Comparação de sistema de partículas

Jonas Costa Campos,
Thiago Gomes Vidal de Mello
Escola Politécnica, PUCRS,
Porto Alegre, Rio Grande do Sul
Email: jonas.campos@edu.pucrs.br , thiago.mello.001@acad.pucrs.br

***Keywords*-Animação; Sistema de Partículas.**

I. Introdução

Em 1983, foi criado o primeiro modelo de sistema de partículas[2], um método utilizado para simular objetos difusos, ou seja, que não tem um formato bem definido, como por exemplo nuvens, água ou fumaça. Neste modelo, Reeves define que cada partícula deve ter sete atributos:

- Posição inicial
- Velocidade inicial
- Tamanho inicial
- Cor inicial
- Transparência inicial
- Formato
- Tempo de vida

Com base nisso, decidimos criar um sistema de partículas responsivo a colisões na Unity[3] e comparar com o sistema que a própria *engine* oferece.

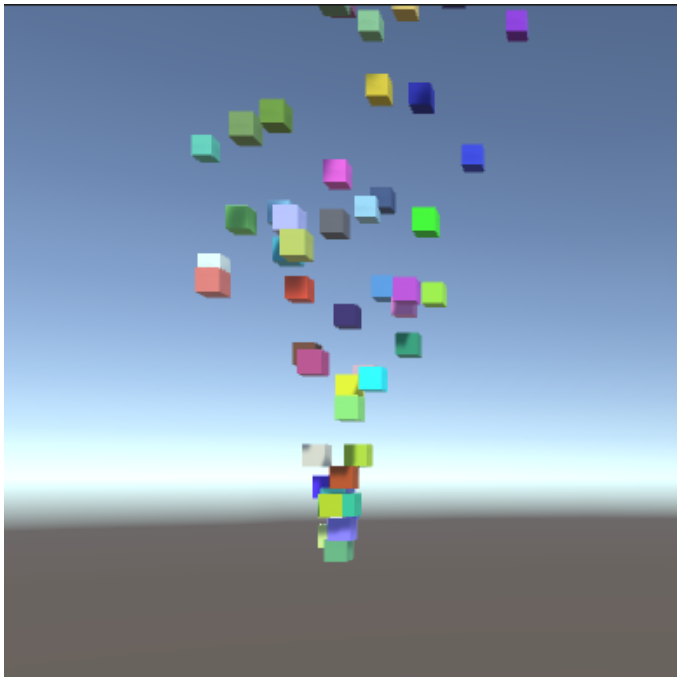

Figure 1. Sistema de partículas implementado

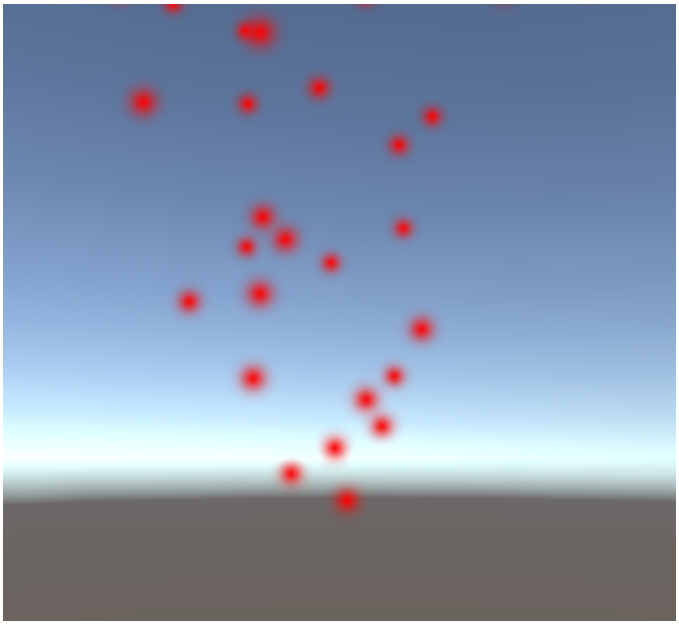

Figure 2. Sistema de partículas da Unity

II. Metodologia

O sistema construído foi programado em C Sharp e consiste em duas classes: CustomParticle e CustomParticleSystem. A primeira tem como objetivo guardar informações de uma partícula, sendo elas:

- Tempo de nascimento
- Tempo de vida
- Posição inicial
- Velocidade inicial
- Fator de aceleração
- Fator de desvio
- Tamanho
- Modificador de cor

Cada partícula recebe valores para estes itens quando instanciada, sendo eles gerados aleatoriamente dentro das faixas que são definidas pelo usuário. A classe da partícula não tem nenhuma função além de guardar estes dados e verificar seu limite de tempo, quando isto for requisitado.

Já a classe CustomParticleSystem é o sistema propriamente dito. O objeto do sistema possui uma estrutura de lista

de instâncias de objetos de partícula para gerenciar o conjunto. Esta é a classe que o usuário irá interagir diretamente e é a que controla o sistema de partículas, configurando faixas de valores para as variáveis mencionadas anteriormente, bem como de rotação e o uso da física de colisão ou até mesmo a gravidade da *engine*. Há dois laços de execução principais, utilizando o funcionamento da própria Unity: o método Update, que é invocada uma vez a cada quadro da execução e o método FixedUpdate, que executa uma vez a cada passo da física (configurado por projeto e possui intervalo padrão de 0,02 segundo, ou 50 vezes por segundo).

Dentro do método Update é feita a instanciação das partículas, dadas as regras de tempo e quantidade estabelecidas pela configuração do usuário. A criação de uma nova partícula é feita a partir do método InstantiateParticle, que aplica valores aleatórios dentro dos limites estabelecidos na configuração.

Já no método FixedUpdate é feito o gerenciamento das partículas. Em um laço que envolve todas as partículas, primeiro ela é excluída caso o tempo de vida tenha-se excedido e depois são feitas as alterações, caso não seja excluída, começando pelo cálculo da movimentação (utilizando o médoto de AddForce do componente de RigidBody da Unity), seguido pela modificação do tamanho e da cor.

Adicionalmente, foi implementado um suporte para funções customizadas pelo usuário, para isto, utilizamos a biblioteca mXparser [1]. Com esta funcionalidade o usuário pode criar caminhos utilizando funções matemáticas como, por exemplo, uma onda variando a coordenada Y utilizando seno de X (vide Figura3).

O sistema que foi implementado aceita qualquer objeto tratado como *prefab* que possua um componente do tipo Renderer, podendo ser tanto 2D como 3D. Desta forma é possível também fazer partículas a partir de objetos que possuam comportamentos customizados.

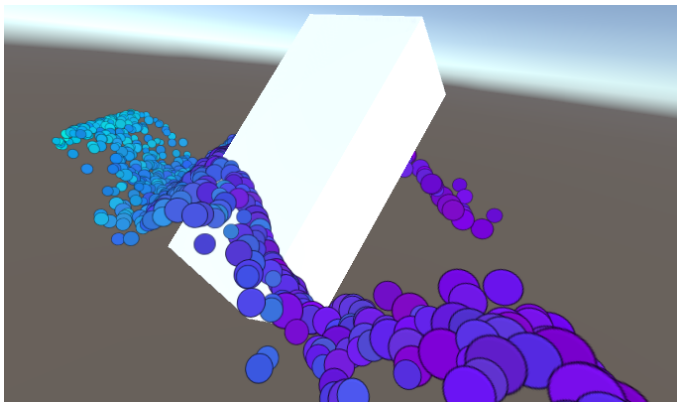

Figure 3. Partículas com trajetória aplicada à função seno

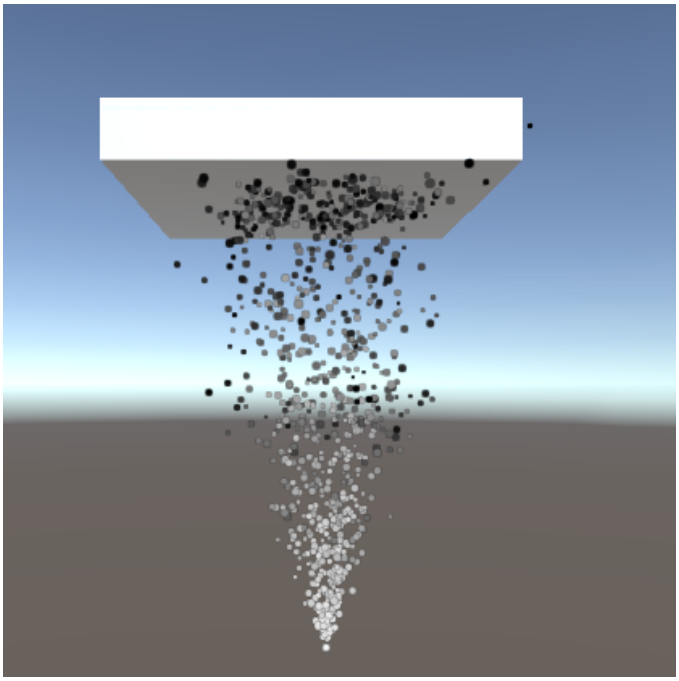

Figure 4. Partículas 2D com colisão e mudança de cor

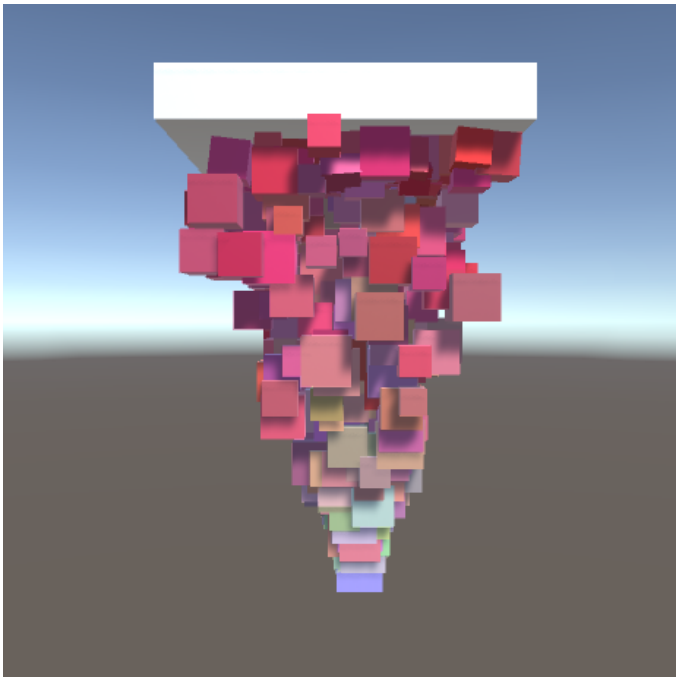

Figure 5. Partículas 3D com colisão e mudança de cor

III. Resultados

Os seguintes testes foram realizados utilizando um computador com processador AMD Ryzen 5 1400 Quad-Core, placa de vídeo NVIDIA GeForce GTX 1050 Ti e 16 GBs de memória RAM.

Executando o sistema de partículas baseado no modelo de Reeves para gerar 1000 partículas simultaneamente, alcançamos aproximadamente 125 *frames* por segundo com partículas 3D e aproximadamente 155 *frames* por segundo com partículas 2D, enquanto o sistema nativo da Unity alcança aproximadamente 390 *frames* por segundo.

Em testes utilizando colisão a quantidade de *frames* por segundo do sistema da Unity se mantém igual, enquanto que no nosso sistema, utilizando partículas 2D temos uma taxa de 75 *frames* por segundo e com partículas 3D temos uma taxa de 68 *frames* por segundo

O próximo teste utiliza a função de mudança de cor das partículas e novamente, a performance do sistema da Unity se mantém inalterada. Testamos isto com colisões ativadas e desativadas e podemos observar que, para partículas 2D, nosso sistema tem uma média de 25 *frames* por segundo com a colisão ativada e 38 *frames* por segundo sem colisão. Para partículas 3D, obtivemos uma média de 23 *frames* por segundo com colisão ativada e 30 *frames* por segundo sem colisão.

Comparação de sistemas de partículas		
	Nosso sistema	Sistema da unity
Com mudança de cor e com colisão	23 - 25 fps	390 fps
Sem mudança de cor e com colisão	68 - 75 fps	390 fps
Com mudança de cor e sem colisão	30 - 38 fps	390 fps
Sem mudança de cor e sem colisão	125 - 155 fps	390 fps

IV. Conclusão

Comparando os dois sistemas de partículas, podemos notar que apesar de muito completo, o sistema da Unity ainda obteve uma performance muito superior, pois mesmo no teste em que obteve os melhores resultados, o sistema implementado alcançou pouco menos de 40% do desempenho obtido no sistema da *engine*. Para o futuro podemos tentar descobrir maneiras de otimizar o nosso sistema para obter uma performance melhor, mesmo com as funções que utilizamos.

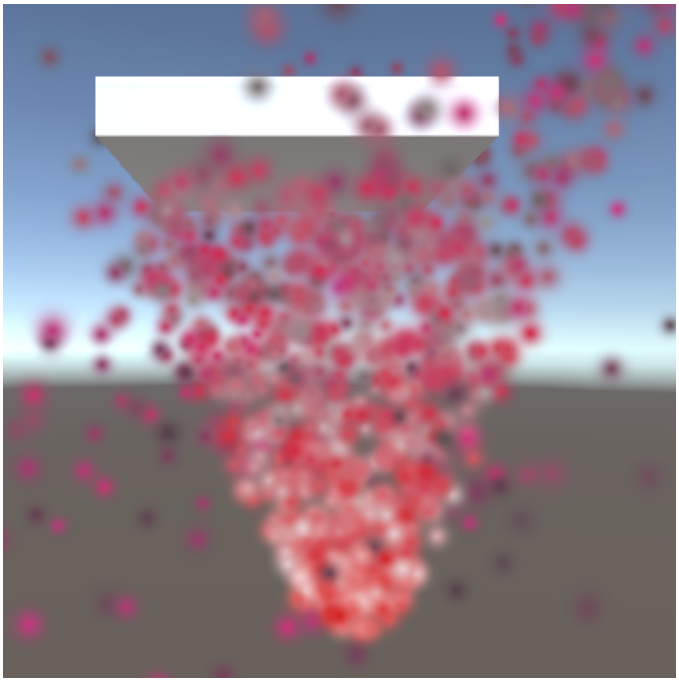

Figure 6. Sistema de partículas da Unity com colisão e mudança de cor

References

[1] MultiMedia LLC. *mXparser – Math Expressions Parser for JAVA, Android, C Sharp, .NET/MONO/Xamarin – Mathematical Formula Parser / Evaluator Library*. URL: https://mathparser.org/ (visited on 06/24/2022).

[2] William T Reeves. "Particle systems—a technique for modeling a class of fuzzy objects". In: *ACM Transactions On Graphics (TOG)* 2.2 (1983), pp. 91–108.

[3] Unity Technologies. *Unity*. URL: https://unity.com/ (visited on 06/27/2022).