6
Resultados

Este capítulo mostra algumas estratégias consideradas antes de chegarmos ao sistema proposto, mostra também os resultados obtidos ilustrados por diversos experimentos computacionais, além de fazer uma breve comparação entre o sistema proposto em diversas plataformas gráficas e a comparação entre o sistema proposto implementado na GPU e na CPU.

6.1.
Estratégias Consideradas

Antes de considerar o uso da estratégia proposta, alguns outros métodos foram testados. A primeira estratégia se baseou no trabalho de Kipfer et al. [9] que propôs fazer uma ordenação para detectar colisão. Sua estratégia ordena as células e a colisão é testada com base nesta ordenação. A coerência entre os quadros era aproveitada e a ordenação era distribuída ao longo de vários quadros. Adotando esta estratégia, o conjunto de potenciais colisores a cada quadro é muito impreciso. Como nossa proposta inclui colisão em ambientes confinados, a ordenação completa deveria ser feita a cada quadro. Para testar esta idéia, optou-se por implementar um *bubble sort*, ao invés do *bitonic merge sort*. Esta implementação parecia razoável porque se pode verificar através de consultas de oclusão (*occlusion queries)* se a lista já está ordenada. Com isso esperava-se que mesmo fazendo a ordenação completa a cada passo, um bom desempenho seria obtido. Após fazer a ordenação, uma busca binária seria feita para criar uma lista que mapeasse as células em sua posição de início e fim neste conjunto ordenado. Com posse disso, o módulo de detecção de colisão poderia verificar em qual célula o centro da partícula está e acessar todos os seus vizinhos. Porém, mesmo existindo uma forte coerência entre os quadros, uma mudança na ordenação pode ser muito grande, o que implicaria em muitas passadas em um só quadro para fazer a ordenação. Como o número de partículas é elevado, esta mudança se torna constante tornando esta implementação ineficiente.

Outra estratégia considerada foi fazer a implementação da montagem da estrutura de *grid* no programa de fragmentos, e não no programa de vértices. Para isso, um *grid* com as posições iniciais das partículas era montado na CPU. A cada quadro seguinte, um programa de fragmentos verificava para cada célula se suas partículas ainda estavam dentro daquela mesma célula. Verificava ainda se alguma partícula das células vizinhas teria passado para aquela célula. Com isso, numa passada, todas as células do *grid* seriam atualizadas. Esta estratégia não se encaixou bem à representação do *grid*, exigindo que um número máximo de partículas por célula fosse definido a priori. Além disso, o *shader* necessário para fazer as atualizações, possui muitas operações de *branch*, o que faz com que o processador gráfico perca eficiência.

Por também exigir muitas operações de *branch*, usar os quatro canais de cores para armazenar as partículas no *grid* se mostrou ineficiente. Uma solução híbrida de CPU e GPU também foi testada. A montagem do *grid* era feito na CPU e o resto da simulação na GPU. Esta solução também se mostrou ineficiente.

6.2. Experimentos Computacionais

Uma série de experimentos computacionais foram feitos para ilustrar o sistema proposto e descrito no Capítulo 5. As taxas medidas são referentes a uma renderização com pontos opacos, não sendo portanto necessário a ordenação das partículas para visualização. Os experimentos foram executados em uma placa de vídeo modelo NVIDIA GeForce 8800 GTX 768 Mb.

6.2.1. Vulcão de Partículas

Neste experimento um vulcão é representado através de um terreno digital. Um emissor de partículas é instanciado, emitindo partículas em posições aleatórias dentro do vulcão. Um total de 32K partículas são emitidas ao longo de mil quadros. As partículas fluem montanha abaixo, como ilustra a Figura 13. Após todas as partículas serem emitidas, o experimento é executado a 109 fps.

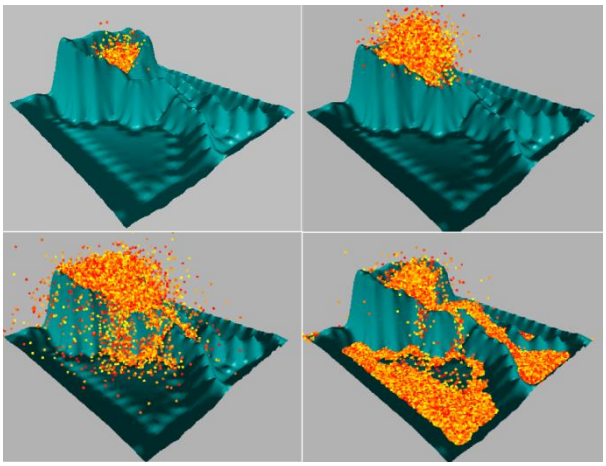

Figura 13 Imagens do Vulcão.

6.2.2. Globo de Neve

Este experimento demonstra a criação de formas não esféricas feitas com o agrupamento de partículas. Um boneco de neve é modelado dentro de um globo combinando diferentes obstáculos (5 esferas, 1 cilindro, e 1 plano). Um emissor gera 21K flocos de neve. Cada floco é composto por três partículas conectadas por barras rígidas: cada partícula possui duas restrições. A Figura 14 ilustra o experimento que é executado a 54 fps.

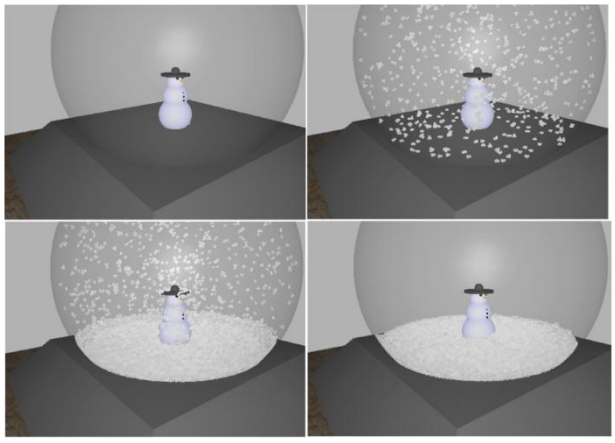

Figura 14 Imagens do globo de neve.

6.2.3. Galão com um Furo

Neste experimento um galão cilíndrico com 128K partículas de tamanhos diferentes é modelado. O raio das partículas varia uniformemente de 1 a 2 unidades. Um pequeno cilindro é usado como condição para criar o furo na parte

de baixo do cilindro maior, como ilustrado na Figura 11. As partículas caem numa caixa, esvaziando o galão. A Figura 15 ilustra o experimento, que é executado a 22 fps.

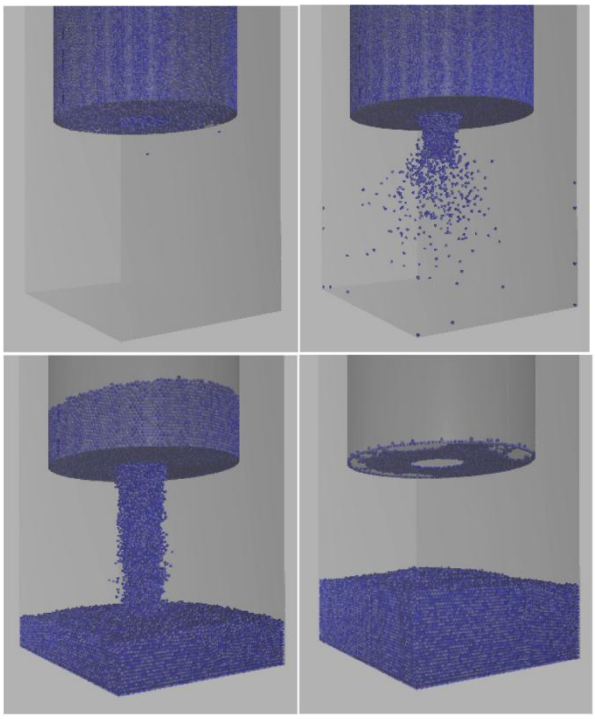

Figura 15 Imagens do experimento do galão.

6.2.4. Barragem de Partículas

Neste experimento é montada uma pilha com 1024K partículas confinadas numa barragem, como mostrada na primeira imagem da Figura 16. No início da simulação, as partículas são liberadas, mas continuam confinadas numa caixa com um cilindro como obstáculo adicional. A simulação é executada a 6 fps.

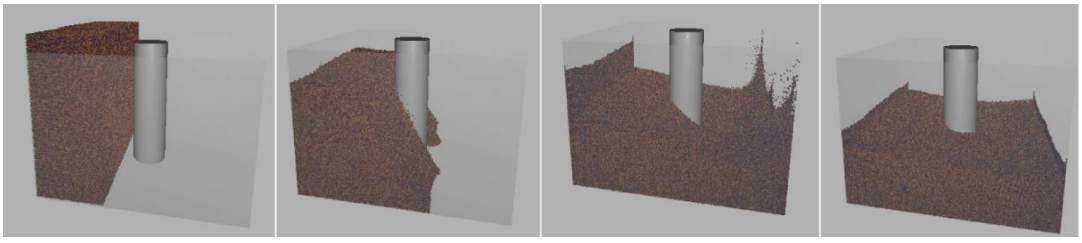

Figura 16 Simulação de 1 milhão de partículas liberadas de uma barragem em um ambiente confinado sendo executada a 6fps.

6.2.5. Sumário e Discussão

A Tabela 2 resume o desempenho obtido em uma GeForce 8800 GTX nos experimentos computacionais.

Experimento	# de Partículas	Desempenho
Vulcão	32K	109 fps
Globo de Neve	64K	54 fps
Galão	128K	22 fps
Barragem	1024K	6 fps

Tabela 2 Desempenho alcançada na GeForce 8800 GTX.

O sistema proposto também pode ser executado em outras placas gráficas. O gráfico da Figura 17 mostra o experimento Barragem de Partículas, implementado com 32K partículas e com 128K partículas, em diversas plataformas gráficas. Este gráfico mostra a tendência de evolução das GPUs, reforçando a idéia de que soluções baseadas na GPU são cada vez mais promissoras.

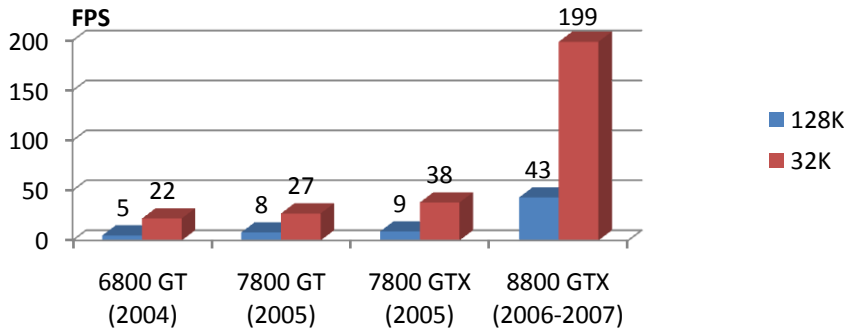

Figura 17 Desempenho do sistema proposto em várias plataformas gráficas

Para grandes números de partículas, soluções baseadas na GPU são facilmente superiores às soluções baseadas na CPU. Para comparação, foi implementado um sistema de partículas correspondente na CPU. A configuração das CPUs testadas eram um AMD Opteron com 2 processadores Dual-Core 2.41GHz e um Pentium Core2Duo 1.8Ghz.

O gráfico mostrado na Figura 18 revela que para experimentos com mais de 32K partículas, o sistema proposto baseado na plataforma gráfica 6800 GT é superior a mesma solução baseadas em uma CPU com quatro núcleos, e quando a plataforma

gráfica é a 7800 GTX, experimentos com mais de 1K partículas já possuem melhor desempenho na GPU.

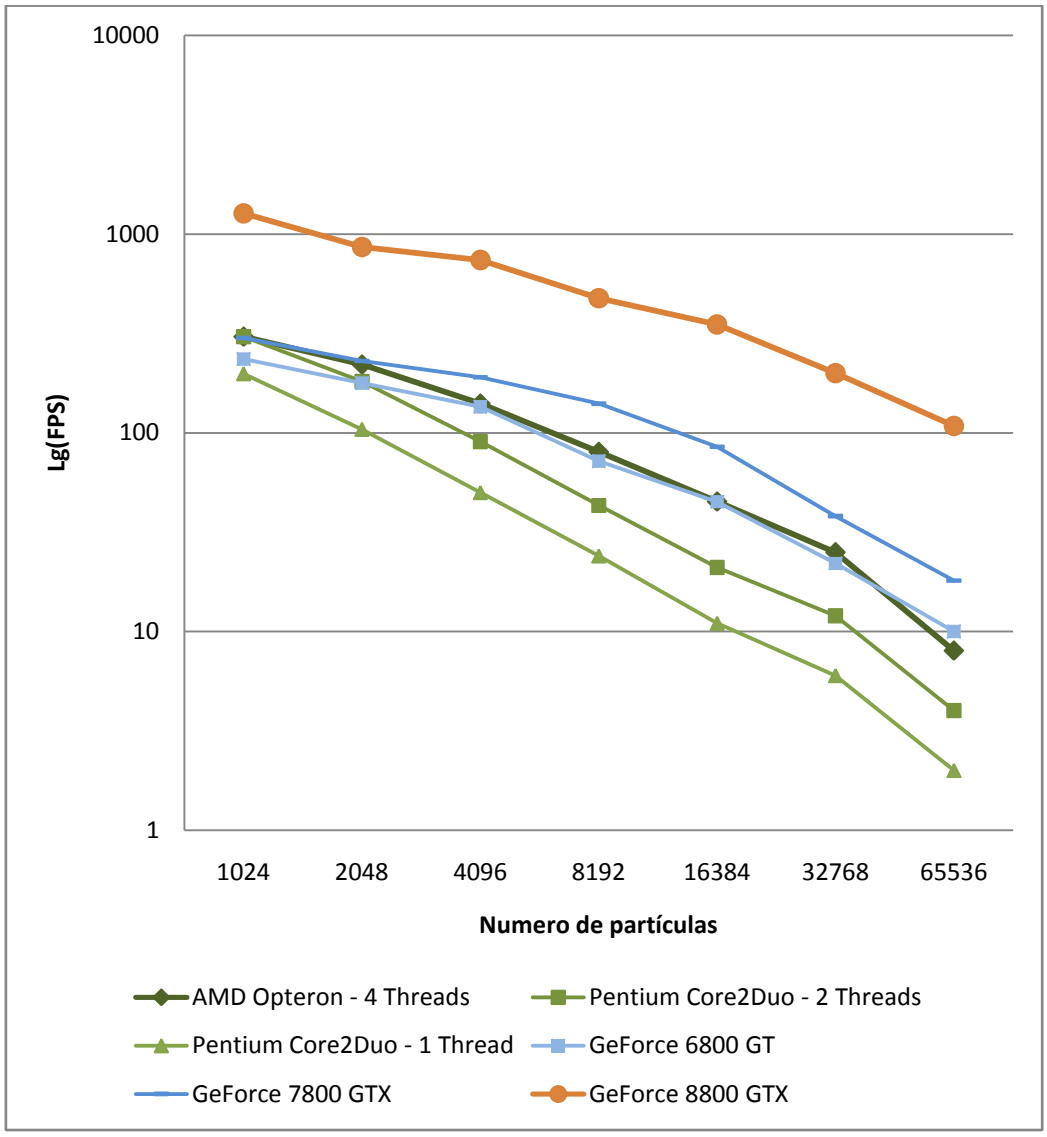

Figura 18 Desempenho do sistema proposto em diversas plataformas

Para comparação com outros trabalhos, em um passo do método de relaxação a resposta à colisão foi computada apenas com a partícula mais próxima. Com isso, houve a necessidade de um número muito maior de iterações em um passo de integração para manter a mesma estabilidade, afetando drasticamente a performance do sistema.

Para testar a estabilidade do sistema, o experimento do Galão foi repetido com distribuições de raio variadas. O número de passadas de rendering necessárias para a construção do grid aumenta quando a variação dos raios

aumenta. O desempenho do sistema diminui, porém o sistema continua numericamente estável, mesmo com apenas uma iteração do método de relaxação.

Para partículas do mesmo tamanho, o número de partículas mapeadas em uma única célula não ultrapassa oito. Com isso, pode-se usar a construção do grid como proposto por Purcell et al. [6]. Porém, a quantidade de memória usada para armazenar o grid se torna proibitiva, já que precisaríamos alocar espaço para oito partícula em cada célula.