**Sistemas de Partículas paramétricos**

**-** Sistema de partículas facilmente processados dentro do pipeline gráfico.

A posição de cada partícula é determinada colocando o parâmetro *t,* de tempo, na equação que define o movimento da mesma.

Esta abordagem tem duas vantagens principais:

Não existe guardar adicionalmente o estado intermédio de uma partícula.

É uma solução analiticamente exata para descrever o movimento de partículas, sem envolver a integração de equações em ordem de determinar a deslocação da partícula. O processo é apenas calculado com base no tempo.

Apresenta como desvantagens:

Uma vez definido o sistema de partículas, a equação utilizada para modelar o comportamento das partículas não pode ser alterada. Isto limita a capacidade do sistema em reagir com o ambiente ou consigo próprio em tempo real.

O uso destes sistemas está limitado a contextos onde o movimento se possa descrever matematicamente, com base numa equação composta por parâmetros dependentes.

Estes sistemas são utilizados para modelar virtualmente propriedades do mundo real, capazes de serem descritas matematicamente.

Implementados a nível dos vertex shader

**Sistemas de Partículas não paramétricos**

O sistema utiliza equações de aceleração que integra ao longo da simulação, para determinar a velocidade instantânea de uma partícula. A equação da velocidade instantânea é também integrada ao longo do percurso da simulação para calcular a posição de cada partícula para cada momento de execução.

Apesar desta abordagem ser menos precisa que a anterior, apresenta a possibilidade do sistema interagir com o seu ambiente e consigo próprio.

**Armazenamento**

Focando os sistema de partículas não paramétricos, é necessário guardar as informações relativas à aceleração e velocidade da partícula para cada instante. Estas duas informações definem aquilo que é limitado como o estado de uma partícula.

Existem duas formas de guardar estas informações:

Utilizando um VBO

Utilizando texturas

Esta abordagem baseada em sistemas não paramétricos obriga a processo de leitura, alteração e escrita, para realizar o update da deslocação de uma partícula (read and modify) e guardar as novas informações para utilizar na iteração seguinte (write). Contudo, operações desta ordem não são permitidas no pipeline gráfico, pelo facto dos objetos serem partilhados entre pipelines que podem estar em estágios de execução distintos. (why?)

Sistemas de partículas em tempo real são atualmente condicionadas pelas limitações de comunicação e troca de informações entre CPU e a GPU, massivamente eficaz na computação de procedimentos gráficos.

Artigo 2

**Passos da simulação de um sistema de partículas**

1. Processo de nascimento e morte
2. Update velocidade e posição
3. Ordenar por alfa blendin (opecional)
4. Transferir dados textura para o vertex data
5. Renderizar partículas

Num primeiro passo, a textura com as informações da velocidade são atualizadas, tendo em conta forças de aceleração ou colisões com outros objetos do ambiente. Numa segunda fase, atualiza-se a textura com as posições de cada partícula de forma similar, utilizando os dados previamente calculados na textura da velocidade.

Em alguns contextos, dependendo do método de deslocação utilizado, a aplicação de forças ou reações a colisões pode ser diretamente processada sobre os dados da textura com as posição das partículas, evitando assim a necessidade da textura com as informações relativas à velocidade de cada partícula.

As posições e velocidades de cada partículas são guardadas como dados numéricos do tipo *float*, utilizando os parâmetros RGBA possíveis de preencher em cada texel da textura. As componentes RGB registam assim as componentes X,Y e Z referentes à posição ou ao vetor da velocidade.

Dado que operação de leitura-modificação-escrita não são permitidas nos componentes programáveis da GPU,

Dado que uma textura não pode ser utilizada no pipeline gráfico como sendo simultaneamente de escrita e de leitura, é utilizada uma técnica de duplicação de dados. De um lado são lidas as informações do estado anterior para calcular as novas posições. Do outro buffer são escritos os novos valores calculados. No final, os fios de execução dos diferentes estados do pipeline são sincronizados, e os buffers trocados. Este método de duplicação de buffers é realizado tanto com texturas como com vertex buffers.

Artigo 3

Definição de três tipos de sistemas de partículas:

Sem interação. Partículas independentes. Complexidade computacional O(n)

Interação curto alcance. Partículas interagem e sofrem influencia de uma vizinhança próxima de si. Complexidade entre O(n^2) e O(n), para estruturas que permitam facilmente conhecer a vizinhança de um ponto, como árvores (k-D trees) ou grafos.

Interação longo alcance. Uma partícula sofre interações e influencias de todas as outras partículas.

**Método de Euler**

Permite obter velocidades precisas para acelerações constantes e posições precisas para velocidades constantes.

Perde capacidade de modelação para intervalos de t consideravelmente espaçados entre si.

Vn+1 = Vn + hAn

Pn+1 = Pn + hVn

**Sistema de partículas**

Coleção de várias partículas atómicas, que em conjunto modelam um objeto com limites indefinidos. Ao longo de um período de tempo, novas partículas são geradas dentro do sistemas, deslocando-se e sofrendo alterações segundo um determinado comportamento do sistema, até terminarem a sua execução.

William T.Reeves 1983

AS partículas podem ser desenhadas como pontos, polígonos com texturas ou geometria básica. Uma partícula pode ainda, para um determinado contexto no seu ciclo de vida, seu um emissor de partículas. Por exemplo, num efeito de chuva, assim que uma gota termina a sua queda atingindo aquilo que no modelo represente o chão, apesar da partícula que represente a gota se extinguir, antes de ser retirada a sua posição passa a ser o emissor das partículas que representam o choque (“spash”) e divisão da gota em partículas de menores dimensões

**Artigo 1**

**Análise de Boids implementados a nível da GPU**

Calculo de interações em sistemas de partículas de longo alcance: Um partícula é influenciada por todas as restantes partículas ativas do sistema. Ou, noutra perspetiva, uma partícula afeta todas as partículas do sistema onde se insere.

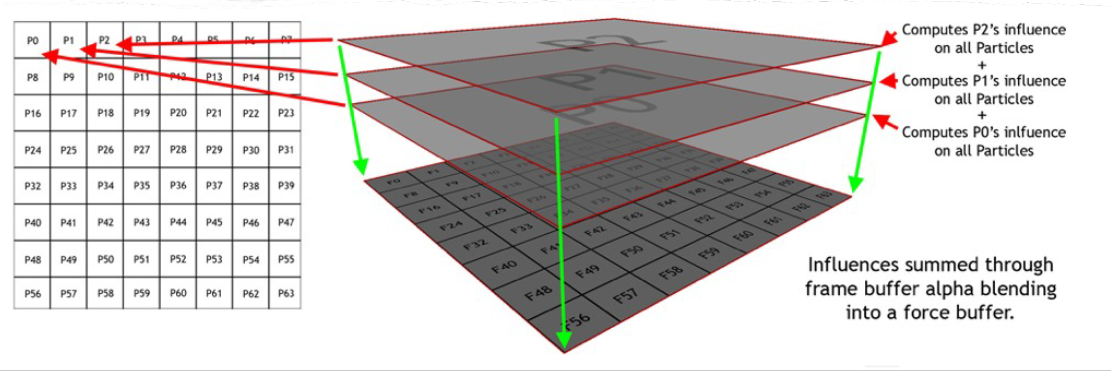
Utilização da técnica de Force Splatting, onde a influencia de uma partícula é projetadas sobre todas as outras partículas, acumulando aditivamente a influencia total das N partículas para cada uma das partículas. O processo é conseguido através do uso de uma nova textura, na qual se armazena a influência acumuladas de todas as partículas.

A influencia de uma partícula sobre todas as restantes, é calculada num quad, com as mesmas dimensões que a textura anteriormente referida. Existirá assim uma stack com N quadas, um para cada partícula do sistema.

Os vértices de cada quad, identificam a partícula Pi que representam e são utilizados para incrementar a influencia da partícula Pi no buffer.

A acumulação das forças é processada pela técnica de apha blending de todos os quads. Apesar do processo de criação dos quads por partícula ser O(n2), o processo tira partido das rápidas capacidades de rasterização das GPUs.

Na fase de *update* da posição de cada partícula, a respetiva força acumulada guardada na textura é dividida pelo valor que representa a massa da partícula. Este parâmetro estará por exemplo guardado na terceira textura anteriormente apresentada. Com este calculo, obtém-se a aceleração instantânea de cada partícula.



**Flocking / boids**

Sistemas de partículas que procuram simular os comportamentos e deslocação de objetos em grupos, como bando de aves, cardumes de peixes ou insetos.

Estes sistemas baseiam-se em três conceitos chave, que definem o comportamento das suas entidadas: EXPLORAR BOIDS

<https://www.red3d.com/cwr/boids/>

Separação: As partículas deslocam-se de forma a evitar aglomerações locais, reduzindo assim eventuais situações de colisão entre si

Alinhamento: as partículas movimentam-se segundo a direção média do bando

Coesão: As partículas deslocam-se para a posição média da sua vizinhança.

A nível de uma implementação na GPU, o conceito de separação pode ser calculado através da técnica de force splatting anteriormente descrita. Neste contexto, a influencia de cada partícula nas restantes descreve a força de repulsão que as leva a afastar, para evitar formar grupos de partículas muito próximos.

A noção de alinhamento e coesão, baseiam-se na média da direção e posição de uma vizinhança de partículas.

Estes dois processos podem facilmente ser calculadas através da técnica conhecida como Fast Avaraging, que tira partido dos métodos de Mip-Mapping automaticamente gerados pela GPU para texturas.

Assim, considerando a textura com os dados da velocidade de cada partícula, o mip map de menor dimensões tem como valores RGB os respetivos valoes de X,Y e Z que descrevem o vetor médio da velocidade.

Este vetor é adicionado ao vetor da força calculado através da técnica de force splatting.

[Artigo 1]

O maior bottleneck surge na transferências das informações dos vértices (geometria) a renderizar para a placa gráfica.

Surgem assim duas abordagem:

realizar o processamento dos vértices a cada frame no lado do CPU, e enviar as informações para a GPU para posterior desenho

Conter todos os dados necessários já no contexto de execução da placa gráfica, evitando assim as limitações de velocidade entre o barramento CPU GPU.

O processamento a nível da GPU, nos contextos em que tal seja possível, é potencialmente muito mais rápidos, obtendo assim ganhos de desempenho significativos.