FACULDADE DE CIÊNCIAS E TECNOLOGIA DA UNIVERSIDADE DE COIMBRA
DEPARTAMENTO DE ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES
MESTRADO EM TECNOLOGIAS DE INFORMAÇÃO VISUAL

DEEC

Arquitectura para sistemas de partículas dinâmicos

Tiago José Adão Pedrosa Carvalheiro, aluno número 2005524238

Orientador:
Dr. José Carlos de Gouveia Teixeira
Sub-orientador:
Lic. João Ramos

Júri:
Dr. Paulo Coimbra (Presidente);
Dr. José Teixeira (Orientador);
Dr. Paulo Peixoto (Vogais);

Julho 2011

Agradecimentos

Pessoalmente, gostava de agradecer, em especial, à família, por todo o apoio fornecido e pelo facto de terem proporcionado a possibilidade de concluir este ciclo com sucesso.
Um grande obrigado para o João Ramos, pela revisão deste trabalho, conselhos fornecidos, sugestões, guias de orientação, tanto na teoria como na prática, pela amizade criada, pelas discussões saudáveis que proporcionaram a minha aprendizagem e pelo apoio que forneceu ao longo deste projecto.
Um especial reconhecido e grato obrigado ao Doutor José Carlos Teixeira, por elevar a competência e capacidade do meu conhecimento, por toda a orientação, apoio, ajuda e conselhos que me deu ao longo de todo o projecto.
Também gostava de agradecer a toda a equipa da MediaPrimer, em especial ao João, Tiago e Andreia, por toda a amizade e apoio. Por último, gostava de agradecer aos meus colegas de curso. Aos quais estarei sempre grato pela companhia, apoio, amizade e momentos passados.

Dedicatória

Gostava de dedicar este trabalho ao meu avô, que faleceu durante a conclusão deste projecto.

Resumo

Este relatório pretende documentar o trabalho realizado na Dissertação de Mestrado em Tecnologias de Informação Visual. O trabalho desenvolvido ambiciona a discussão de uma arquitectura para sistemas de partículas dinâmicos, onde se tenciona agrupar e definir todos os aspectos que são intervenientes nesta área, de uma forma intuitiva e construtiva. Pretende-se recorrer à utilização de um motor físico para proporcionar uma evolução da arquitectura, disponibilizando novas ferramentas que permitem atingir novos patamares de dinâmica nos movimentos das partículas e consequentemente a concepção de novos sistemas. A simulação de sistemas de partículas dinâmicos tem sido empregada em animação computacional durante vários anos, e recentemente a sua maior usabilidade é introduzida em sistemas de simulação em tempo real na área científica, ambientes virtuais, bem como na utilização de efeitos na indústria dos jogos. A sua importância tem sido reconhecida como um bloco de desenvolvimento essencial para o enriquecimento detalhado de fenómenos em ambientes que simulam elementos naturais. As aplicações actuais são na sua grande maioria limitadas pela enorme transferência de dados que existe entre o processador principal (CPU) e o processador gráfico (GPU). Existem muitas investigações que exploraram várias maneiras diferentes de computação em sistemas de partículas, mas até ao momento as pesquisas e discussões sobre o modo de organizar uma arquitectura com uma interface sustentável para implementação, uso e futuras extensões dos trabalhos realizados, foram escassas. Para analisar o desafio proposto, deve-se encontrar uma resposta para as seguintes questões:

- Que tipo de abstracção será mais útil para a especificação de sistemas de partículas?
- Que tipo de arquitectura pode ser construído, de modo a que novos efeitos possam ser introduzidos?
- Qual o conjunto de funcionalidades que devem ser disponíveis na aplicação ao nível de CPU, como ao nível de equipamento especializado (GPU)?

Palavras-Chave: Arquitectura; Sistemas de partículas dinâmicos; Modelação e Animação Física; Motor Físico; Computação Gráfica; Dinâmica;

Abstract

The aim of this report is to document the work done in the Master's Dissertation in Visual Information Technologies. This work aims for a discussion on developing an architecture of dynamic particle systems, where it´s intended to group and define all the aspects of it, in an intuitive and constructive form. It is intended to use a physics engine, used for a physically-based modeling, to allow a natural evolution on the architecture, captivating the advantages of the new tools in order to provide a new level of dynamism in particle movement and the creation of new systems. Simulations using dynamic particle systems have been used in computer animation for several years, and recently it has been used in real-time systems such as science research, computer graphics, and in the game industry. Its value has been greatly recognized as a central block of development for simulation and creating detailed natural phenomena in virtual scenes. Real-time applications are greatly limited by the enormous data transfers between the main processor (CPU) and the graphics processor (GPU). There has been a great number of investigations regarding different approaches on the computation of dynamic particle systems, but until today, research and discussions about creating a substantial architecture interface for implementation, usage and future extensions, mas been insufficient. In order to analyze the proposed challenge, we should find the answer to the following questions:

- What form of abstraction will be more useful for defining a dynamic particle system?
- What nature of architecture can be built, so that new effects could be introduced?
- What are the functionalities that should be available to the application on the CPU level as well on the specialized equipment (GPU)?

Key Words: Architecture, Dynamic particle systems, Physically-based modeling, Physics engine; Computer Graphics; Dynamics

Índice

Lista de Figuras:

Lista de Tabelas:

Lista de Abreviaturas:

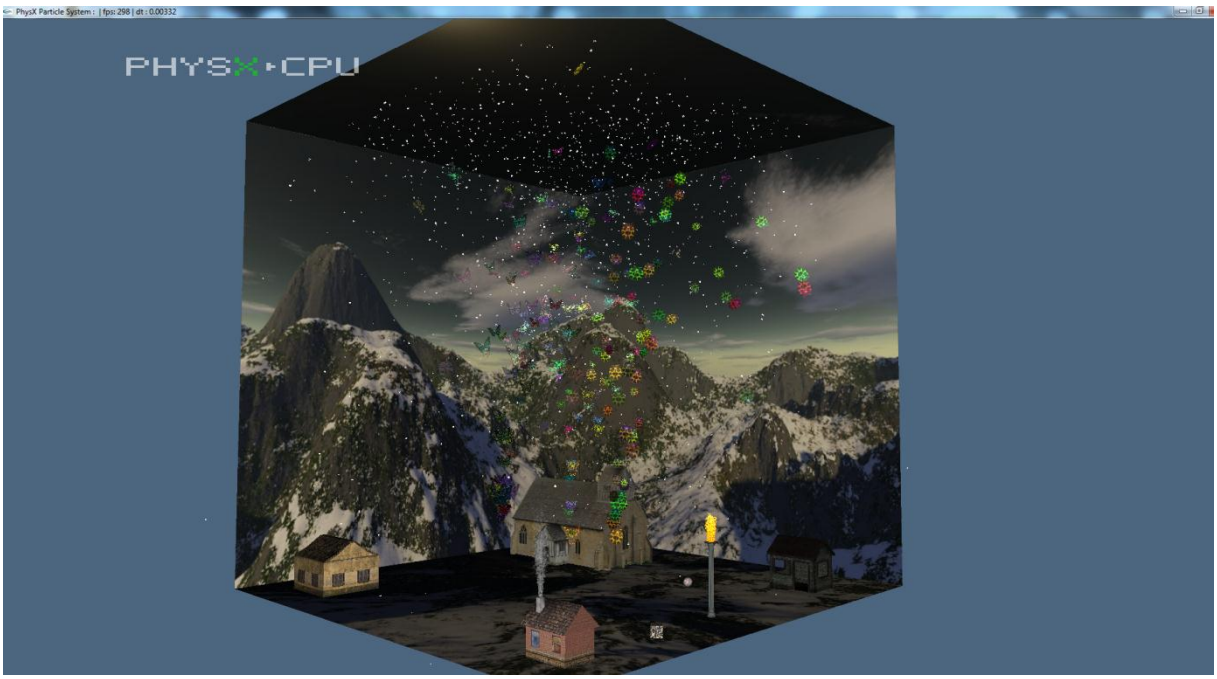
CPU – Central Processing Unit

GPU – Graphics Processing Unit

LOD – Level of Detail

SPH – Smoothed Particle Hydrodynamics

1. Introdução a uma arquitectura de um sistema de partículas



"The beginning of knowledge is the discovery of something we do not understand."

Frank Herbert, Author and Writer

1.1.Introdução

À medida que a computação gráfica vai evoluindo cada vez mais, a exigência e demanda para o realismo, qualidade e interacção também vai aumentando, tanto na concepção de efeitos, como na execução de simulações geradas computacionalmente. Para apresentar uma resposta eficiente, estamos perante uma necessidade urgente de integrar e desenvolver conceitos abstractos de componentes funcionais que enfrentam uma ambição e requerimento cada vez maior. Estes factos reforçam a construção de arquitecturas cada vez mais consistentes e que contenham uma interface portável perante as mudanças constantes ao nível de equipamento, como ao nível da interface das funcionalidades implementadas. A elaboração de uma arquitectura de um sistema de partículas deve ser feita de modo independente duas principais arquitecturas de computação gráficas disponíveis, OpenGL e Direct3D, que são abstracções das operações de baixo nível das simulações em tempo real. A arquitectura de escolha para o desenvolvimento e base deste trabalho foi OpenGL. No entanto tivemos em conta o Direct3D em algumas das escolhas de implementação feitas, de modo que a portabilidade de uma arquitectura gráfica para a outra seja quase directa, sem que fosse necessário proceder a alterações no núcleo da arquitectura proposta.

As simulações deste tipo de sistemas estão limitadas pela comunicação entre CPU e GPU e também pela velocidade que um GPU consegue desenhar cada quadro. Este último factor é limitativo quando existem um grande número de pixéis para processar e sobretudo quando existem também uma grande repetição de escrita dos mesmos. Esta limitação pode ser ultrapassada pela redução do tamanho de cada partícula, ao mesmo tempo que o efeito de realidade aumenta, necessitamos de aumentar a sua quantidade para que a redução no tamanho seja quase imperceptível. Assim, podemos contornar um pouco este problema, mas ainda temos que lidar com o primeiro, visto que agora a limitação principal da simulação situa-se na comunicação. Portanto, o novo desafio situa-se em contornar este obstáculo, por isso, é desejável que a quantidade de transferência de dados seja minimizada, e isto só pode ser obtido integrando as duas partes, ou seja, efectuar a simulação e o desenho do quadro em simultâneo no GPU, de modo a contornar esta grande limitação comum neste tipo de simulações, e ao mesmo tempo libertando o CPU para poder realizar mais trabalho noutras áreas da aplicação.

A introdução de detecção de colisão e movimento dinâmico foi efectuado através da introdução de um motor de física, que incorpora e oferece as funcionalidades para o

manuseamento de objectos e suas propriedades físicas. Normalmente, estes motores detêm interfaces para a sua integração perante as arquitecturas gráficas. A escolha do motor físico foi um processo importante para a organização interna.

1.2.Motivação

Hoje em dia, a indústria dos jogos e as simulações científicas são algumas das áreas que mais contribuíram para a investigação e consequente evolução de sistemas de partículas, desde a simulação de fumo num cenário virtual, à simulação de sangue numa cirurgia de treino assistida por computador. Este tipo de aplicações possui algo em comum, um bom sistema de partículas, ao mesmo tempo, simulam efeitos que são tecnologicamente difíceis de atingir. Para realizar estes efeitos é necessário construir um sistema de partículas, mas a implementação de um sistema simples não é suficiente. Será portanto necessário um bom planeamento e concepção de um sistema avançado, que apresente uma arquitectura estruturada que possa entre outras características preservar as seguintes qualidades: rapidez, flexibilidade e extensibilidade. Todas as funcionalidades devem ser encapsuladas de modo que os programadores não necessitem de as implementar outra vez, facilitando assim a portabilidade da arquitectura para outras plataformas. O interface deverá ser desenvolvido para que a sua integração com equipamento mais especializado não afecte a portabilidade do sistema.

A arquitectura descrita nesta investigação assume que a partícula seja calculada como um ponto num espaço tridimensional, acartando consigo determinados atributos dinâmicos. O sistema tem a responsabilidade sobre o cálculo dos valores que fazem parte desses atributos de cada uma das partículas, através do tempo. Este modelo proposto usufrui da implementação de alguns métodos rápidos de desenho.

1.3.Ambição

Este trabalho ambiciona vários módulos de simulação, desde os mais simples aos mais complexos como dinâmica de fluidos, que fazem uso da capacidade do processamento

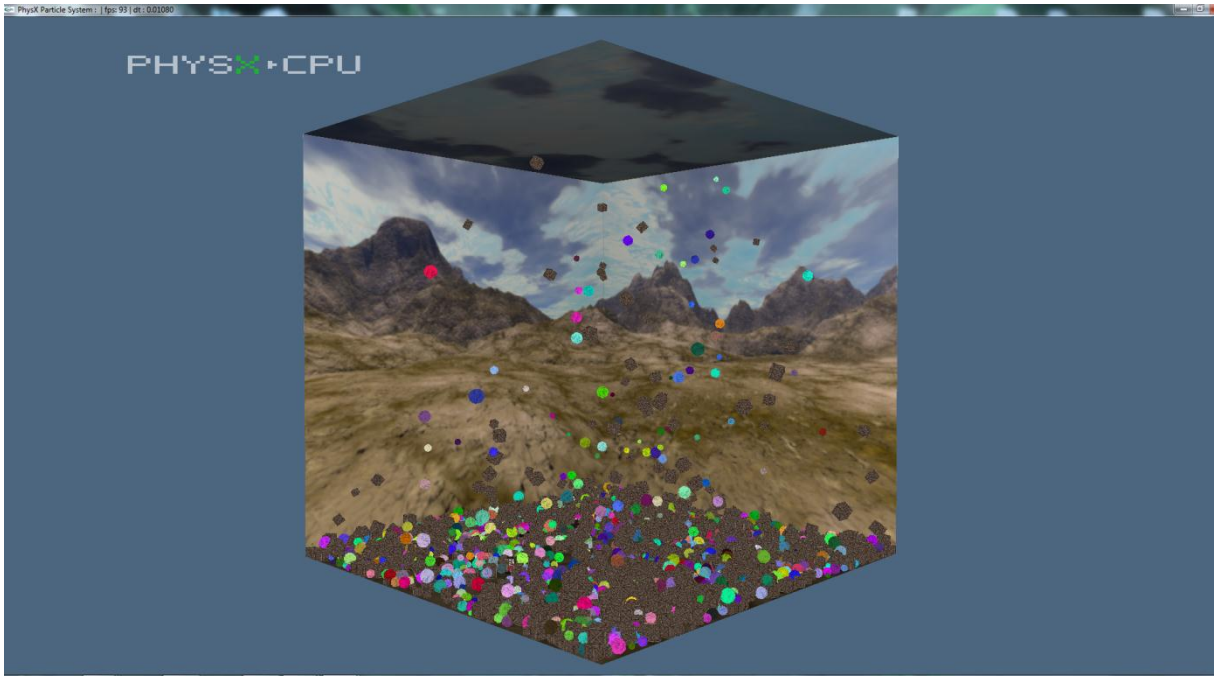
paralelo para efectuar cálculos físicos no GPU. Esta novidade marca do início da próxima geração de simulações físicas, tirando partido do facto que estes sistemas apresentam uma enorme escalabilidade para absorver as vantagens que o processamento em paralelo oferece, nomeadamente o aumento do número de operações por ciclo, relativamente ao processamento em linha. Esta inovação permite que as simulações possam agora ser mais complexas, atingindo novos patamares que antes não eram possíveis executar em tempo real. A realidade das simulações situa-se em grande parte na complexidade de movimento que pode ser simulado, portanto pretende-se uma combinação flexível de movimentos imprevisíveis, que partem na sua maioria de operações sobre posições, recorrendo a utilização de um conjunto de forças e da colisão com primitivas geométricas. O resultado desta pesquisa e implementação deverá ser uma arquitectura para sistemas partículas dinâmicos, criada e desenvolvida num ambiente de C++, constituída por um conjunto de bibliotecas que proporcione as funcionalidades necessárias e permita que aplicações simulem a dinâmica de conjuntos ou sistemas de partículas. A arquitectura deve ser pensada e criada de modo a facilitar a introdução de novas variedades de efeitos baseados em partículas nas aplicações interactivas. A introdução do movimento dinâmico das partículas está relacionado com os princípios da física de Newton, que foram utilizados para a simulação de movimentos mais realistas.

1.4. Trabalho previamente desenvolvido

Sistemas de partículas para simulações e animações em computação gráfica foram concebidos primordialmente por [1]. A investigação levada a cabo e a especificação deste sistema, nomeadamente os atributos de cada partícula e a forma de interacção e computação do sistema prevaleceram nas pesquisas consequentes até aos dias de hoje. Neste trabalho publicado, as partículas eram geradas através de 2 formas: esféricas e rectangulares. A velocidade inicial de cada partícula estava acumulada com a sua posição inicial. Os restantes atributos, como a cor, eram gerados estocasticamente recorrendo ao uso de uma média e uma variância. A dinâmica introduzida nesta investigação original era bastante simples. A única força que afectava as partículas após a sua criação era a gravidade, criando assim trajectórias no formato de parábolas. A grande complexidade atingida por este sistema derivava apenas da atribuição de valores iniciais aleatórios e não de uma dinâmica complexa. A investigação que sucedeu pelo mesmo autor [2], introduzia uma dinâmica de partículas mais sofisticada para a

simulação de movimentos, introduzindo relações entre partículas. Este trabalho serviu como ponto de partida, pois estabelece as bases necessárias para a compreensão e linhas gerais destes sistemas. A primeira investigação publicada sobre arquitectura de sistemas de partículas foi publicada por [7], onde apresenta um conjunto de funcionalidades bastante atraente, entre elas, grupos de sistemas de partículas, listas de acções (forças e colisões), que, apesar de não incluir um motor de física, incorpora um diferenciador de equações usado para calcular forças e colisões com objectos, incluindo também a utilização de processamento em paralelo recorrendo a *threads*. No mesmo ano [8] publica um artigo sobre considerações a ter na concepção de arquitecturas para sistemas de partículas, que apesar de apresentar uma abordagem bastante mais simples que [7], não deixa de ser uma boa referência para o desenvolvimento deste trabalho. A introdução a movimentos e dinâmica geral aplicada a um sistema de partículas foi publicada por [10], onde é realçado o método dos integradores de Euler, apresentando uma vista geral sobre como integrar e aplicar forças a sistemas de partículas.

2. A definição de um sistema de partículas



"Electricity is actually made up of extremely tiny particles called electrons that you cannot see with the naked eye unless you have been drinking."

Dave Barry, Pulitzer Prize winning author in "The Taming of the Screw"

2.1. Introdução

Um sistema de partículas é um conjunto de elementos individuais, ou partículas. Cada partícula contém atributos próprios, como velocidade, cor, tempo de vida, entre outros. Existem essencialmente dois tipos de sistemas de partículas: sistemas em que cada partícula age de modo autónomo, ou seja, sem considerar as acções de outras partículas e sistemas em que cada partícula age em conjunto com outras partículas através de dependências. Este trabalho tem em consideração o uso de estes dois tipos diferentes de sistemas, que vão ser definidos de forma distinta mais a frente.

Partículas que pertencem a um determinado sistema de partículas, de um modo geral, partilham entre si um conjunto de atributos semelhantes, que apesar existir ou não uma interdependência entre elas, no seu conjunto agem para criar um determinado efeito comum. Uma explosão é um exemplo simples para a visualização de um sistema de partículas, em que cada faísca podem ser vista como a representação de uma partícula. Através de um uso avançado de texturas e outras propriedades, os sistemas de partículas podem ser criados para simular uma grande quantidade de efeitos naturais e não só: fogo, fumo, líquidos como água ou sangue, neve, constelações de estrelas, trilhos de vapor, entre outros. Claramente, o objectivo era criar um projecto que tivesse potencialidade e que fosse ao mesmo tempo versátil, de modo que fosse capaz simular qualquer um destes efeitos.

Devido à estrutura natural em que consiste um sistema de partículas, a sua arquitectura orienta-se para uma implementação orientada a objectos, e foi precisamente esse caminho que este projecto tomou. Existem exactamente dois caminhos que se podem tomar na criação de uma arquitectura base, utilizando classes em C++: pode-se criar uma classe base para um sistema de partículas e derivar outra classe filha para cada tipo de sistema de partículas (por exemplo: a criação de um sistema de partículas para fogo, fumo, entre outros), ou pode-se criar uma arquitectura mais genérica com um conjunto de atributos ao qual se possa determinar tipo de sistema de partículas definido.

A organização proposta será analisada mais a frente, mas antes necessitamos de definir uma partícula e um sistema de partículas antes de podermos analisar em detalhe a implementação desta arquitectura.

2.2. A organização de uma partícula

2.2.1. A definição de uma partícula

Uma partícula define-se com um ponto num espaço tridimensional, que reúne um conjunto de atributos dinâmicos e métodos associados. Esta completa um ciclo de vida: nasce, cresce e acaba por morrer. Ao longo deste ciclo de vida, podemos ajustar os parâmetros que influenciam este ciclo, desfrutando assim da possibilidade de criar vários efeitos diferentes, sugerindo que cada um tenha uma evolução ligeiramente diferente. Isto tudo acontece através do tempo. Apesar do seu comportamento ser único, estocástico e na maioria dos casos individual, expecto quando existem casos de interdependências, esta faz sempre parte de um conjunto de elementos semelhantes, que pretendem atingir um objectivo comum durante uma simulação.

Nos tópicos seguintes vamos descrever cada um desses atributos e os métodos de cada partícula, no entanto é necessário ter em conta que outros atributos ou métodos podem ser adicionados de modo a enriquecer as possibilidades na variação de comportamentos da mesma. Estes atributos representam um bom ponto de partida e devem ser suficientes para a maioria das aplicações, obviamente não conseguimos prever todas as situações que podem ser simuladas, muito provavelmente no futuro esta lista deverá crescer, alargando assim as capacidades do sistema. Tendo em conta a diversidade no tipo de sistemas que é possível criar nesta arquitectura, estes atributos são os comuns a todos os tipos de sistemas.

2.2.2. A estrutura interna

Ao criar uma partícula, necessitamos de decidir quais os atributos necessários para a controlar uma simulação do seu ciclo de vida. Uma partícula pode incluir os seguintes atributos:

- *Posição* – este será um dos atributos mais óbvios, precisamos de saber a posição da partícula no espaço para a podermos actualizar a sua posição após a interacção de forças e ter a capacidade de a representar graficamente.

- *Posição anterior* – para poder fazer outro tipo de efeitos e obter representações geométricas mais complexas, necessitamos de guardar a posição anterior.

- *Duração de vida* – para controlar o seu ciclo de vida, necessitamos de guardar o tempo que a partícula já viveu ou o tempo que resta de vida (pode também ser considerado como a energia restante da partícula). Este atributo é muito importante, porque na maioria dos casos poderá ser utilizado para manipular outros atributos.

- *Tamanho* – o tamanho é um atributo que pretende representar a área de representação gráfica de cada partícula. Podemos ainda ter outro atributo associado ao tamanho, e que visa guardar a forma como a partícula se transforma através do tempo.

- *Massa* – este atributo determina a quantidade e o efeito que forças externas têm sobre o comportamento de uma partícula.

- *Cor* – para melhorar a realidade dos elementos naturais que um dado sistema de partículas represente, é essencial que cada partícula tenha uma cor atribuída, no caso de utilização de texturas, a cor da partícula pode ser utilizada para fazer uma mistura da sua cor com a cor de uma textura, alargando assim a possibilidade de encontrar um efeito mais realista. A cor é algo que pode ser alterado ao longo do ciclo de vida da partícula, por isso podemos ter outra cor associada ao sistema, e ao longo do ciclo de vida efectuar uma interpolação linear entre essas duas cores.

- *Alfa* – este atributo está relacionado com o nível de transparência que cada partícula, e é utilizado nos efeito mais detalhados, como a simulação de fenómenos naturais: fumo ou fogo. Pode ser um atributo fixo ou um atributo que se altere com o ciclo de vida. Para atingir efeitos mais complexos é necessário adicionar outros atributos que controlem a forma como este atributo se altera.

- *Velocidade* – este atributo representa um vector, que guarda a velocidade e direcção actual de cada partícula. Este atributo vai ser influenciado por forças externas como o vento ou a gravidade.

- *Aceleração* – se uma partícula for capaz de acelerar, então este atributo vai influenciar a velocidade da mesma.

Apesar de alguns destes atributos possam parecer que pertencem exclusivamente as propriedades de cada partícula, isso não é verdade. A razão deste facto é que, a não ser que o atributo se altere durante o ciclo de vida ou exista uma variação no atributo de cada partícula quando é criada, não existe a necessidade de guardar esse atributo, devido ao desperdício de memória ocupada por cada e um desperdício de ciclos de instruções no processador para alterar este atributo em milhares de partículas, quando será muito mais fácil apenas guardar este atributo no sistema de partículas, caso ele seja fixo. Por isso é necessário ponderar o custo das decisões quando consideramos a concepção da base do sistema.

2.2.3. Métodos associados.

Para além dos atributos, ao planear o modelo de construção de uma partícula, a sua classe vai exigir métodos próprios. Estes métodos são criados para ajudar a organização da estrutura interna do sistema e ao mesmo tempo criar uma arquitectura mais limpa. Sendo assim, a classe de cada partícula vai possuir dois métodos, um de iniciação dos seus atributos quando um objecto deste tipo é criado. O outro método, bastante importante, é um método de actualização das suas propriedades, que leva como parâmetro o somatório das forças resultantes desse sistema.

2.3. A organização de um sistema de partículas dinâmico

2.3.1. Introdução

A modelação de fenómenos naturais apresenta um problema difícil de resolver. É necessário encontrar um método que tivesse a possibilidade de modelar objectos que não usufruíssem de limites bem definidos, e que não apresentassem superfícies bem definidas ou uma quantidade de brilho constante. Este método devia ter aptidão para criar superfícies irregulares, tanto na sua forma como na sua aparência. Também devia ter a característica de

manipular um conjunto de atributos que fossem capaz de reproduzir dinâmica e fluidez no movimento. Estes conjuntos de objectos não são rígidos, nem as suas transformações podem ser descritas como uma série de simples transformações afins, que são comuns nas representações de superfícies geométricas. Conservando estes princípios em mente, este capítulo tem como âmbito de definir por completo um sistema de partículas [1].

2.3.2. Sistemas de tempo real: parâmetros em função do tempo

Na área da computação, sistemas de tempo real, são sistemas que estão sujeitos a um constrangimento de tempo real. São considerados como sistemas que efectuam uma tarefa crítica. Nas simulações, o termo tempo real deriva da implicação que a computação levada a cargo pelo sistema, seja suficientemente rápida para que possa prosseguir a uma velocidade que permita a interactividade. Em sistemas de partículas os parâmetros físicos e visuais estão directamente relacionados em função do tempo. Estes sistemas são considerados como sistemas que são dependentes do tempo para criar efeitos realistas e necessitam de que os parâmetros variem em função do tempo através de amostras discretas.

2.3.3. O funcionamento de um sistema de partículas

Antes de partirmos para definições precisamos de entender o funcionamento básico de um sistema de partículas. Um sistema de partículas é um conjunto de elementos básicos, que formam conjuntos que não são estáticos, pelo contrário, a dinâmica é uma palavra de ordem nestes sistemas. Outro ponto de referência a ter em conta em sistemas de partículas é o facto que estes sistemas serem naturalmente caóticos. Isto significa que não executam um ciclo ou um caminho pré-determinado, cada atributo constituinte de cada partícula é formado a partir de um conjunto de dados que são introduzidos de uma forma limitada e casual, modificando assim o seu comportamento. Deste modo temos a certeza que cada partícula possui o seu ciclo independente, elevando o grau de realismo da simulação. Este elemento casual é chamado de processo estocástico, esta denominação resulta do facto que as propriedades estão submetidas

as leis do acaso, isto faz com que os resultados da simulação aconteçam de uma forma mais orgânica e natural [5].

2.3.4. A definição de um sistema de partículas

Um sistema de partículas define-se como um sistema que sugere a modelação de um objecto que é composto por um conjunto de primitivas simples. Este sistema funciona como um controlador destes elementos. Utiliza um grupo de regras que determinam a dinâmica, a morfologia e a representação, de acordo com o estabelecimento de algumas rotinas. Este sistema é essencialmente definido como um formalismo matemático usado para descrever fenómenos que respeitam as seguintes normas:

- Dinâmicos e dependentes do tempo;
- Altamente paralelizáveis com elementos pequenos e individuais;
- Complexos;

Estes sistemas são insensíveis ao contexto, significando que podem ser utilizados na modelação de diferentes situações. São sistemas que não implicam a especificação de casos de uso na definição dos mesmos. Ao mudar apenas alguns parâmetros, podemos responder uma quantidade diversa de problemas, seguindo as linhas gerais descritas anteriormente. A sua aplicação e definição pretendem alcançar um objecto dinâmico que seja apto a representar movimento e alterações na sua forma, assim a complexidade desejada emerge de uma forma natural. A representação de um sistema de partículas difere em três aspectos muito básicos utilizados na criação de objectos em imagens virtuais. Em primeiro lugar, um objecto não é representado por conjunto de superfícies geométricas com polígonos complexos ou parte constituintes dos mesmos, que poderão representar áreas limites de um dado objecto, em vez disso, pretende-se representar um volume. Em segundo lugar, um sistema de partículas não é uma entidade estática, as partículas mudam de forma e de aspecto ao longo do tempo, novas partículas são criadas ao mesmo tempo que outras são eliminadas. Em terceiro lugar, um

objecto representado por um sistema de partículas não é determinista, visto que a sua forma e aspecto não são completamente especificados. Em vez disso, este tipo de sistema tem a vantagem de ser um processo estocástico que é utilizado como um norma de criar, mudar a forma e a aparência de um conjunto de elementos.

Este método apresenta várias vantagens na tentativa de modelar fenómenos naturais. Partindo do princípio, que cada partícula é tratada como um único ponto, imediatamente este factor é vantajoso pois é mais fácil de manusear um ponto do que qualquer outra primitiva geométrica, sendo esta a representação mais simples que existe. Por isso, este facto é aproveitado porque o tempo que exige na sua computação é muito reduzido, podendo aumentar o número de cálculos sobre elementos deste tipo mais do que qualquer outro, produzindo assim uma representação mais complexa. A definição do modelo que pretendemos é progressiva e controlada pela definição de atributos. Assim, para obtermos um modelo detalhado e complexo não requer muita implementação. Devido ao facto de ser um sistema progressivo, um sistema de partículas tem a capacidade de ajustar o seu nível de detalhe, de modo a adaptar-se a determinadas situações de visualização, por exemplo, a medida que nos aproximamos do sistema, podemos aumentar o seu nível de detalhe. Este tipo de sistema trata as partículas como se fossem objectos "vivos", manipulando o seu ciclo de vida, permitindo assim a definição de formas complexas, dinâmicas e arbitrárias, algo que não se consegue atingir com outros métodos de concepção e manipulação de superfícies geométricas. Deste modo, as representações volumétricas apresentam uma proposta alternativa e viável para problemas que primitivas geométricas comuns não conseguem resolver.

2.3.5. Modelos de interacção entre partículas: sistemas locais e globais

Podemos dividir estes sistemas essencialmente em 2 grandes grupos, ao qual vamos denominar de sistemas locais e sistemas globais. A definição de sistema local consiste em sistemas de partículas que modelam o comportamento dos elementos individuais como se cada um fosse autónomo. Assim, as interacções entre partículas não são consideradas, e todos os cálculos são locais a cada partícula, daí a razão da sua designação. Em contraste, existem sistemas de partículas globais, que incluem partículas que interagem e reagem com outras partículas. Desta forma podemos definir distintamente quais os principais tipos de sistemas

que existem. Como veremos mais à frente, ambos os grupos apresentam diferentes vantagens e problemas que requerem técnicas específicas de programação, que se adeqúem ao tipo de sistema. Obviamente que um tipo de sistema global produz um comportamento muito mais complexo e interessante, mas esta riqueza não é gratuita. O cálculo necessário para considerar as interacções e influência entre partículas, aumenta drasticamente o custo computacional requerido. Por isso, sistemas de partículas globais devem ser manuseados com muito cuidado e ponderação. Esta arquitectura utiliza estes dois tipos de sistemas de modo a oferecer uma boa plataforma de utilização e futura extensão [16].

2.3.6. Estrutura interna

Cada sistema de partículas controla um agrupado de partículas, que de acordo com a dinâmica implementada, actuam como elementos que pertencem a um conjunto e que partilham atributos comuns. Cabe ao sistema a função de atribuir valores a este conjunto de atributos de modo a que, colectivamente, possam atingir o efeito desejado. Estes sistemas são compostos por um conjunto de atributos e um conjunto de métodos, que devem servir como as ferramentas de controlo destes elementos individuais.

2.3.7. Atributos associados

Estes sistemas são compostos por um conjunto de atributos que servem de controlo e manipulação destes elementos [11]:

- *Posição*: ao criar um sistema, este necessita de uma localização para determinar uma posição espacial onde as partículas vão iniciar o seu movimento. Este atributo é normalmente modelado como um único ponto representativo no espaço, mas nem sempre será o caso.

- *Forças*: adicionando uma ou mais forças externas para actuar no sistema de partículas pode criar novos patamares, aumentando o grau de realismo da simulação, portanto é

necessário ter pelo menos uma força vectorial no sistema. Deve-se evitar uma partilha de forças entre todos os sistemas criados, assim cada um terá uma simulação independente.

- *Velocidade*: caso o sistema apresente movimento, podemos incluir este atributo, assim, em cada quadro de desenho, podemos alterar a sua posição de um modo progressivo e realista sem proceder manualmente à sua alteração.

- *Numero máximo de partículas*: depois de seleccionar o tipo de partícula e definir as suas características, o próximo passo será escolher o número máximo de partículas que o sistema deve gerir. Neste passo, existem essencialmente dois problemas a considerar: a limitação da performance do sistema e o grau de complexidade do objecto que pretendemos simular. Por exemplo, ao simular uma nuvem de fumo, o número de partículas a utilizar deve coincidir com a densidade dessa mesma nuvem. Existem casos onde a aplicação poderá reduzir o número de partículas dinamicamente, devido a limitações de GPU ou porque a área ocupada pelo sistema no ecrã não revela uma necessidade de apresentar um nível de detalhe máximo. Limitações a nível de performance são umas das limitações mais simples de quantificar, e por isso, é necessário fazer vários testes de performance na aplicação, sendo um elemento chave na determinação do limite no número de partículas que podem ser simuladas, sem prejudicar a velocidade do sistema, tanto a nível de CPU como GPU. Estes testes de performance vão essencialmente impor um limite, caso contrário, o nível de detalhe de cada sistema pode rapidamente ficar fora de controlo e crescer exponencialmente, acartando o elevado risco de consumo de memória indeterminado, podendo em casos extremos levar a uma quebra total da aplicação.

- *Emissão de partículas*: a emissão de partículas é normalmente controlado por intervalo de tempo e determina o número de partículas emitidas nesse mesmo intervalo. O intervalo utilizado como tempo de referência é normalmente um segundo, mas este é apenas um valor, as aplicações podem escolher o intervalo que mais lhes convêm. De modo a manter uma emissão de elementos regular, o sistema necessita de guardar a diferença do tempo entre quadros de desenho, assim que o tempo máximo por emissão seja ultrapassado, o sistema deve reiniciar o valor para que um novo ciclo se inicie. Nem todos os sistemas ou todos os tipos de efeitos requerem este modo de controlo, mas um bom sistema tem um controlo geral sobre todos os aspectos. É necessário não confundir este atributo com o

máximo número de partículas do sistema ou a quantidade do tempo de vida das partículas. Caso as partículas estejam vivas por muito tempo, o sistema pode atingir o número máximo de elementos e a emissão de novos elementos vai ser obstruída. Este efeito pode ser o objectivo da simulação mas caso contrário é necessário a manipulação do sistema para obter os resultados pretendidos, configurando o controlo de emissão e o valor de vida das partículas.

- *Lista de partículas*: o principal atributo de um sistema deste tipo será obviamente uma lista de partículas, para que o sistema possa aceder as estas e comandá-las.

- *Conjunto de regras para atributos de partículas*: quando um sistema cria uma nova partícula, este deve inicializar os atributos consoante um grupo de regras associado a um conjunto de valores. O sistema necessita de saber quais são os conjuntos de valores válidos. Alguns atributos das partículas podem mudar consoante o tempo, por isso, pelo menos os valores iniciais devem ser conhecidos. Quando se inicializam os valores dos atributos de cada partícula, normalmente procede-se a uso de alguma variação e aleatoriedade para que estas adquiram algum realismo e não aparentem nem se comportem da mesma forma. Estes sistemas, excepto casos especiais, não têm que necessariamente proceder a uma modelação física exacta do mundo, na sua grande maioria o interesse fica por aproximar uma modelação que alcance bons resultados visuais. Por exemplo, a massa associada a cada partícula pode não corresponder a valor de uma medida representativa no mundo real, assim como as forças podem não corresponder a valores exactos de forças equivalente do mundo real. Outro bom exemplo desta situação é o facto de a gravidade pode não corresponder ao valor exacto de uma aceleração de $9.8m/s^2$, desde que atraia as coisas para baixo consoante um valor satisfatório. Estes valores e a sua exactidão dependem muito do tipo de simulação a realizar.

- *Estado actual*: na grande maioria isto implica simplesmente a activação e desactivação do sistema. Existem um número indeterminado de casos onde este controle é bastante prático, por exemplo, quando o sistema tem um tempo limite de acção, como uma explosão. Neste caso, passado um determinado tempo, é requerido que o sistema pare de emitir novas partículas. Outro caso, será quando o sistema está fora do campo de visualização, assim não existe razão nenhuma para continuar a actualizar as partículas,

nem razão para enviar os dados para o GPU em cada quadro. Para efectuar este tipo de controlo deve-se criar uma variável que permita a mudança entre os diferentes estados de cada sistema.

- *Modelação dos atributos gráficos*: a manipulação de atributos gráficos representa um elemento importante na tentativa de simulação de elementos reais. Estes sistemas recorrem a uma manipulação gráfica complexa que associada a uma dinâmica de movimento pretende atingir um elevado grau de realismo na simulação. Uma das formas mais conhecidas de manipulação é efectuada através do canal alfa. Cada sistema deve possuir o seu próprio modo de manipulação deste canal, por isso este atributo será específico a cada sistema. Existem outras formas de manipulação que exigem um planeamento mais complexo que pode representar um tópico para uma evolução futura desta arquitectura.

- *Representação gráfica*: cada sistema está associado a uma superfície geométrica que pretende representar graficamente cada partícula, seja esta um ponto, uma linha, um quadrado, etc. Esta informação deve ser guardada no sistema. Perante esta situação é necessário proceder a uma decisão de arquitectura: fazer um sistema geral de modo a que suporte os tipo de superfícies mencionadas ou deve-se criar novos objectos que derivam do sistema base de modo a representarem as diferentes superfícies? Será necessário todas estas formas? Não existe uma resposta correcta, tudo depende da necessidade e da preferência de cada um. A arquitectura implementada utiliza uma aproximação de herança, principalmente porque é um aproximação mais limpa e deixa espaço para futuras implementações e extensões alternativas que possam surgir.

2.3.8. Métodos associados

Um sistema de partículas necessita de realizar trabalhos em várias áreas, por isso, é necessário um conjunto de métodos que proporcionem uma boa organização interna do sistema, com uma boa interface de utilização. A seguinte lista enumera de alguns desses métodos utilizados:

- *Destrutor*: este método simplesmente liberta toda a memória que foi necessária alocar para cada sistema.

- *Inicialização*: este método procede à configuração do sistema, reservando a memória necessária para alocar todas as partículas, a lista de atributos das partículas também deve ser iniciada.

- *Actualização*: esta função vai receber como parâmetro a diferença de tempo que a aplicação demorou para efectuar o cálculo do quadro anterior e o quadro actual, denominado tempo delta. Esta função itera por todas as partículas de modo a realizar os cálculos necessários para cada uma, utilizando esta diferença de tempo para proceder à actualização. As actualizações são executadas nas propriedades físicas e gráficas. Este tempo também vai ser utilizado na geração de partículas.

- *Adicionar partícula*: este método adiciona uma nova partícula ao sistema, não necessita de alocar mais espaço de memória para fazer, deve encontrar uma partícula que está actualmente com o estado "não activa", alterar o seu estado para "activa" e reiniciar os seus atributos.

- *Remover partícula*: este método serve para alterar o estado de cada partícula, caso esta seja seleccionada para ser removida da simulação, este método deve actualizar o seu estado "activo" para "não activo".

- *Render*: o sistema deve saber qual a representação gráfica associada, portanto esta função necessita de percorrer todas as partículas, associando uma primitiva geométrica a cada uma.

- *Movimentação*: este método apenas altera a posição actual do sistema.

- *Obter/alterar força*: deve-se utilizar estes métodos para proceder a alteração ou simplesmente consultar o valor de uma dada força no sistema.

O conjunto dos métodos descritos constitui uma boa referência como ponto de partida, mas à medida que os sistemas se tornam mais complexos podem-se adicionar novos métodos e criar novos objectos. A arquitectura proposta utiliza estes métodos e outros mais específicos de acordo com as necessidades de cada tipo de sistema [11].

2.3.9. Dinâmica de partículas

Cada partícula, dentro de um sistema, move-se num espaço tridimensional, a sua posição espacial vai-se alterando através de um conjunto de forças. Este movimento deve-se à transição de posição. O movimento efectua-se matematicamente, adicionando o seu vector de velocidade a sua posição actual. Para adicionar mais complexidade a uma partícula, pode-se introduzir um vector de aceleração para modificar o seu vector de velocidade entre quadros. Ao adicionar este parâmetro o sistema pode agora fazer uma simulação com gravidade, porque o seu movimento agora irá ter o efeito de uma parábola em vez de uma linha recta. A estrutura de uma partícula pretende definir a maneira como vai ser simulada, o seu modo de comportamento dever ser uma mimica específica do fenómeno que ocorre durante o tempo. A escolha mais comum será a implementação de uma dinâmica que pretende simular movimentos correspondentes a fenómenos do mundo real. Algumas simulações podem ser consideradas como exemplos mais complexos, como por exemplo, a simulação de chuva. Este tipo de simulação requer que os objectos possuam um movimento bastante rápido. Recorrendo a cálculos de física temos:

$$P = pi + v * \Delta t \qquad (1)$$

Esta equação pode ser reescrita da seguinte forma:

$$dP = v * \Delta t \qquad (2)$$

Onde dP representa a mudança de posição, v é a velocidade constante durante um intervalo de tempo e Δt (em segundos) é a diferença de tempo ao qual queremos avaliar a posição correspondente a este intervalo. Neste caso estamos a considerar que a velocidade é válida em todos os intervalos de tempo, esta generalidade é falsa. Estamos a partir de uma amostra e a generaliza-la para um intervalo de tempo correspondente. Esta prática faz deste

tipo de simulação denominada de integradores de Euler, que são muito instáveis em sistemas mais complexos. Mas na prática, para sistemas de partículas, este é o método escolhido devido à sua simplicidade de cálculo e elegância. Necessitamos de substituir Δt pelo diferencial de tempo consecutivo entre o ciclo da simulação, e utilizar a velocidade como parâmetro. Em seguida recorremos à utilização da física de Newton para integrar a equação de movimento. Uma força é toda e qualquer entidade que permite alterar o estado duma partícula. As forças baseadas na física são já nossas conhecidas, pois ocorrem frequentemente na natureza e provêem dos diferentes elementos naturais. Alguns exemplos destas forças são: gravidade, vento, entre outras. Uma força, ou um conjunto de forças f, determina o comportamento do sistema. Tipicamente podem existir três tipos de forças a actuar sobre um conjunto de partículas:

1. Forças partícula-partícula (ex: repulsão ou atracção entre partículas)
2. Forças aplicadas ao sistema (ex: vento, gravidade)
3. Forças partícula-objecto (ex: choque entre uma partícula e um objecto físico)

Finalmente pode dizer-se que o estado duma partícula é obtido através de métodos numéricos, que permitem obter uma aproximação ao conjunto das funções e equações diferenciais que regem o sistema. Assim, cada interacção permite obter um novo estado para o sistema de partículas. A segunda lei de Newton diz-nos que uma partícula sujeita a uma força f, com uma dada massa m, ganha uma aceleração a, dada pela seguinte relação:

$$f = m * a \qquad (3)$$

Uma das consequências mais imediatas da física de Newton é que, uma partícula tem o seu estado completamente definido pelo vector velocidade e pelo vector aceleração. Convertendo temos:

$$a = f/m \qquad (4)$$

Mas na realidade, a aceleração é a segunda derivada de uma posição:

$$a = d^2x/dt^2 \qquad (5)$$

A primeira derivada de uma posição, é o vector de velocidade:

$$\frac{d}{dt}\overrightarrow{P} = \overrightarrow{v} \qquad (6)$$

A primeira derivada da velocidade é aceleração, confirmando a equação 5:

$$\frac{d}{dt}\vec{V} = \vec{a} \qquad (7)$$

Estas equações estabelecem as relações básicas entre cinemática (aceleração) e dinâmica (forças). Podemos assim definir a qualquer momento a posição e a velocidade da partícula. A equação 5 pode ser reescrita como um par de equações de 1ª ordem:

$$v = \Delta x \,/\, \Delta t \qquad (8)$$

$$a = \Delta v \,/\, \Delta t \qquad (9)$$

Desta forma, podemos representar as forças e calcular as acelerações entre elas, utilizando os diferenciais de tempo e os integradores de Euler. Este é o modelo de movimento implementado na configuração de um sistema de partículas mais simples, o tipo1. Uma futura funcionalidade atraente será a introdução de perturbação numa dada força ou conjunto de forças. Esta perturbação é conhecida como uma rotina muito popular, o ruído, recorrendo a uma técnica denominada de "Perlin Noise". Este método é capaz de gerar ruído num espaço 3D (ou em qualquer outro espaço), permitindo simular padrões de movimento [7][16]. A introdução e funcionalidades oferecidas pelo motor de física podem ser consultadas no anexo A.

2.3.10. Grupos de forças

Actualmente os sistemas de partículas implementados, no que diz respeito a forças aplicadas ao sistema, mais conhecidas por forças externas, só têm em consideração a gravidade, que é representada por um vector e representa a sua aceleração, com significado de metros por segundo ao quadrado, m/s^2. No futuro, seria uma boa extensão dos sistemas a introdução novas forças na simulação, elevando a dinâmica para um patamar bastante atractivo.

2.3.11. Actualização de partículas

Um sistema de partículas só poderá funcionar correctamente se a etapa de actualização estiver implementada correctamente e de modo eficiente, para que possa suportar o comportamento do sistema a uma velocidade que permita a interactividade. Esta etapa deve apresentar uma aproximação cuidada e estruturada, devido ao elevado número de iterações.

2.3.12. Extinção de partículas

Quando uma partícula é criada, atribui-se um tempo de vida que decresce ao longo do tempo que a aplicação demora entre a geração de quadros consecutivos. Depois de ter completado o seu ciclo de vida, esta partícula deve ser removida da simulação, para que o sistema possa dar vida a uma nova partícula. Existem outros mecanismos que podem tomar a decisão de eliminar as partículas da simulação, devido ao acontecimento de uma acção predefinida, como o uso de um determinado limiar máximo ou mínimo ligado a energia da mesma, ou uma colisão com um objecto. Uma partícula que se dirige numa determinada direcção (ex: ecrã) ou uma determinada distância (ex: volume de visualização), também pode ser eliminada, outras condições podem ser introduzidas, como o facto de não estarem a contribuir para a imagem final. Estes mecanismos podem ser utilizados na eliminação de partículas da simulação porque consideram, mediante os limites impostos, que estas já se encontram fora da região de interesse. Durante este processo é desaconselhado o recurso a novas operações sobre a memória, devido ao elevado consumo de tempo que é necessário para serem realizadas e também ao facto de estarem a contribuir para uma sucessiva fragmentação. Este problema pode ser facilmente contornado ao optar por uma reiniciação do elemento assim que este seja seleccionado para ser removido. Desta forma podemos reciclar as partículas.

2.3.13. Modelação visual de interacções entre partículas

Uma vez que todos os parâmetros necessários para a visualização tenham sido calculados em cada quadro, necessitamos de representar cada partícula graficamente. A

representação gráfica deste tipo de sistemas é complexa e contém um conjunto semelhante de propriedades gráficas, como o uso de polígonos ou a utilização de superfícies curvas, manipulação de cor, entre outros. A manipulação ou o ajuste dos parâmetros gráficos correspondentes a cada partícula isoladamente não são suficientes para a criação de efeitos realistas. O controlo da interacção visual entre partículas é muito importante e associada à simulação de movimento, pode levar a resultados visuais muito convincentes. Um dos problemas a enfrentar nesta etapa situa-se no facto que muitos elementos vão obstruir a visualização dos outros que estão ocultos do ponto de vista da profundidade do ecrã, expecto o caso onde a suas posições sejam muito dispersas. Por isso, a cor de cada uma poderá contribuir para um somatório no efeito final. A escolha da interacção gráfica entre partículas está relacionada directamente com as características físicas do conjunto. Assim, podemos ter partículas que são transparentes ou opacas, podem absorver ou emitir luz, podem-se apresentar num modo homogéneo ou heterogéneo, correspondendo a áreas de maior ou menor densidade, onde podem aparentar diferentes características de iluminação consoante a sua posição no conjunto. Portanto este tipo manipulação tem em conta a interacção com o ambiente em que estão inseridas, podendo constituir uma obstrução ou modelação com o cenário envolvente. Muitas das interacções visuais entre partículas podem ser efectuadas utilizando uma manipulação da componente de transparência, associada à cor ou manipulando a componente de profundidade. Assim, para atingir uma modelação visual satisfatória, é preciso uma implementação correcta e determinista, aplicando técnicas apropriadas para alcançar o efeito final desejado. Outro problema a ter em conta é o facto que estes sistemas encontram-se inseridos em ambientes onde existem outras primitivas geométricas, onde provavelmente vão existir intercepções entre ambos. No sistema implementado, partimos de dois pressupostos, visto que temos dois tipos de sistemas que diferem neste aspecto, sistemas que oferecem detecção de colisão com primitivas ou superfícies geométricas arbitrárias e sistemas em que não se efectua a detecção de colisão.

2.3.14. Criação de partículas utilizando um emissor

Um dos métodos mais importantes a implementar será o controlo sobre a forma como as partículas são criadas. Este controlo só poder ser obtido se criarmos um emissor, que é o elemento responsável por inicializar os parâmetros de cada partícula. Para que o sistema

usufrua de um comportamento minimamente interessante, os valores dos atributos das partículas devem ser iniciados aleatoriamente. Desta forma, o movimento de cada partícula aparenta ser mais dinâmico e representa um ponto importante na visualização do sistema. Existe uma variedade de emissores que são frequentemente usados. Um desses emissores, e provavelmente o mais conhecido, será o emissor pontual, que cria todas as partículas a partir de um único ponto no espaço, na tentativa de simular uma explosão. Mas as explosões, na realidade, não nascem a partir de um único ponto. As explosões possuem um volume e uma forma associada, portanto, as partículas são geradas numa área envolvente ao objecto que explodido. Este ponto pode ser perturbado de modo que as partículas sejam criadas numa área envolvente e não exactamente na posição que o ponto ocupa. Para atingir este objectivo necessitamos de um método que tenha a possibilidade de introduzir alguma aleatoriedade, para que o resultado final não apresente regularidade e de alguma forma algorítmico. Este método deve descrever a adição de quantidades controladas de ruído, para reduzir o aspecto artificial da simulação. Emissores podem ter outros formatos, como a utilização de um plano como fonte de criação de partículas, esta alternativa é utilizada na simulação de chuva, granito ou neve, existem diferentes formatos de emissores que podem ser implementados. Imaginemos que pretendemos efectuar uma simulação de chuva, muito provavelmente não será necessário encher todo o cenário com milhares de gotas individuais. Não há razão nenhuma para criar gotas em zonas ou distâncias que não vão ser observadas pelo utilizador. Neste caso, é muito vantajoso se houvesse um caminho que conduzisse a criação das gotas apenas onde o utilizador as possa observar com mais detalhe, ou seja, directamente na sua frente. Este tipo de emissor é designado como um emissor de ecrã, que serve para produzir efeitos apenas baseados no espaço do ecrã. As partículas são criadas em qualquer lado mas estão sempre referenciadas á janela de visualização da aplicação. [16]

Este conjunto de funcionalidades é bastante atraente para adicionar ao sistema de partículas, pois aumenta a capacidade e o número de efeitos visuais que o sistema pode atingir. Este trabalho desenvolvido oferece a possibilidade de criar partículas através de pontos, linhas e usando uma *bounding box*. Uma evolução deste sistema pode ser feita ao introduzir novas formas de emissores, alargando assim as possibilidades de simulações que se podem efectuar.

2.3.15. Efeitos especiais

Os efeitos configuram-se através da escolha dos atributos que estabelecem comportamentos que permitem a simulação de um efeito particular. Mas isto requer algum trabalho e experimentação para obter o resultado desejado. Existe uma grande diversidade de efeitos que podem ser simulados, o problema situa-se na quantidade de variáveis que podem fazer parte cada um. Por isso, de seguida estão enunciadas algumas considerações a ter quando queremos criar um novo efeito:

- *Reflectir*: parte dos atributos das partículas servem para representar propriedades físicas do mundo. Deste modo, deve ser usado algum senso comum como ponto de partida para a implementação.

- *Física*: para obter uma dinâmica de movimento mais aproximada do mundo real, é necessário considerar o uso de uma modelação de física mais avançada. Obter conjuntos de movimentos complexos corresponde a uma grande parte do desafio.

- *Outras fontes*: adquirir novas ideias a partir de outros trabalhos, existem alguns exemplos de sistemas de partículas espalhados pela internet, deve-se fazer uma pesquisa e reunir as principais ideias que estes apresentem, experimentar novos conceitos pode dar origem a novos efeitos ou sistemas.

2.3.16. Controladores de eventos: scripts

Este projecto não tem este modo implementado mas é uma boa referência para uma futura evolução. Para obter uma melhor utilidade de um sistema de partículas temos de adicionar umas propriedades extra, mais concretamente a introdução de paradigma que proporcione mais flexibilidade – um script. Isto significa que, podemos obter um sistema que apresente uma maior flexibilidade e que ao mesmo tempo mantenha o código limpo e rápido. Uma aproximação á introdução deste paradigma e consequentes enriquecimentos poderiam ser feitos caso a caso. Mas este processo pode levar vários dias de desenvolvimento, e

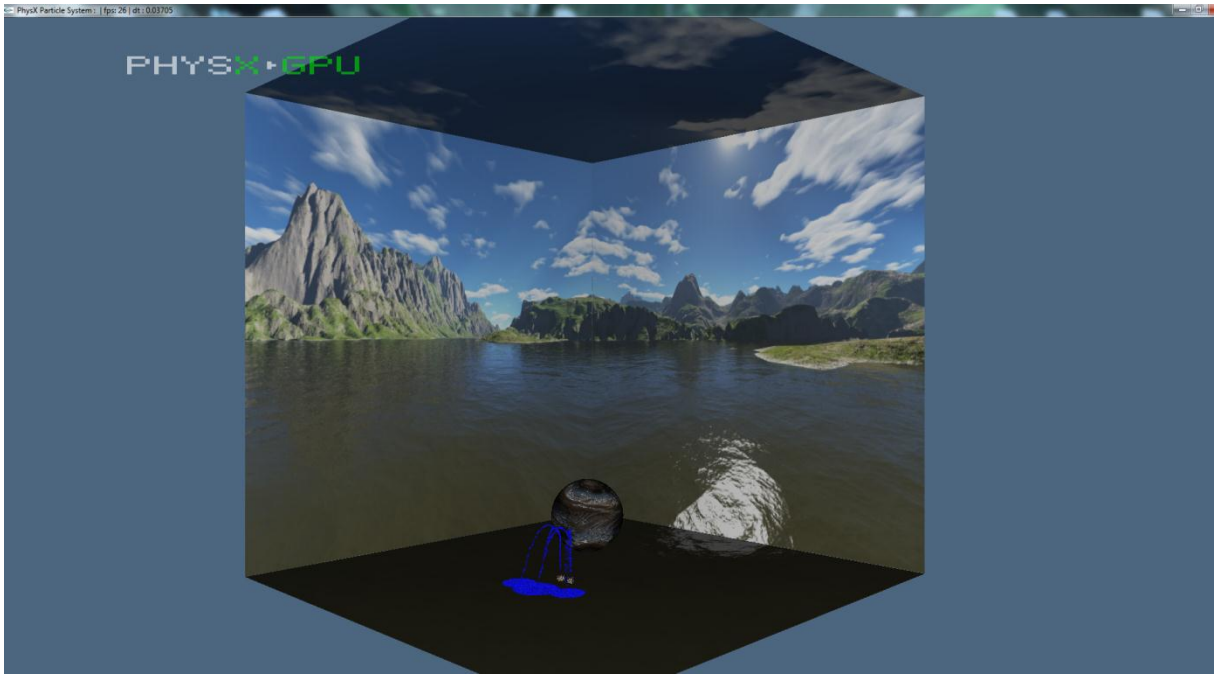
introduzir algumas limitações no sistema. Em vez disso, pretende-se obter um sistema mais complexo com um conjunto de propriedades adicionais. O sistema pode ter a possibilidade de obter uma inteligência acrescida, pois desta forma, tem a capacidade de variar no tempo, assim como variam as partículas. Pretende-se o seguinte, anteriormente fala-mos de uma cor inicial e uma cor final de uma partícula, e uma interpolação linear entre estes acontecimentos. Isto até pode parecer uma boa política, mas na realidade não é muito flexível, se por outro lado pudéssemos ter um método em que tivesse a característica de introduzir um número arbitrário de cores, cada uma associada com um tempo particular e o sistema respeitar essas mudanças de cor, sem dúvida que esta aproximação oferece mais flexibilidade. Ao aplicar a mesma filosofia com outras propriedades do sistema até aparenta ser uma boa solução, mas ao examinar mais aproximadamente a sua implementação, rapidamente chegamos a conclusão que iriamos necessitar de lidar com vários vectores e que estes iriam ser uma duplicação dos atributos do sistema. Ou seja, o que parecia como uma boa solução inicial acaba por se tornar um pesadelo. É necessário recorrer a outro método para atingir a flexibilidade que ansiamos, por isso é necessário outra abordagem do problema. Em vez de termos um vector que guarde todas as mudanças de cada atributo, necessitamos de um vector que guarde todas as mudanças do sistema, recorrendo a eventos e sequências de eventos. Esta nova aproximação pode criar uma grande flexibilidade no sistema. Um evento é essencialmente uma mudança das propriedades do sistema de acordo com o tempo. As sequências de eventos são um conjunto de vectores que identificam as mudanças de um dado sistema e do comportamento das partículas desse sistema. Ao implementar esta funcionalidade podemos criar um sistema capaz de encarar várias situações e mais do que isso, transforma um sistema que de certo modo era limitado, adquirindo a capacidade de enfrentar uma cadeia de acontecimentos, sem que seja necessário criar um outra nova versão do sistema. [16]

2.3.17. Hierarquia de partículas

Esta investigação não tem implementado este modo, mas vai ser abordado porque seria uma boa funcionalidade para uma futura evolução. A dinâmica do mundo real, em muitos dos casos, resulta de uma combinação entre vários estados de modo a compor uma ilusão complexa. Por exemplo, uma explosão criada por um míssil exige fogo a sair dos escapes do míssil, de seguida surge uma explosão em que são libertadas grandes quantidades

de fumo e posteriormente é gerado vapor de água. No meio desta acção toda, não podemos esquecer que existem milhares de detritos e partículas resultantes do objecto destruído, que foram separados da estrutura principal e que viajam a grandes velocidades com trajectórias muito diferentes. Como podemos caracterizar este acontecimento? Poderá ser um sistema de partículas? A resposta é bastante clara, não. Porque temos de lidar com vários tipos de partículas e cada grupo de partículas está a ser orientado por um sistema independente que possui uma dinâmica própria. Este acontecimento pode ser visto como um grupo: um sistema de sistemas de partículas. É necessário criar um sistema que possa agrupar e criar uma cadeira de sistemas de partículas, capaz de ordenar sequências de acontecimentos. Estes sistemas podem definir conjuntos de acções a executar e decidir o comportamento do sistema. Como exemplo, esta cadeia de acções durante uma simulação de uma explosão pode ser considerada como algo que após a existência de uma destruição, activa um sistema de fumo. Este tipo de arquitectura é utilizado na simulação de efeitos reais. Assim, uma hierarquia de partículas é um mecanismo que efectua uma manipulação e controlo de um sistema de partículas. Este sistema também abre uma porta a novas possibilidades, como a criação de um sistema de partículas em que as partículas são em si outro sistema de partículas. Neste caso, quando um sistema de partículas "pai" é transformado, todos os sistemas descendentes e as suas partículas associadas também são transformadas. A estrutura de dados que é utilizada na implementação deste tipo de hierarquias são árvores binárias. Este tipo de hierarquias é utilizado para estender o controlo global de simulações mais complexas, que são compostas por muitos sistemas de partículas em cadeia. É possível agrupar conjuntos de sistemas de partículas em que o sistema "pai" pode agrupar todos estes sistemas num conjunto e assim controlar os sistemas de maneira global, influenciando os seus movimentos e aparências. [16]

3. Arquitectura para sistemas de partículas dinâmicos



3.1. Introdução

Quando se ambiciona obter performance em tempo real e ao mesmo tempo um distinto grau de realismo, a elevada complexidade no detalhe e comportamento dos objectos modelados, sugerem claras vantagens e desvantagens. Por este motivo, é necessário identificar os vários problemas que afectam a performance, visto que muitas das técnicas existentes penalizam o realismo da simulação em troca de performance, para manter a interactividade da simulação. Devido à diversidade de situações que este tipo de sistemas pode simular, é um risco elevado declarar argumentos definitivos sobre o modo de implementação que estes tipos de sistemas devem seguir. No entanto, poderá ser útil a descrição de algumas técnicas e regras de conduta geral sobre estes sistemas, tendo em conta que o uso de qualquer conduta ou técnica pode ser contornado por causa de excepções. Estas técnicas são apenas diferentes caminhos que podemos tomar na elaboração de cenários reais. A produção de uma realidade extrema requer uma atenção especial a cada pormenor no cenário. A utilização de um grande conteúdo de realismo será mais favorável onde o utilizador focaliza mais atenção.

3.2. Objectivos gerais da arquitectura

A arquitectura proposta teve objectivos específicos para a sua definição, esses objectivos estão nomeados de acordo com a sua importância para o resultado final [7]:

Eficiência em tempo real – o objectivo principal da arquitectura é de realizar simulações de sistemas de partículas em tempo real, que apresentem uma dinâmica complexa de movimento. Isto requer que os cálculos sejam executados de forma eficiente de modo que o CPU tenha possibilidade de proceder a outros cálculos que sejam necessários.

Flexibilidade – a arquitectura deve ser criada de forma que o utilizador tenha a liberdade de criar e adicionar uma série de efeitos diferentes.

Extensibilidade – a arquitectura proposta deve possibilitar da sua extensão, como resposta a diferentes especificações que cada aplicação possa exigir.

Independência dos parâmetros – a concepção de uma nova extensão deste trabalho pode ser bastante grande, por isso a arquitectura deve disponibilizar a maior quantidade de parâmetros, para que uma extensão não apresente limitações, ou seja, todos os atributos devem estar ao dispor do utilizador para que possua a liberdade total de escolher quais os que pretende alterar ou ter acesso.

Abstracção e equipamento específico – a estrutura criada deve ser uma verdadeira abstracção da funcionalidade, de modo a que uma aplicação funcione na sua integridade caso adquira uma integração desta arquitectura, os sistemas devem trabalhar de modo independente mas também devem fornecer canais de ligação directos, para que possam comunicar.

Instrução fácil – este sistema deve ser criado com o intuito de fornecer uma interface que proporcionasse uma aprendizagem fácil, e por isso, houve a consideração de criar uma estrutura que fosse elegante, ao mesmo tempo que limitasse o número que falhas que pudessem ocorrer na chamada a funções internas da arquitectura.

3.3. Requerimentos

Os sistemas de partículas mais avançados podem resultar em grandes quantidades de código. Cada sistema tem um comportamento único, por isso existem inúmeras possibilidades de parametrizar o seu comportamento. Há diversas maneiras de influenciar um sistema em qualquer momento da sua utilização, portanto é necessário introduzir esta flexibilidade dentro do sistema e da arquitectura. É bastante claro que cada sistema vai ser muito específico na concretização de certas tarefas. Esta afirmação anterior leva-nos a outra questão, e se quisermos controlar um sistema tão específico que a flexibilidade concebida não permite a sua concepção? Nestes casos muitos específicos, não existe outra possibilidade a não ser a concepção manual desse mesmo sistema e integra-lo na arquitectura existente, de forma a realizar realizando o seu trabalho sem impedir o funcionamento geral dos outros sistemas. Devemos considerar a capacidade de integração da arquitectura criada em outras arquitecturas existentes, o seu funcionamento não deve entrar em conflito ou impedir acções realizadas por outros.

3.4. Performance

3.4.1. Introdução

Qualquer equipamento específico tem a uma limitação de performance, por isso, os objectivos do sistema deve ser limitados. Em alguns casos, se tivermos uma análise e um planeamento cuidado do sistema que vamos criar, podemos aproveitar a capacidade que os equipamentos disponibilizam. Existem várias técnicas que podem ser utilizadas para obter maior velocidade na execução os cálculos, criando assim a possibilidade de conceber sistemas de partículas complexos. Esta secção está destinada a descrever algumas técnicas que podem ser utilizadas para aumentar a riqueza e a qualidade dos resultados. Apenas um conhecimento profundo sobre o sistema a criar pode ajudar na determinação as técnicas que o podem beneficiar.

3.4.2. Manuseamento da memória

Como já foi referido anteriormente, estes sistemas podem baixar significativamente a velocidade da aplicação. Um dos factores que contribuem para este acontecimento deriva da ocorrência de "picos" de performance, que provêm do mau manuseamento de memória. Uma das optimizações a implementar nestes sistemas é baseada na observação das operações sobre a memória, que são operações demorosas. Assim, chamadas às funções de manuseamento de memória dentro de ciclos devem ser evitadas o mais que possível. Existem mecanismos de manuseamento de memória bastante eficientes, mas mesmo estes apresentam uma sobrecarga significativa para cada operação. Estes mecanismos fazem uma procura dos blocos de memória que se encontram disponíveis e seleccionam o mais adequado. As operações de memória associadas a cada partícula acartam um impacto significativo na performance. Especialmente em sistemas com grandes números de partículas e que possuam, por sua vez, uma duração de vida muito curta. Uma alternativa para o uso de operadores *new* e *delete* deve ser encontrada, para contornar este custo. A aproximação mais fácil ao problema será o uso de reciclagem, logo que cada partícula seja seleccionada para ser removida ou introduzida na simulação, deve-se actualizar o seu estado, como viva ou morta, utilizando uma variável booleana. Desta forma, as rotinas de reservar e libertar memória podem ser esquecidas. Não

necessitamos de reservar ou libertar a blocos de memória constantemente. Em vez disto, as partículas estão colocadas nas mesmas posições mas apenas com o seu estado actualizado. Usando esta técnica, o sistema pode conter um número de partículas constante, evitando as chamadas desnecessárias aos operadores *new* e *delete*.

3.4.3. Indexação espacial

A indexação espacial permite-nos fazer uma busca rápida sobre localizações espaciais. O cálculo da distância de um ponto a um conjunto de objectos ou selecção de pares mais próximos num determinado espaço, podem agravar consideravelmente o sistema. Esta técnica é mais empregada quando estamos a utilizar um sistema global e permite um aumento significativo na performance. Um sistema global com efeitos globais apresenta no pior cenário um custo nas operações de $O(n^2)$, onde n, é o numero de partículas. Este custo está relacionado com o facto de cada partícula ser testada com todas as outras, devido as interdependências. Portanto uma indexação espacial permite a busca de relações entre partículas numa vizinhança, podendo reduzir o custo para $O(n * k)$, onde k é um número constante que determina a quantidade espacial.

3.4.4. Sistemas com nível de detalhe

Os sistemas de partículas possuem cerca de centenas ou milhares de elementos, e cada um destes elementos requerem um conjunto de actualizações dinâmicas. Os métodos utilizados para executar o seu desenho também podem apresentar um custo significativo. No entanto é possível obter um aumento significativo de performance recorrendo a utilização de um paradigma de nível de detalhe (LOD). Uma nuvem de fumo que é modelada por partículas pode ser observada a uma distância da sua posição, esta situação pode ser aproveitada para evitar alguma computação e mesmo assim obter um efeito realista. Quando nos aproximamos, podemos aumentar o número de partículas gradualmente e alguma interacção extra pode ser aplicada ao sistema para obter o realismo desejado. A performance ganha poderá compensar pequenos artefactos que o sistema possa apresentar. Esta técnica, de uma maneira geral pode funcionar em duas áreas distintas, em termos de nível de detalhe. A primeira será uma clara

optimização do código responsável pelo desenho gráfico do sistema. Em segundo lugar podemos optimizar os métodos de actualização de partículas. A optimização no código de desenho pode ser bastante difícil. As velocidades das placas gráficas existentes no mercado sugerem que os métodos de desenho não apresentam grandes atrasos nos sistemas. Por outro lado, a actualização das partículas requer um consumo de operações bastante significativo. A concentração de performance deve-se situar mais nesta área. A optimização do método de actualização em termos de nível de detalhe é relativamente simples. Necessitamos apenas de definir o que vai variar, dependendo da distância, de uma resolução e como vai variar. A técnica mais fácil para realizar esta tarefa, será a criação de uma variação do número de partículas do sistema. Desta forma, podemos proceder a uma interpolação desde uma resolução mais baixa até uma resolução máxima, dependendo da distância ao sistema. Assim, optimizamos sistema de forma ideal, convergindo o seu nível, diminuindo ou aumentando o detalhe. A única desvantagem desta técnica situa-se no facto de requerer que a velocidade de deslocação do utilizador seja sempre inferior a velocidade do aumento de detalhe. Outra técnica para a optimização do sistema, envolve a simplificação dos cálculos internos efectuados a cada partícula. Alguns sistemas podem conter cálculos com equações bastante exaustivas de modo a assegurar uma visualização correcta. Portanto alguns destes cálculos também podem ser desactivados, poupando assim, ciclos de processador.

3.5. Criar a estrutura de dados

3.5.1. O objecto: partícula

3.5.1.1. Hierarquia

A definição da estrutura de dados correspondentes a cada partícula, significa responder essencialmente a uma questão: o que estamos a modelar? fogo? água? fumo? O problema situa-se no facto que diferentes fenómenos requerem diferentes parâmetros, ou seja, diferentes tipos de sistemas e diferentes versões do mesmo tipo de sistema requerem parâmetros próprios. Tendo em conta que a arquitectura proposta apresenta 3 tipos de sistemas diferentes, significa que vamos obrigatoriamente trabalhar com 3 tipos de partículas diferentes, que diferem nos seus atributos e métodos. Portanto a resposta para o problema inicial encontra-se na criação de objectos hereditários que especifiquem as necessidades de cada tipo de sistema.

3.5.1.2. Definição geral

A escolha da estrutura de dados mais adequada a cada partícula, implica uma selecção de parâmetros rigorosa que apresente uma resposta suficiente para a maioria das situações. De tal modo que não torne a estrutura demasiado pesada. Outro aspecto importante a considerar na especificação dos parâmetros será devido á qualidade da simulação. Todos os parâmetros declarados devem ter um propósito de utilização de modo que o aumento de memória requerido para cada partícula seja justificado. Por isso, a utilidade de cada parâmetro deve indicar a funcionalidade de controlo sobre cada partícula. Assim, é necessário encontrar um meio-termo que satisfaça a maioria das necessidades requeridas sem exagerar. Na elaboração deste objecto, não iniciamos um construtor especializado, porque não queremos quaisquer valores originais no momento em que precisamos de instanciar este objecto. Estes valores vão variar de acordo com cada tipo de sistema. As escolhas dos parâmetros podem-se agrupar em 2 conjuntos:

- Parâmetros associados com a dinâmica (físicos)
- Parâmetros associados com o aspecto visual (gráficos)

Os parâmetros de dinâmica devem ser suficientes para a simulação de movimentos. A análise do movimento a simular é essencial para obter uma compreensão dos parâmetros necessários para realizar esta tarefa. Uma boa recolha fonte de informação será obtida na leitura de livros de física, em especial os que mencionam artigos sobre a dinâmica entre partículas ou objectos rígidos. É essencial adquirir uma compreensão e visão generalizada da importância e da contribuição que cada força pode ter sobre a dinâmica do movimento. Cada partícula deve guardar o somatório de forças que actuam sobre ela, os tipos de forças devem ser guardados no sistema. Por exemplo, não faz sentido nenhum que cada partícula guarde a gravidade, que será uma constante e que é global a todas as partículas num dado sistema. Uma vez se determinaram os parâmetros necessários para a simulação de movimento, necessitamos de especificar quais vão ser os parâmetros que vão influenciar a representação gráfica de cada partícula. Neste campo as possibilidades são inúmeras, por isso cada tipo de sistema vai necessitar de parâmetros específicos. Realizou-se um estudo prévio sobre o conjunto de técnicas que vão ser utilizadas para a manipulação visual de cada partícula, para agrupar o

conjunto de parâmetros utilizados por estas técnicas e de modo que estas manipulam o aspecto visual. A hierarquia dos tipos de partículas criadas pode ser vista pelo seguinte esquema:

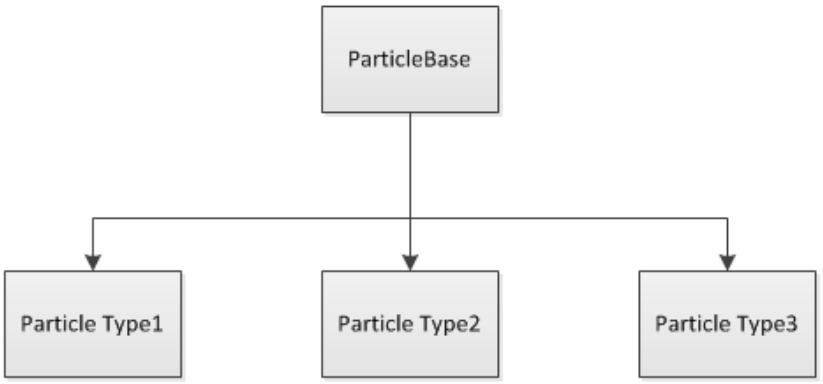


Figura 1 – Hierarquia que relaciona e organiza os diferentes tipos de partículas.

Para definir a hierarquia dos objectos que representam as partículas, começamos por definir um objecto abstracto, denominado ParticleBase, do qual novos objectos derivados de partículas possam ser especializados. Desta forma, estes objectos partilham o que é comum entre eles e especializam as suas diferenças. Os objectos de partículas derivados pertencem ao um determinado tipo de sistema e devem particularizar os atributos de acordo com a sua funcionalidade perante o sistema que se inserem.

3.5.2. O objecto: sistema

3.5.2.1. Hierarquia

Este trabalho analisou a implementação necessária para diferentes tipos de sistemas de partículas. Esta análise conduzio a exploração de dois caminhos essenciais. Estes dois caminhos existentes são aptos a para utilização destes sistemas em dois cenários diferentes. Por um lado, quando o âmbito de utilização destes sistemas é bastante abrangente e complexo deve-se usar uma hierarquia de objectos, portanto é criado um objecto abstracto de base, ao qual em seguida derivam-se objectos filhos que especifiquem os diferentes tipos de sistemas. Estes objectos filhos são implementados usando uma herança de um objecto abstracto. A utilização desta técnica permite-nos agrupar todos os tipos de sistemas em vectores separados, possibilitando o seu acesso de forma linear. Cada sistema poderá ser

ligeiramente diferente, devido á diferença na utilidade que pretende simular. Estes detalhes devem estar escondidos do programador, que acede a estes sistemas através de uma interface comum. Outra filosofia explorada, foi a criação de um sistema bastante complexo, onde o tipo de sistema requerido, não será nada mais do que uma derivação da configuração de parâmetros inseridos nesse mesmo objecto. Esta opção leva a que os parâmetros constituintes do objecto sirvam como uma definição para quase tudo, visto que todos os diferentes sistemas de partículas vão estar incluídos dentro desse mesmo objecto. Esta prática poderá envolver uma quantidade significativa de valores booleanos para activar ou desactivar determinadas funcionalidades. Uma vantagem desta segunda aproximação será obviamente a sua simplicidade, há muitos sistemas deste tipo podem ser implementados recorrendo a um código central, utilizando um controlo de comportamentos. Existe no entanto algumas desvantagens na utilização deste método, provavelmente a mais importante será a degradação na performance do sistema, devido aos inúmeros controlos introduzidos na tentativa de modelar todos os diferentes sistemas. Outro problema situa-se na grande confusão de código, o que poderá gerar uma complicação acrescida para futuras extensões do sistema. Claramente estes dois caminhos apresentam vantagens e desvantagens, e a escolha mais acertada deverá servir como resposta ao que queremos desenvolver. Ou seja, se quisermos criar sistemas que apresentam uma natureza muito distinta e a partilha de código é pouca ou nenhuma, ou se por outro lado queremos sistemas que podem reciclar uma grande parte do código principal e que podem ser implementados num só ficheiro, acartando consigo uma enorme quantidade de parâmetros. A escolha será dependente das diferentes utilidades que o sistema pode tomar. Este trabalho escolheu uma implementação por hierarquia.

3.5.2.2. Definição geral

Este objecto é o coração da arquitectura, a actualização das partículas, a definição de formas geométricas e os valores a aplicar as partículas encontram-se dentro deste, assim como todos os outros tópicos abordados no capítulo 2 quando introduzimos uma definição generalizada de sistemas de partículas. A maioria das funções devem ser declaradas como virtuais, para que este objecto alcance a possibilidade, caso seja necessário, de proceder a uma extensão hierárquica para outros objectos filhos, podendo assim criar diferentes versões do mesmo sistema. Provavelmente cada versão derivada de um sistema, terá de implementar a

sua versão de algumas funcionalidades. Ao mesmo tempo que este tipo de aproximação torna a arquitectura mais flexível e extensível, origina código mais organizado e consideravelmente mais limpo. Esta aproximação força a criação de um destrutor virtual na classe de base. Estes sistemas de partículas diferentes possuem construtores únicos. Estes construtores devem ser inicializados com a seguinte informação:

- Número de partículas inicial do sistema
- Posição do sistema
- Tipo de sistema
- Parâmetros originais das partículas
- Forma geométrica das partículas

Podemos adicionar estas informações específicas como um conjunto de argumentos do construtor, utilizando uma estrutura de dados auxiliar ao sistema que contenha toda a informação necessária a sua construção. Esta estrutura deve ser declarada como um argumento do tipo ponteiro, caso este seja nulo, o sistema funciona com os valores originais. A razão principal ao qual utilizamos a estrutura deve-se ao facto que a estrutura pode ter uma função que contenha valores originais e visto que os parâmetros podem ser muitos, é mais elegante e mais limpo utilizar esta aproximação, do que lidar com funções que têm cerca de 14 ou até mais parâmetros. Outra razão situa-se no facto que as novas extensões ou alterações são efectuadas na própria estrutura, limitando assim os lugares onde os erros possam ocorrer, simplificando o trabalho nas futuras alterações.

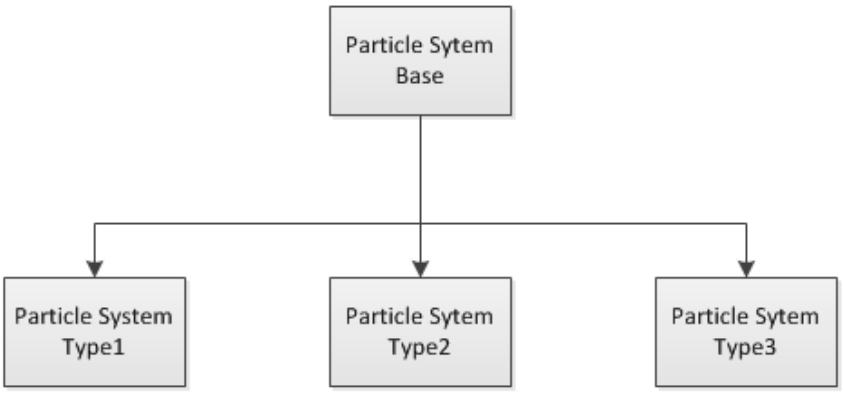


Figura 2 – Hierarquia que relaciona e organiza os diferentes tipos de sistemas.

Para definir a hierarquia dos objectos que representam os sistemas de partículas, deve-se começar por definir um objecto abstracto, denominado ParticleSystemBase, ao qual novos

objectos derivados possam ser especializados. O sistema de partículas do tipo 1 denomina-se ParticleSystemNoPhysX, pelo facto de não estar ligado ao motor físico. A sua utilização deve empregada para a visualização de efeitos visuais que simulem fenómenos que não dependem de cálculos físicos muito exigentes, portanto tem a característica de ser o sistemas mais rápido e que permite a utilização de um grande número de partículas para definir o detalhe desejado que se pretende simular. As simulações de fumo, fogo, neve, entre outros devem ser simuladas por este sistema. O segundo tipo de sistema de partículas chama-se ParticleSystemPhysX, que utiliza o motor físico da NVIDIA, o PhysX. Para obter mais informações sobre o PhysX, deve-se consultar o anexo A. Este sistema utiliza actores como partículas recorrendo a objectos rígidos com propriedades físicas para simular a sua dinâmica e colisão. Este sistema está limitado a duas formas físicas, cubos e esferas. Contém colisão com todos os objectos que sejam criados a partir do motor físico, incluindo os fluidos e outros objectos que compõem cada cena. Este sistema deve ser utilizado em simulações que requerem a interacção entre elementos físicos. O terceiro tipo de sistema intitula-se por ParticleSystemPhysXFluids e ambiciona a simulação de fluidos, recorrendo ao motor PhysX. Este sistema é o mais complicado, devido a complexidade de cálculos físicos requeridos para a sua simulação. Este sistema pode simular fluidos no estado líquido como no estado gasoso.

3.6. O gestor de sistemas de partículas

3.6.1. Introdução

Quando estamos a trabalhar com muitos sistemas deste tipo, temos de considerar a concepção de uma camada superior – um gestor. A sua relação com os sistemas deve ser semelhante a relação existente entre sistemas e partículas. Desta forma, temos de criar um gestor de sistemas de partículas, que possa orientar todos os sistemas. Este gestor tem como principal objectivo de criar, destruir, actualizar, movimentar, mudar as forcas que actuam nos sistemas, desenhar e determinar quais os sistemas que estão actualmente activos. A elaboração de um sistema avançado que seja flexível, rápido e extensível, requer tempo e um bom planeamento na consideração do seu âmbito e utilidade. Esta arquitectura utiliza objectos hereditários, portanto isto abre a possibilidade de criar novos tipos de sistemas, promovendo assim uma extensão do sistema existente, alargando as funcionalidade e criando um novo

âmbito. No futuro, as novas extensões da arquitectura, poderiam ser adicionadas por um sistema auxiliar do tipo “plug-in”.

3.6.2. Atributos do gestor de partículas

Um dos atributos deste gestor será um vector de ponteiros que possa alocar memoria para guardar todos os sistemas criados, escolhemos esta aproximação essencialmente porque facilita este tipo de operações. O gestor também pode ter a capacidade de reter elementos comuns a conjunto de sistemas semelhantes, por exemplo, se um dado conjunto de sistemas utiliza a mesma textura, não há necessidade de desperdiçar memoria na gráfica ao alocar os dados correspondentes á mesma textura em duas zonas diferente, este recurso pode simplesmente ser alocado uma única vez e posteriormente acedido pelos diferentes sistemas.

3.6.3. Métodos do gestor de partículas

Ao trabalhar com estes sistemas, requer-se uma iniciação que seja fácil de executar. Assim, não é desejável que se tenha de procurar quais são os sistemas activos e verificar se as partículas de um dado sistemas estão activas para proceder posteriormente a sua eliminação, libertando o recurso da memória. O gestor tem a capacidade de verificar se cada sistema contém partículas activas. Desta forma, este gestor pode automaticamente remover o sistema, caso seja indicado. Para utilizar sistemas esporádicos (sistemas em que as partículas são eliminadas depois de algum tempo), foi bastante útil a concepção de um método que verifique se um dado sistema já fora removido da simulação, ou seja, se ainda está activo dentro do gestor. Este gestor tem a seguinte lista de métodos:

- *Iniciar* – este método só tem um argumento, um ponteiro para lista de sistemas que vão ser criados.

- *Adicionar* – este método permite alocar a memória necessária para adicionar cada sistema ao gestor, consoante o tipo de sistema requerido.

- *Remover* – esta funcionalidade permite a eliminação de um sistema particular, libertando o recurso alocado e removendo do gestor.

- *Actualizar* – quando estamos a proceder a actualização da cena, devemos chamar esta função para podermos proceder a actualização todos os sistemas de uma forma sequencial. Como não verificamos se todos os sistemas contêm partículas activas em cada quadro, ao chamar esta função de actualização de cada sistema, este devemos informar o gestor se um determinado sistema ainda contém partículas activas. Retornado um valor de "verdadeiro" ou "falso", se o gestor obtiver um retorno verdadeiro, então procede-se a actualização do dado sistema, caso contrário deve-se proceder a sua eliminação.

- *Desenhar* – depois de os sistemas terem sido actualizados, quando desejarmos podemos chamar esta função para proceder ao desenho de todos os sistemas activos.

- *Esta activo* – este método verifica se um sistema continua activo na simulação.

3.7. Arquitectura proposta

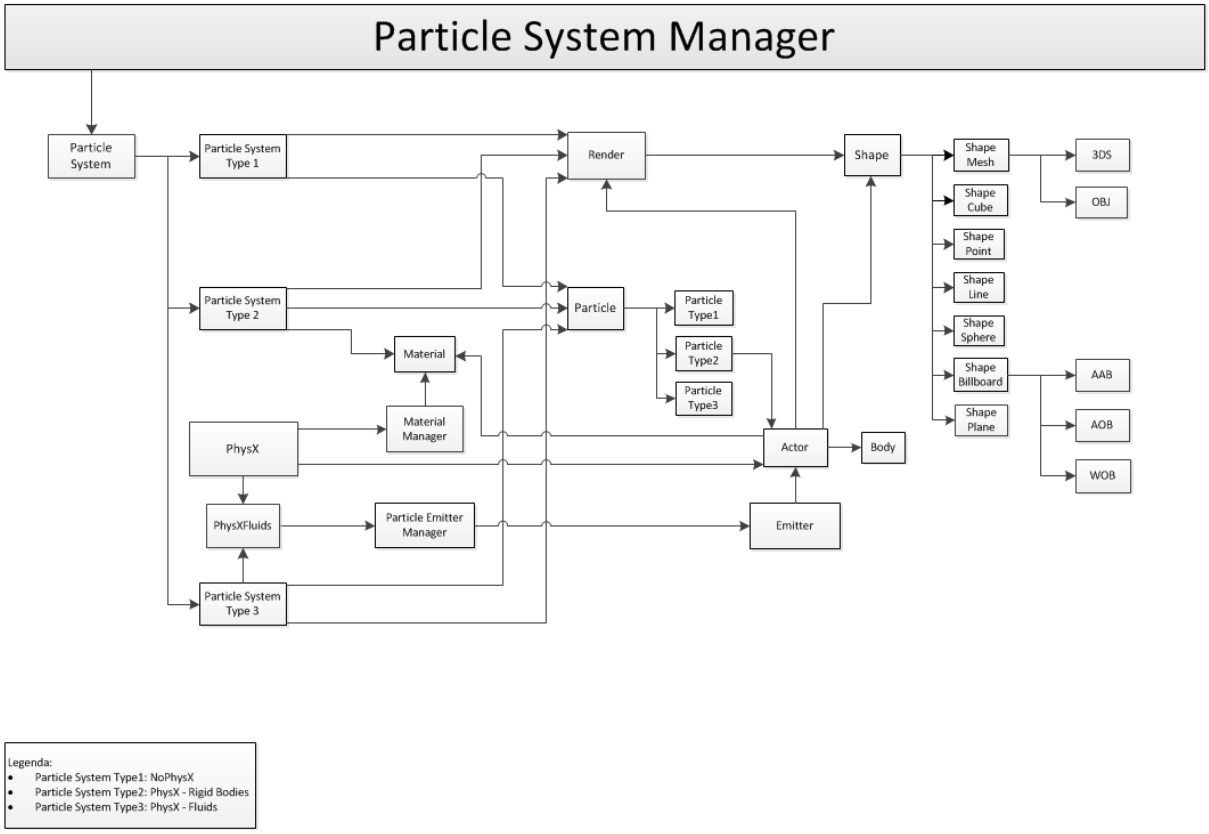
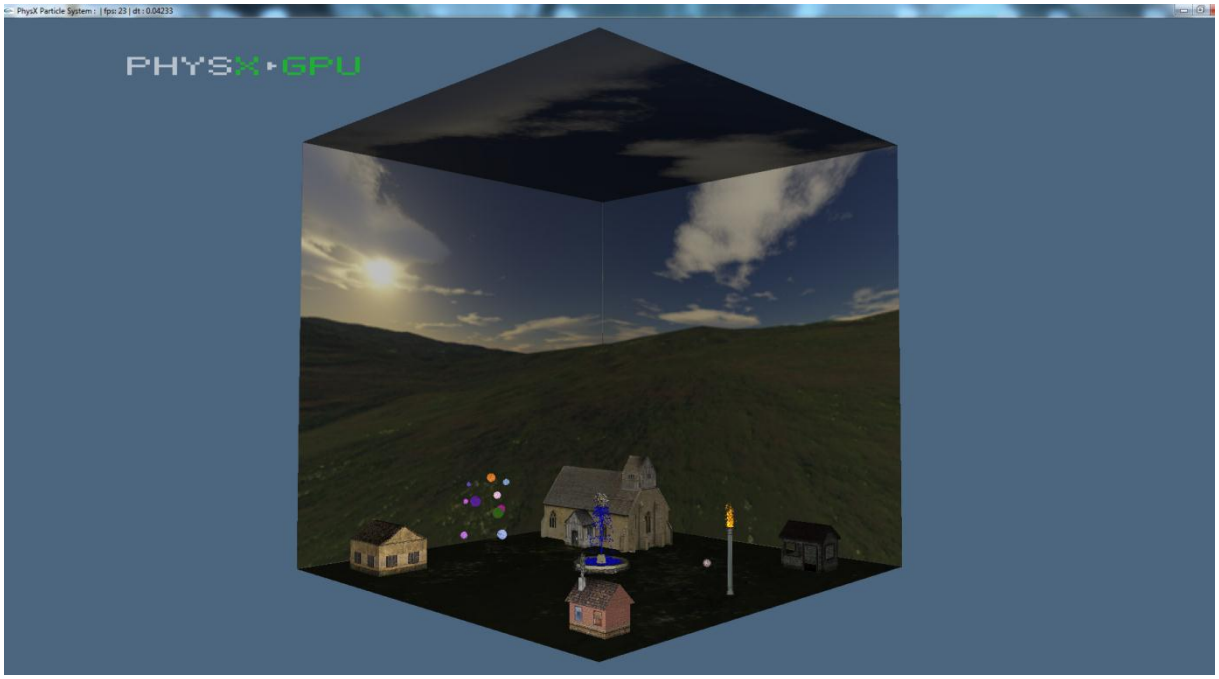


Figura 3 – Esquema que representa a arquitectura proposta.

4. Representação gráfica



"Powers of observation lie with the mind Luke, not with the eyes."

Obi Wan Kenobi, Star Wars: The Old Republic

4.1. Atributos das partículas

Uma das funcionalidades desenvolvidas neste projecto é o facto de podermos escolher um conjunto de atributos relacionados com as partículas para serem enviados para a placa gráfica. Estes atributos podem ser utilizados para construir algoritmos de desenho especializados utilizando *shaders*. Esta funcionalidade foi introduzida e desenvolvida com a mentalidade de ser portável entre as bibliotecas gráficas, OpenGL como o Direct3D. Apesar de o trabalho desenvolvido não apresentar a utilização de *shaders*, este foi concebido com o instinto de promover novas extensões e por isso a independência dos sistemas perante arquitecturas gráficas ou novos horizontes foi uma regra quase geral no desenvolvimento de todas as funcionalidades.

4.2. Formas reproduzidas de cada sistema

	Pontos	Linhas	Billboards	Cubos	Esferas
ParticleSystemNoPhysX	✓	✓	✓	✓	✓
ParticleSystemPhysX	×	×	×	✓	✓
ParticleSystemPhysFluids	✓	✓	✓	✓	✓

Billboarding é uma das técnicas mais conhecidas para aumentar a qualidade, realismo e performance das aplicações gráficas. É empregado maioritariamente pela sua capacidade de redução no número de polígonos e consequente na redução das transferências entre CPU e GPU. Este modelo é muito utilizado em sistemas de partículas, pois na grande maioria das vezes é o modelo geométrico seleccionado para representar graficamente cada partícula. Billboard é basicamente um objecto, normalmente um quadrado que se orienta para um determinado eixo, como a câmara do utilizador. Devido a sua capacidade de manipulação perante um eixo, esta técnica permite que este polígono se oriente de acordo com esse mesmo eixo.

4.3.Resultados obtidos

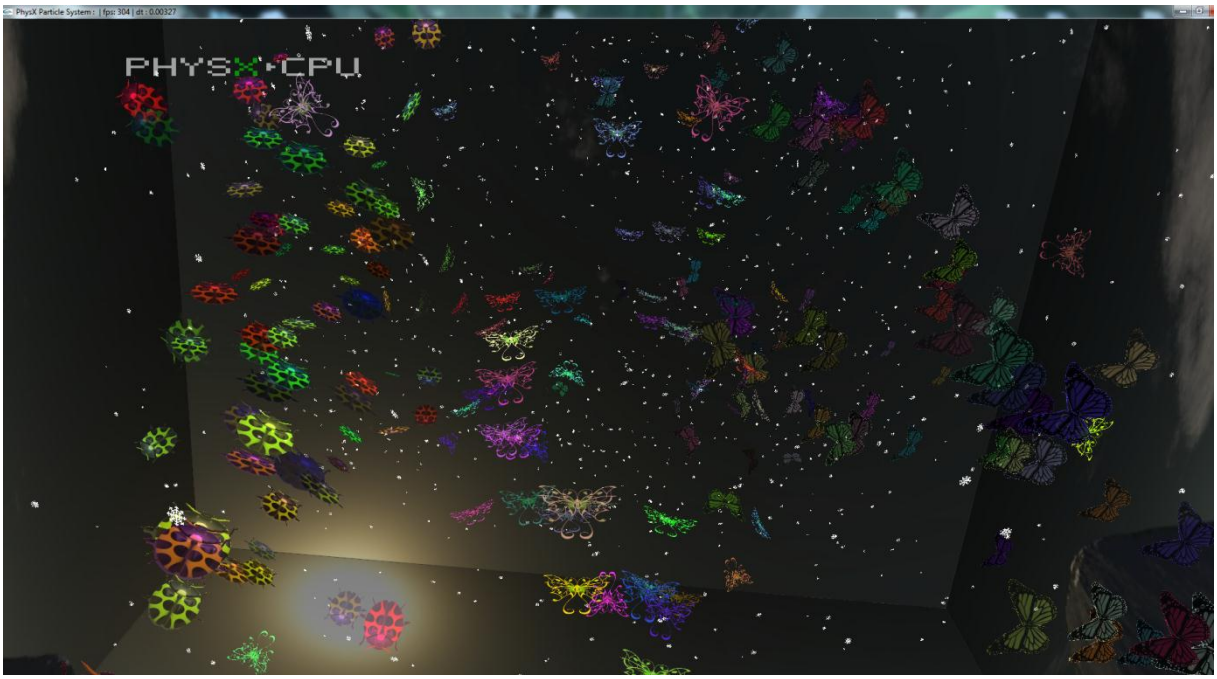


Figura 4 – Sistema NoPhysx, visualização de vários insectos.

Figura 5 - Sistema NoPhysx, simulação de neve.

Figura 6 – Sistema NoPhysX, visualização geral de todos os sistemas.

Figura 7 – Sistema NoPhysX, simulação de fumo.

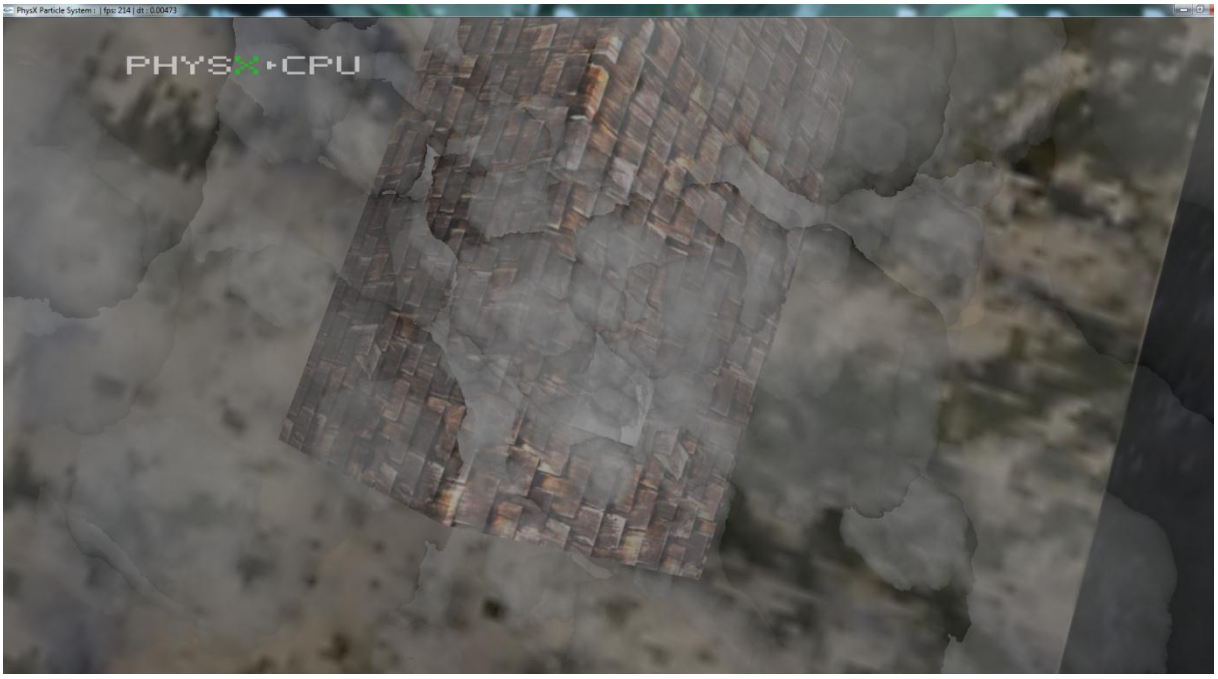


Figura 8 – Sistema NoPhysX, simulação de fumo.

Figura 9 – Sistema NoPhysX, simulação de fogo.

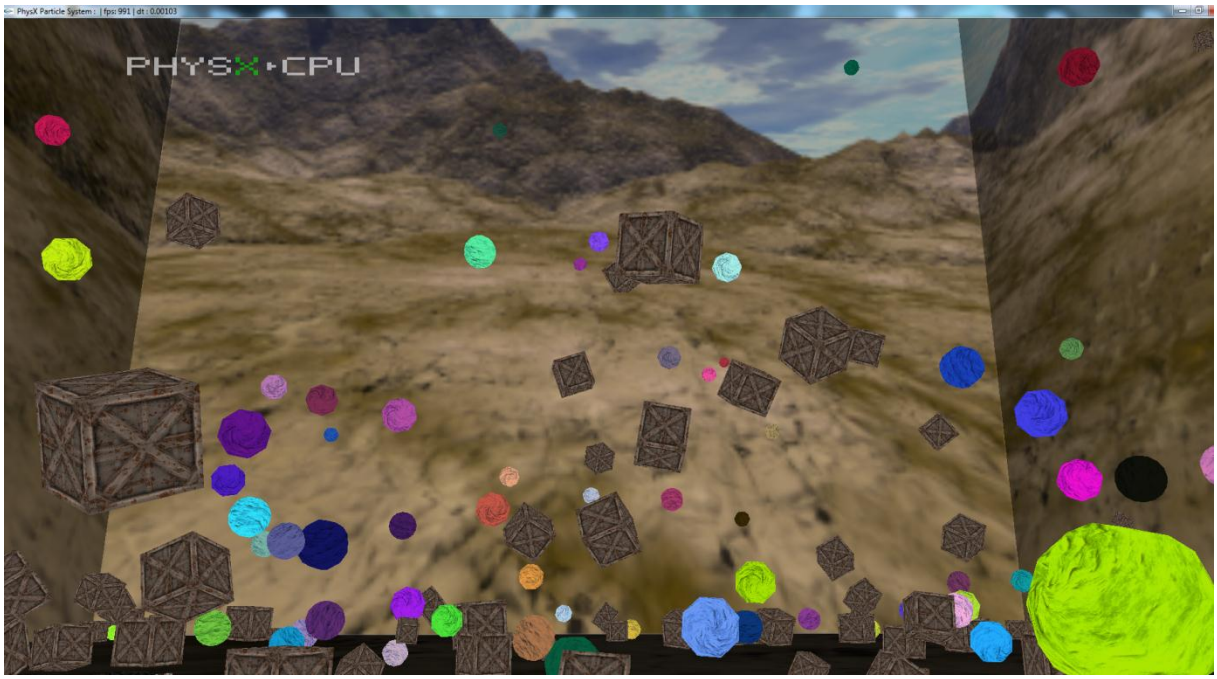


Figura 10 – Sistema PhysX, colisão entre esferas e cubos.

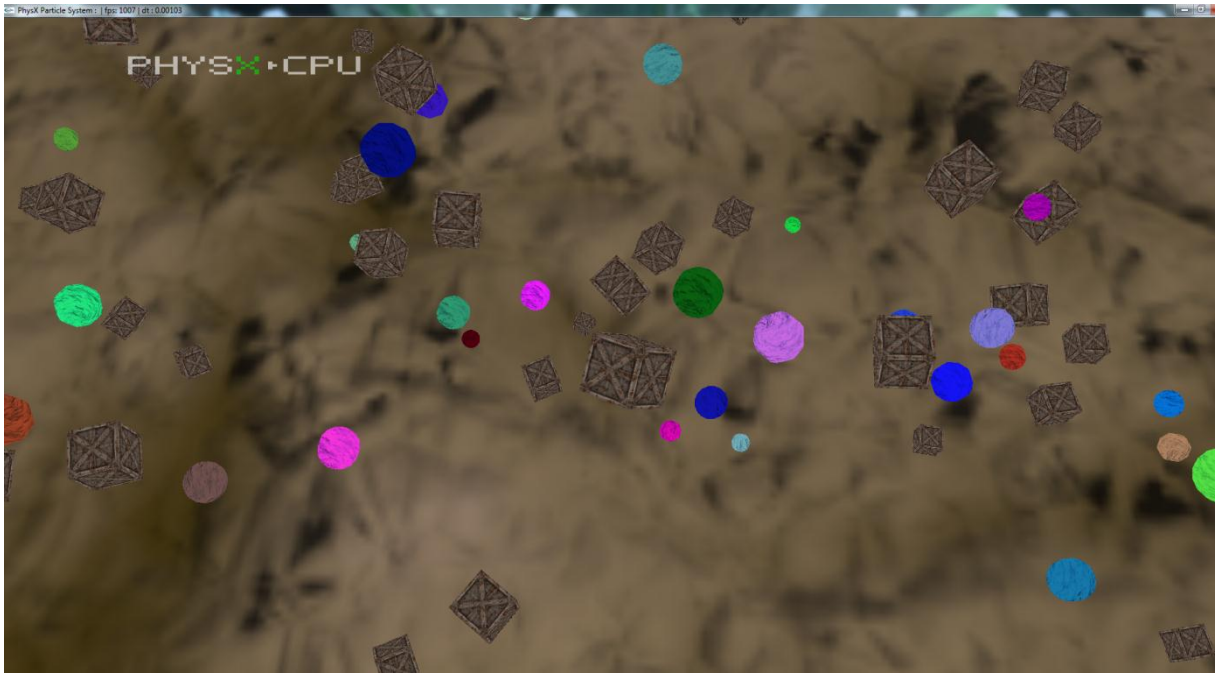


Figura 11 – Sistema PhysX, colisão entre esferas e cubos.

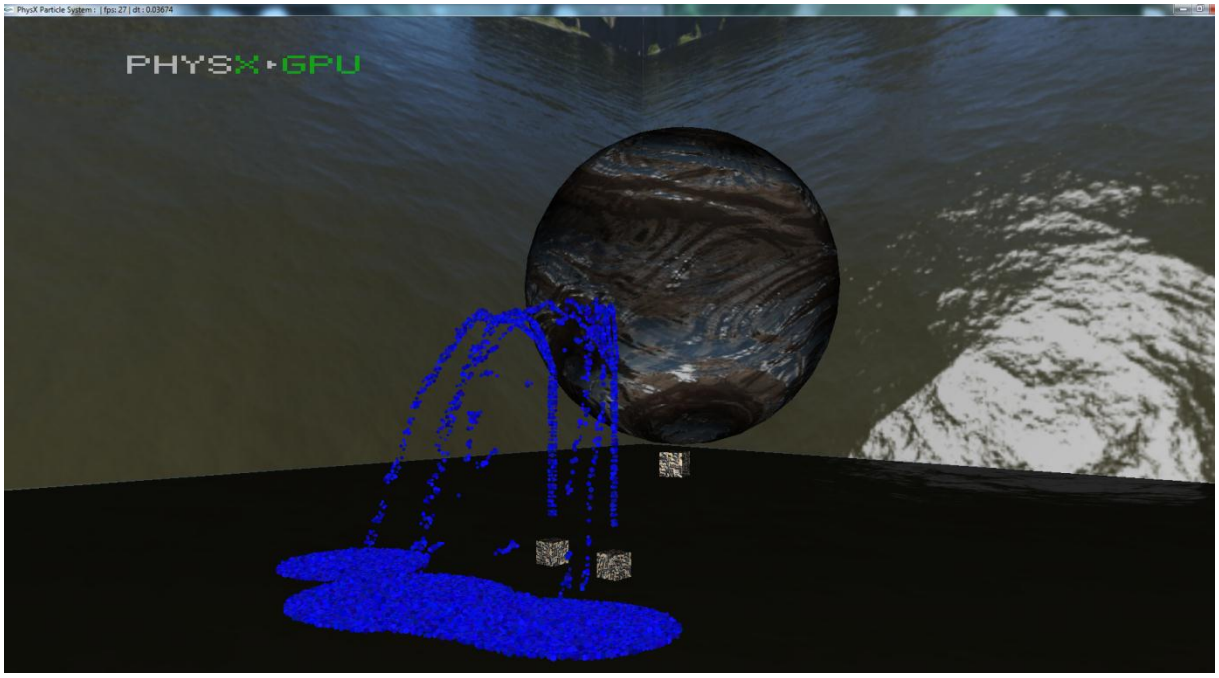


Figura 12 – Sistema PhysX Fluids, simulação de fluido, utilizando dois emissores.

Figura 13 – Sistema PhysXFluids, simulação de fluido, colisão com objectos.

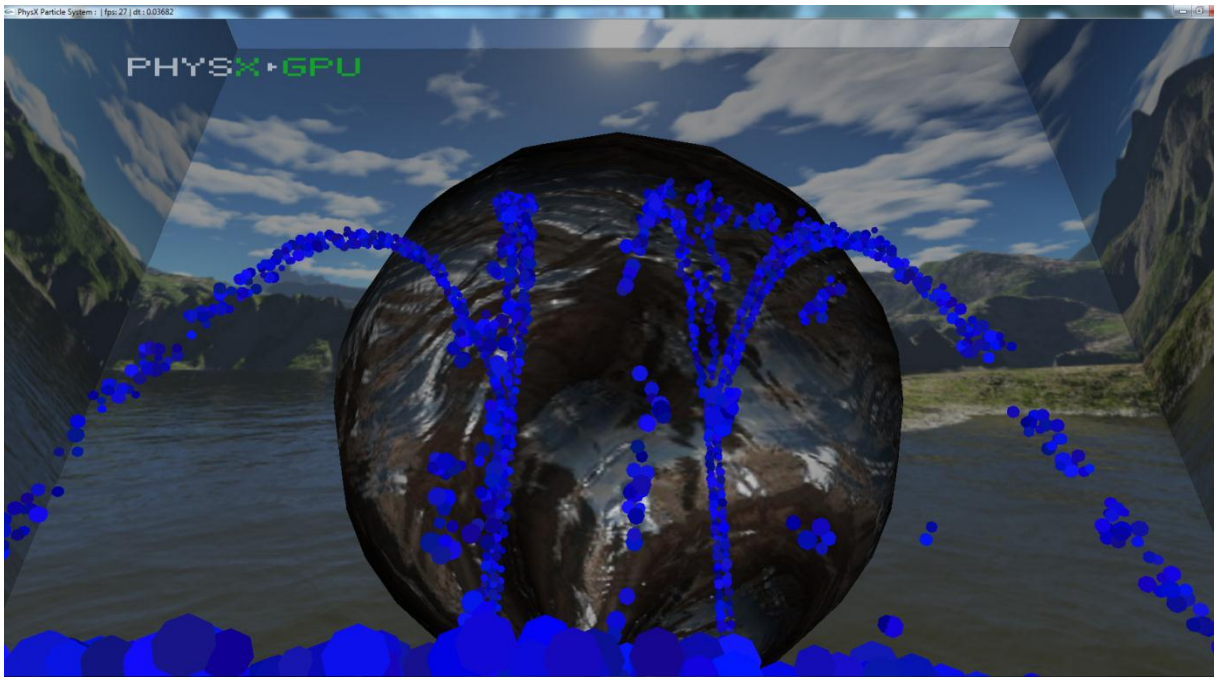


Figura 14 – Sistemas PhysXFluids, simulação de fluido, interacção entre partículas.

Figura 15 – Sistemas PhysXFluids, interacção entre partículas.

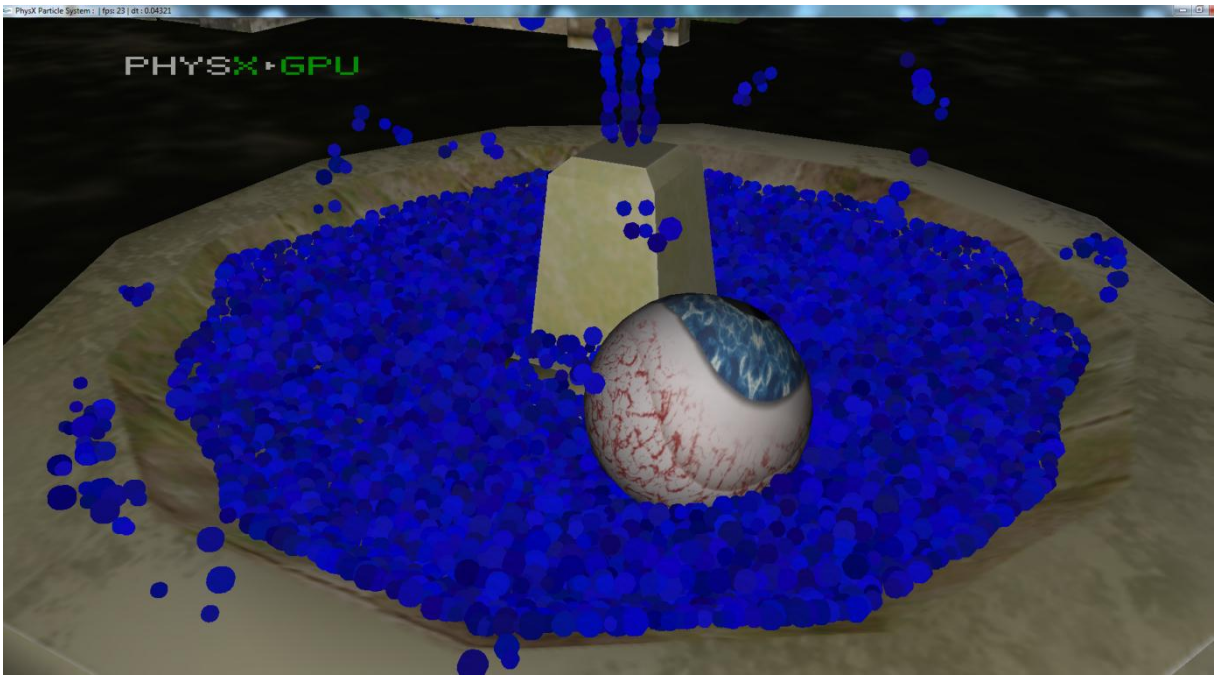
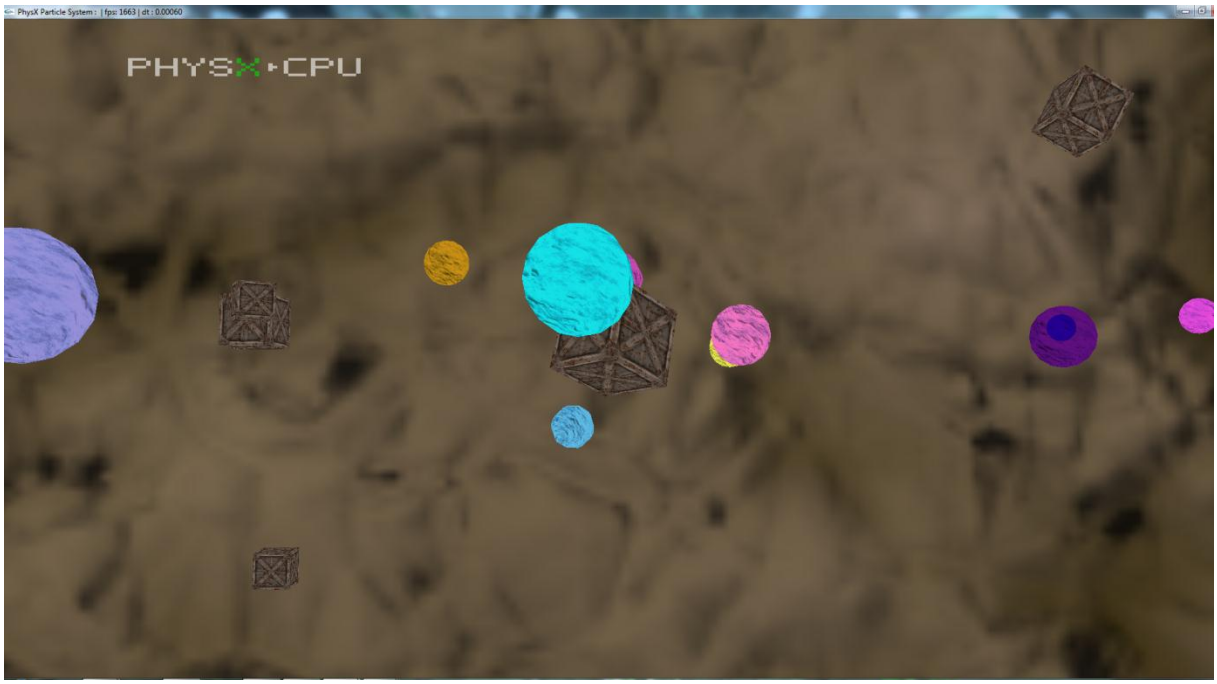


Figura 16 – Sistemas PhysXFluids, interacção entre partículas e objectos dinâmicos.

Figura 17 – Integração de todos os tipos de sistemas num cenário final.

5. Conclusão e futuro desenvolvimento



5.1.Conclusão

Este trabalho apresenta o conteúdo e funcionamento geral de sistemas de partículas dinâmicos, classificando-os como sistemas que apresentam uma grande capacidade de modelação de vários tipos de fenómenos naturais e eficientes para se adaptarem a diferentes ambientes. Estes sistemas são uma grande ferramenta para enriquecer ambiente gráficos, ou para efectuar simulações na área científica. Devido a sua simplicidade, a simulação de realismo destes sistemas encontra-se muitas vezes limitada pela potência do hardware e também do software. Ao recorrer a modelos estocásticos para controlar a sua dinâmica e aspecto gráfico, estes sistemas ganham uma enorme flexibilidade de produzir uma grande quantidade de efeitos e detalhes sem grande esforço por parte do programador. Devido ao facto de serem sistemas progressivos, estes podem gerar quantidades de detalhes necessárias às simulações, reduzindo significativamente o tempo de computação. Este trabalho tinha como ambição a criação de uma arquitectura que fosse capaz de criar e manipular vários tipos de sistemas de partículas. Como resultado, propõe-se uma arquitectura flexível que apresenta uma interface de utilização fácil e intuitiva, contendo um modelo de estrutura que permite a integração de futuras evoluções. Os objectivos deste trabalho foram alcançados, propõe-se uma arquitectura onde é possível criar três tipos diferentes de sistemas de partículas, capazes de enfrentar as mais diversas simulações, entre si diferem na dinâmica que cada um é capaz de simular como na diferença de elementos gráficos que cada é capaz de desenhar. A introdução de um motor físico veio alargar as possibilidades de extensão deste trabalho e proporcionou uma grande evolução, capacidade inovação e simulação de dois novos tipos de sistemas de partículas. Ao mesmo tempo desenvolveu-se um pequeno gestor de cenários ao qual é possível adicionar objectos físicos que interagem com os sistemas de partículas criados. Este gestor de cenários permite, entre outros tipos de objectos, adicionar formas arbitrárias que estão devidamente ligadas ao motor físico, elevando assim a capacidade de criar as mais diversas demonstrações.

5.2. Futuro desenvolvimento

Para futuro desenvolvimento e extensão do trabalho realizado são propostas as seguintes tarefas:

- Melhoramento da arquitectura e algoritmos utilizados, reorganização de funcionalidades;

- Escalar o algoritmo base para a utilização de processamento em paralelo usando *threads*;

-Introdução de grupos de acções: colisão e forças;

- Introdução de domínios de acção usando primitivas geométricas;

- Introdução de conjunto de emissores;

- Algoritmo de reconstrução de superfície 3D, utilizando o GPU, para fluidos;

- Evolução para a versão 3.0 do PhysX;

- Explorar novas funcionalidades do PhysX e integrar novas formas de interacção dos sistemas;

- Introdução de gerador de eventos;

- Criação de hierarquias de partículas;

6. Referências bibliográficas

[1] Reeves, T. (1983). "*Particle Systems – Technique for Modeling a Class of Fuzzy Objects*". SIGGRAPH Proceedings.

[2] Reeves, T., Blau, R. (1985). "*Approximate and Probabilistic Algorithms for Shading and Rendering Structured Particle Systems*". SIGGRAPH Proceedings, San Francisco, California.

[3] Reynolds, W. (1987). "*Flocks, Herds and Schools: A Distributed Behavioral Model*". SIGGRAPH Proceedings, Anaheim, California.

[4] Sims, K, (1990). "*Particle Animation and Rendering Using Data Parallel Computation*". SIGGRAPH Proceedings.

[5] Lander, J. (1998). "The Ocean Spray In Your Face". Game Developer Magazine, Graphic Content.

[6] McReynolds, T., Blythe, D. (1998). "*Advanced Graphics Programming Technics Using OpenGL*". Silicon Graphics ´98 Course, Chapter 11: Natural Phenomena, pp. 139-149.

[7] McAllister, David K. (2000). "*The Design of an API for Particle Systems*". Technical Report, Department of Computer Science, University of North Carolina at Chapel Hill.

[8] Burg, J. (2000). "*Building and Advanced Particle System*". Game Developer Magazine, March.

[9] Eberly, D. (2000). "*3D Game Engine Design: A Practical Approach to Real-Time Computer Graphics*". Morgan Kaufman, Chapter 13.7: Particle Systems, pp. 432-433.

[10] Witkin, A. (2001). "*An Introduction to Physically Based Modeling: Particle System Dynamics*". Pixar Studios, SIGGRAPH Curse Notes 2001, Physically Based Modeling.

[11] Hawkins, K., Astle, D., (2001). "*OpenGL Game Programming*". Prima Tech, Chapter 15: Particle Systems, pp. 433-462.

[12] McCuskey, M. (2002). "*Special Effects Game Programming with DirectX*". Premier Press, Chapter 18: Rain, Smoke, Magic and more: The joy of particle systems, pp. 585-632, Chapter 19: Advanced Particle Systems, pp. 633-680.

[13] Luna, F. (2003). "*Introduction to 3D Programming with DirectX 9.0*". Wordware, Chapter 14: Particle Systems, pp.235-256.

[14] Adams, J. (2003). "*Advanced Animation with DirectX*". Premier Press, Chapter 15: Using Particles in Animation, pp. 244-272.

[15] Barron, T. (2003). "*Strategy Game Programming with DirectX 9.0*". Wordware, Chapter 13: Particle Rendering.

[16] Dalmau, D. (2003). "*Core technics and Algorithms in Game Programming* ". New Riders, Chapter 19: Particle Systems, pp. 476-492.

[17] McReynolds, T., Blythe, D. (2005). "*Advanced Graphics Programming Using OpenGL*". Morgan Kaufman, Chapter 18: Natural Detail, pp. 467-478.

[18] Thorn, A. (2005). "*DirectX 9.0 Graphics: The Definitive Guide to Direct3D*". Wordware, Chapter 10: Point Sprites and Particle Systems, pp. 249-260.

[19] Jones, W. (2005). "*Beginning DirectX 9*". Thomson Course Technology, Chapter 8: Point Sprites, Particles and Pyrotechnics, pp. 177-197.

[20] Eberly, D. (2005). "*3D Game Engine Architecture: Engineering Real-Time Applications with Wild Magic*". Morgan Kaufman, Chapter 3.3: Particles, pp.202-203; Chapter 4.5.3: Points and Particles, pp. 406-410; Chapter 7.2: Particle Physics, pp. 576-580;

[21] Luna, F. (2006). "*Introduction to 3D Programming with DirectX 9.0 – A shader approach*". Wordware, Chapter 19: Particle Systems, pp.235-256.

Anexo A: NVIDIA PHYSX

"The world these days isn´t flat anymore, it´s parallel."

Nvidia @ GPU Technology Center, September 2010, San Jose Convention Center

1. Introdução

Este capítulo pretende divulgar alguns conceitos sobre a plataforma do PhysX. Os próximos tópicos salientam a motivação na utilização de motores de física e a ligação com a arquitectura proposta neste trabalho.

1.1. Motivação

No mundo-real, os objectos estão sempre em interacção com outros objectos, cada um apresenta diferentes comportamentos devido ao facto de serem compostos por diversas matérias, devido ao peso, tipo de superfícies, tipo de material constituinte, entre outras características. Propriedades como colisões, fricção, forças, entre outras, são essenciais para oferecer a sensação de realismo. As aplicações que façam uso integral de simulações físicas requerem alguma complexidade nos cálculos, assim a forma mais fácil de escalar uma aplicação para este nível será a integração de um motor de física. Devido ao grande custo de processamento que é requerido para efectuar os cálculos necessários para simular o comportamento do mundo-real, uma solução alternativa será a utilização de um processador que seja dedicado para efectuar esta tarefa. Da mesma forma que um GPU tem a tarefa de processar gráficos, podemos hoje em dia tirar vantagem das velocidades que os melhores GPUs oferecem para dedicar um GPU para efectuar simulação física, libertando assim o CPU para efectuar outros cálculos necessários e oferecendo a aplicação uma maior disponibilidade para realizar outras tarefas. Esta é a solução utilizada como modelo de apresentação deste trabalho, espera-se a composição de simulações complexas para fazer uma pequena amostra dos limites que hoje em dia podem ser atingidos em tempo-real ao mesmo tempo preservando a interactividade.

1.2. Motores físicos

Um motor de física é um programa que simula a física de acordo com as leis de Newton, de forma a predicar um acontecimento do mundo real através de uma simulação.

Existem essencialmente dois tipos de motores de física, motores de tempo-real e motores de alta precisão. Motores de alta precisão requerem uma grande quantidade de cálculos para reflectir acontecimento do mundo-real, são utilizados em aplicações científicas e simulações de efeitos especiais realizados em filmes. Para jogos e aplicações interactivas, a simulação física necessita de ser calculada em tempo real, enquanto a aplicação está em funcionamento, por isso os modelos normalmente utilizados são um pouco mais simplificados para atingir esse requerimento. Estes motores implicam uma interactividade entre o comportamento de objectos e actores muito próximo da realidade.

Existem motores que são de livre utilização e outros são comerciais, mas ambos podem ser ligados a um motor gráfico, estabelecendo assim uma facilidade no desenvolvimento das aplicações.

A base do desenvolvimento de motores físicos situa-se na modelação de comportamentos utilizando equações matemáticas, construindo algoritmos de simulação que resolvem a evolução da simulação e a alteração ao longo do tempo. Normalmente os motores de física disponíveis lidam com detecções de colisão e dinâmica de um corpo, integrando mais recentemente a simulação de fluidos, roupa e objectos deformáveis.

1.3.PhysX SDK

O NVIDIA PhysX SDK, anteriormente conhecido como Ageia PhysX SDK, é uma ferramenta bastante poderosa, oferecendo uma arquitectura com interface de programação para a simulação de ambientes físicos dinâmicos em diversas plataformas, PC, PS3 e XBOX. Foi a primeira arquitectura assíncrona (processamento em paralelo) disponível, capaz de aproveitar a potência de cálculo que os multiprocessadores hoje em dia oferecem. Apresentando uma arquitectura bastante flexível que expõe a capacidade de criar ambiente físicos complexos. Esta arquitectura utiliza uma tecnologia de simulação avançada, com suporte para a concepção de um número arbitrário de cenários físicos, ao qual podemos enunciar as seguintes características:

- Simulação complexa de dinâmica em objectos rígidos e detecção de colisão, simulação realista do movimento, interacção de personagens, permitindo ao programador uma

concepção de colisões bastante naturais e o acumulamento entre estas para consequentes efeitos secundários.

- Pontos de referência e molas são úteis na concepção de ferramentas para criar mecanismos complexos que possibilitam ir além do movimento de personagens, portas, folhas, movimento de veículos, a possibilidade de o utilizador pegar nos objectos e manipula-los.

- A simulação volumétrica de fluidos para enriquecer a visualização de efeitos especiais em cenários, como a simulação de ondas na superfície do fluido, possibilidade de utilização de armas com fluidos ou a criação de fontes de água (ou outro liquido qualquer).

- Apresenta um módulo para simulação de sistemas de partículas, este módulo é criado a partir da utilização da simulação de fluidos, oferecendo a possibilidade de criar fogo, fumo, nevoeiro e chuva, de forma muito natural e realista.

- A simulação de roupa foi introduzida para aumentar a realidade da simulação de uma cena, oferecendo a possibilidade de criar vários objectos, como cortinados, roupas para as personagens entre outros.

Esta arquitectura contém os integradores capazes de resolver todo o tipo de simulações físicas, permitindo que fluidos, partículas, roupa e objectos rígidos interajam correctamente. Explora o paralelismo a qualquer nível de utilização, o que permite atingir elevadas performances na simulação. A abstracção a nível do *hardware* elimina a necessidade de proceder a uma compreensão de arquitecturas das próximas gerações.

1.4.PhysX no GPU

O enriquecimento de simulações físicas aumenta significativamente a utilização de cálculos matemáticos. Assim, a utilização de um processador dedicado para efectuar os cálculos físicos simboliza a próxima geração de simulações físicas ao mesmo tempo que retira uma grande carga de trabalho ao CPU. Este pode agora respirar melhor e proceder a realização de outros trabalhos enquanto o GPU trata da simulação física. Assim é possível obter uma simulação de comportamentos mais complexos sem que a performance da aplicação seja penalizada. Para obter um melhor aproveitamento das simulações físicas, é aconselhável a utilização de um CPU e GPU de alta performance, estas simulações acompanhadas com objectos tenham um desenho mais detalhado, oferecem resultados bastante realistas. Nos esquemas seguintes pretende-se esclarecer melhor a vantagem e inovação que este motor físico possui relativamente as outros:

Esquema 1: Single CPU

Figura 18 – Esquema 1, um CPU

Neste esquema analisamos apenas 1 CPU sem núcleos, como podemos verificar o CPU encontra-se muito atarefado, devido a excessiva carga de trabalho a que está sujeito. Desta forma, o programador está limitado a complexidade de cálculos físicos, pois a aplicação poderá rapidamente perder a performance. Podemos ainda verificar, nesta situação que a placa gráfica fica a maioria do tempo a espera que os cálculos avancem.

Esquema 2: Multicore CPU

Figura 19 – Esquema 2, CPU com vários núcleos.

Este esquema é mais recente que o anterior e marca a evolução nos núcleos em CPU. Podemos verificar que agora o trabalho do CPU é distribuído pelos vários núcleos que o compõem. As aplicações obtêm mais performance, existindo a possibilidade de aumentar ligeiramente os cálculos físicos, uma vez que o CPU adquiriu a capacidade de realizar mais trabalho. Apesar da evolução neste esquema relativamente ao anterior, podemos verificar que a gráfica continua na grande maioria do tempo a espera que o trabalho acabe por ser realizado, portanto é necessário encontrar outra solução.

Esquema 3: Symmetric Multiprocessor

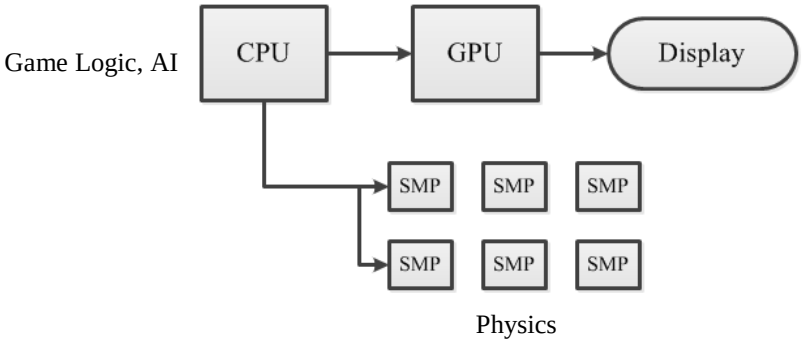


Figura 20 – Esquema 3, multiprocessadores simétricos.

Pela 1ª vez houve uma aproximação na tentativa de retirar todos os cálculos da física do CPU, recorrendo a um esquema de multiprocessadores simétricos (SMP), ao qual envolve uma arquitectura com múltiplos processadores estão interligados por um bus comum e partilham a memória principal, sendo controlados pela mesma instância do sistema operativo. Obviamente esta solução esta muito longe de se tornar uma solução económica e comercializável para os utilizadores comuns. Apesar desta nova aproximação ao problema, podemos verificar que a gráfica continua a espera e por isso esta solução foi abandonada.

Esquema 4: Ageia Processor

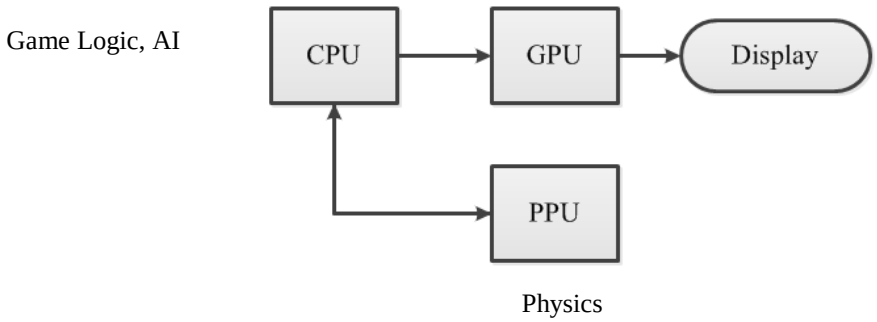


Figura 21 – Esquema 4, processador Ageia.

A solução apresentada neste esquema, aproveita a ideia mencionada no esquema anterior, retira os cálculos físicos do CPU, mas apresenta uma evolução mais inovadora, pois agora a física pode ser completamente simulada num processador dedicado e integrado numa placa própria, sem recorrer a vários processadores. A simulação física agora é processada mais rapidamente, permitindo aumentar a sua complexidade, devido ao processador exclusivo. Uma solução certamente mais atraente e economicamente viável, apesar disto, ainda existem desvantagens, como a necessidade de adquirir equipamento específico para realizar os cálculos físicos e o facto o GPU continuar a grande maioria do tempo a espera.

Esquema 5: GPU

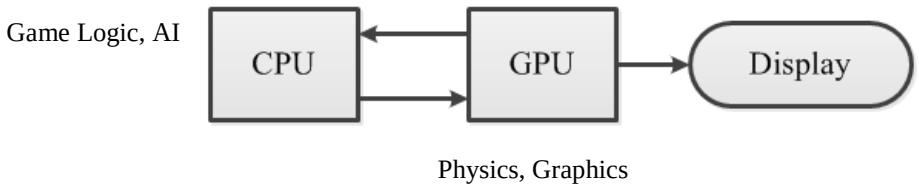


Figura 22 – Esquema 5, GPU.

Pela primeira vez encontra-se uma solução óptima, juntando as duas aproximações dos esquemas anteriores e resolvendo os problemas que assombravam as soluções propostas. Assim, este esquema representa o futuro das simulações físicas e marca a diferença entre os vários motores físicos existentes no mercado, escalando o seu funcionamento e performance para um novo patamar. Agora, temos um processador que se encarrega de efectuar os cálculos físicos e é também responsável pelo desenho da cena, permitindo que o CPU realize mais trabalho e ao mesmo tempo, preenchendo a gráfica com mais trabalho, agora encarrega-se da

simulação física. Esta solução é economicamente viável, pois todos os computadores necessitam de uma gráfica, que desta forma, pode apresentar uma nova funcionalidade.

Esquema 6: SLI

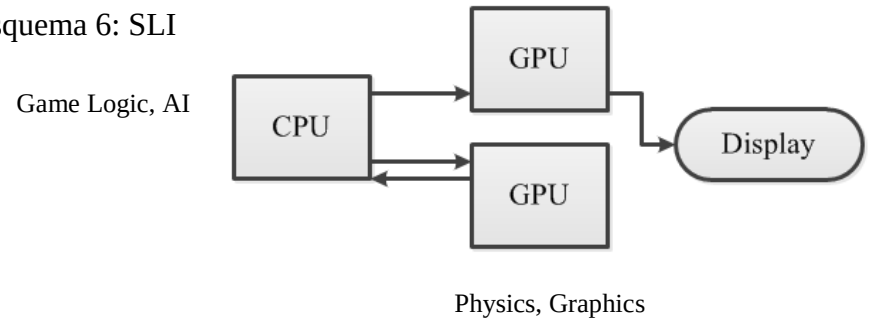


Figura 23 – Esquema 6, SLI.

As exigências das simulações podem chegar ao extremo, recorrendo a algoritmos de computação bastantes pesados. Tirando partido da evolução dos GPU, esta solução apresenta-se como a arquitectura mais recente de simulações físicas, absorvendo as vantagens das soluções anteriores, ao mesmo tempo que apresenta uma resposta para os programadores que requerem que mais poder de cálculo. Um GPU encarga-se da simulação física, enquanto este está a realizar o trabalho, o outro GPU pode desenhar objectos que não sejam dependentes da física, assim que a física estiver simulada, os dois em conjunto tratam do desenho da cena. As simulações nesta solução podem atingir níveis e complexidades nunca antes vistos. A única desvantagem é a necessidade de adquirir material da mais alta gama. Esta é a solução utilizada por este trabalho como base de desmonstração. Pretende-se mostrar a complexidade que as simulações físicas hoje em dia podem atingir em tempo-real.

2. Arquitectura e componentes do PhysX

Esta secção trata de esclarecer os aspectos importantes sobre a arquitectura do PhysX. A compreensão dos conceitos principais que constituem esta arquitectura e o conhecimento geral sobre as simulações físicas utilizados neste trabalho são mencionados nas seguintes secções.

2.1.Mundo

O PhysX encontra-se implementado em C++ e está organizado como uma hierarquia de objectos. Cada objecto contém um conjunto de funcionalidades acessíveis através de interfaces implementadas pelo utilizador. Estes objectos são abstractos, que formam classes de base. As interfaces dos objectos seguem algumas convenções de implementação:

- Todos os objectos contêm o mesmo nome que o ficheiro onde o objecto está declarado;
- Todos os tipos declarados e objectos começam com letras maiúsculas;
- Objectos de interface começam sempre com o prefixo "Nx";
- Parâmetros e valores de retorno onde o valor do tipo "NULL" é aceite, estão programados com uma sintaxe de ponteiros;
- Parâmetros e valores de retorno onde o valor do tipo "NULL" não é aceite, estão programados com uma sintaxe de referências;
- Se alguma funcionalidade estiver dependente do utilizador, este terá de implementar uma interface, incluindo o manuseamento de memória.

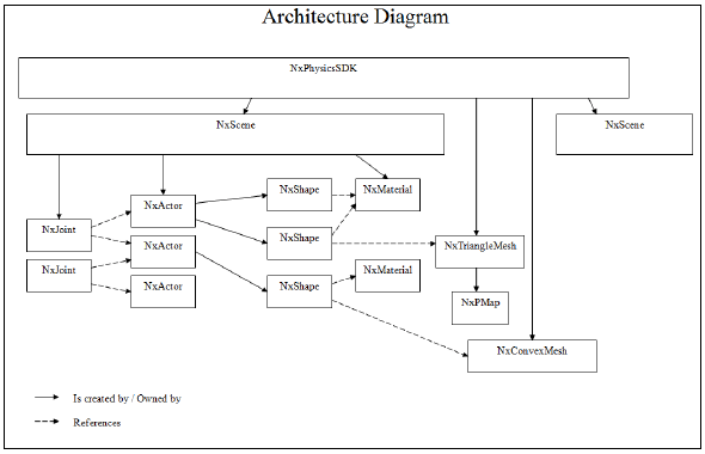


Figura 24 – Arquitectura do NVIDIA PhysX.

O objecto abstracto NxPhysicsSDK é o objecto usado para instanciar o grupo de todos os outros objectos que vão ser considerados na simulação, incluindo as várias cenas e também a iniciação dos parâmetros globais que vão afectar a simulação das mesmas. Para obter uma instancia deste objecto, deve ser chamada a seguinte função NxCreatePhysicsSDK(). O parâmetro NX_PHYSICS_SDK_VERSION assegura que a aplicação esta a utilizar a mesma

versão dos ficheiros. Este trabalho utiliza a versão 2.8.4, recentemente foi criada uma nova versão 3.0 que apresenta uma restruturação total do PhysX. Os programadores podem, por opção, indicar objectos que fazem o manuseamento de memória, preservando o estado da simulação.

2.2.Cenas

A definição de uma cena situa-se no agrupamento de conjuntos e restrições de objectos físicos, ao qual proporciona a interactividade da simulação. Cada cena simula o comportamento destes objectos através do tempo, mas estes conjuntos e restrições não podem ser partilhados entre diferentes cenas. Cada cena pode manipular um conjunto de elementos que constituem um ambiente e internamente encontra-se representada pelo objecto NxScene.

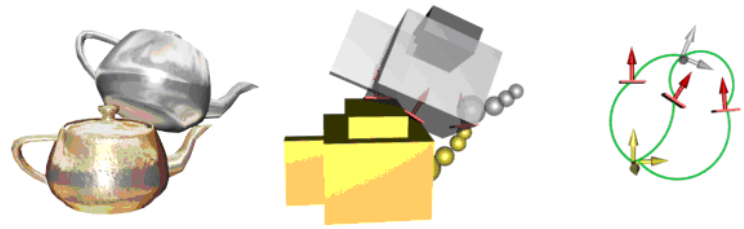
2.3.Materiais

Os materiais são conceitos relacionados com as reacções de colisão. Estes estão ligados a forma física de um dado actor. Assim, a descrição de um material, irá fornecer ao motor a informação para resolver as situações onde existem colisões com objectos, definindo o formato das reacções a aplicar aos objectos, como impulsos, forças aplicadas ou quebras de inércia. Todas as formas devem ter um material associado, caso o programador não associe um material pré-definido o motor irá associar um material com as definições originais. As características de cada material podem ser definidas da seguinte forma:

- Restituição (estática e dinâmica): está relacionado com a quantidade de impulso que é aplicado a cada objecto quando existe colisão, deve-se atribuir um valor com um intervalo entre 0 e 1, quanto mais perto o valor dor de 1, mais impulso será aplicado ao objecto.

- Atrito (estático/dinâmico): esta propriedade define o valor de atrito para cada forma, implica um valor entre 0 e infinito.

2.4. Actores



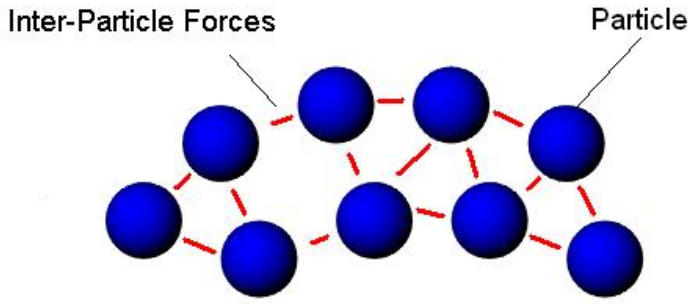
2.4.1. Introdução

Os actores formam uma base para qualquer contexto de simulação. Representam objectos físicos que fazem parte de uma determinada cena. Existem muitas propriedades que estão ligadas a estes actores, orientação e posição global, interacção com outros objectos, entre outros. No contexto de comportamento, podemos ter dois conjuntos de actores, dinâmicos e estáticos, dependendo da forma como reagem a forças ou ao contacto com outros objectos da simulação.

2.4.2. Actores estáticos e dinâmicos

Actores estáticos são actores que fazem parte de um cenário envolvente, são objectos que não se movimentam, como edifícios. Estes objectos têm o propósito de apenas fornecer detecção de colisão, não possuem massa, cinética, inércia ou qualquer tipo de propriedades físicas, devido ao facto de a sua posição não se alterar durante a simulação. Actores dinâmicos representam a definição oposta, constituindo-se através de um conjunto de definições físicas que permitem que estes sejam dinâmicos, podendo alterar a sua posição durante a simulação e reagir as diferentes situações físicas. Cada actor pode ter um conjunto de formas associadas. Durante a simulação pode-se alterar as definições dos actores, podendo modificar as descrições que o constituem o seu comportamento, como alterar as interacções de forças que os influenciam a sua interacção no mundo.

2.5.Fluidos



2.5.1. Introdução

Este módulo permite simulações físicas de fluidos no estado líquido e gasoso, utilizando sistemas de partículas e emissores. É possível simular fontes de água, permitindo que o programador tenha uma influência sobre o controlo físico da simulação, manipulando a quantidade de emissão, velocidade inicial, a variância, ângulo de emissão, ente outros. Existe a possibilidade de criar drenos, por exemplo, criar uma cascata de água e ao fundo criar um dreno para que as partículas possam ser retiradas da simulação, permitindo que novas partículas possam ser criadas. Outra forma de eliminar partículas será a especificação de um tempo de vida associado a cada uma, que vai decrescendo a cada passo que a simulação avança. Isto é especialmente útil quando pretendemos simular fenómenos gasosos onde o tempo de vida associado com propriedades gráficas poderá criar simulações realistas.

2.5.2. SPH

A modelação de fluidos é utilizada no dia-a-dia, desde a previsão de tempo, engenharia aeronáutica, computação gráfica, etc. Os fluidos são modelados através das equações discretas de movimento em materiais contínuos, chamadas as equações de Navier-Strokes. As equações de Navier-Strokes podem ser resolvidas utilizando uma diferença finita, volume finito, elementos finitos e métodos de partículas. Hidrodinâmica de partículas (SPH) foi concebida para simular fenómenos assimétricos em astrofísica. Era requerido um método

que apresentasse uma manipulação fácil e resultados coerentes. Este método satisfazia estes requerimentos, apresentava resultados que eram sensíveis a situações particularmente difíceis e poderia ser expandido a cálculos mais complexos de física sem muito trabalho adicional. O SPH é um método que funciona através de partículas, ao contrário de outros métodos que funcionam a base de células, este não necessita de células para calcular a derivadas espaciais. Em vez disso, as derivadas são encontradas por diferenciação analítica de uma função de interpolação. As equações da quantidade de movimento e energia tornaram-se um conjunto de equações diferenciáveis que são de compreensão fácil em termos mecânicos e termodinâmicos. Por exemplo, o gradiente de pressão tornou-se numa força entre partículas. O SPH foi criado para resolver as equações de Navier-Strokes em 3 dimensões. A capacidade de efectuar simulações realistas de modelos de fluidos têm sido usados recentemente para a aplicações de computação gráfica em tempo-real.

O fluido é modelado através de um conjunto de partículas ao qual se relacionam através de quantidades relativas. O SPH é um método que funciona através de interpolações de partículas. Estas quantidades são analisadas numa região fixa com raio simétrico, através de posições de partículas discretas, avaliadas em qualquer posição no espaço. Assim, uma quantidade escalar A é interpolada numa localização r através de uma média do somatório da contribuição de todas as partículas vizinhas. Se ρ, representar a quantidade que especifica a densidade de cada partícula, então podemos interpolar qualquer quantidade A no ponto r através da seguinte fórmula:

$$A(r) = \sum_j m_j \frac{A_j}{\rho_j} W(r - r_j, h) \qquad (10)$$

Onde j itera por todas as partículas, mj é a massa da partícula j , r_j representa a sua posição , ρ_j a sua densidade, A_j é a quantidade a avaliar em r_j. A função de $W(r, h)$ é denominada de máscara espacial com raio h. São apenas consideradas máscaras com suporte finito, h é utilizado como suporte finito nessa formulação. Caso W seja equivalente (ex: $W(r, h) = W(-r, h)$) e normalizada, então esta interpolação tem uma precisão de segunda ordem. A máscara é considerada normalizada se:

$$\int W(r)dr = 1 \qquad (11)$$

A massa e a densidade de uma partícula aparecem na equação 13 porque cada partícula i representa um certo volume, $V_i = {}^{m_i}/_{\rho_i}$. Enquanto a massa m_i é constante durante toda a simulação e também é constante para todas as partículas, a variação acontece apenas a densidade ρ_i e necessita de ser avaliada em todos os passos. Ao substituir a equação 13, a densidade de cada partícula pode ser dada pela seguinte forma:

$$\rho_s(r) = \sum_j m_j \frac{\rho_j}{\rho_j} W(r - r_j, h) = \sum_j m_j W(r - r_j, h) \qquad (12)$$

Na grande maioria das equações de fluidos, as derivadas das quantidades dos campos necessitam de ser avaliadas, ao utilizar a aproximação a este método, estas derivadas só afectam a mascara espacial, deste modo o gradiente de uma quantidade A é aproximada pela seguinte forma:

$$\nabla A_s(r) = \sum_j m_j \frac{A_j}{\rho_j} \nabla W(r - r_j, h) \quad (13)$$

E o laplaciano pode ser obtido da seguinte forma:

$$\nabla^2 A_s(r) = \sum_j m_j \frac{A_j}{\rho_j} \nabla^2 W(r - r_j, h) \qquad (14)$$

A interpolação espacial no método de SPH é de segunda ordem, desde que a máscara seja equivalente e normalizada. Por isso, deve-se escolher um conjunto de máscaras que se adaptem ao cálculo dos vários elementos e que ao mesmo tempo melhorem a computação e a estabilidade da simulação. Este método tem vantagens e desvantagens, em comparação com os métodos tradicionais de Eulor. Uma das vantagens é que a conservação de massa é automaticamente garantida, facilitando assim a modelação de interface do fluido, ao mesmo

tempo que oferece a facilidade na resolução de condições complexas dos limites de colisão, comparativamente aos métodos de Eulor, especialmente em três dimensões, apresentando uma resolução mais simples e mais rápida. As desvantagens situam-se no facto que a simetria de forças e a conservação do momento não são automaticamente preservadas (as equações foram criadas de modo a forçar a simetria), é difícil a imposição de incompreensibilidade no fluido, a geração de superfícies suaves, existem um conjunto de parâmetros que necessitam de ser configurados para atingir uma boa performance.

2.5.3. Especificação de um fluido

Cada fluido contém um conjunto de propriedades que definem o seu comportamento durante a simulação, as mais importantes definem-se da seguinte forma:

- *maxParticles*: numero máximo de partículas utilizadas para a simulação do fluido.
- *restParticlesPerMeter*: numero de partículas por metro, utilizando uma resolução linear avaliada quando o fluido se apresenta no estado relaxado.
- *restDensity*: densidade do fluido no estado relaxado.
- *kernelRadiusMultiplier*: influência radial para a interacção entre partículas.
- *packetSizeMultiplier*: paralelização do fluido.
- *stiffness*: pressão do fluido, relacionado com a interacção entre partículas
- *viscosity*: define a viscosidade do fluido.
- *damping*: o amortecimento de velocidade aplicado globalmente a cada partícula.
- *externalAcceleration:* aceleração aplicada a cada partícula em cada passo da simulação.
- *simulationMethod*: define se o método de simulação deve considerar a interacção entre partículas ou não.

2.5.4. Estado das partículas

Cada partícula tem um estado individual associado, como descrito em seguida:

- *Posição* – posição da partícula.
- *Velocidade* – velocidade linear de cada partícula.
- *Densidade* – densidade do fluido na região envolvente da partícula, a massa de todas as partículas que pertencem a um fluido são constantes.
- *Vida* – a vida restante de cada partícula, medido em segundos.

2.5.5. Interacção entre partículas

A simulação de fluidos pode ocorrer em dois módulos diferentes, um chamado SPH, onde as forças entre partículas são consideradas e outro módulo ao qual estas forças não são consideradas. O módulo de SPH é importante na simulação de fluidos mais densos, como líquidos, mas não é o módulo que apresenta a melhor performance.

As interacções entre partículas recorrendo ao modo de simulação SPH, utilizam um conjunto de campos de forças associados com cada partícula. O campo de força relacionado com a pressão tem o efeito de afastar as partículas e o campo relacionado com a viscosidade tem o efeito de reduzir a velocidade relativa entre partículas. O tipo de simulação para fluidos pode ser configurado da seguinte forma:

- NX_F_SPH – simulação de fluido, tendo em conta a interacção entre partículas.
- NX_F_NO_PARTICLE_INTERACTION – simulação de fluido sem interacção entre partículas.
- NX_F_MIXED_MODE – alterna a simulação entre os dois estados anteriores (oferecendo mais performance que o SPH, mantendo algumas características densas).

A mudança do tipo de interacção de um fluido durante uma simulação pode levar a resultados instáveis. A tensão da superfície é modelada através de forças atractivas entre partículas. No interior do fluido a tensão da superfície é balanceada através das forças. Mas as partículas que

se situam na superfície do fluido actuam na direcção oposta à normal da superfície, para reduzir a curvatura geral da superfície.

Durante a simulação de um fluido a massa de cada partícula permanece constante, mas a sua densidade altera-se durante o tempo, devido a compressibilidade do fluido. A mudança desta densidade é alcançada devido ao movimento das partículas. As áreas onde existe uma grande densidade no fluido são consequência de um grande número de partículas juntas. Neste momento, diferentes tipos de fluidos não interagem entre si.

2.5.6. Criar um fluido

Para criar um fluido, basta chamar a função NxScene::createFluid() com o descritor associado. Existem essencialmente duas formas de adicionar partículas ao fluido, especificar os valores manualmente, como a velocidade, posição, entre outros ou utilizando um emissor. Podemos especificar o estado inicial das partículas usando a estrutura NxParticleData e o membro do descritor NxFluidDesc::initialParticleData. Se não desejarmos criar um conjunto inicial de partículas, basta apenas iniciar a quantidade a simular indicando o valor no membro numParticlesPtr e iniciar initialParticleData a NULL. Para obter os resultados da simulação temos de especificar um buffer de escrita (particleWriteData), este buffer pode ser utilizado para algoritmos de desenho ou actualização das partículas. Se desejarmos a especificação inicial do estado das partículas é permitido utilizar o mesmo buffer nessa iniciação (NxParticleData) como buffer de escrita (particleWriteData). Ao criar um fluido para serem actualizados são os seguintes campos:

- bufferPos: a posição de cada partícula individualmente;
- bufferVel: a velocidade de cada partícula
- bufferLife: a duração de vida de cada.

O buffer de escrita é apenas actualizado durante a simulação. Podemos adicionar mais partículas a simulação manualmente mas existe um limite no número de partículas que podem ser adicionadas desta forma, dependendo do número existente actualmente na simulação. Existe um limite máximo de partículas que podem ser adicionadas por cada quadro de desenho, originalmente é de 4096.

A simulação de fluidos pode ser feita em software (CPU ou em hardware (GPU), para escolher esta opção basta apenas activar a opção NX_FF_HARDWARE no descritor do fluido. A simulação em software vai ser consideravelmente mais lenta do que hardware, especialmente em cenas que contenham um grande número de objectos dinâmicos.

2.5.7. Emissores de fluidos

A especificação de um emissor ou um conjunto de emissores de um fluido fornece uma forma alternativa para criar partículas. Estes podem ser utilizados para criar uma grande variedade de efeitos. Existem dois módulos de simulação quando se utiliza emissores:

- Pressão constante: neste caso, o estado do fluido é analisado, o emissor tenta igualar o estado das partículas ao estado do fluido que se encontra em repouso na sua posição espacial. É possível desta forma criar jactos de água.
- Fluxo constante: neste caso, o emissor apenas mantêm uma emissão constante de partículas a cada quadro.

Para activar ou desactivar um emissor, basta alterar a opção NX_FEF_ENABLED no descritor do fluido. Ao criar um emissor, utiliza-se um descritor associado (NxFluidEmitterDesc) ao qual se especifica o seu modulo e em seguida deve-se adicionar ao fluido através da função NxFluid::createEmitter(). Alternadamente poderá revelar-se mais fácil adicionar um conjunto de emissores quando estamos a criar um fluido.

Parâmetros importantes na especificação de um emissor:

- *relPose*: posição relativa a forma do actor;
- *type*: tipo de emissor, pressão constante (NX_FE_CONSTANT_PRESSURE) ou fluxo constante (NX_FE_CONSTANT_FLOW_RATE);
- *shape*: forma do emissor, pode ser rectangular (NX_FE_RECTANGULAR) ou elíptica (NX_FE_ELLIPSE);

- *particleLifeTime*: tempo de vida associado as partículas, se for introduzido o valor 0, então a partícula vive até colidir com o dreno.
- *maxParticles*: numero máximo de partículas que são emitidas por este emissor. O emissor cria novas partículas até ao limite indicado, assim que o limite for atingido este irá parar a sua emissão e inicia uma nova emissão assim que o número de partículas activas baixar do limite imposto.
- *fluidVelocityMagnitude:* a magnitude de velocidade ao qual as partículas são iniciadas, a velocidade máxima de cada partícula está associada com o parâmetro NxFluid::motionLimitMultiplier;
- *rate*: controlo de emissão de partículas num segundo, este parâmetro só tem significado se o tipo de simulação for de fluxo constante (NX_FE_CONSTANT_FLOW_RATE).
- *Dimension(X/Y)*: tamanho do emissor, nas direcções do 1º e 2º eixo da sua orientação.
- *randomAngle:* intervalo do desvio padrão da direcção de emissão. A direcção de emissão é especificada pelo 3º vector de orientação.
- *randomPos:* limite da posição aleatória em que as partículas podem ser criadas. O valor z especifica o limite da aleatoriedade ao longo da direcção da emissão.