $4 \cdot 2 \cdot 3\mathbf{n}_2 \approx 4 \cdot 2 \cdot 6\mathbf{n}_0 \approx 48\mathbf{n}_0$ bytes para carregar o nível *small–sized* da estrutura.

No nível *medium–sized*, adicionamos uma referência para a directed–edge *previous* de cada *directed–edge* da malha, sendo assim precisamos de $4 \cdot 3 \cdot 3\mathbf{n}_2 \approx 4 \cdot 3 \cdot 6\mathbf{n}_0 \approx 72\mathbf{n}_0$ bytes de memória para utilizar o nível *medium–sized*.

Por fim, no nível *full–sized* adicionamos uma referência para o vértice final e uma referência para a *directed–edge next* de cada *directed–edge* da malha. Logo armazenamos no total 5 referências explicitas de outras entidades do modelo para cada *directed–edge*, e como cada triângulo é composto de 3 *directed–edges*, precisamos de $4 \cdot 5 \cdot 3\mathbf{n}_2 \approx 4 \cdot 5 \cdot 6\mathbf{n}_0 \approx 120\mathbf{n}_0$ bytes para carregar a topologia de uma malha no nível *full–sized*.

CHE: Como apresentamos no capítulo 4, CHE é também uma estrutura de dados escalonável para superfícies, composta pelos níveis 0, 1, 2 e 3.

No nível 0, a CHE não armazena nenhuma informação sobre a adjacência dos triângulos da malha, portanto o espaço ocupado pela estrutura em memória é extremamente pequeno. Neste nível guardamos apenas o índice do vértice inicial de cada *half–edge* no contêiner V[]. Logo, para alocar o nível 0, supondo que cada valor inteiro ocupa 4 bytes, utilizamos apenas $4 \cdot 3\mathbf{n}_2$ bytes de memória e, usando a aproximação deduzida a partir da fórmula de Euler, o nível 0 da CHE consome aproximadamente $4 \cdot 6\mathbf{n}_0 \approx 24\mathbf{n}_0$ bytes de memória.

No nível 1, adicionamos a CHE informações sobre a vizinhança de cada triângulo através do contêiner O[], ou seja, precisamos armazenar a *half–edge* oposta de cada *half–edge* da malha. Com isso, a quantidade de memória que precisamos para o nível 1 da CHE é igual a $4 \cdot 3\mathbf{n}_2 + 4 \cdot 3\mathbf{n}_2 = 24\mathbf{n}_2$, e colocando o custo em função do número de vértices do modelo, precisamos de aproximadamente $48\mathbf{n}_0$ bytes de memória. Com isso dobramos a quantidade de memória utilizada pela estrutura do nível 0 para o nível 1, mas, em contrapartida, aumentamos bastante a eficiência do cálculo das funções de resposta, como vimos na seção 4.6.

O nível 2 da CHE adiciona uma representação explícita para cada célula do modelo. Desta maneira podemos associar a cada uma delas atributos significativos para uma dada aplicação onde deseja-se utilizar a estrutura. Propomos neste nível a associação de cada vértice a uma de suas *half–edges* incidentes através do contêiner Vertex Half–Edge, já que assim otimizamos bastante o cálculo da estrela do vértice, e ainda a criação da Edge Map para a representação explícita de arestas. Com isso, para construir o nível 2 da CHE precisamos alocar ao menos $4\mathbf{n}_0$ bytes de memória para o contêiner Vertex Half–Edge e $20\mathbf{n}_1 \approx 60\mathbf{n}_0$ bytes para a Edge Map. Assim seu custo total do nível 2 da CHE é, aproximadamente, $64\mathbf{n}_0 + 48\mathbf{n}_0 \approx 112\mathbf{n}_0$ bytes.

No último nível da CHE, adicionamos uma representação explícita para cada componente de bordo da superfície. Tal representação é feita armazenando uma *half–edge* incidente a cada curva de bordo da superfície. Assim precisamos de $\mathbf{n}_\partial$ de memória para armazenar a *half–edge* representante de cada curva de bordo. Logo, precisamos para o nível 3, um total de memória de aproximadamente $\mathbf{n}_\partial + 112\mathbf{n}_0$. Como na maioria das vezes a quantidade de componentes de bordo é pequena, o espaço de memória necessário é de aproximadamente $112\mathbf{n}_0$ bytes.

Comparações: Cada estrutura de dados apresentada trata de forma diferente o balanceamento entre o uso de memória e o custo computacional. Dentre as estruturas estudadas, a CHE é a mais flexível já que propõe uma implementação escalonável que se adapta a uma larga escala de aplicações.

Mesmo quando comparada às estruturas mais compactas, como a *Corner–Table*, a CHE se mostra bastante eficiente já que no nível 2, por exemplo o custo da CHE e da *Corner–Table* são os mesmos.

Em relação a estruturas mais sofisticadas como a *Handle–Edge* e a *Directed–Edges* a CHE também apresenta vantagens. A CHE consegue tratar o conceito de escalabilidade de maneira mais ampla e flexível que a *Directed–Edges*, uma vez que a mudança de nível proporciona muito mais que um

equilíbrio no uso de memória e processador, ela também nos dá acesso a novas informações sobre a malha, não conseguidas diretamente a partir dos níveis anteriores. Ainda, comparada a *Handle–Edge* a CHE representa um grande ganho no uso de memória já que o nível 3 da CHE e a *Handle–Edge* apresentam a mesma característica, que é a representação explícita das curvas de bordo, mas o nível 2 da CHE é bem mais compacto que a *Handle–Edge*.

Uma desvantagem da CHE em relação à *Handle–Edge* é que, a inserção e remoção de células é menos eficiente, já que, tendo em vista que a CHE trabalha com contêineres de índices inteiros, a mudança no número de células requer a reorganização dos índices. Em contrapartida, a *Handle–Edge*, por trabalhar com ponteiros, necessita apenas do redirecionamento de referências, o que sem dúvida requer menos esforço do que a reorganização de índices da CHE.

Na tabela 6.2 fazemos um quadro comparativo dos gastos de memória de cada estrutura.

	Custo em Bytes
Handle–Edge	$204\mathbf{n}_0$
Corner–Table	$52\mathbf{n}_0$
Directed–Edges *Small*	$48\mathbf{n}_0$
Directed–Edges *Medium*	$72\mathbf{n}_0$
Directed–Edges *Full*	$120\mathbf{n}_0$
CHE *Nível* 0	$24\mathbf{n}_0$
CHE *Nível* 1	$48\mathbf{n}_0$
CHE *Nível* 2	$112\mathbf{n}_0$
CHE *Nível* 3	$112\mathbf{n}_0$

Tabela 6.1: Custo de memória para representação da topologia.

6.2 Estruturas de Dados para 3–Variedades

Nesta seção, analisaremos o consumo de memória das estrutura de dados para variedades de dimensão 3 apresentadas neste trabalho e os compararemos com o custo da CHF. Assim como fizemos na seção anterior, expressaremos tais custos em função do número de vértices da malha, e para isso mais utilizaremos a equação de Euler para variedades de dimensão 3: $\mathbf{n}_0 - \mathbf{n}_1 + \mathbf{n}_2 - \mathbf{n}_3 = \chi(\mathsf{M})$, onde $\chi(\mathsf{M})$ é a característica de Euler da 3–variedade M.

Supondo que a característica de Euler de uma 3–variedade é pequena se comparada ao número de vértices, e ainda imaginando que a quantidade de faces de bordo é pequena se comparada ao número de faces de interior (Gumhold *et al.* 1999) mostra que podemos trabalhar com as seguintes aproximações: $\mathbf{n}_0 : \mathbf{n}_1 : \mathbf{n}_2 : \mathbf{n}_3 \approx 1 : 6.5 : 11 : 5.5$.

Por padrão, adotaremos que o espaço necessário para armazenar um valor inteiro em memória é de 4 bytes.

Handle–Face: Para representar a topologia de uma 3–variedade, a estrutura de dados *Handle–Face* necessita de 136 ponteiros por tetraedro, 5 ponteiros por face, 4 ponteiros por aresta e 3 ponteiros por vértice. Para estimar o espaço de memória utilizado por uma modelo carregado com a *Handle–Face*, utilizaremos as aproximações deduzidas a partir da formula de Euler e apresentadas em (Gumhold *et al.* 1999) e descritas anteriormente. Desta forma, como cada ponteiro requer 4 bytes, a *Handle–Face* precisará de $4 \cdot (136\mathbf{n}_3 + 5\mathbf{n}_2 + 4\mathbf{n}_1 + 3\mathbf{n}_0) \approx 4 \cdot (748\mathbf{n}_0 + 55\mathbf{n}_0 + 26\mathbf{n}_0 + 3\mathbf{n}_0) \approx 3328\mathbf{n}_0$ bytes de memória.

Indexed Data–Structure with Adjacency: A estrutura de dados *Indexed data–structure with adjacency*, a *Ia*, é bastante concisa. Como a estrutura armazena apenas o k–esqueleto e a relação de adjacência entre os k–simplexos da malha, precisamos apenas de $4 \cdot 8\mathbf{n}_3 \approx 32\mathbf{n}_3$ bytes para armazenar as informações todas as relações topológicas da estrutura.

Novamente utilizando a fórmula de Euler podemos expressar o consumo de memória em função no número de vértices do modelo. Estaremos supondo, novamente, que o número de genus e o número de faces de bordo são pequenos, e, assim, teremos $\mathbf{n}_3 \approx 5.5\mathbf{n}_0$, e a quantidade de memória consumida pela *Ia*, será, aproximadamente, $176\mathbf{n}_0$ bytes.

NMIA: A *Non–manifold indexed data structure with adjacencies*, ou *NMIA* por simplicidade, foi desenvolvida para representar não–variedades de dimensão 3, mas proporcionando uma boa adaptatividade na representação de 3–variedades. A *NMIA* estende a *Ia* codificando as múltiplas componentes conexas existentes na estrela de vértices ou arestas não–variedade.

O custo em memória para construirmos a *NMIA* é de $8\mathbf{n}_3 + 3\mathbf{n}_2^t + 2\mathbf{n}_1^t + C_e + C_v$, onde C_e representa o número total de clusters em arestas não–variedades, C_v representa o número total de componentes conexas nas estrelas dos vértices, $\mathbf{n}_2^t$ representa o número de *dangling–faces*, e $\mathbf{n}_1^t$ o número de *wire–edges* do modelo.

Para compará-la com a CHF, analisaremos o consumo de memória da *NMIA* apenas para malhas de tetraedros que são 3–variedades. Como complexos que são variedades não possuem *dangling–faces*, *wire edges*, e as estrelas de vértices e de arestas tem apenas uma componente conexa, $\mathbf{n}_2^t = \mathbf{n}_1^t = C_e = 0$ e $C_v = \mathbf{n}_0$. Lodo o custo em bytes de armazenamento da estrutura será igual ao da Ia, ou seja aproximadamente $4 \cdot 44\mathbf{n}_0 \approx 176\mathbf{n}_0$.

CHF: Como apresentamos no capítulo 5, a CHF é uma estrutura de dados escalonável para 3–variedades, composta por 4 níveis.

No nível 0, a CHF não armazena informações sobre a adjacência dos tetraedros da malha, logo o espaço ocupado pela estrutura em memória

é extremamente pequeno. Como armazenamos apenas o vértice oposto a cada *half–face*, no tetraedro incidente, utilizamos apenas de $4 \cdot 4\mathbf{n}_3$ bytes de memória neste nível. Ainda, como $\mathbf{n}_3 \approx 5.5\mathbf{n}_0$, o nível 0 da CHF consome aproximadamente $88\mathbf{n}_0$ bytes de memória.

No nível 1, adicionamos a CHF informações sobre a vizinhança de cada tetraedro, ou seja, armazenamos a *half–face* oposta de cada *half–face* da malha. Assim, a quantidade de memória que precisamos para o nível 1 da CHF é igual a $4\cdot4\mathbf{n}_3+4\cdot4\mathbf{n}_3 = 32\mathbf{n}_3$ bytes, e colocando em função do número de vértices do modelo, precisamos de aproximadamente $176\mathbf{n}_0$ bytes de memória. Com isso dobramos a quantidade de memória utilizada pela estrutura do nível 0 para o nível 1, mas em contrapartida aumentamos bastante a eficiência do cálculo das funções de resposta, como vimos na seção 5.6.

O nível 2 da CHF adiciona uma representação explícita para cada célula do modelo. Desta maneira podemos associar a cada uma delas atributos significativos para uma dada aplicação onde deseja-se utilizar a estrutura. Propomos neste nível a associação de cada vértice e aresta a uma de suas *half–faces* incidentes através dos contêineres Vertex Half–Face e Edge Map, já que assim otimizamos bastante o cálculo das estrelas de vértices e arestas. Propomos também, neste nível, a criação do contêiner Face Map, para a representação explícita de faces. Como vimos na seção 2.8, precisamos de aproximadamente 20 bytes por aresta e face para a construção de contêineres do tipo map, logo o nível 2 da CHF precisa alocar $4\mathbf{n}_0 + 20\mathbf{n}_1 + 20\mathbf{n}_2$ bytes a mais que o nível 1, e seu custo total em memória é de aproximadamente $4\mathbf{n}_0 + 20\mathbf{n}_1 + 20\mathbf{n}_2 + 176\mathbf{n}_0 \approx 4\mathbf{n}_0 + 130\mathbf{n}_0 + 220\mathbf{n}_0 + 176\mathbf{n}_0 \approx 530\mathbf{n}_0$ bytes.

No último nível da CHF, adicionamos uma representação explícita para as superfícies de bordo da 3–variedade. Tal representação é feita implementando o nível 1 da CHE. Assim precisamos de $48\partial\mathbf{n}_0$ bytes a mais que no nível 2 da CHF, onde $\partial\mathbf{n}_0$ é o número de vértices da superfície de bordo. Logo, precisamos para o nível 3, um total de memória de aproximadamente $530\mathbf{n}_0 + 48\partial\mathbf{n}_0$.

Comparações: Cada estrutura de dados gerencia de forma diferente o uso de memória e o custo computacional. A CHF mostrou-se mais flexível que as demais, já que é a única que propõe uma implementação escalonável que se adapta a uma larga escala de aplicações.

Mesmo quando comparada às estruturas mais compactas, como a Ia ou a NMIA restrita a variedades, a CHF é bastante competitiva, já que no nível 1, por exemplo, o custo da CHF é o mesmo destas estruturas.

A CHF consegue tratar o conceito de escalabilidade de maneira bastante ampla e flexível pois, assim como na CHE, a mudança de nível proporciona

	Custo em Bytes
Handle–Face	$3328\mathbf{n}_0$
IA/ NMIA *Manifold*	$176\mathbf{n}_0$
CHF *Nível* 0	$88\mathbf{n}_0$
CHF *Nível* 1	$176\mathbf{n}_0$
CHF *Nível* 2	$530\mathbf{n}_0$
CHF *Nível* 3	$530\mathbf{n}_0 + 48\partial\mathbf{n}_0$

Tabela 6.2: Custo de memória para representação da topologia.

muito mais que o equilíbrio entre o uso de memória e processador, ela também nos dá acesso a novas informações sobre a malha, não conseguidas diretamente nos níveis anteriores.

Em relação a estruturas mais sofisticadas como a *Handle–Face* a CHF representa um grande ganho no uso de memória já que o nível 3 da CHF e a *Handle–Face* apresentam a mesma característica principal, que é a representação explícita das superfícies de bordo, mas o nível 3 da CHF é bem mais compacto que a *Handle–Face*.

Uma desvantagem da CHF em relação à *Handle–Face* é que, a inserção e remoção de células é menos eficiente, já que, tendo em vista que a CHF trabalha com contêineres de índices inteiros, a mudança no número de células requer a reorganização dos índices de tempos em tempos via um procedimento de "recolhimento de lixo" (*Garbage Collection*). Em contrapartida, a *Handle–Face*, por trabalhar com ponteiros, necessita apenas do redirecionamento de referências, o que sem dúvida requer menos esforço.

Na tabela 6.2 fazemos um quadro comparativo dos gastos de memória de cada estrutura.