

Biblioteca Modular para Construção de Sistemas de Partículas em Ambiente Java

Michael Janner Marques Marcelo de Gomensoro Malheiros

Centro Universitário UNIVATES

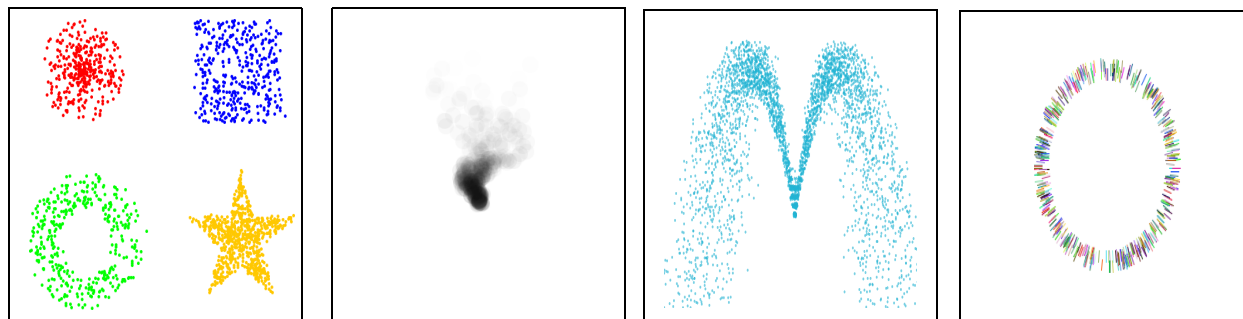


Figura 1: Exemplos de sistemas de partículas criados pela biblioteca.

Resumo

Java é uma plataforma que se tornou bastante popular para o desenvolvimento de aplicações, tanto comerciais quanto para entretenimento, como é o caso de jogos. Este trabalho descreve uma biblioteca de classes Java que tem por objetivo simular uma grande variedade de sistemas de partículas. Com base nas funcionalidades que a biblioteca provê, um jogo pode incorporar efeitos que imitam fenômenos físicos como fumaça, fogo, neve, explosões ou chuva. Esta biblioteca é relevante pois recursos como estes não estavam anteriormente disponíveis de forma modular e em código aberto na plataforma Java, obrigando que desenvolvedores sempre implementassem seus próprios sistemas de partículas.

Palavras-chave: sistemas de partículas, simulação, plataforma Java

Contato:

michaelycus@gmail.com
mgm@univates.br

1. Introdução

A plataforma Java destacou-se bastante na última década por estar disponível em um grande número de sistema operacionais e dispositivos, permitindo ao desenvolvedor executar a mesma aplicação em diferentes ambientes.

Ao longo dos anos esta tecnologia incorporou um grande número de desenvolvedores. Em particular, muitos desses desenvolvedores utilizaram Java para a criação de jogos. E apesar dos programadores contarem com a ajuda de *engines* para facilitar o desenvolvimento de aplicações, muitos recursos ainda não estão livremente disponíveis, forçando o desenvolvedor muitas vezes ter de implementar

novamente funcionalidades bastante úteis, como é o caso para sistema de partículas.

Sistemas de partículas são largamente utilizados na Computação Gráfica e geralmente não têm interação com os personagens do jogo, mas são usados para criar efeitos que aumentam o realismo e a diversão do mesmo. Efeitos como fumaça, explosões, fogo e chuva são facilmente criados a partir de um sistema de partículas. Este trabalho apresenta a implementação de uma biblioteca modular para simulação de sistemas de partículas na plataforma Java. O objetivo é servir de apoio ao desenvolvedor que deseja utilizar efeitos animados complexos em suas aplicações.

A Seção 2 mostra como funciona tipicamente um sistema de partículas. A Seção 3 descreve trabalhos que serviram de inspiração para esta biblioteca. A Seção 4 faz a descrição da implementação da mesma, explicando os conceitos de cada elemento envolvido no sistema de partículas. Essa seção também descreve as principais funcionalidades desenvolvidas. A Seção 5 brevemente relata os resultados obtidos, enquanto na Seção 6 são traçadas as conclusões deste trabalho, enfatizando as contribuições e sugerindo aprimoramentos futuros.

2. Sistemas de Partículas

Sistemas dinâmicos de partículas foram introduzidos por Reeves [1983]. Segundo este autor, um sistema de partículas representa um objeto como uma nuvem de primitivas que definem o seu volume. Em um período de tempo, partículas são geradas em um sistema, deslocadas e têm sua forma alterada dentro do mesmo, para então desaparecer. Partículas geralmente são representadas por um ponto, por motivos de economia computacional. Entretanto, podem ser representadas por um quadrado, um cubo, uma imagem, ou qualquer outra forma. Também por motivos de economia de processamento, partículas não colidem entre si e não são influenciadas por ações de renderização, como

sombra e refração. Entretanto, partículas podem interagir com um corpo externo, por exemplo, colidindo contra uma parede ou acumulando-se sobre uma superfície.

Um sistema de partículas simples é geralmente formado por três entidades principais:

Emissor – lugar no espaço onde as partículas são geradas. Um emissor pode ser um ponto, uma curva ou um objeto qualquer. O emissor também controla a direção, velocidade e idade inicial da partícula.

Partícula – é a partícula em si, com seus atributos.

Força – é a ação que faz a partícula alterar seu movimento, afetando o comportamento do sistema.

Cada sistema de partículas pode ser descrito por parâmetros distintos de cada uma destas entidades. Dependendo da configuração desses parâmetros, o modelo resultante muda.

3. Trabalhos Relacionados

A biblioteca apresentada neste trabalho foi inspirada na implementação desenvolvida por McAllister [2000], descrita a seguir. Além desta, são conhecidas poucas implementações abertas e disponíveis de sistemas de partículas, citadas na Seção 3.2.

3.1 Particle System API

A Particle System API foi implementada em C++ e descrita em [McAllister 2000]. Esta vem sendo desenvolvida como código aberto desde sua criação (estando disponível em www.particlesystem.org).

Segundo McAllister [2000], nesta API uma partícula é um objeto com um conjunto de atributos muito semelhantes àqueles originalmente introduzidos em [Reeves 2003]. Foi desenvolvida com alguns objetivos específicos de projeto, entre os quais pode-se citar: eficiência para uso em tempo real, flexibilidade para a criação de efeitos e facilidade de aprendizado.

Nessa API, todas as partículas existem dentro de um grupo, que é um conjunto de partículas modificadas pelas mesmas forças. Muitos grupos podem existir ao mesmo tempo, mas todas as funções da API são aplicadas ao grupo selecionado no momento. Todas as ações são aplicadas para as partículas deste grupo.

Ações são as funções na API que modificam os atributos das partículas do grupo corrente. A maior parte das ações é usada para simular forças físicas como, por exemplo, gravidade. Outras alteram o aspecto visual da partícula. As ações foram desenvolvidas para funcionar em blocos, de maneira a criar efeitos especiais, e não foram projetadas para reproduzir efeitos individualmente. A maioria dos efeitos consiste de três a oito ações que se unem.

Domínios proporcionam um mecanismo uniforme para representar uma região tridimensional na API. Em outras palavras, domínios são regiões onde ações são empregadas em partículas que ali se encontram. Por exemplo, em determinado domínio partículas são criadas com uma série de atributos. Em outro domínio, partículas têm a direção invertida, sendo rebatidas. Já

em outro domínio, partículas podem morrer, como se atingissem um limite. Domínios são então usados como parâmetros de algumas ações, podendo ter diversas formas geométricas.

Como dito anteriormente, a Particle System API foi desenvolvida em C++, porém sem empregar orientação a objetos, sendo preciso continuamente alterar o estado da API. Seguiu o modelo empregado em OpenGL. Entretanto, esta abordagem de máquina de estados não foi julgada adequada para uso em aplicações Java.

3.2 Outras Implementações

A Java Particle System Library (JOPS) é uma biblioteca aberta desenvolvida em Java. Possui um editor gráfico que possibilita a criação e edição interativa de efeitos 3D diversos [Gomes 2008], porém não vem sendo atualizada desde 2008 e não tem seu código ou utilização documentados.

Também para Java, a empresa Indie Technologies desenvolveu uma solução comercial chamada Genesis FX Particle System, sendo que uma versão gratuita para testes está disponível [Indie 2007].

Davison [2005] tem um capítulo dedicado à criação de um sistema de partículas simples com Java 3D.

Ilmonen e Kontkanen [2003] apresentam uma extensão do sistema clássico de partículas, na qual unifica partículas, fontes de partículas e geradores de forças em um sistema de partículas denominado de segunda ordem. Neste sistema as fontes de partículas e as forças geradoras são dependentes das forças, assim como as partículas visuais. A implementação de referência é escrita em C++. Dada a maior complexidade computacional de seu modelo, este acabou por não ser utilizado.

4. Implementação

Ao contrário da biblioteca desenvolvida por McAllister [2000], a biblioteca aqui apresentada utilizou-se dos recursos de orientação a objetos de Java, tanto para melhor organização do código como para promover a extensibilidade da mesma.

A decisão de restringir os efeitos de partículas a duas dimensões, e não três como na API, foi baseada no fato da grande maioria de jogos escritos em Java trabalharem em 2D. Além disso, a maioria das *engines* 3D normalmente já incorporam recursos de efeitos com partículas dentre as suas funcionalidades.

4.1 Partícula

Uma partícula na biblioteca é similar ao modelo de McAllister [2000]. Entretanto, na presente biblioteca foram adicionados mais elementos para caracterizar a partícula, entendendo-se que isso não representaria perda de desempenho e possibilitaria maior flexibilidade para o desenvolvimento de efeitos.

Alguns atributos incluídos na classe **Particle**, são: posição atual, posição anterior, posição alvo, tempo inicial, tempo final, atividade (**alive**) e dado (**data**).

Dos atributos apresentados anteriormente é interessante esclarecer alguns pontos: a posição anterior recebe o valor da posição atual imediatamente antes desta ser atualizada e a posição alvo é usada para indicar a posição que se deseja que a partícula alcance através de uma ação. O tempo inicial é definido na criação da partícula a partir do tempo do sistema e o tempo final é geralmente utilizado para indicar quando a partícula irá deixar de existir. O atributo **alive** é uma variável booleana que define se a partícula faz ou não mais parte do sistema. O atributo **data** é uma referência para o tipo **Object** do Java, sendo assim possível armazenar qualquer tipo de informação.

A classe **Particle** contém apenas métodos que definem e informam os valores de seus atributos.

4.2 Grupo de partículas

Todas as partículas de um sistema pertencem a um grupo, representado pela classe **ParticleGroup**.

Entre os principais atributos do grupo de partículas estão: delta t (**dt**) e domínio.

O propósito da simulação do sistema de partículas é calcular cada atributo da partícula – posição, tamanho, cor – à medida que estes variam no tempo. Para calcular a variação da posição da partícula ao longo do tempo o grupo utiliza o método de integração numérica de Euler, com o valor **dt** especificado pelo usuário para ajustar a exatidão do incremento. Então, quanto menor o valor de **dt** menor será o deslocamento da partícula e maior será a precisão no cálculo da posição. Domínio é a região no espaço onde as partículas serão criadas.

Cada grupo possui ainda uma série de ações que afetam as partículas nele incluídas, detalhadas a seguir.

4.3 Ações

Ações são as funções da biblioteca que modificam os atributos das partículas pertencentes ao grupo. As ações ficam inseridas em uma lista e sua ordem de execução depende da ordem de inserção dentro do grupo, ou seja, a primeira ação inserida no grupo irá modificar todas as partículas para então a próxima ação começar a ser executada. Uma única ação não é suficiente para representar um efeito particular. Os efeitos são geralmente criados com a combinação de pelo menos três ações diferentes e a ordem de execução das ações também pode afetar o efeito final.

Por uma questão de flexibilidade, optou-se por dividir as ações da seguinte maneira: **SourceActions**, que são ações de criação, e **ParticleActions**, que são ações que atuam sobre as partículas.

Quando novas partículas são criadas dentro de um grupo, essas possuem atributos nulos. As ações de criação são então responsáveis por definir os atributos iniciais das mesmas. Assim, atuam sobre as partículas uma única vez, dando o impulso inicial. Existem várias ações de criação, destacando-se: **ColorParticleSource**, que define uma cor inicial para as partículas do grupo; **EndTimeParticleSource**, que define quanto tempo a

partícula existirá; e **VelocityParticleSource**, que define a velocidade inicial para as partículas.

Depois de criadas e tendo seus atributos iniciais definidos, as partículas começam a sofrer alterações das ações sobre partículas. Diferente das ações de criação, as ações sobre partículas irão modificar o valor de cada partícula várias vezes, afetando a mesma enquanto essa existir no grupo. Destacam-se: **GravityParticleAction**, que atua sobre a partícula alterando sua velocidade em determinada direção; **BounceParticleAction**, que faz com que as partículas sejam rebatidas quando atingirem certa região do espaço definidas por um domínio; **KillParticleAction**, que faz as partículas morrerem quando alcançam determinado tempo de vida; e **SinkParticleAction**, que especifica uma região no espaço aonde as partículas serão mortas ao entrarem.

4.4 Gerenciador de Grupos

Muitos grupos de partículas podem existir de uma vez no sistema e todos eles são controlados por um gerenciador de grupos, definido por **GroupManager**. Este possui um método que executa o laço principal de operações da biblioteca, descrito pelo pseudocódigo apresentado abaixo:

para todos os grupos de partículas

***moverPartículas()** // atualiza posição das partículas*

***criarPartículas()** // cria e define valores iniciais*

***aplicarAções()** // aplica sobre todas as partículas*

***removerPartículasMortas()** // remove partículas*

fim para

// concluído o laço, as partículas são desenhadas na tela

É importante notar que, por uma questão de flexibilidade, a renderização das partículas foi deixada a cargo do programador. Isto se justifica pois é possível renderizar as partículas das mais diversas formas (linhas, círculos ou até mesmo imagens), o que é diretamente dependente da aplicação.

4.5 Domínios

Nesta biblioteca domínios são fundamentalmente utilizados para representar regiões do espaço 2D.

A principal função do domínio é definir onde as partículas surgirão, daí o motivo do grupo de partículas possuir um atributo deste tipo. Entretanto, domínios também podem ser usados por outras ações para, por exemplo, limitar determinada região onde as partículas não devem entrar.

Os domínios estendem a classe **Domain**, implementando os seguintes métodos: **createParticles**, que cria as partículas no interior do domínio; **contains**, que testa se determinado ponto está contido no domínio; e **bounce**, para calcular o efeito de rebatida após uma colisão.

No momento, a biblioteca conta com domínios que implementam um círculo, um retângulo, um anel e um polígono.

4.6 Estrutura da Biblioteca

A estrutura da biblioteca foi dividida em pacotes e projetada com o objetivo de ser flexível a atualizações e inserção de novos recursos, conforme a Figura 2.

Os pacotes estão assim divididos: **data**, contém as classes elementares do projeto, como das partículas; **domain**, contém as classes que representam os domínios; **particleActions**, contém as ações que modificam os atributos das partículas; **sourceActions**, contém ações de criação, que modificam os atributos iniciais das partículas; e **util**, fornece algumas classes de apoio à biblioteca. Outros pacotes que não fazem essencialmente parte do sistema foram incorporados à estrutura, servindo de exemplo ao desenvolvimento de aplicações: **particleEffects**, classes que implementam efeitos que servem de referência para a criação de outros efeitos; **panes**, que contém implementações que também servem de modelo para a interface gráfica.

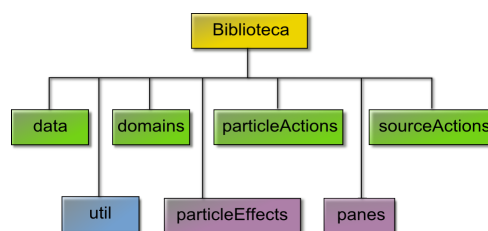


Figura 2: Estrutura de pacotes da biblioteca.

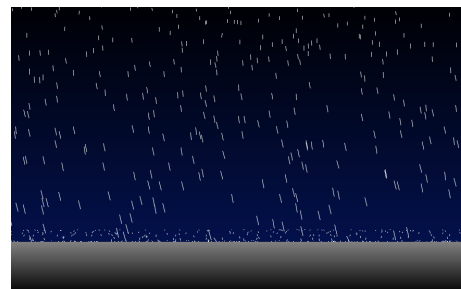


Figura 3: Resultado da simulação de chuva.

5. Resultados

A biblioteca está disponível sob licença GPL, sendo denominada **J2DParticles** e sendo hospedada em code.google.com/p/j2dparticles.

Para validação da biblioteca, vários efeitos complexos foram criados, ilustrados na Figura 1. Então mediu-se o desempenho, em quadros por segundo, da simulação ocorrendo em tela inteira em máquinas de processamento diferente, sempre com a mesma implementação (usando a classe **Graphics** do Java).

A título de exemplo, uma simulação de chuva é brevemente detalhada. Para renderizar as partículas utilizaram-se linhas que iniciam da posição atual, avançando até a posição anterior, o intensifica a noção de velocidade das gotas. Ao final, as partículas se chocam com uma superfície plana, desaparecendo logo em seguida. A Figura 3 mostra o resultado. Na Listagem 1, é exibido o código utilizado para criar todo o sistema de partículas.

```

// cria grupo de partículas
ParticleGroup pg = new ParticleGroup();
pg.setMaxParticles(2000);
pg.setCreationInterval(20);
pg.setRate(15);
// define domínio
pg.setSourceDomain(
    new RectangleDomain(-500,-20,1800,20));
// cria ações de criação
pg.addAction(new VelocityParticleSource(20, 0, 0, 0));
pg.addAction(new ColorParticleSource(Color.white));
// cria ações sobre partículas
pg.addAction(new GravityParticleAction(0, 8));
pg.addAction(
    new BounceRectangleParticleAction(0,600,1024,50,0.2));
pg.addAction(new KillParticleAction(3000));
  
```

Listagem 1: Código para simulação de chuva.

Os resultados dos testes em quatro máquinas é apresentado na Tabela 1. As últimas colunas apresentam o número médio de *frames* por segundo e o máximo de partículas criadas ao mesmo tempo.

Sistema	Processador	Memória	Vídeo	FPS	Max
Windows 7	Core 2 Duo E6750 2,66 GHz	2 GB	ATI RADEON 4870 512 MB	32	1470
Windows 7	Core 2 Quad Q8200, 2,33 GHz	4 GB	nVidia GeForce GTX275 896MB	32	1469
Open Suse 11.1	Athlon 64 3200+	2,0 GB	nVidia GeForce 6600 LE	16	749
Windows XP	Intel Pentium 4 1,8GHz	1,0 GB	nVidia GeForce FX5200	19	900

Tabela 1: Testes executados.

6. Conclusões

Este trabalho apresentou a implementação de uma biblioteca de sistemas de partículas que possibilita a criação de variados efeitos de animação em aplicações Java. Esta apresenta uma estrutura bastante flexível que permite que novas ações e domínios sejam criados.

Referências

- DAVISON, A., 2005. *Killer Game Programming in Java*. O'Reilly Media, 1st edition.
- GOMES, G., 2008. *Java Particle System Library* [online]. Disponível em: code.google.com/p/jops [Acessado em 23 de maio de 2010].
- ILMONEN, T. e KONTKANEN, J., 2003. The Second Order Particle System. In: *WSCG Proceedings, 2003*.
- INDIE TECHNOLOGIES, 2007. [online]. Disponível em: <http://www.indietechnologies.com/> [Acessado em 15 de setembro de 2010].
- MCALLISTER, D.K., 2000. *The Design of an API for Particle Systems*. University of North Carolina, Chapel Hill.
- REEVES, W.T., 1983. Particle Systems: A Technique for Modeling A Class of Fuzzy Objects. In: *PROC. OF SIGGRAPH '83, Detroit, Michigan, July, 1983*.