2
Trabalhos Relacionados

As pesquisas em simulação de sistemas de partículas são muito anteriores à criação dos processadores gráficos programáveis. Um dos maiores desafios é computar a interferência entre partículas de uma maneira eficiente. Este problema nos leva naturalmente a uma solução quadrática, o teste de cada partícula contra todas as outras. Alder e Wainwright [1] idealizaram a subdivisão do espaço em uma grade regular de células. As partículas são associadas às células que contém seu centro. Chamaremos esta estrutura de dados de *grid*. Com isso, cada partícula só tem interferência sobre as partículas que estão em sua própria célula ou nas células vizinhas. Para diminuir mais os testes entre partículas, pode-se tirar vantagem de coerências temporais e espaciais do movimento das partículas [12,19], porém essas otimizações não são mapeadas de maneira simples para o modelo de computação em paralelo usado na GPU.

Latta [10] apresentou uma implementação de um sistema de partículas sendo executado inteiramente na GPU. Esta implementação incluía simulação, renderização e colisão contra cena, e conseguia lidar com um milhão de partículas em tempo real; porém, esta proposta não tratava a colisão entre partículas. Kipfer et al. [9] apresentaram um sistema de partículas na GPU que computava colisão entre partículas. O identificador da célula, atribuído a cada partícula, era usado como chave de ordenação das partículas. Para a ordenação eficiente das partículas na GPU, foi proposto uma melhoria em relação à implementação na GPU do algoritmo de ordenação bitônica de Purcell et al. [17]. Outros algoritmos de ordenação baseados na GPU têm sido apresentados, como o de Govindaraju et al. [7]. Com as partículas ordenadas, o conjunto de potenciais colisores de cada partícula é extraído com base em sua vizinhança no vetor ordenado. O sistema proposto por Kipfer et al. [9] só trata a colisão da partícula mais próxima. Por esta razão, acreditamos que não seja possível fazer simulações de partículas numericamente estáveis em ambientes confinados (pelo menos, tais exemplos não foram fornecidos).

Purcell et al. [17] propuseram um algoritmo alternativo para a construção de um *grid*. O algoritmo usa o programa de vértices para organizar as partículas (neste caso, fótons) nas células do *grid*, e usa o *stencil buffer* para lidar com múltiplas partículas mapeadas na mesma célula. O algoritmo proposto por Purcell et al. [17] monta o *grid* da seguinte maneira: Cada célula usa $m \times m$ texels, podendo armazenar $m \times m$ partículas. No programa de vértices é computada a posição de cada partícula nesta textura, e esta partícula é escrita como um ponto de tamanho *m*. Para lidar com múltiplas partículas sendo escritas na mesma posição, o *stencil buffer* é usado. O *stencil buffer* é inicializado com cada texel da célula numerado de 0 até m^2-1. Na montagem, o teste de stencil é habilitado para escrever apenas quando o valor de *stencil* for igual a m^2-1, e sempre fazer o incremento do valor de *stencil*. Assim quando um ponto é desenhado, apenas um fragmento é escrito, mas toda a região do ponto tem seu valor de stencil incrementado, assegurando que o próximo ponto desta célula será desenhado em um texel diferente. Se o número máximo de partículas que pode ser inserido numa única célula é conhecido, este método constrói o *grid* em uma única passada, ao contrário do método de ordenação que exige várias passadas.

Se todas as partículas tivessem o mesmo tamanho, e tendo o tamanho da célula igual ao diâmetro das partículas, asseguramos que no máximo oito partículas (caso não haja interpenetração)são inseridas numa única célula. Porém, se o tamanho das partículas é variável, o número máximo de partículas numa única célula pode ser muito grande e não pode ser computado *a priori*. Portanto o algoritmo proposto por Purcell et al. [17] não atende a nosso objetivo de simular partículas de diâmetros variáveis.

Neste trabalho é proposto um novo algoritmo para a construção do *grid*. Como na proposta de Purcell et al. [17], o algoritmo proposto usa o programa de vértices para a montagem do *grid*, porém a organização das partículas que estão na mesma célula se dá através de uma lista encadeada. Este algoritmo requer *n* passadas, onde *n* é o número máximo de partículas numa única célula.

Kipfer et al. [9] usou o método de integração numérica de Euler para calcular as equações de movimento das partículas. A resposta a colisão é feita calculando o tempo de colisão e ajustando a posição e velocidade de cada partícula. Neste trabalho, foi adotado o método de integração de Verlet para a

atualização da posição das partículas [5]. A resposta a colisão é computada através do método da relaxação [16], descrita por Jakobsen [8].

Simulação de partículas baseadas em Smoothed Particle Hydrodynamics (SPH) também tem sido usada para agregar realismo à animação de fluidos [13,4,14]. SPH também é considerado como um método numérico alternativo para a simulação de modelos com grandes deformações [11]. O sistema aqui proposto pode ser adaptado para simulações com SPH. Em SPH, precisamos computar o conjunto de partículas que estão no domínio de uma função de suavização. Podemos calcular eficientemente a vizinhança das partículas colocando o tamanho da célula igual ao raio do domínio de suporte da função de suavização.