7
GBPM: Integração e busca por vizinhos

Para integrar as equações de movimento de Newton (2-1), é necessário levar em consideração o seguinte fato: as forças e os torques atuando sobre as partículas, variam rapidamente. Por esse motivo, o integrador a ser adotado deve ser robusto, no sentido de lidar com oscilações de alta frequência.

Além disso, como o gargalo do algoritmo está na etapa de calcular as forças, é primordial adotar um esquema de integração, que a cada passo de tempo, consiga avaliar as forças apenas uma vez.

7.1
Esquema de integração de Gear

O algoritmo de Gear (1, 29, 30) é um método de integração do tipo preditor-corretor. Ele é uma boa opção para integrar as equações de movimento devido as características mostradas nos próximos parágrafos.

Primeiramente, o algoritmo de Gear consegue lidar com o fato da alta freqüência que aparece nas acelerações, desde que seja escolhida uma ordem relativamente grande do método.

A maior vantagem desse método está no fato de ser necessária apenas uma avaliação das forças por passo de tempo. Como já foi comentado, o gargalo do DEM, e conseqüentemente do GBPM, está na etapa de determinação das forças.

Outro esquema de integração que se adapta bem aos requisitos do método, segundo o trabalho de Allen (1), é o método de integração de Verlet. Seria interessante comparar e verificar as diferenças nos resultados das simulações, empregando diferentes tipos de integrador, e também verificar o *"trade-off"* entre o grau do integrador e a qualidade e velocidade do resultado obtido. Essas direções não foram investigadas no curso dessa tese e esses pontos são apontados como direção de futura pesquisa.

7.1.1
Descrição formal do esquema de integração de Gear

O método de integração de Gear consiste de dois passos principais (1, 29, 30): o preditor e o corretor.

No passo preditor, são aproximadas as posições $\overrightarrow{r_i}^{\mathrm{pr}}$, velocidades $\frac{d}{dt}\overrightarrow{r_i} = \overrightarrow{v_i}$ e derivadas de ordem mais alta no tempo $t + \Delta t$, extrapolando os valores atuais usando uma expansão em série de Taylor.

$$
\begin{aligned}
\overrightarrow{r_i}^{\mathrm{pr}}(t+\Delta t) &= \overrightarrow{r_i}(t) + \Delta t \overbrace{\frac{d}{dt}\overrightarrow{r_i}(t)}^{\overrightarrow{v_i}(t)} + \frac{\Delta t^2}{2}\frac{d^2}{dt^2}\overrightarrow{r_i}(t) \\
&\quad + \frac{\Delta t^3}{6}\frac{d^3}{dt^3}\overrightarrow{r_i}(t) + \frac{\Delta t^4}{24}\frac{d^4}{dt^4}\overrightarrow{r_i}(t) + \cdots \\
\overbrace{\frac{d}{dt}\overrightarrow{r_i}^{\mathrm{pr}}(t+\Delta t)}^{\overrightarrow{v_i}^{\mathrm{pr}}(t+\Delta t\ )} &= \frac{d}{dt}\overrightarrow{r_i}(t) + \Delta t\frac{d^2}{dt^2}\overrightarrow{r_i}(t) + \frac{\Delta t^2}{2}\frac{d^3}{dt^3}\overrightarrow{r_i}(t) \\
&\quad + \frac{\Delta t^3}{6}\frac{d^4}{dt^4}\overrightarrow{r_i}(t) + \cdots \\
\frac{d^2}{dt^2}\overrightarrow{r_i}^{\mathrm{pr}}(t+\Delta t) &= \frac{d^2}{dt^2}\overrightarrow{r_i}(t) + \Delta t\frac{d^3}{dt^3}\overrightarrow{r_i}(t) + \frac{\Delta t^2}{2}\frac{d^4}{dt^4}\overrightarrow{r_i}(t) + \cdots \\
\frac{d^3}{dt^3}\overrightarrow{r_i}^{\mathrm{pr}}(t+\Delta t) &= \frac{d^3}{dt^3}\overrightarrow{r_i}(t) + \Delta t\frac{d^4}{dt^4}\overrightarrow{r_i}(t) + \cdots \\
\frac{d^4}{dt^4}\overrightarrow{r_i}^{\mathrm{pr}}(t+\Delta t) &= \frac{d^4}{dt^4}\overrightarrow{r_i}(t) + \cdots \\
&\vdots
\end{aligned}
$$

No passo corretor as coordenadas preditas $\overrightarrow{r_i}^{\mathrm{pr}}$ e suas derivadas $\frac{d}{dt}\overrightarrow{r_i}^{\mathrm{pr}}$, $\frac{d^2}{dt^2}\overrightarrow{r_i}^{\mathrm{pr}}$,etc., são utilizadas para determinar as forças $\overrightarrow{F_i}(\overrightarrow{r_i}^{\mathrm{pr}}, \overrightarrow{v_i}^{\mathrm{pr}}, \overrightarrow{\varphi_i}^{\mathrm{pr}}, \overrightarrow{\omega_j}^{\mathrm{pr}})$ e torques $\overrightarrow{M_i}(\overrightarrow{r_i}^{\mathrm{pr}}, \overrightarrow{v_i}^{\mathrm{pr}}, \overrightarrow{\varphi_i}^{\mathrm{pr}}, \overrightarrow{\omega_j}^{\mathrm{pr}})$, que atuam no sistema.

Fazendo uso dos valores atualizados das forças e torques, são obtidos: a aceleração linear $\frac{d^2}{dt^2}\overrightarrow{r_i}^{\mathrm{cor}}(t+\Delta t)$, e angular $\frac{d^2}{dt^2}\overrightarrow{\varphi_i}^{\mathrm{cor}}(t+\Delta t)$. Isso é conseguido empregando as equações de Newton.

Definimos o erro de predição para a aceleração como a diferença dos valores preditos e corrigidos.

$$
\Delta\frac{d^2}{dt^2}\overrightarrow{r} \equiv \frac{d^2}{dt^2}\overrightarrow{r_i}^{\mathrm{cor}}(t+\Delta t) \ - \frac{d^2}{dt^2}\overrightarrow{r_i}^{\mathrm{pr}}(t+\Delta t) \tag{7-1}
$$

Na ultima etapa do passo corretor, a predição sofre uma correção, que é proporcional ao erro $\Delta\frac{d^2}{dt^2}\overrightarrow{r}$. Essa correção é balanceada com pesos, $c_0, c_1, ...$ para cada derivada, que são empregados conforme a equação:

$$\begin{pmatrix} \overrightarrow{r_i}^{\mathrm{cor}}(t+\Delta t) \\ \frac{d}{dt}\overrightarrow{r_i}^{\mathrm{cor}}(t+\Delta t) \\ \frac{d^2}{dt^2}\overrightarrow{r_i}^{\mathrm{cor}}(t+\Delta t) \\ \frac{d^3}{dt^3}\overrightarrow{r_i}^{\mathrm{cor}}(t+\Delta t) \\ \frac{d^4}{dt^4}\overrightarrow{r_i}^{\mathrm{cor}}(t+\Delta t) \\ \vdots \end{pmatrix} = \begin{pmatrix} \overrightarrow{r_i}^{\mathrm{pr}}(t+\Delta t) \\ \frac{d}{dt}\overrightarrow{r_i}^{\mathrm{pr}}(t+\Delta t) \\ \frac{d^2}{dt^2}\overrightarrow{r_i}^{\mathrm{pr}}(t+\Delta t) \\ \frac{d^3}{dt^3}\overrightarrow{r_i}^{\mathrm{pr}}(t+\Delta t) \\ \frac{d^4}{dt^4}\overrightarrow{r_i}^{\mathrm{pr}}(t+\Delta t) \\ \vdots \end{pmatrix} + \begin{pmatrix} c_0 \\ c_1\frac{1!}{(\Delta t)^1} \\ c_2\frac{2!}{(\Delta t)^2} \\ c_3\frac{3!}{(\Delta t)^3} \\ c_4\frac{4!}{(\Delta t)^4} \\ \vdots \end{pmatrix} \frac{\Delta t^2}{2} \quad \underbrace{\Delta \frac{d^2}{dt^2}\overrightarrow{r}}_{\text{Erro de predição.}} . \tag{7-2}$$

Os coeficientes dependem da ordem do algoritmo, e também, do tipo da equação diferencial. Para um algoritmo de sexta ordem e as equações de movimento de Newton os coeficientes são:

$$c_0 = \frac{3}{16}, c_1 = \frac{251}{360}, c_2 = 1, c_3 = \frac{11}{18}, c_4 = \frac{1}{6}, c_5 = \frac{1}{60}$$

para quinta ordem:

$$c_0 = \frac{19}{90}, c_1 = \frac{3}{4}, c_2 = 1, c_3 = \frac{1}{2}, c_4 = \frac{1}{12}$$

e para quarta ordem:

$$c_0 = \frac{1}{6}, c_1 = \frac{5}{6}, c_2 = 1, c_3 = \frac{1}{3}.$$

Para mais detalhes sobre esses coeficientes veja o livro de Allen (1).

Finalmente avançamos o contador de tempo $t := t + \Delta t$, e começamos o próximo passo de integração.

7.2 Busca por vizinhança

Numa simulação com o GBPM, o gargalo está na avaliação das forças e dos torques devido às iterações partícula-partícula. Isso ocorre, pois a cada passo de tempo é necessário determinar quais pares de partículas estão em contato mecânico. E nesse trabalho é sugerida uma nova estrura de dados para resolver esse problema.

A maneira mais ingênua para resolver esse problema é usar uma tática de força bruta. Mas em geral, existem muitas partículas, e essa estratégia torna-se dispendiosa. Para reduzir a complexidade do problema é importante empregar uma estrutura de dados de busca espacial.

Para melhorar o desempenho da simulação, buscamos uma estrutura de dados que tire vantagem da coerência, isso é, o fato que a cada passo de tempo todas as partículas se movem muito pouco, com relação ao passo de

tempo anterior. Também é desejável que essa estrutura possa ser utilizada num contexto de paralelização.

Formalmente, dado um conjunto de pontos $S_0 = \{r_i \in \mathbb{R}^3; i = 1..N\}$, o problema é determinar eficientemente a função

$$b_\epsilon : \mathbb{R}^3 \mapsto \overbrace{P(S_0)}^{\text{Conjunto das partes}}$$
$$b_\epsilon(X) = \{(r_i \in S_0; \|r_i - X\| < \epsilon)\}$$

onde $\epsilon \in \mathbb{R}$ é o raio de busca.

Nesse trabalho, é proposta uma estrutura. Ela tira vantagem da forte coerência, entre dois passos consecutivos da simulação. Ela determina $b_\epsilon(X)$ em tempo constante. É paralelizável e tira proveito da hierarquia de memória dos processadores modernos.

7.3 Estrutura de dados para a busca espacial

Iniciamos a estrutura com uma malha regular sobre o conjunto de grãos. O raio dos grãos deve ser menor que 2ϵ. A célula, ocupando a linha l e a coluna c, é denotada por $C_{l,c}$.

Cada $C_{l,c}$ é constituída por uma lista de ponteiros $L_{l,c}$, que é iniciada vazia.

Para cada grão $i \in C_{l,c}$,é adicionado um ponteiro de i em $L_{l,c}$, e nas listas das células vizinhas a $C_{l,c}$ denotadas por viz$(C_{l,c})$

$$\text{viz}(C_{l,c}) = \{C_{l+u,c+v}; u \in \{-1, 0, 1\}; v =\in \{-1, 0, 1\}\}.$$

Com isso, cada partícula gera nove ponteiros no caso 2D. Esse procedimento é ilustrado na figura 7.1.

Para buscar os vizinhos de um ponto $X \in C_{l,c}$, basta ler a lista $L_{l,c}$,veja as imagens 7.2 e 7.3.

Esse método é direto, evitando percorrer a malha em busca dos vizinhos. Com isso, agrupando a busca na memória, e otimizando a utilização da memória cache do processador.

A cada passo de tempo, é executada uma verificação para determinar as partículas que mudaram de célula. E em seguida, as listas afetadas são atualizadas.

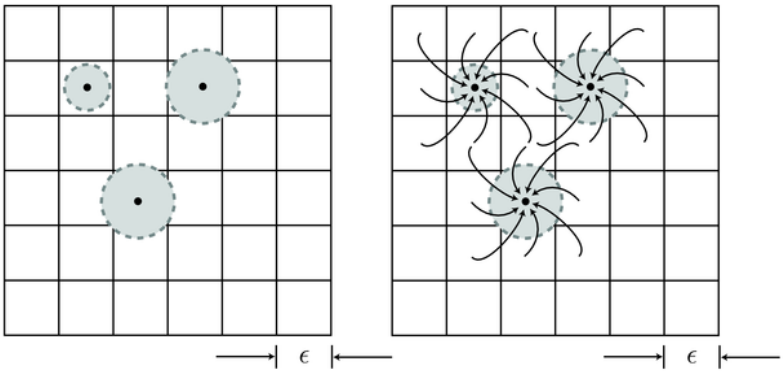


Figura 7.1: Cada célula da malha regular é composta por uma lista com ponteiros para as partículas em sua vizinhança.

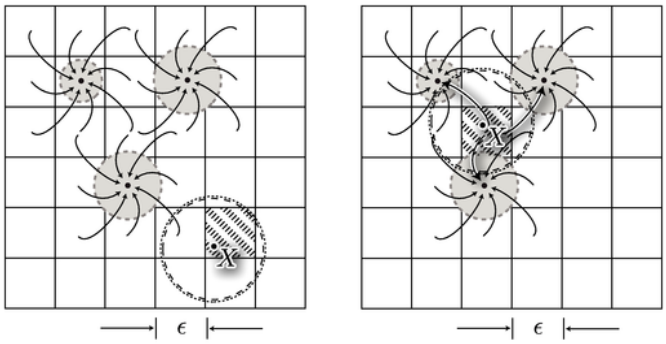


Figura 7.2: A resposta da busca é o valor da lista na célula. Esquerda: nenhum vizinho, direita: três vizinhos.

Perceba que no momento que uma partícula i passa de uma célula $C_{l,c}$ para outra célula vizinha $C_{l,c+1}$, alguns vizinhos são mantidos. Por isso, o número de listas a serem alteradas é reduzido. De fato, só se faz necessário inserir ponteiros nas listas do conjunto $\text{viz}(C_{l,c+1}) - (\text{viz}(C_{l,c+1}) \cap \text{viz}(C_{l,c}))$, e remover nas listas das células de $\text{viz}(C_{l,c}) - (\text{viz}(C_{l,c+1}) \cap \text{viz}(C_{l,c}))$.

O processo de atualização das listas está ilustrado nas figuras 7.4 e 7.5 e . No caso 3D, o processo é análogo, e o número de ponteiros por partícula aumenta para 27.

Concluindo, as vantagem de utilizar essa estrutura são: velocidade na busca pelos vizinhos, simplicidade de implementação, permite o desenvolvimento para algoritmos em paralelo. A grande desvantagem está no consumo de memória.

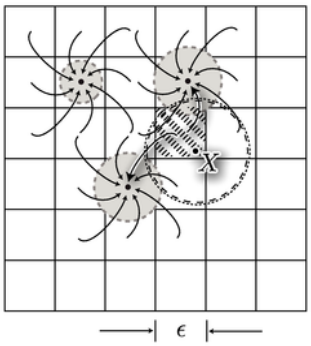


Figura 7.3: A resposta da busca é o valor da lista na célula em destaque. Nesse caso apenas um vizinho.

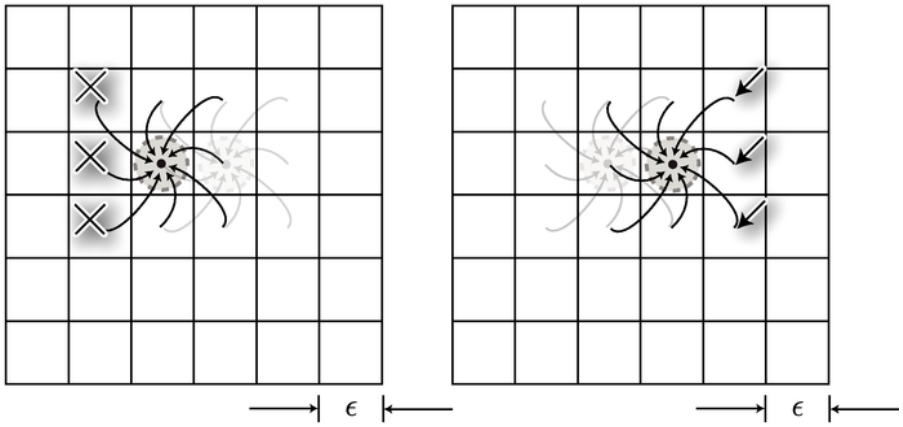


Figura 7.4: Alteração da posição da partícula, caso simples, temos que efetuar três operações de remoção de ponteiro e mais três de adição.

7.4 Comparação da estrutura de dados

Para efetuar a comparação da estrutura de dados proposta escolhemos três diferentes estruturas de dados, que são indicadas para esse tipo de simulação (55, 1), as estruturas escolhidas para a comparação são: Força bruta, Lista Verlet e Método de Malha Regular, vamos dar uma breve explicação sobre cada uma das estruturas.

Força bruta

Nesse método, para encontrar os vizinhos em um raio determinado são testadas todas as partículas.

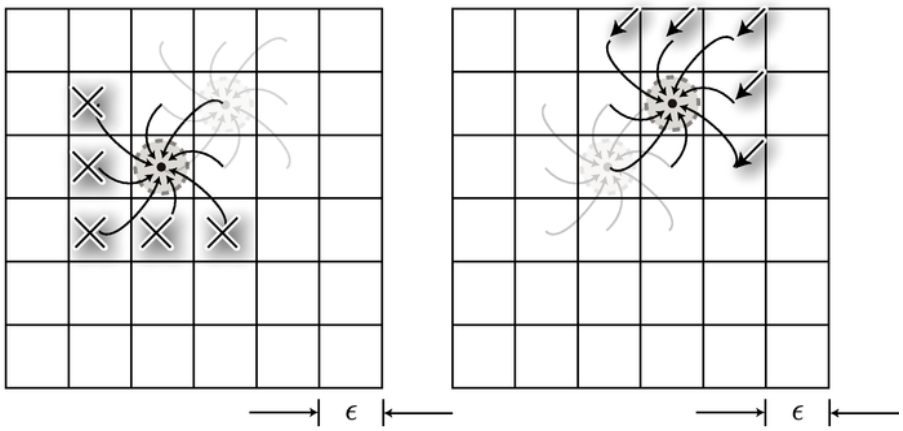


Figura 7.5: Alteração da posição da partícula, caso elaborado, remoção em cinco lista e adição em outras cinco.

Lista Verlet

Essa estrutura de dados mantem uma lista para cada partícula. Nessa lista estão todos os possíveis vizinhos para um determinado raio, a cada passo de tempo é verificada a validade da lista. Assim que uma das listas perde a validade é necessário recalcular todas as listas utilizando o método de força bruta. Em simulações com partículas bastante esparsas esse método pode apresentar alguma vantagem (55).

Malha regular

O método da malha regular consiste em dividir o domínio de simulação em cubos, e guardar em cada cubo é guardada uma lista com os ponteiros para as partículas que ocupam o mesmo. Para verificar quais as partículas estão em um raio de um determinado ponto, basta checar as listas dos cubos vizinhos, em 2D são nove e em 3D são vinte e sete.

Toda vez que uma partícula passa de um cubo para o outro, temos que atualizar duas listas.

7.5 Comparando a velocidade

Para efetuar a comparação vamos realizar a seguinte simulação: Inicializamos a simulação com um determinado número de partículas distribuídas uniformemente em um paralelepípedo, essas partículas inicialmente estão paradas e caem, em queda livre, até uma caixa onde elas ficam confinadas. Como se estivéssemos enchendo uma caixa de areia.

Essa simulação é interessante, pois temos dois regimes extremos de densidade de partículas partículas: no começo as partículas se encontram dispersas e no final elas estão amontoadas.

Efetuamos a simulação três vezes. Em cada uma dessas vezes modificamos o número de partículas, efetuamos a simulação e medimos a velocidade durante a simulação. O resultado dessas medidas é apresentado na figura 7.6. Na sequência vamos analisar cada um dos casos.

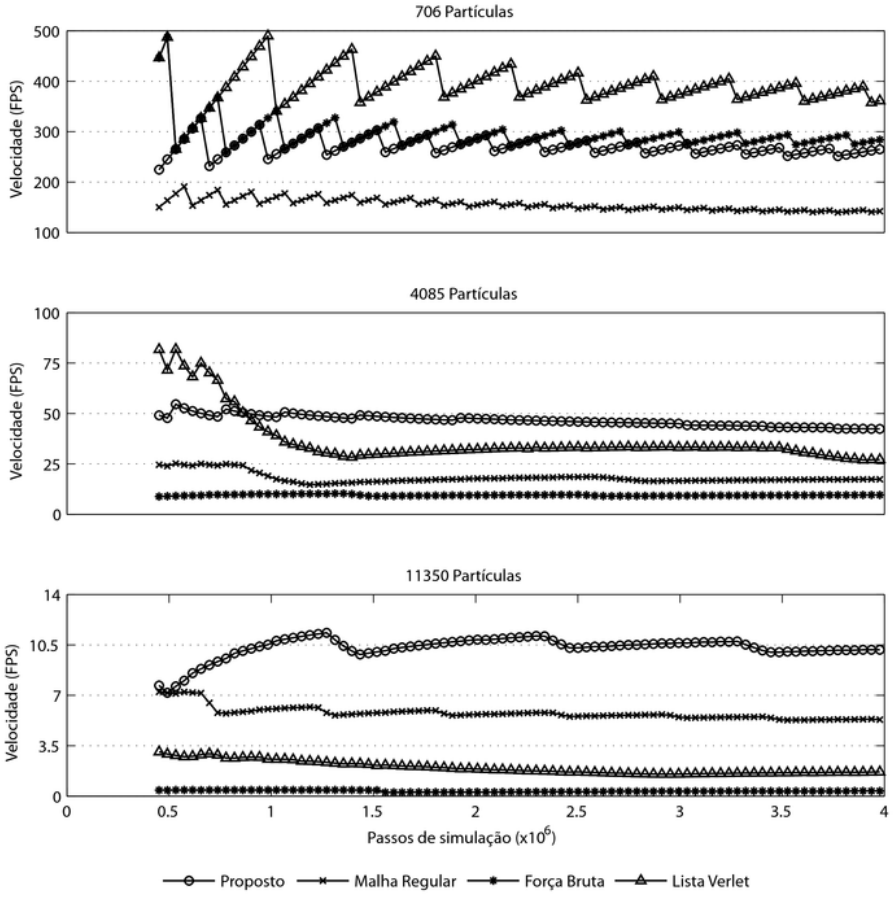


Figura 7.6: Comparação das velocidades de simulação (quanto maior melhor). Quanto maior o número de partículas e maior a densidade maior o ganho de velocidade que o método proposto representa.

Simulação com número pequeno de partículas (706)

Para a simulação com um número pequeno de partículas (706) o método da lista Verlet é o vencedor. O serrilhado que aparece no gráfico se deve ao overhead de recalcular as listas quando uma delas perde a validade, isso acontece de maneira uniforme pois as partículas estão se movendo de acordo com a lei da gravidade. A estrutura proposta obteve desempenho equivalente

ao método de força bruta e o método da malha regular foi o que apresentou o pior desempenho.

O serrilhado aparece naturalmente no método proposto e no método da malha regular. Esse artefato é efeito do overhead de recalcular as listas quando as partículas estão trocando de célula da malha.

Uma pergunta interessante a ser feita é a seguinte: Por quê apareceu o efeito de serrilhado no método força bruta? A resposta dessa pergunta é: Mesmo no método força bruta existe um teste para excluir rapidamente uma partícula que esteja longe do ponto, esse teste é baseado na distância em cada uma das coordenadas. Quando as partículas ocupam determinada posição, esse teste falha. Com isso a partícula deve passar por um segundo teste mais rigoroso, reduzindo a velocidade de simulação e consequentemente gerando o pico.

Com o passar do tempo, esse serrilhado é suavizado em todos os métodos. Isso porquê as partículas se movem cada vez mais devagar (lembre-se que elas estavam em queda livre e em seguida estão se acomodando no interior de uma caixa).

Simulação com número moderado de partículas (4060)

Com um número maior de partículas, o método proposto perde somente do para as listas Verlet durante o incio da simulação. Isso acontece pois nessa etapa as partículas estão bem espalhadas.

O efeito do serrilhado é suavizado em todos os métodos, devido ao fato de que o overhead de atualizar as listas é quase insignificante. Isso acontece pois estamos efetuando a busca por vizinhos um número muito grande de vezes. E portanto, vale a pena gastar para determinar o mais rápido possível os vizinhos.

Simulação com número relativamente grande de partículas (11350)

A partir desse número de partículas o comportamento do último gráfico da figura 7.6 se repete.

O método proposto é o mais rápido e o método de malha regular fica em segundo. Para encontrar os vizinhos o Método de Malha Regular deve varrer os vinte e sete cubos vizinhos, e fazer isso milhões de vezes se torna computacionalmente caro se comparado ao método proposto, que contém toda a informação necessária dentro da célula.

Quanto maior o número de partículas maior o ganho de velocidade que o método proposto oferece. Claro isso não acontece de graça, estamos trocando velocidade por capacidade de memória. Para cada partícula temos que guardar

vinte e sete ponteiros enquanto o método da malha regular mantém apenas um ponteiro.

7.6 Comparação de falsos vizinhos

Um falso vizinho é uma partícula que a estrutura de dado retorna de forma errada. Esse falso vizinho está próximo do ponto pesquisado, mas fora do raio de busca.A figura 7.7 mostra o número de falsos vizinhos retornado por cada estrutura de dados.

O método que retorna o menor número de falsos vizinhos é o método de força bruta, depois de testar todas as partículas, ele retorna exatamente as que estão no raio de busca, isso é, o método de força bruta retorna zero falsos vizinhos, por isso ele não aparece na figura 7.7.

O método Verlet fica em segundo lugar, e os métodos baseados em malha regular retornam o mesmo número de falsos vizinhos.

Nos métodos que possuem malha regular (Método proposto e Malha Regular), o número de falsos vizinhos depende da resolução da malha. Já no método Verlet esse número depende de um parâmetro que diz quantas partículas entram na lista de Verlet.

Mesmo fornecendo um número maior de falsos vizinhos, os métodos de malha regular são mais eficientes, devemos a isso a implementação de testes muito eficientes para determinar se uma partícula está realmente na vizinhança. Esses testes são realizados em um pipeline e a maioria dos falsos vizinhos é descartada muito rapidamente.

7.7 Comparação da utilização de memória

A ultima comparação foi baseada na quantidade de memória que cada tipo de estrutura utilizou. O resultado dessa comparação é apresentado na figura 7.8. Essa figura apresenta a memória total do programa (não somente da estrutura) para os três casos de simulação.

O método de força bruta não utiliza nenhuma memória extra para encontrar os vizinhos. Assim, seu gráfico representa a quantidade de memória que foi utilizada para armazenar os contatos e as propriedades das partículas.

O método proposto é que consome a maior quantidade de memória. Em segundo fica o método Verlet e em terceiro o método da malha regular. De maneira simplista a quantidade de memória que o método proposto utiliza

pode ser calculado da seguinte maneira:

$$\begin{aligned}\text{Mémoria total} = \#\text{células na malha} \times \text{overhead da lista} \\ + 27 \times \#\text{partículas} \times \#\text{bits por ponteiro.}\end{aligned}$$

Isso significa que desprezando o overhead da implementação da lista podemos simular aproximadamente 1.1 Milhões de partículas com cada 1Gb de memória.

Na prática, conseguimos metade disso. Isso acontece devido à forma que implementamos a lista: utilizamos as estruturas da STL. Essas podem ser otimizadas de maneira a melhorar a utilização da memória.

Mesmo com uma implementação simples a estrutura proposta serve para problemas práticos que utilizam alguns milhões de partículas. Se for necessário ir para a escala de centena de milhões de partículas, torna-se proibitiva a aplicação da estrutura proposta.

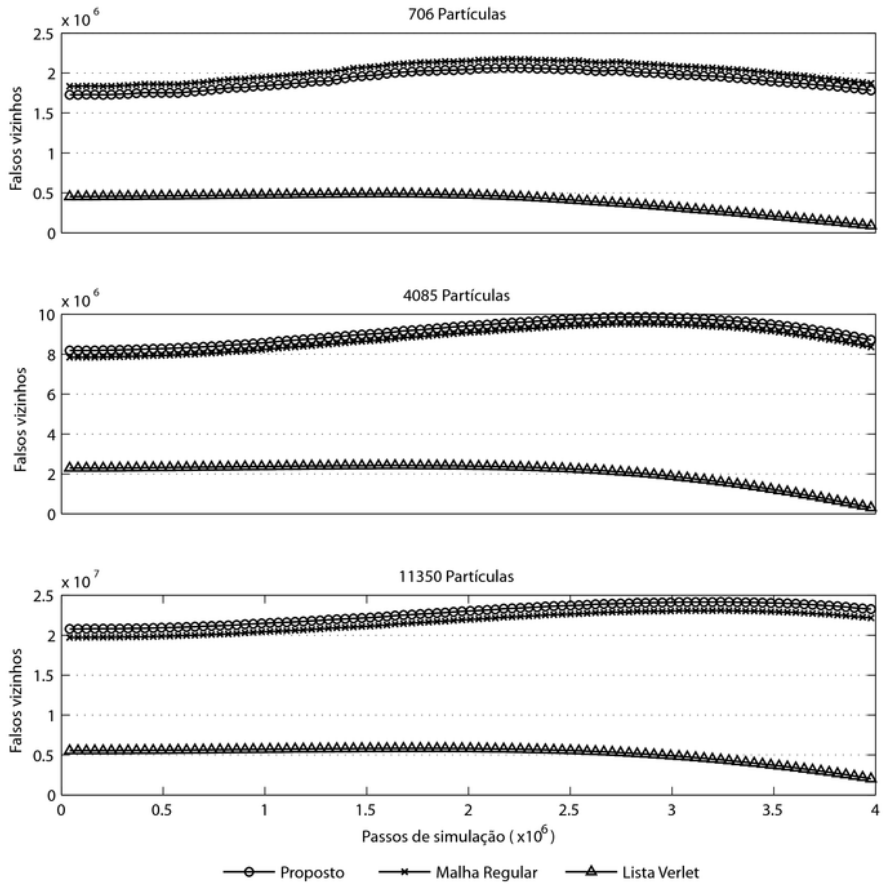
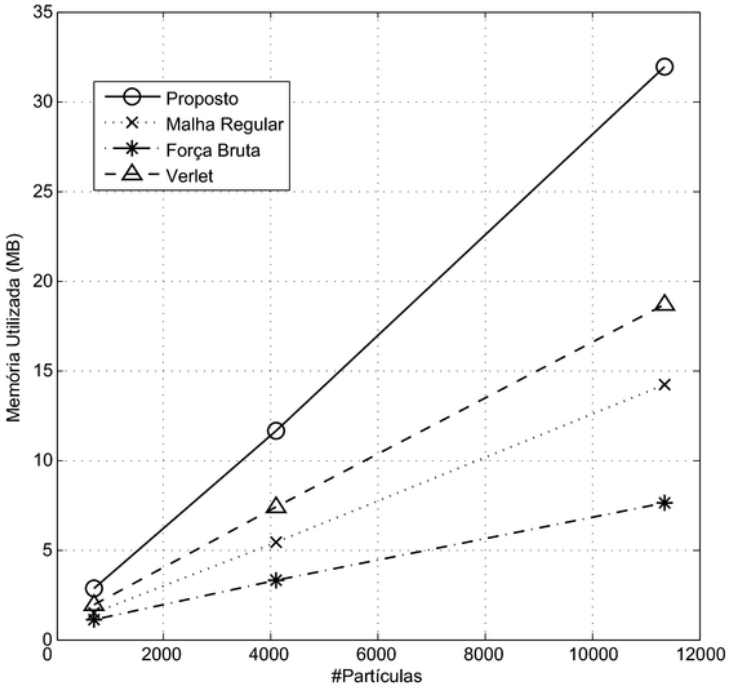


Figura 7.7: Número de falsos vizinhos (quanto menor melhor). Tanto o método proposto quanto o método da malha regular retornam o mesmo número de falsos vizinhos, isso acontece pois os dois são baseados em malha, e esse número depende da resolução da malha. O método da lista Verlet retorna uma quantidade proporcional ao parâmetro que determina o tamanho da lista. Mesmo fornecendo um número maior de falsos vizinhos, os métodos de malha regular são mais eficientes, devemos a isso a implementação de testes muito eficientes para determinar se uma partícula está realmente na vizinhança. Esses testes são realizados em um pipeline e a maioria dos falsos vizinhos é descartada muito rapidamente.



	#706	#4085	#11350
Proposto	2808	11620	31916
Malha Regular	1500	5396	14220
Força Bruta	1084	3264	7628
Verlet	1932	7388	18656

Figura 7.8: Utilização de memória (quanto menor melhor). O método proposto é o que mais utiliza memória, com ele conseguimos simular aproximadamente 0.5 milhões de partículas com cada 1Gb de memória.