Interacción y Animación: Simulación de partículas (fuego, lluvia, explosiones) con sistemas de partículas

Alexis J. Aguirre Narváez[1], Edwin Florez[2]
[1]*alexis.aguirre1@upr.edu*
[2]*edwin.florez@upr.edu*
Universidad de Puerto Rico, Recinto de Mayagüez

11/12/2025

Resumen

Presentamos un sistema de partículas modular desarrollado en C++17 y renderizado en tiempo real utilizando la biblioteca gráfica SFML 3.0. El motor se basa en una arquitectura genérica compuesta por partículas, emisores, fuerzas y un módulo opcional de interacción entre partículas, lo que permite simular fenómenos naturales como fuego, lluvia, fuentes de agua y explosiones. Además de demos independientes para cada fenómeno, se construyó una escena compuesta que integra el ascenso de un cohete de fuego, múltiples explosiones aéreas y una fuente de agua. Describimos la arquitectura del sistema, las clases principales, los detalles de implementación y los resultados visuales obtenidos, así como posibles líneas de trabajo futuro.

1. Introducción

Los sistemas de partículas son una técnica fundamental en gráficos por computadora para representar fenómenos complejos, tales como fuego, humo, lluvia, chispas o efectos atmosféricos, a través de un conjunto de entidades simples que siguen reglas locales [1, 2]. En lugar de resolver ecuaciones completas de fluidos o gases, se modela una gran cantidad de partículas discretas con propiedades básicas (posición, velocidad, vida) cuyo comportamiento colectivo genera el efecto visual deseado.

En este trabajo describimos el diseño e implementación de un sistema de partículas escrito en C++17 y visualizado con SFML 3.0. El sistema está construido alrededor de un motor central (`ParticleSystem`) que administra partículas y aplica fuerzas, mientras que los fenómenos específicos se encapsulan en emisores (*emitters*) intercambiables: `FireEmitter`, `RainEmitter` y `ExplosionEmitter`.

El objetivo general es:

> Diseñar e implementar un sistema de partículas modular y extensible en C++ que permita simular fenómenos naturales (fuego, lluvia/fuente, explosiones) en tiempo real, utilizando SFML 3.0 como motor gráfico.

A partir de este objetivo se plantean metas específicas: crear una arquitectura basada en interfaces, implementar fuerzas físicas simples (gravedad, turbulencia), integrar interacción entre partículas y demostrar el sistema en demos independientes y en una escena combinada.

2. Metodología

En esta sección se describe la arquitectura del sistema, las clases principales, la representación de

partículas, las fuerzas externas, el modelo de interacción entre partículas, así como detalles de implementación y escenas de demostración.

2.1. Arquitectura General

La arquitectura se organiza en torno a cuatro componentes principales:

- **Partículas** (`Particle`): entidades básicas con estado físico.
- **Emisores** (`IEmitter` y derivados): generan nuevas partículas.
- **Fuerzas** (`IForce` y derivados): modifican la dinámica.
- **Motor central** (`ParticleSystem`): administra el ciclo de simulación.

La comunicación se realiza mediante interfaces abstractas para emisores y fuerzas, lo que permite extender el sistema con nuevos comportamientos sin modificar el núcleo.

2.2. Representación de Partículas y Vectores

Cada partícula se modela mediante la clase `Particle`, que almacena al menos:

- posición (`Vec2 position`),
- velocidad (`Vec2 velocity`),
- tiempo de vida restante (`float life`),
- vida máxima (`float maxLife`).

La estructura `Vec2` representa vectores 2D y se utiliza para posiciones, velocidades y fuerzas. Esto simplifica la implementación de operaciones vectoriales básicas (suma, escala, normalización).

2.3. Clase `ParticleSystem`

`ParticleSystem` actúa como motor central, con las siguientes responsabilidades:

- almacenar un conjunto de partículas en un arreglo de tamaño fijo,
- recibir nuevas partículas emitidas mediante `emit(const Particle& p)`,
- registrar emisores (`addEmitter(IEmitter*)`) y fuerzas (`addForce(IForce*)`),
- actualizar todas las partículas cada frame, aplicando fuerzas y decrementando `life`,
- descartar partículas muertas,
- gestionar colisión simple con el suelo mediante `enableGround(y, restitution)`,
- ejecutar, de forma opcional, un paso de interacción entre partículas (repulsión local).

El método `update(float dt)` coordina estas tareas para cada intervalo de tiempo.

2.4. Interfaces `IEmitter` y `IForce`

Para favorecer la extensibilidad, se definieron interfaces genéricas para emisores y fuerzas. En LaTeX usamos `listings` para ilustrar fragmentos sin romper la maquetación:

Listing 1: Interfaz genérica de emisores.

```
struct IEmitter {
   virtual ~IEmitter() = default;
   virtual void emit(ParticleSystem&
      system,
                float dt) = 0;
};
```

y de forma análoga para fuerzas:

Listing 2: Interfaz genérica de fuerzas.

```
struct IForce {
   virtual ~IForce() = default;
   virtual void apply(Particle& p,
                 float dt) = 0;
};
```

Cualquier clase que herede de estas interfaces puede integrarse fácilmente al sistema. Esto permite que `ParticleSystem` sea agnóstico al tipo específico de emisor o fuerza.

2.5. Emisores Específicos

2.5.1. FireEmitter

`FireEmitter` genera partículas en una posición fija (por ejemplo, la base de un cohete). Las partículas se emiten con una velocidad predominantemente vertical ascendente y un ángulo aleatorio en torno a la vertical para simular combustión turbulenta. La vida útil es corta y se mapeará posteriormente a gradientes de color cálidos (amarillo, naranja, rojo) en la fase de dibujado.

2.5.2. ExplosionEmitter

`ExplosionEmitter` crea chispas radiales a partir de un punto central. Cada partícula se emite con una dirección uniforme en $[0, 2\pi)$ y una magnitud de velocidad aleatoria. Los tiempos de vida varían también para generar un comportamiento visual rico. La gravedad descendente hace que las chispas eventualmente caigan.

2.5.3. RainEmitter (Lluvia y Fuente)

`RainEmitter` es responsable de la simulación de agua en dos modos:

Modo lluvia. Las partículas nacen en una banda horizontal amplia, ubicada por encima del área visible. La velocidad inicial apunta hacia abajo con ligera variación horizontal. La gravedad es alta para simular gotas rápidas; opcionalmente se activa rebote en el suelo.

Modo fuente. El mismo emisor se reconfigura para funcionar como una fuente:

- el área de emisión se reduce a una región pequeña en la parte inferior (boca de la fuente),
- la velocidad base apunta hacia arriba,
- se distinguen dos subconjuntos de partículas: *chorro central* (ángulo estrecho, velocidad alta) y *spray* (ángulo más amplio, velocidad algo menor),
- la gravedad y la turbulencia definen trayectorias parabólicas y dispersión lateral.

La implementación concreta crea un `Particle p` local y lo pasa a `ParticleSystem::emit`, asignando posición aleatoria dentro del área de emisión y velocidad según los parámetros de chorro o spray.

2.6. Fuerzas: Gravedad y Turbulencia

2.6.1. GravityForce

`GravityForce` aplica una aceleración constante $\vec{g}$ al vector velocidad de cada partícula:

$$\vec{v}_{t+dt} = \vec{v}_t + \vec{g}\,dt.$$

Dependiendo del fenómeno:

- fuego: gravedad con componente vertical negativa (flotación hacia arriba),
- fuente y lluvia: gravedad positiva hacia abajo, de mayor magnitud,
- explosiones: gravedad moderada aplicada tras la expansión inicial.

2.6.2. TurbulenceForce

`TurbulenceForce` introduce una componente de ruido controlada por un parámetro de intensidad. Esto rompe alineaciones perfectas y aporta naturalidad tanto al fuego como al agua y a las chispas.

2.7. Interacción entre Partículas

Para mejorar la estabilidad visual en regiones densas, `ParticleSystem` incluye un módulo opcional de interacción. El modelo es de repulsión radial: si dos partículas se encuentran dentro de un radio R, se aplica una fuerza

$$\vec{F}_{ij} = k\left(1 - \frac{d}{R}\right)\hat{u},$$

donde d es la distancia entre partículas, k es una constante de intensidad y $\hat{u}$ es el vector unitario que apunta de una partícula a la otra. La interacción se activa mediante:

```
system.setInteractionsEnabled(true);
system.setInteractionRadius(R);
system.setInteractionStrength(k);
```

En el fuego se usan radios pequeños ($R \approx 15$–18) e intensidades moderadas ($k \approx 50$–70) para evitar solapamientos en la base sin abrir demasiado la llama. En la fuente de agua se utilizan radios mayores ($R \approx 25$–30) e intensidades más notables ($k \approx 120$–150) para estabilizar el chorro y dar volumen al spray.

2.8. Implementación y Herramientas

El sistema se implementó en C++17. SFML 3.0 se utiliza para:

- crear la ventana principal (`sf::RenderWindow`),
- controlar el bucle de eventos,
- calcular el *delta time* con `sf::Clock`,
- dibujar partículas (círculos o rectángulos) y aplicar rotaciones.

CMake estructura la construcción del proyecto y genera varios ejecutables independientes:

- `particles` (demo de fuego),
- `particles_rain` (lluvia/fuente),
- `particles_explosion` (explosiones),
- `particles_scene` (escena combinada).

La organización del código separa cabeceras (`include/core`), fuentes (`src`) y artefactos de compilación (`build`).

2.9. Escenas de Demostración

2.9.1. Demo de Fuego

Utiliza `ParticleSystem`, `FireEmitter`, `GravityForce` (ascendente), `TurbulenceForce` y el módulo de interacción suave. Cada partícula se representa como un círculo y su color se interpola desde blanco/amarillo hasta naranja/rojo según la fracción de vida restante.

2.9.2. Demo de Lluvia/Fuente

En modo lluvia, las partículas se dibujan como rectángulos alargados orientados según la velocidad. En modo fuente, se usan círculos pequeños que describen trayectorias parabólicas y rebotan en el suelo para simular salpicaduras.

2.9.3. Demo de Explosiones

Cada detonación genera una nube de chispas que se dispersan radialmente y cambian de color a medida que pierden energía. La gravedad produce una caída progresiva que refuerza la sensación de explosión aérea.

2.9.4. Escena Combinada

La escena `particles_scene` integra:

1. fuego que simula el ascenso de un cohete,
2. tres explosiones secuenciales en la parte alta de la pantalla,
3. una fase final con la fuente de agua activada.

Cada sistema se actualiza con su propio conjunto de fuerzas e interacción, demostrando la flexibilidad del motor.

3. Resultados

Las simulaciones se ejecutan en tiempo real con miles de partículas activas por sistema sin pérdida significativa de rendimiento en un entorno de escritorio convencional.

El fuego muestra una columna fluctuante con base densa y extremo superior difuso, coherente con un efecto de combustión estilizada. La interacción entre partículas ayuda a evitar solapamientos visibles en la zona de mayor densidad.

Las explosiones producen chispas radiales con expansión rápida y posterior caída, con gradientes de color que sugieren enfriamiento progresivo. La secuencia de tres explosiones en la escena combinada aumenta el impacto visual.

La fuente de agua presenta un chorro central pronunciado y un spray lateral distribuido. La combinación de gravedad fuerte, turbulencia e interacción genera trayectorias parabólicas creíbles y una región de impacto con rebotes que simulan salpicaduras.

3.1. Análisis Técnico

3.1.1. Rendimiento

El sistema mantiene tiempo real con miles de partículas. El cuello de botella proviene del módulo de interacción $O(N^2)$.

3.1.2. Complejidad

- Fuerzas: O(N)
- Emisión: O(P)
- Interacción: $O(N^2)$

3.1.3. Discusión

La arquitectura modular facilita la extensión hacia fluidos SPH, campos de fuerza o shaders.

3.2. Fenómenos Simulados

3.2.1. Fuego

- Emisor puntual.
- Fuerza ascendente (flotación).
- Turbulencia suave.
- Interacción para evitar aglomeración.

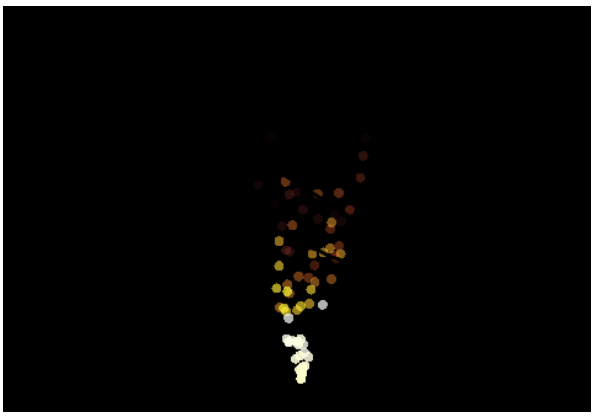

Figura 1: Resultado visual del fuego.

3.2.2. Fuente de agua

- Emisor puntual modificado.
- Dos modos: chorro estrecho y spray lateral.
- Gravedad dominante.

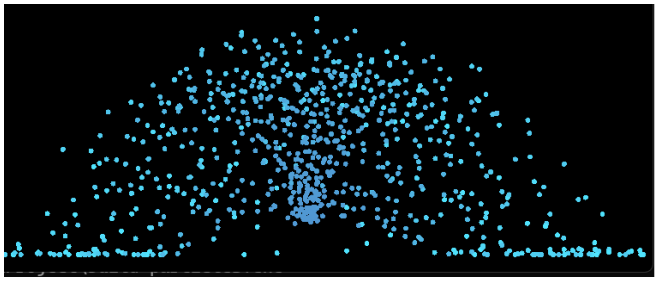

Figura 2: Resultado de la fuente.

3.2.3. Explosión

- Emisión radial desde un punto.
- Velocidad inicial aleatoria.

4. Trabajos Futuros y Aplicaciones Potenciales

El sistema desarrollado constituye una base sobre la que se pueden explorar diversas extensiones e investigaciones futuras:

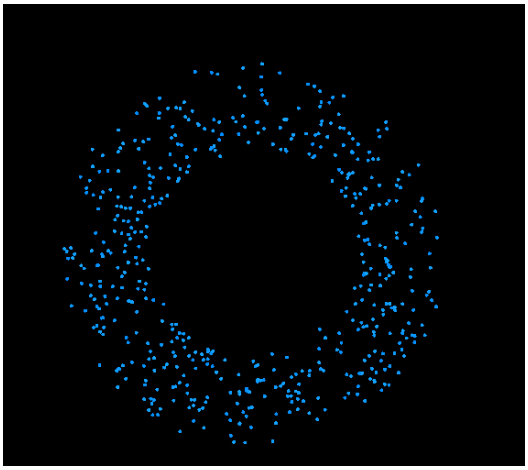

Figura 3: Resultado explosión simulada.

4.1. Nuevas Fuerzas y Fenómenos

- Incorporar fuerzas de vórtice o campos direccionales para simular remolinos, tormentas de arena o humo arrastrado por ventiladores.
- Modelar fuerzas atractivas (gravitación local) para simular enjambres, bandadas o partículas orbitando alrededor de un núcleo.
- Extender la interacción actual, basada en repulsión, a modelos de cohesión y alineamiento, conectando con simulaciones tipo *boids*.

4.2. Integración con Shaders y GPU

- Migrar parte de la simulación o del renderizado a la GPU mediante shaders, permitiendo manejar órdenes de magnitud mayores de partículas.
- Aplicar texturas y *billboards* para cada partícula, creando efectos visuales más realistas (fuego volumétrico, humo, niebla).

4.3. Aplicaciones Interactivas y Educativas

- Integrar controles en tiempo real para que estudiantes ajusten parámetros (gravedad, intensidad de emisores, turbulencia) y observen su impacto.
- Usar el sistema como base de un laboratorio virtual de gráficos por computador, donde se puedan implementar nuevos emisores como ejercicios.

4.4. Aplicaciones en Videojuegos y Visualización

- Emplear el motor como módulo de efectos especiales en prototipos de videojuegos 2D (explosiones, magia, clima dinámico).
- Integrar el sistema en herramientas internas de previsualización de efectos para experimento MonteCarlo o simulaciones de materia activa.

Estas líneas muestran que el proyecto no solo tiene valor como práctica académica, sino como plataforma de experimentación y desarrollo posterior.

5. Conclusión

Hemos presentado un sistema de partículas modular en C++17 que, mediante la composición de emisores, fuerzas e interacción entre partículas, permite simular diversos fenómenos naturales en tiempo real. La utilización de SFML 3.0 simplifica el renderizado y la gestión del bucle gráfico, mientras que CMake facilita la organización en múltiples demos ejecutables.

La arquitectura basada en interfaces hace posible agregar nuevos efectos con un esfuerzo incremental reducido. El módulo de interacción, aunque simple, mejora significativamente la estabilidad visual en fenómenos densos como fuego y fuentes de agua.

Como trabajo futuro se plantea la integración de nuevos tipos de fuerzas, el uso de shaders y GPU, y la aplicación del motor en contextos interactivos, educativos y de videojuegos.

Referencias

[1] W. T. Reeves. Particle Systems – A Technique for Modeling a Class of Fuzzy Objects. In *Proceedings of SIGGRAPH*, 1983.

[2] R. Bridson. *Fluid Simulation for Computer Graphics*. CRC Press, 2nd edition, 2015.

[3] M. Saito and T. Takahashi. Comprehensible Rendering of Particle Systems. In *Proceedings of SIGGRAPH Asia*, 1991.

[4] SFML Development Team. *SFML 3.0 Documentation*. `https://www.sfml-dev.org/documentation/3.0/`.