Particle System

Contents

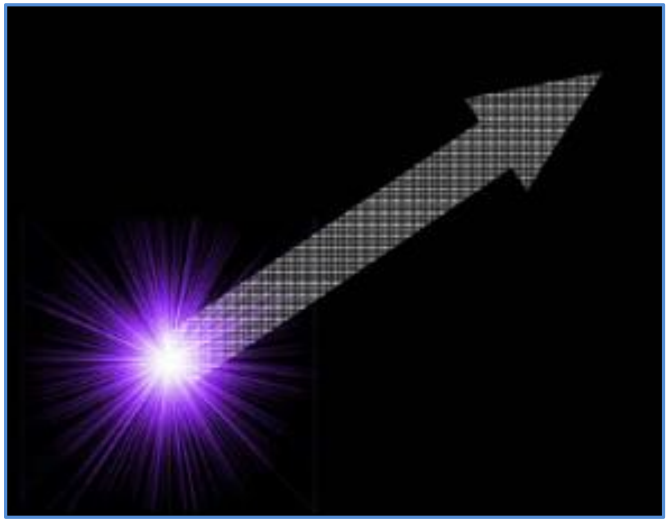
- **Introduction**
  - System Configuration
- **Collision Detection and Response**
  - Algorithm: Collision and Response
  - Acceleration Method
- **Rendering Methods**
  - Basic Rendering
  - Texture Mapping
  - Environment Mapping
  - Ray Tracing
  - Fuzzy Particles

Introduction

What is Particle System?

- A particle system is collection of individual elements (particles)
  - Controls a set of particles which act autonomously but share some common attributes. Ex:
    - **Position(2D/3D)**
    - **Velocity (vector: speed and direction)**
    - **Color+(transparency)**
    - **Life time**
    - **Size**
    - **Shape**
    - **Etc.**

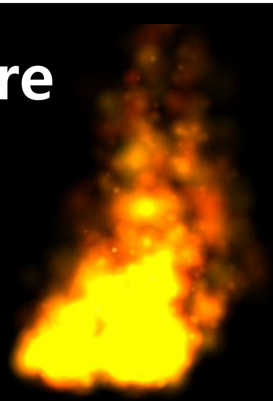
What control handles do we want/need?



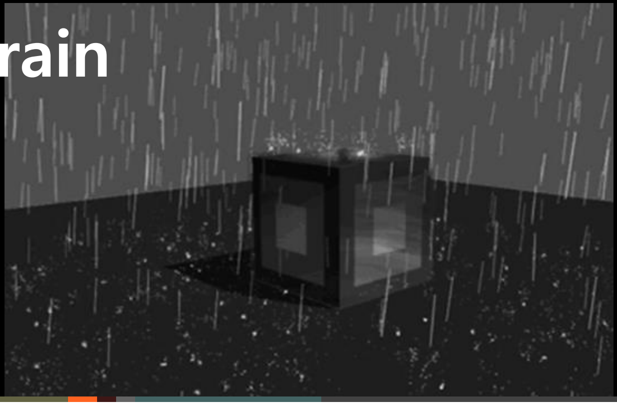
Particle System Applications

- Particle system is the solution to modeling **fuzzy, amorphous(changeable),dynamic and fluid objects** like:

smoke


fire

sparks

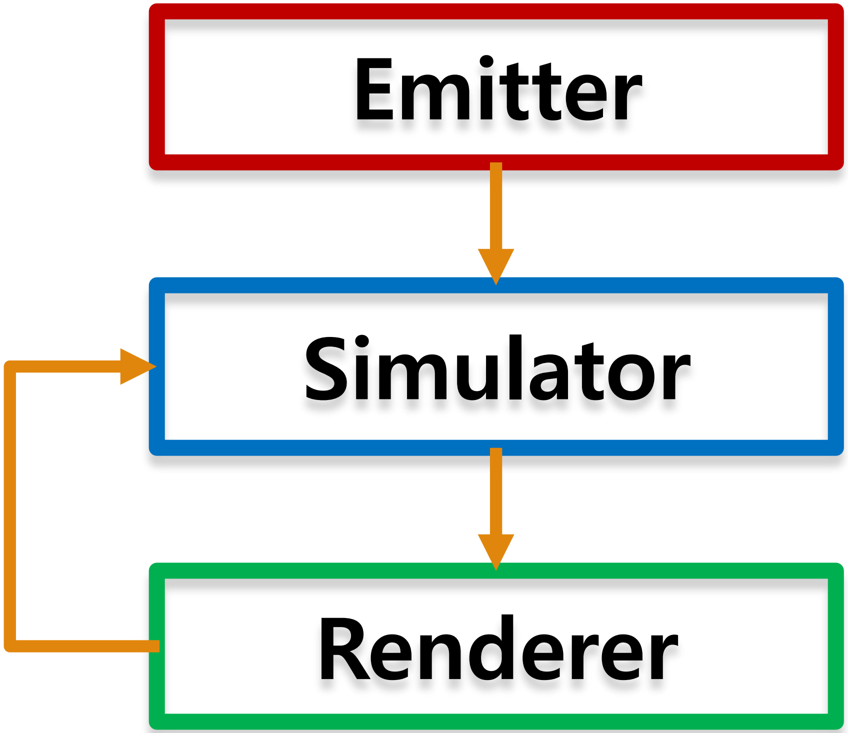

rain

snow

galaxies

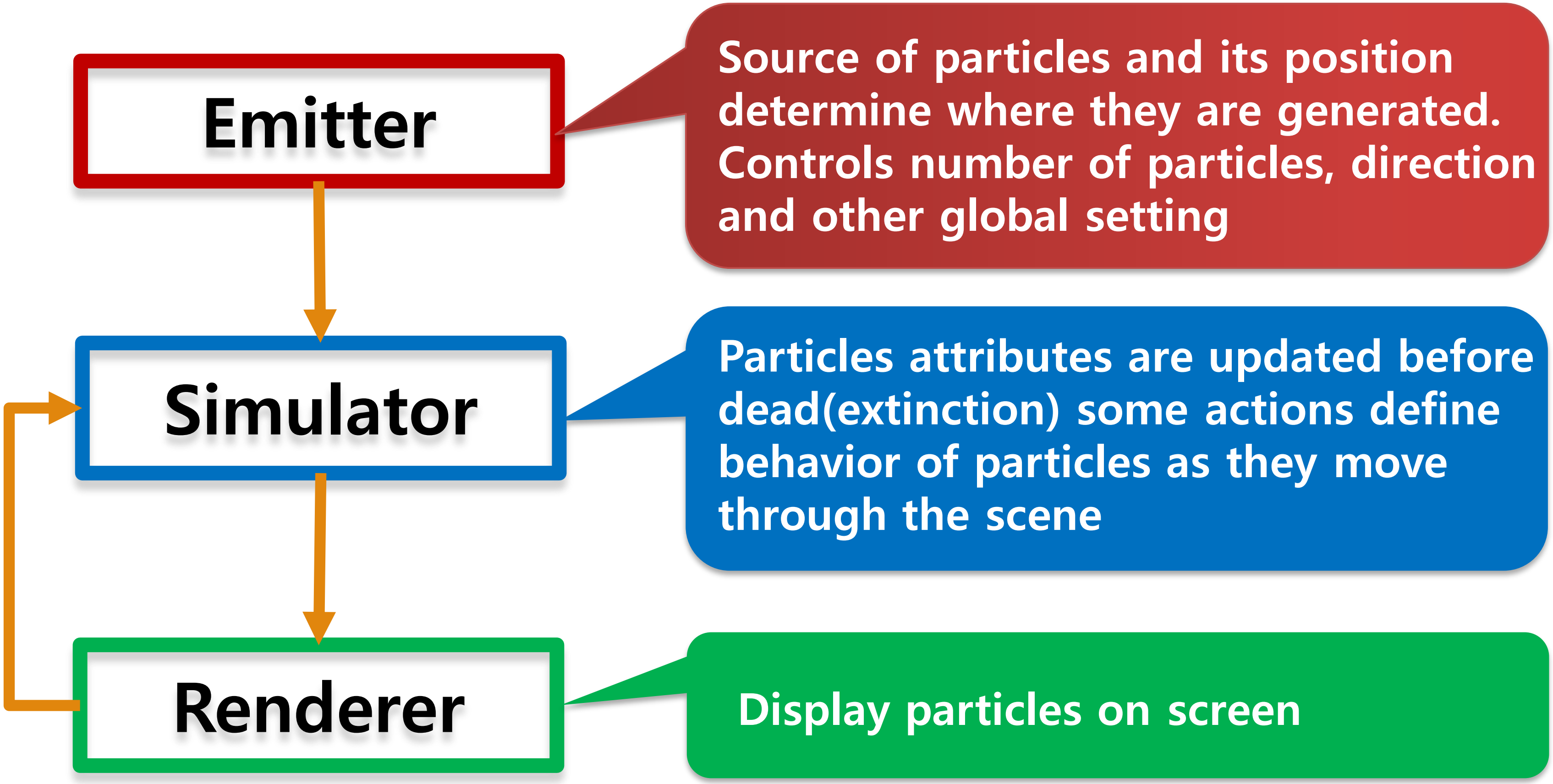
Particle Systems Configuration

- A particle system implement with **3 parts**



Particle Systems Configuration

- A particle system implement with **3 parts**



Emitter

Source of particles and its position determine where they are generated. Controls number of particles, direction and other global setting

Simulator

Particles attributes are updated before dead(extinction) some actions define behavior of particles as they move through the scene

Renderer

Display particles on screen

The Full Model of Particle Systems

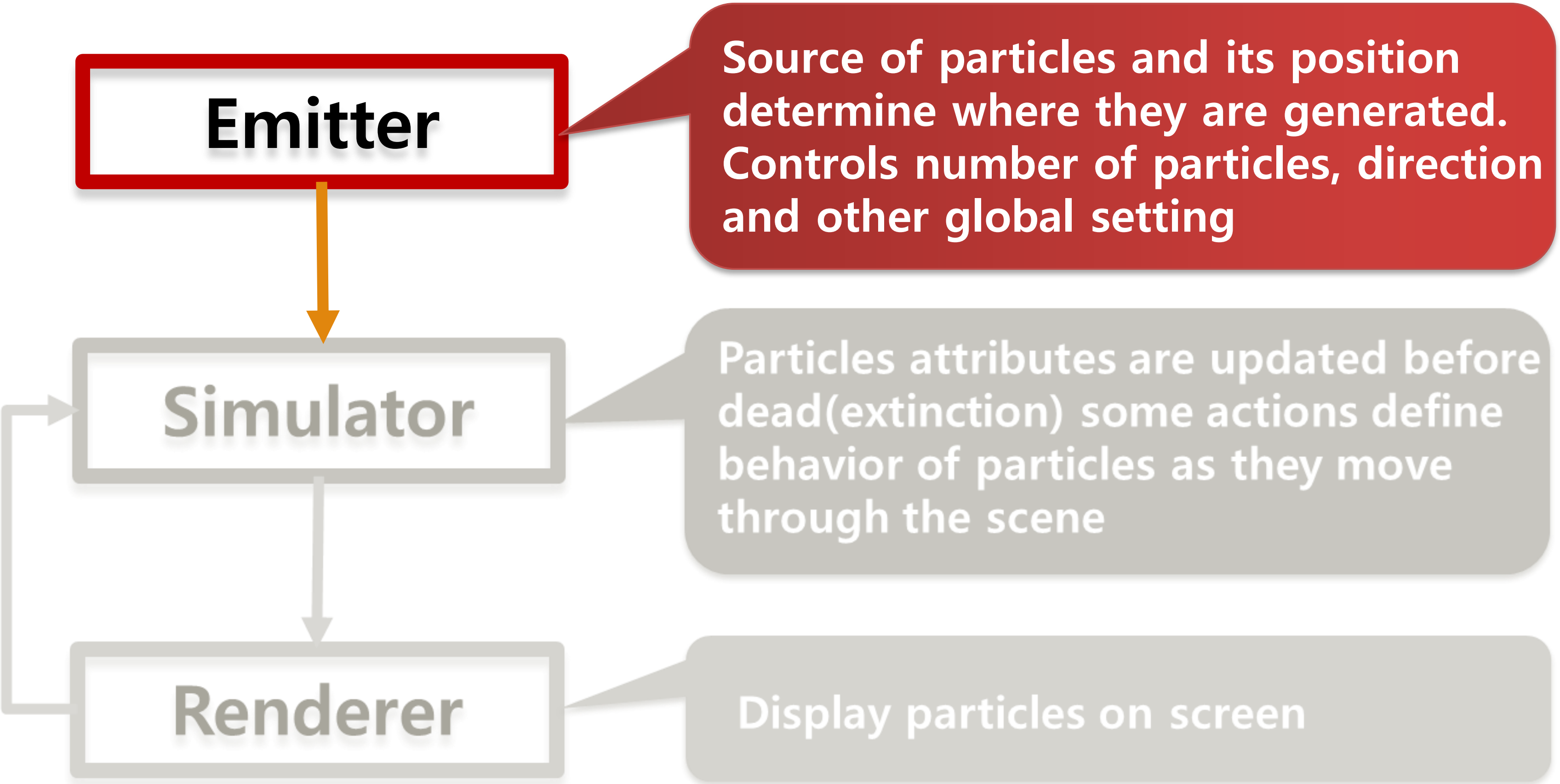
- **For each frame:**

  {

  - **Generate** new particles and assign attributes → **Emitter**
  - **Update** particles based on attributes and physics → **Simulator**
  - **Delete** any expired particles
  - **Render** particles → **Renderer**

  }

Emitter



Emitter

Source of particles and its position determine where they are generated. Controls number of particles, direction and other global setting

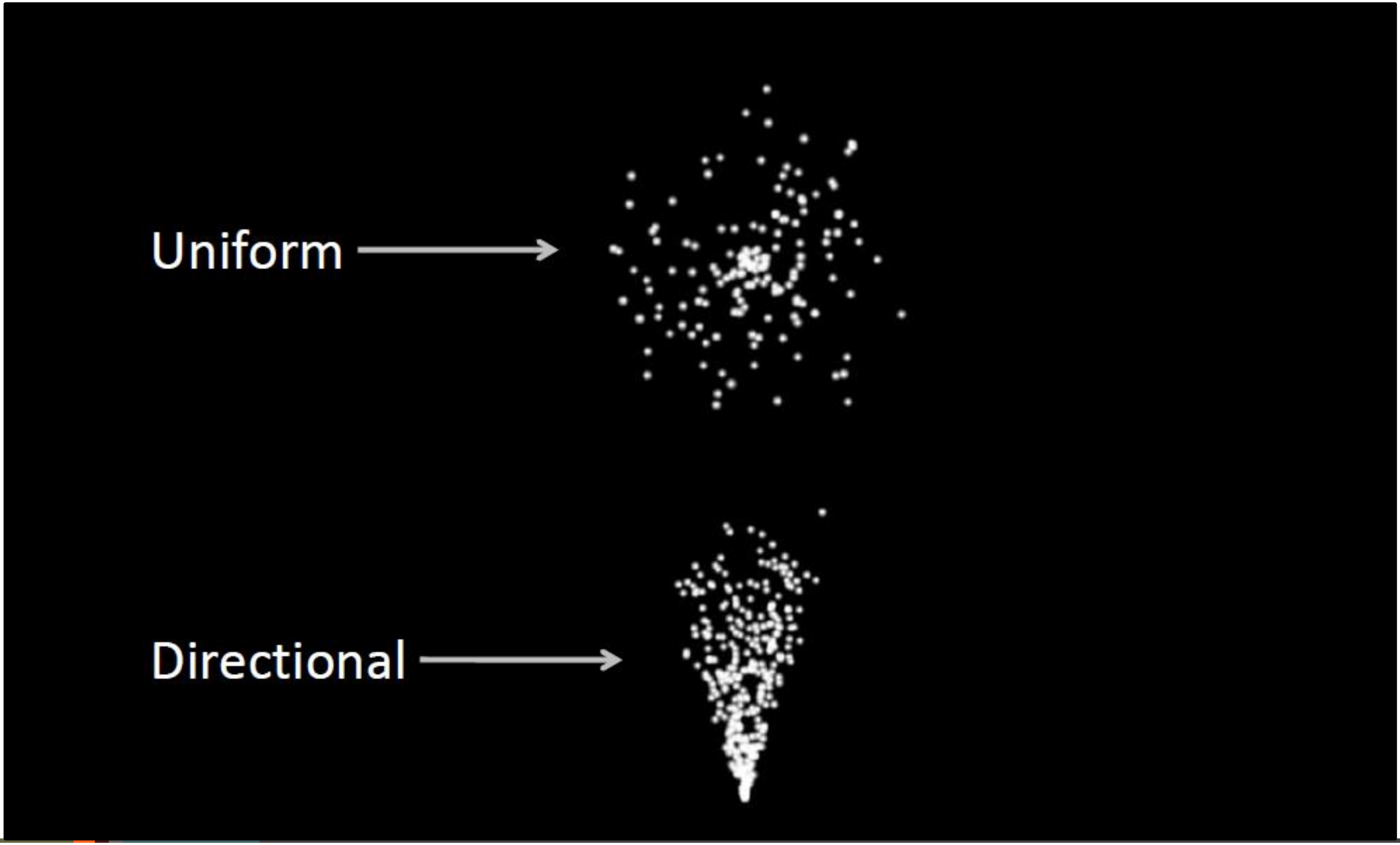
Simulator

Particles attributes are updated before dead(extinction) some actions define behavior of particles as they move through the scene

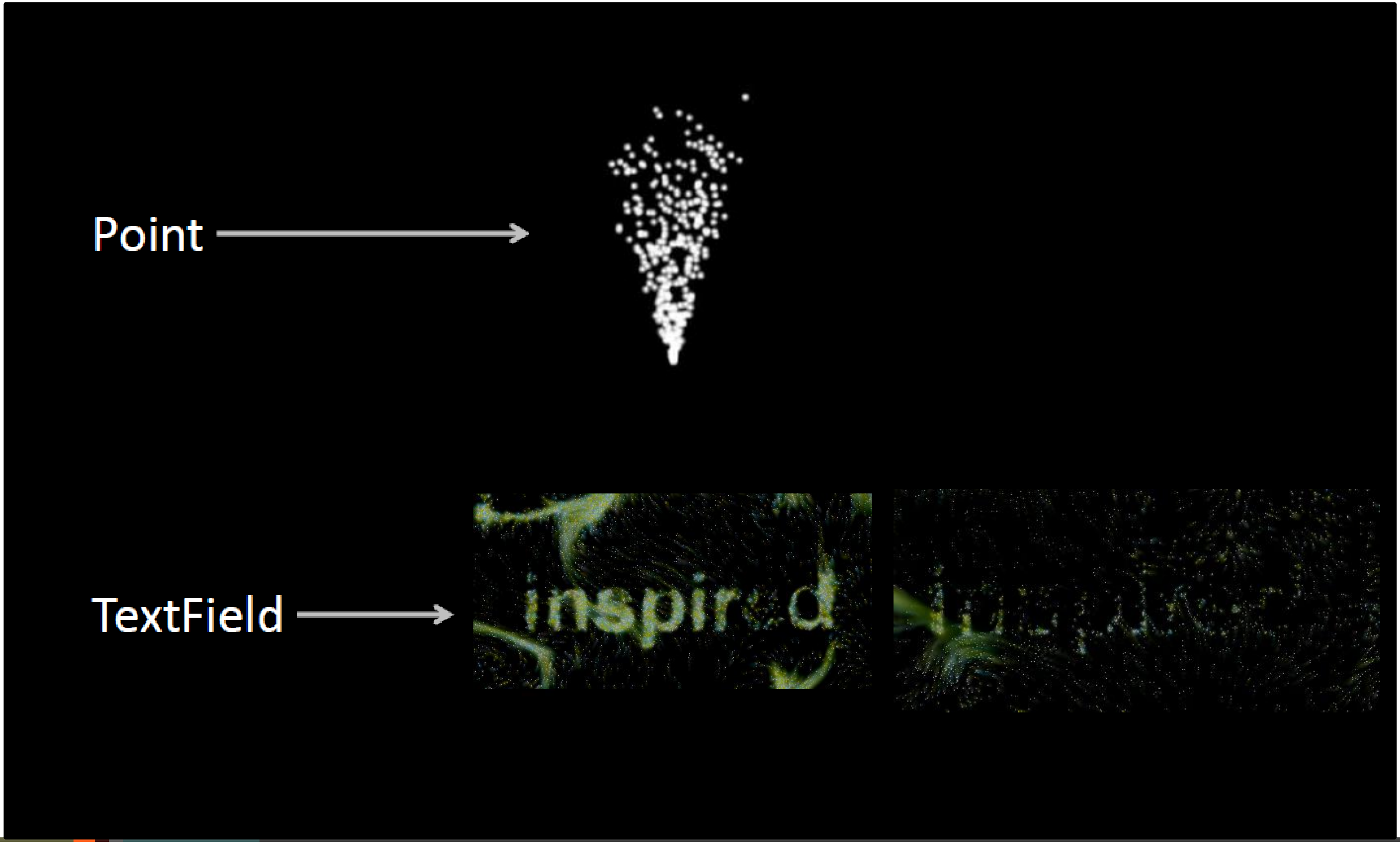
Renderer

Display particles on screen

Emitter Type(Control)



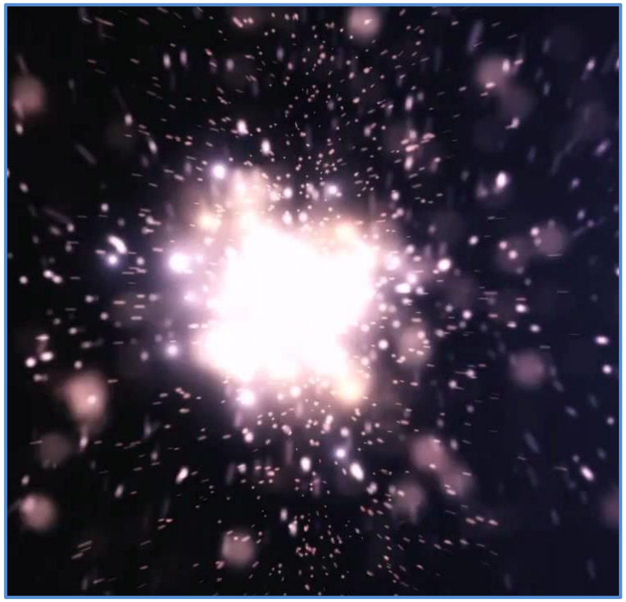
Emitter Type(Zone)

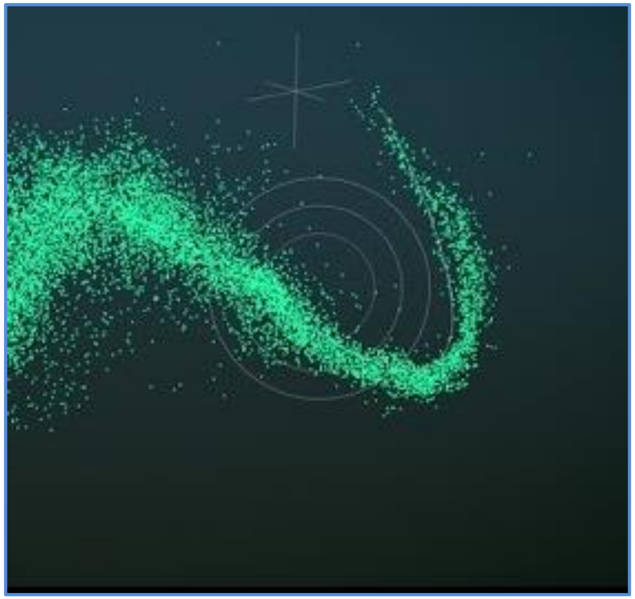

Point

TextField

Creating Particles

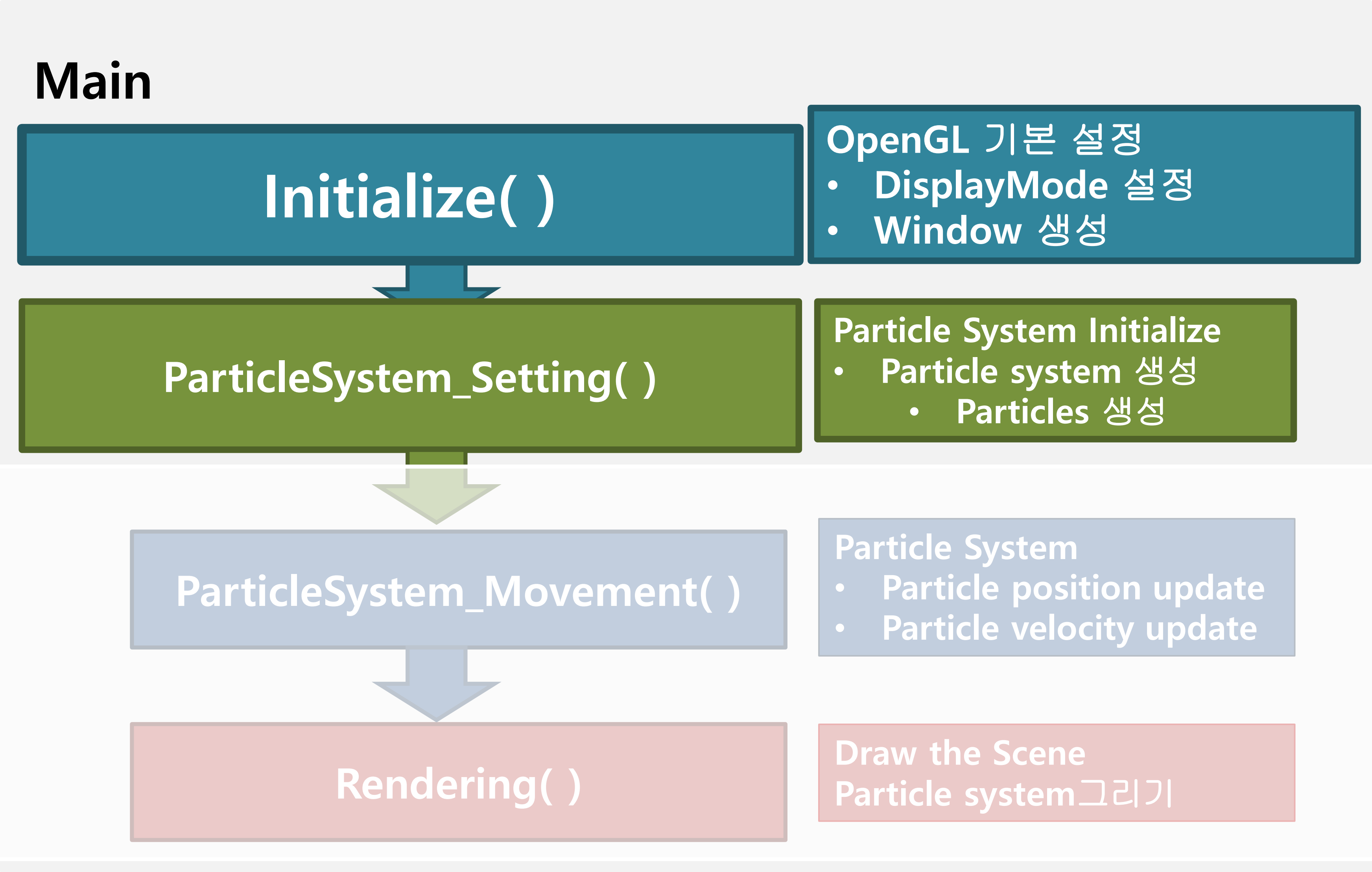
- **Where and How to create particles?**
  - Around some center
  - Along some path
  - Surface of shape
  - Where particle density is low or high





This is where user controls animation

Code skeleton: Initialize()



Main

Initialize( )

OpenGL 기본 설정
- DisplayMode 설정
- Window 생성

ParticleSystem_Setting( )

Particle System Initialize
- Particle system 생성
  - Particles 생성

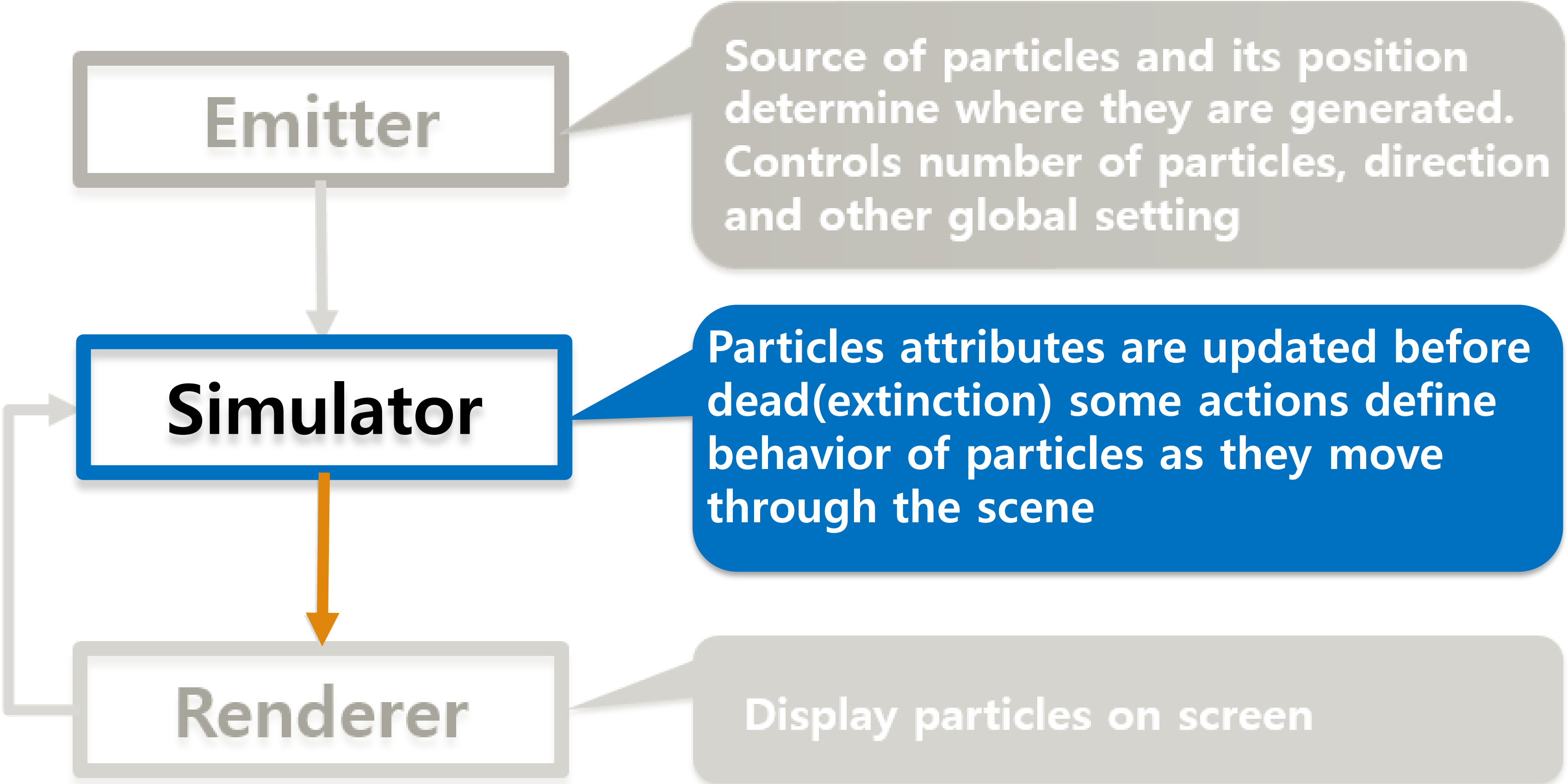
ParticleSystem_Movement( )

Particle System
- Particle position update
- Particle velocity update

Rendering( )

Draw the Scene
Particle system그리기

Simulator



Source of particles and its position determine where they are generated. Controls number of particles, direction and other global setting

Particles attributes are updated before dead(extinction) some actions define behavior of particles as they move through the scene

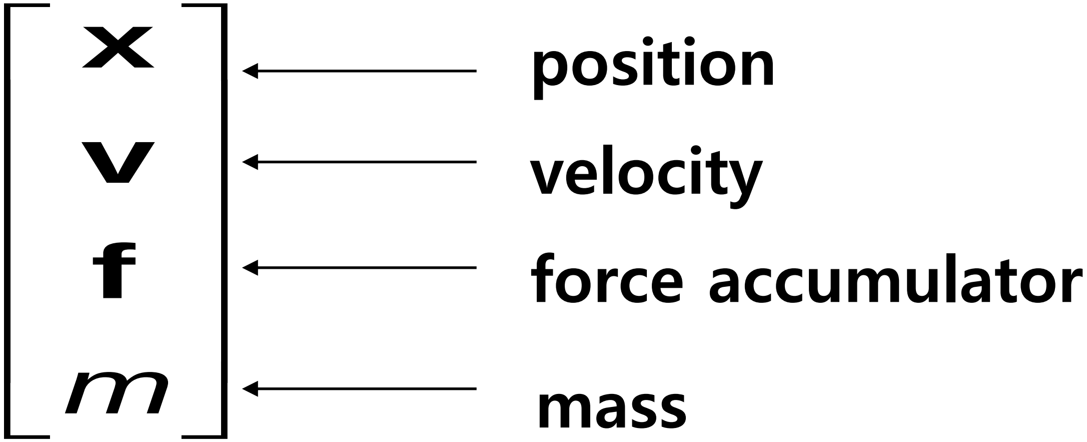
Display particles on screen

Physics of Particles

- Physics system controls the motion of **every particle**
- A particle's position in each succeeding frame can be computed by its velocity
- This can be modified by an acceleration force for more complex movements

Particle Structure

- How do we represent a particle?



$$\begin{bmatrix} \mathbf{x} \\ \mathbf{v} \\ \mathbf{f} \\ m \end{bmatrix}$$

position

velocity

force accumulator

mass

Update Step

- **For each particle:**

{

- Acceleration

$$a = f/m$$

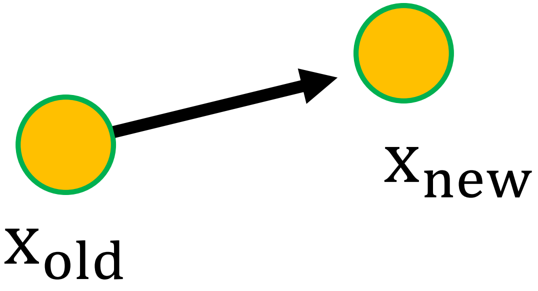
- Velocity

$$\mathrm{v}_{\mathrm{new}} = \mathrm{v}_{\mathrm{old}} + a\Delta t$$

- Position

$$\mathrm{x}_{\mathrm{new}} = \mathrm{x}_{\mathrm{old}} + \mathrm{v}_{\mathrm{old}}\Delta t$$

}



Time Integration (Velocity)

- **Velocity (speed + direction)**
  - Rate at which position changes

$$\frac{\Delta\mathbf{x}}{\Delta\boldsymbol{t}} = \boldsymbol{V}$$

  - Multiply by time

$$\mathbf{x} = \mathbf{x} + \mathbf{v}\Delta\boldsymbol{t}$$

    - Also called **Forward Euler**

Forward Euler (Example)

$$\boldsymbol{X}^{\boldsymbol{t}+\Delta\boldsymbol{t}} = \boldsymbol{X}^{\boldsymbol{t}} + \boldsymbol{V}\Delta\boldsymbol{t}$$

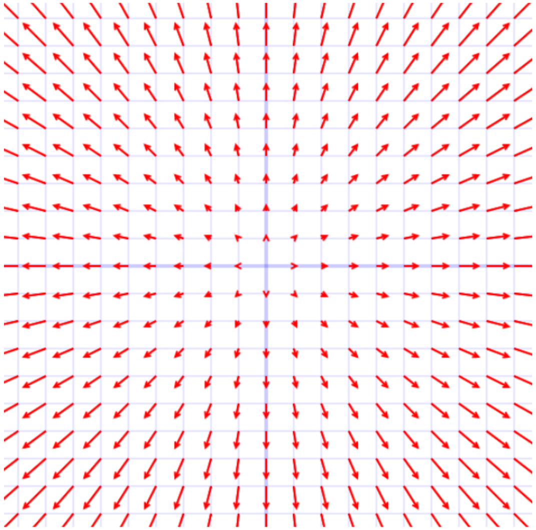
- Example:

•X(t=0) = (0, 1)
•V = (-y, x),
•Δt=0.5
Find X(t=1.5).

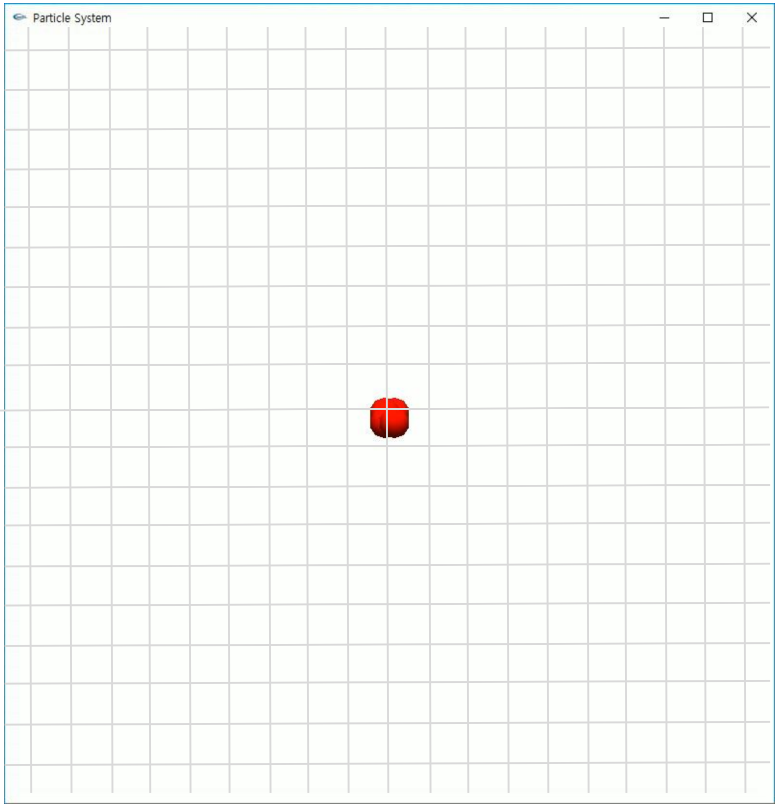
X(0.5) = (0,1) + (-1,0)x0.5 = (-0.5, 1)
X(1) = (-0.5, 1) + (-1, -0.5)x0.5 = (-1, 0.75)
X(1.5) = (-1,0.75) + (-0.75,-1)x0.5 = (-1.375, 0.25)
…

Vector Fields(Radial expansion)

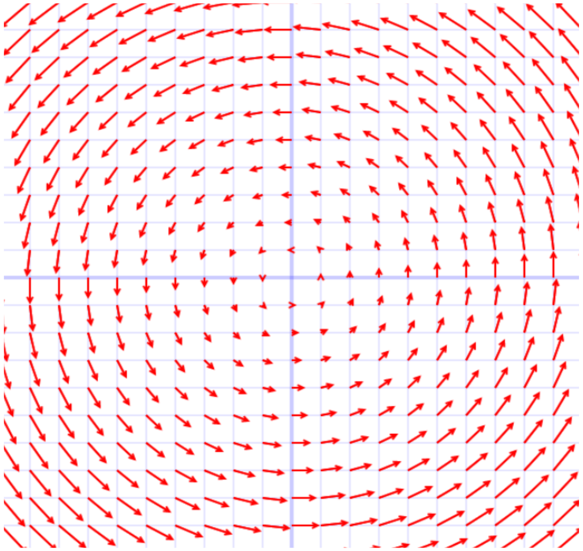
- Particle motion(): position to velocity vector, i.e., a vector field.



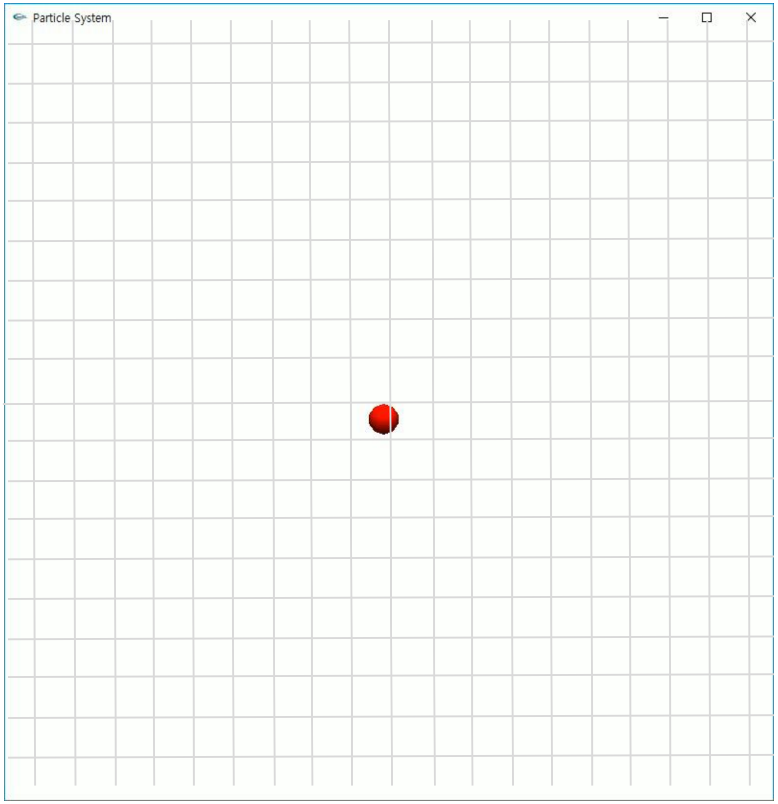
Radial expansion: $V=(x,y)$



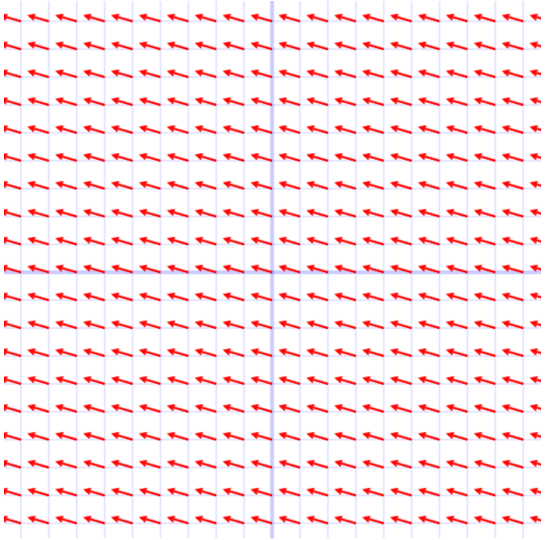
Vector Fields(Rotational vortex)



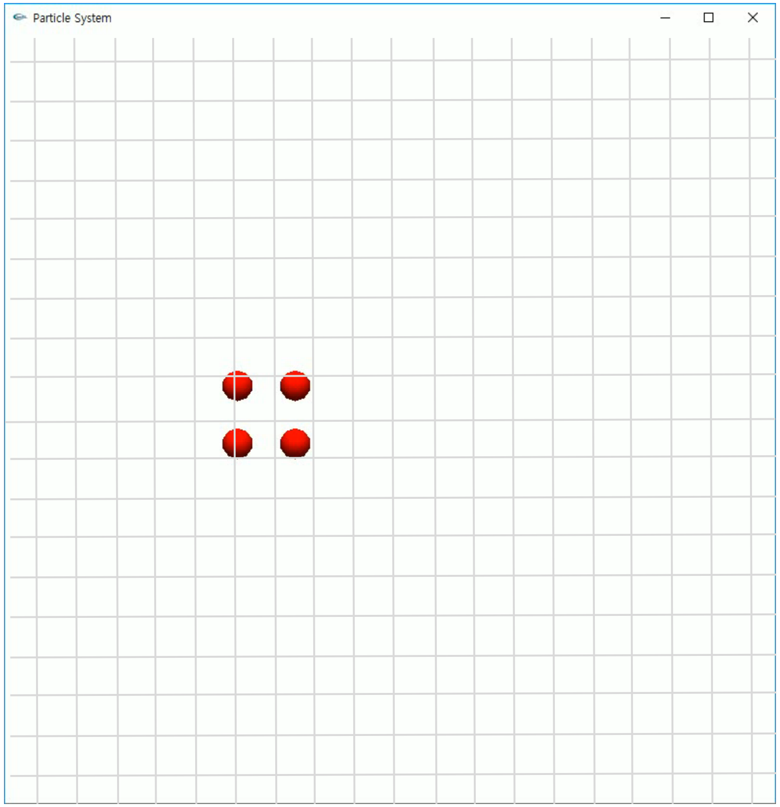
Rotational Vortex: $V=(-y,x)$



Vector Fields(Constant wind)



Constant Wind: V=(-7,2)



Time Integration (Forces)

- Newton's 2nd law
  - Acceleration
    - The rate that velocity changes

$$\Delta\mathbf{v}/\Delta\boldsymbol{t}$$

    - Useful for gravity , spring, wind etc.

$$\mathbf{v} = \mathbf{v} + a\Delta\boldsymbol{t}$$

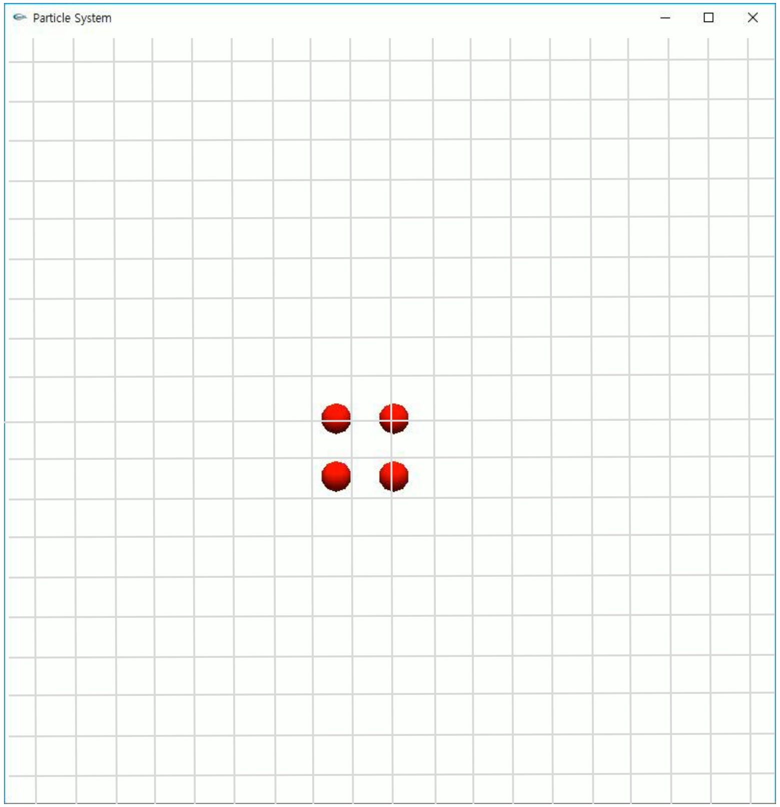
Type of Forces: Gravity

- **Constant**: gravity
  - Force Law:

$$f = mg$$

- *g:* gravity acceleration

Example: gravity=-9.8



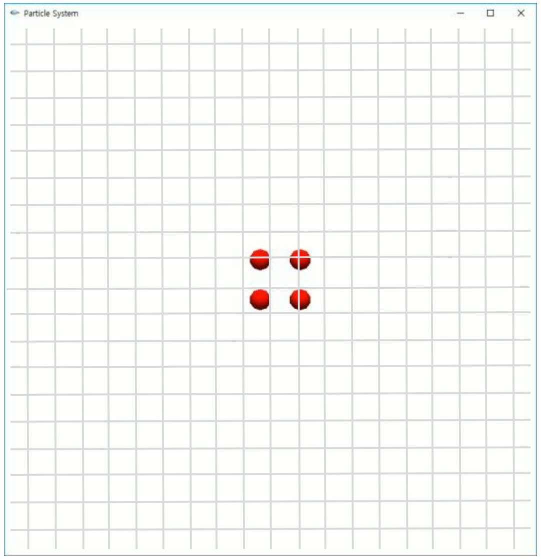
Type of Forces: Drag

- **Velocity dependent**: drag
  - Force Law:

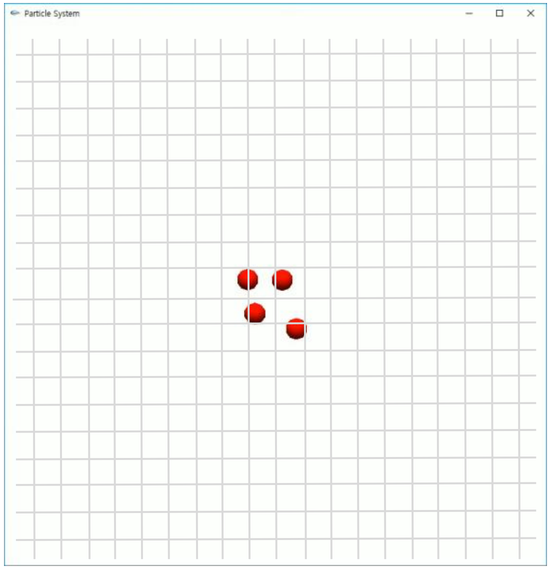
$$f = -k_{drag}v$$

    - *k:* drag constant

Example: k=-0.3



Initialize velocity: Same



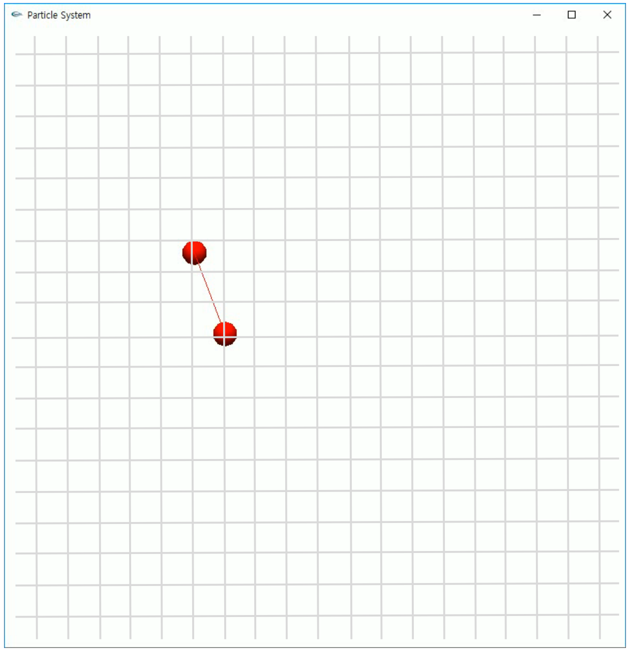
Initialize velocity: Different

Type of Forces: Springs

- **Distance dependent**: springs
  - The spring forces between a pair of particles at position ***a*** and ***b***
    - Force Law:

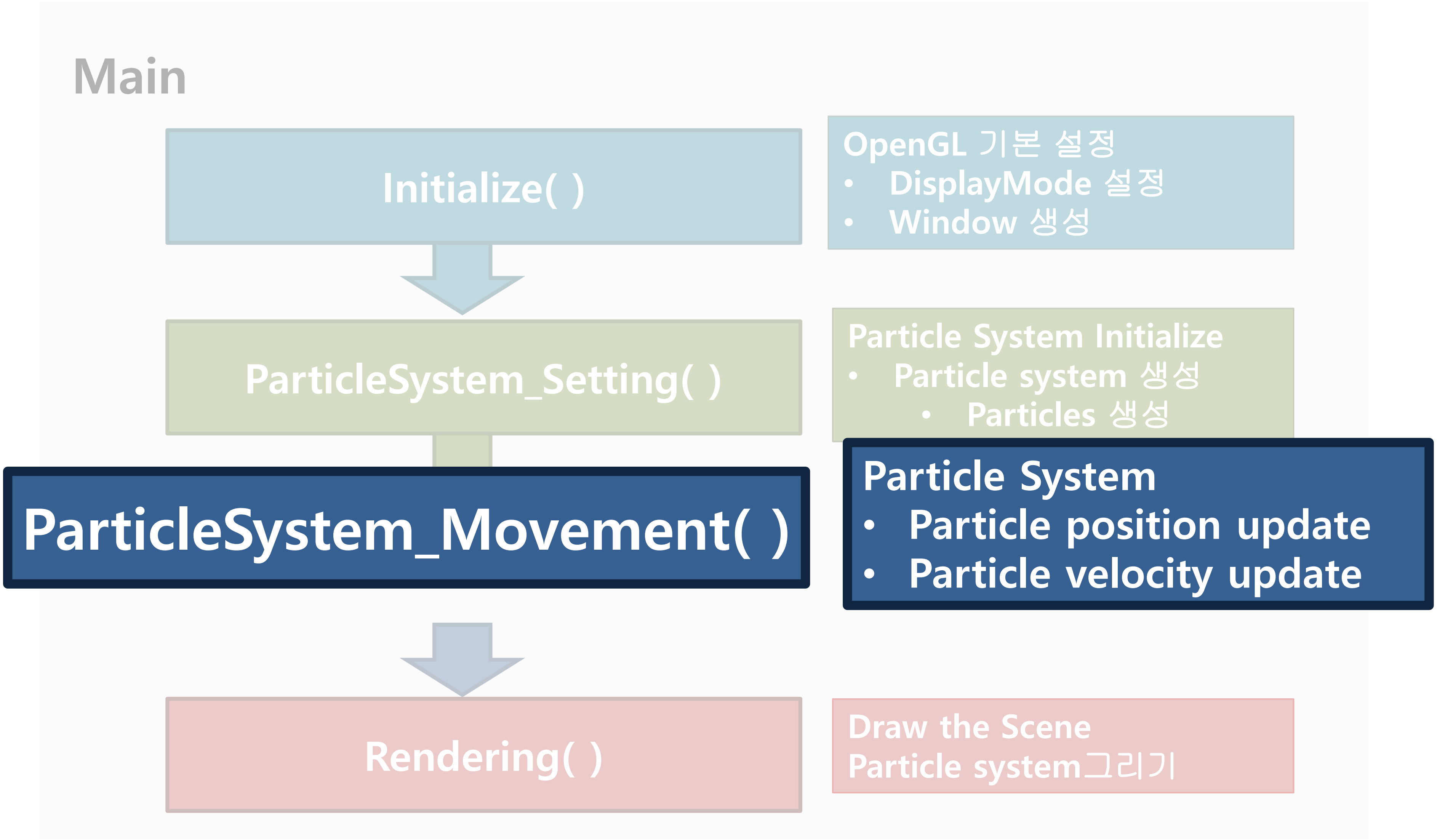
$$f_a = -k_s \Delta x, \qquad f_b = -f_a$$

    - Δx: （$a - b$） - L
    - L: spring length
    - k_s: spring constant



[one of the particle is fixed in there]

Code skeleton: Movement_Update()



Main

Initialize( )

OpenGL 기본 설정
- DisplayMode 설정
- Window 생성

ParticleSystem_Setting( )

Particle System Initialize
- Particle system 생성
  - Particles 생성

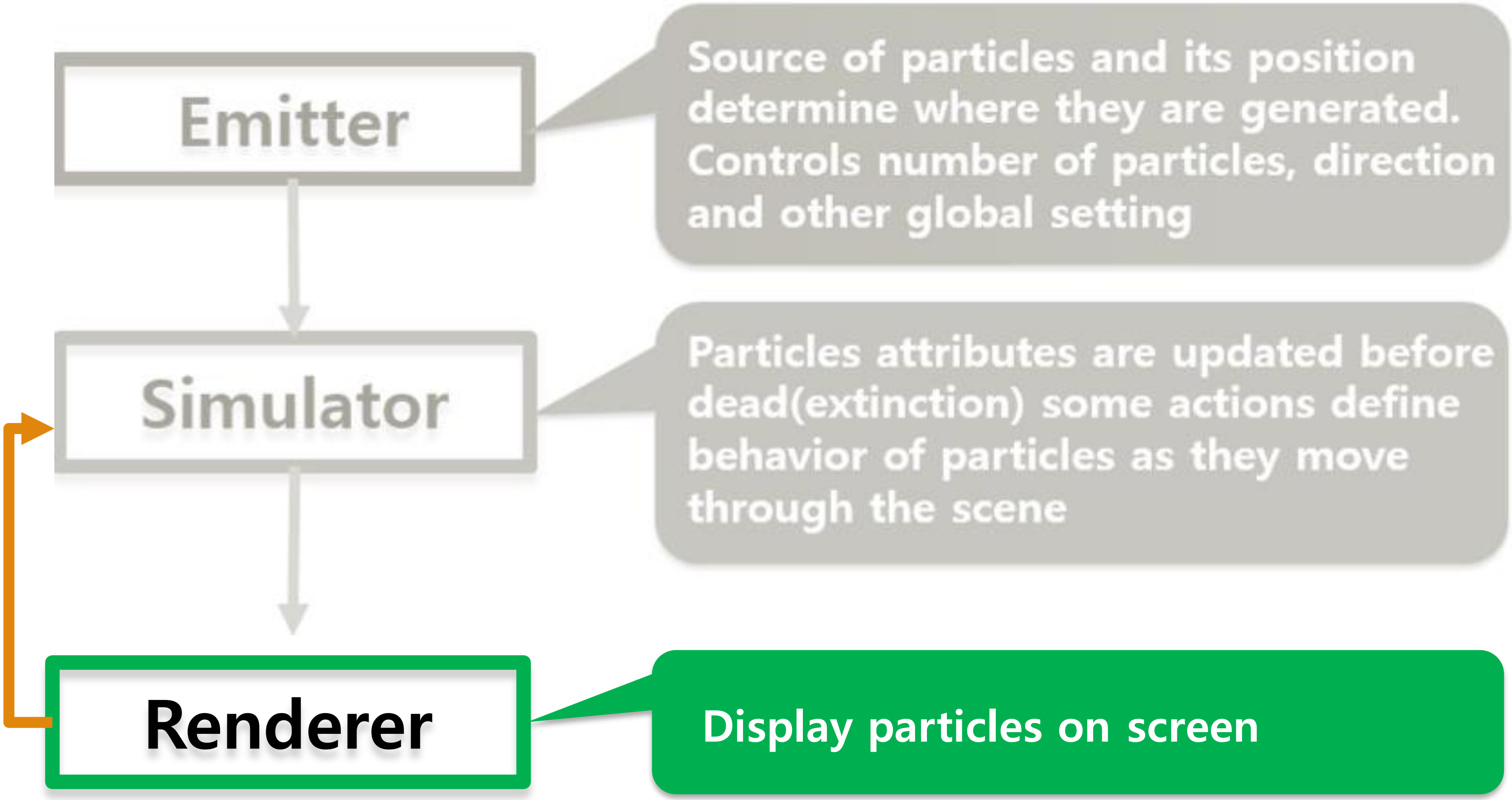
ParticleSystem_Movement( )

Particle System
- Particle position update
- Particle velocity update

Rendering( )

Draw the Scene
Particle system그리기

Renderer



Emitter

Source of particles and its position determine where they are generated. Controls number of particles, direction and other global setting

Simulator

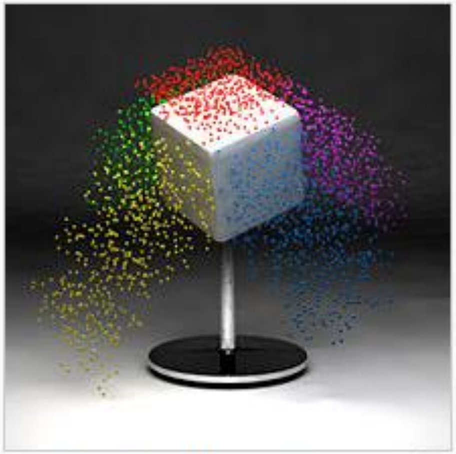
Particles attributes are updated before dead(extinction) some actions define behavior of particles as they move through the scene

Renderer

Display particles on screen

Display Particle System on Screen

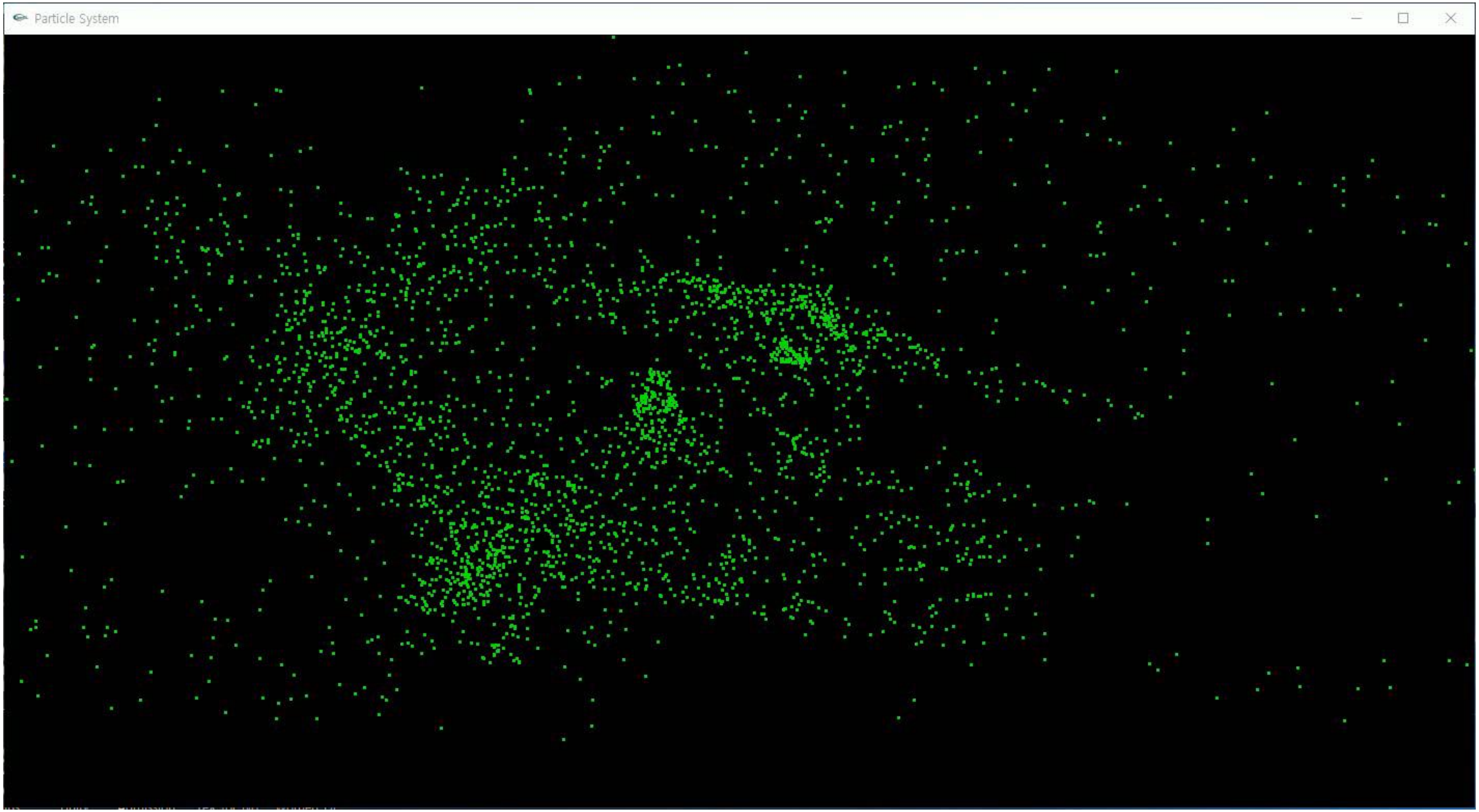
- After the simulation stage, the particle need rendering
- The particle render type of a particle object specifies the form of its particles
  - E.g. you can display as
    - Points
    - Lines(from last position to current position)
    - Spheres
    - Texture
    - Geometry(small objects)



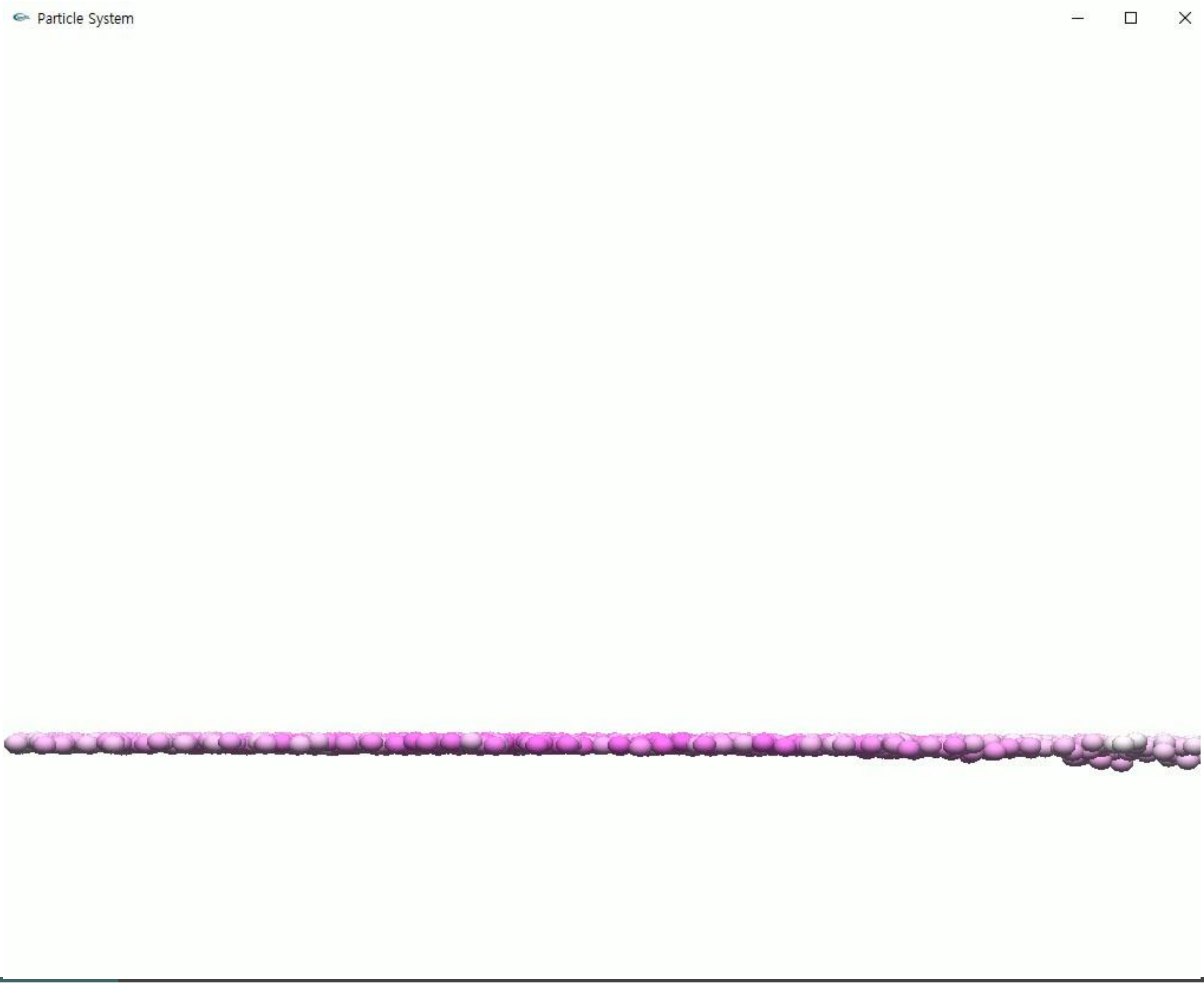
A cube emitting 5000 animated particles, obeying a "gravitational" force in the negative Y direction.

The same cube emitter rendered using static particles, or strands.

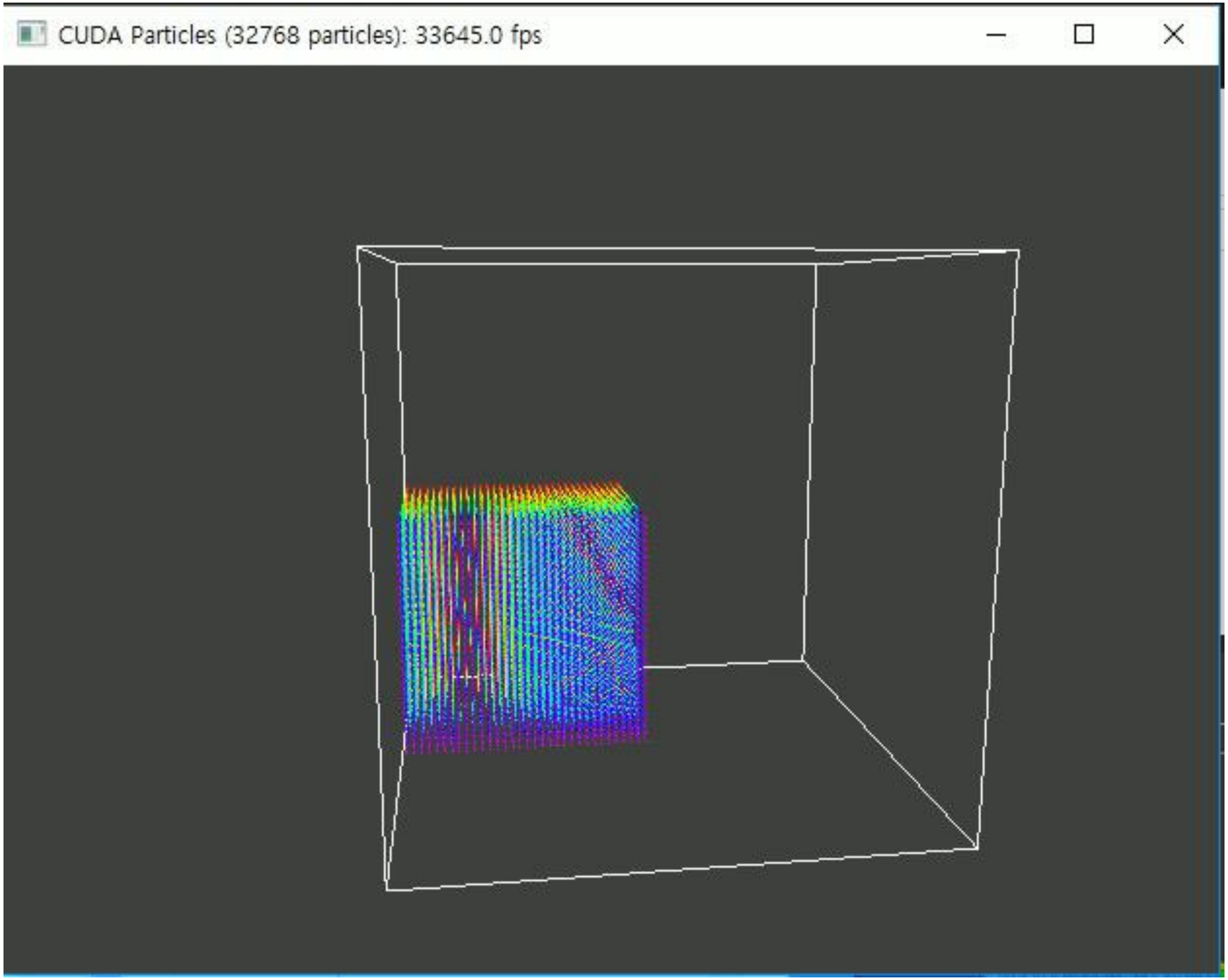
Particle Rendering: Points



Particle Rendering: Spheres

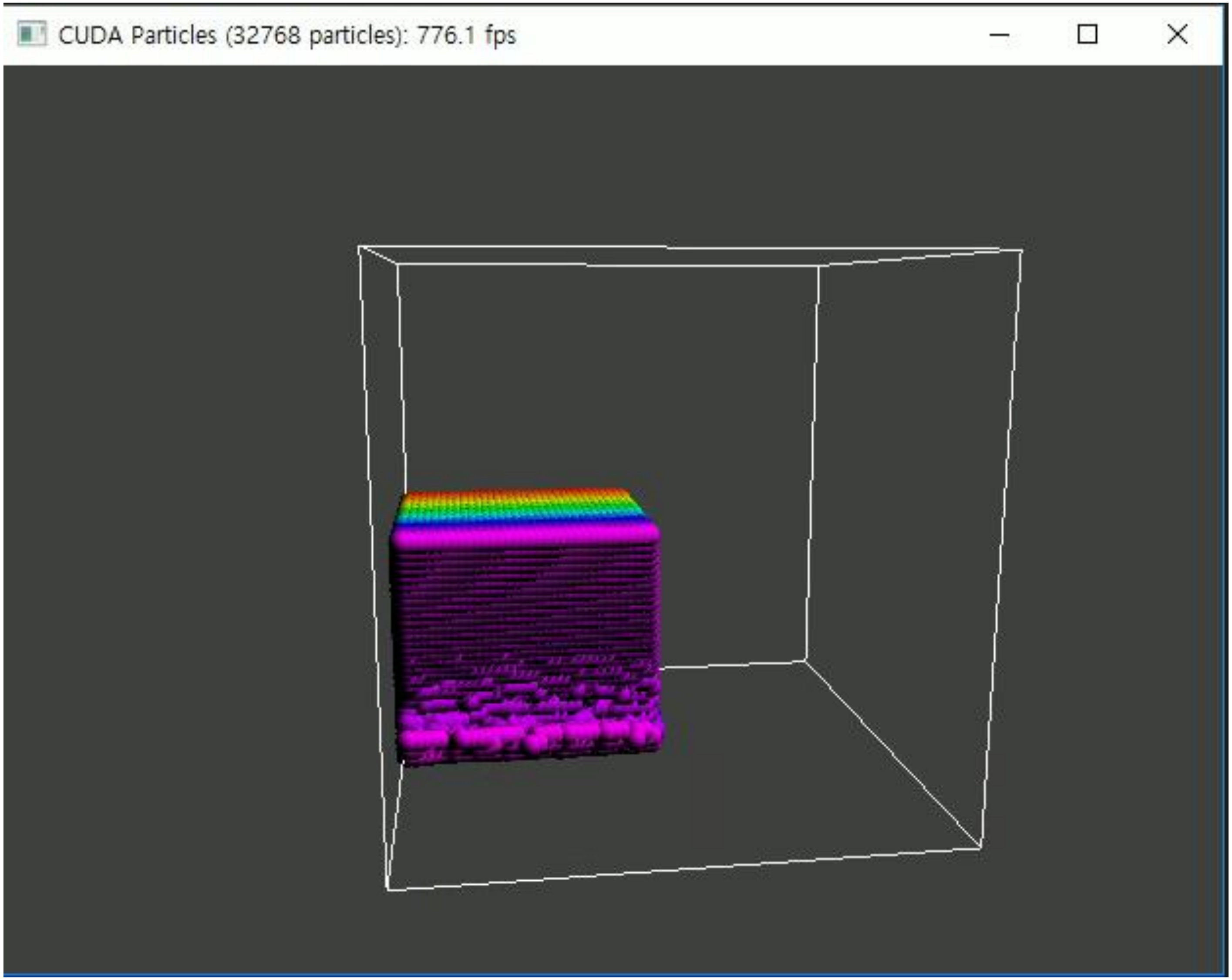


GPU Shading with Points



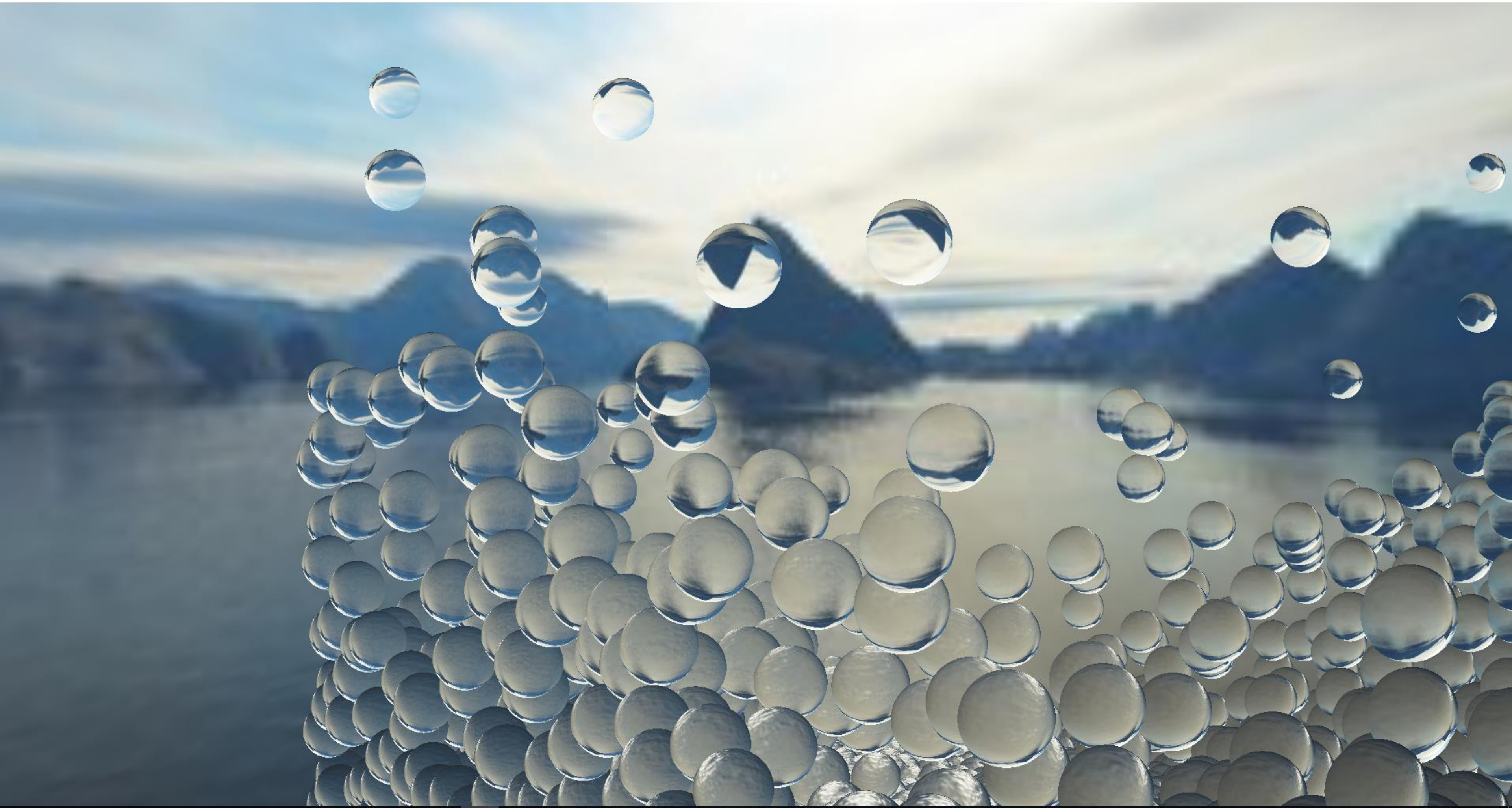
CUDA Particles (32768 particles): 33645.0 fps

GPU Shading with Sphere

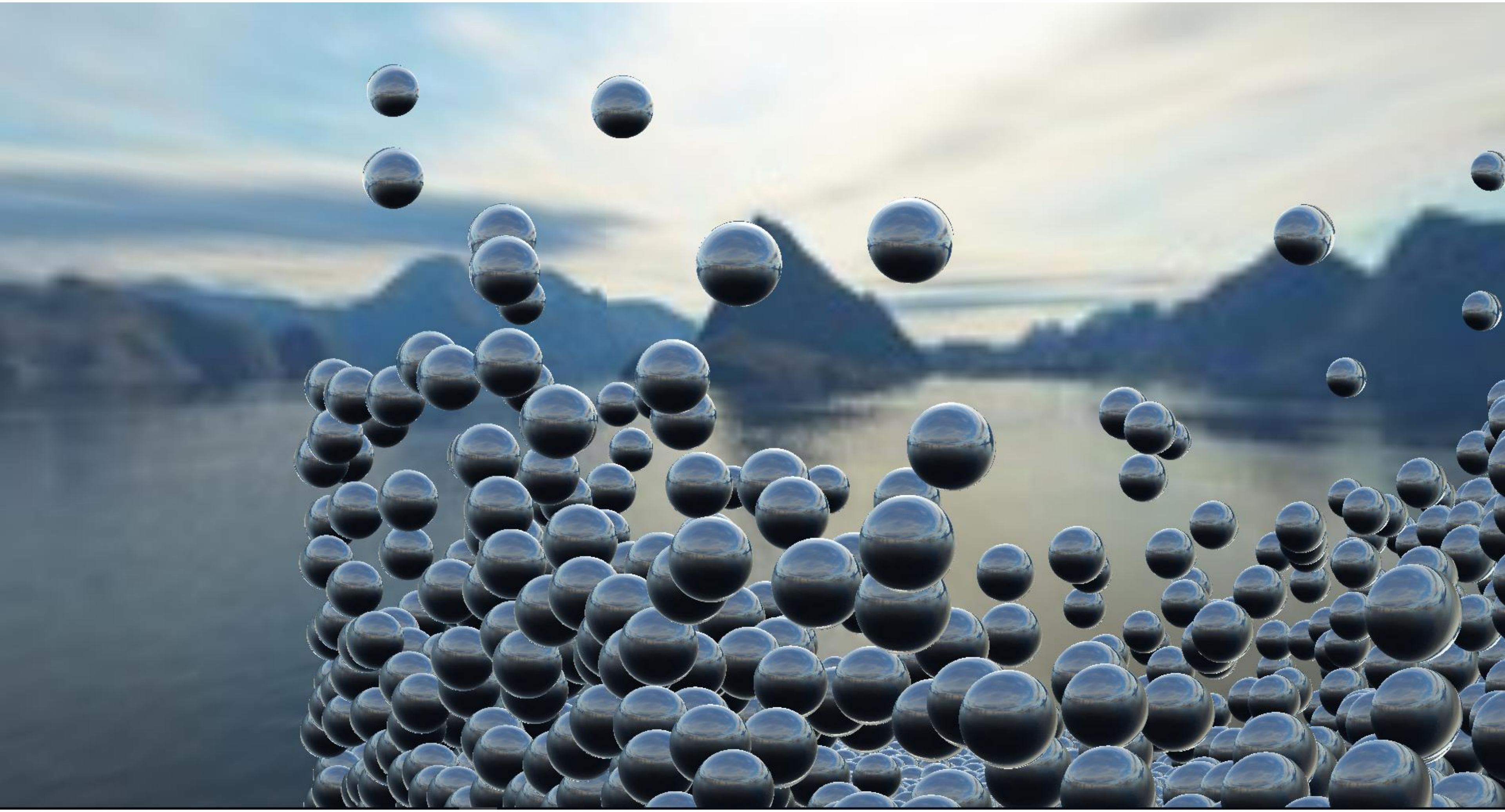


CUDA Particles (32768 particles): 776.1 fps

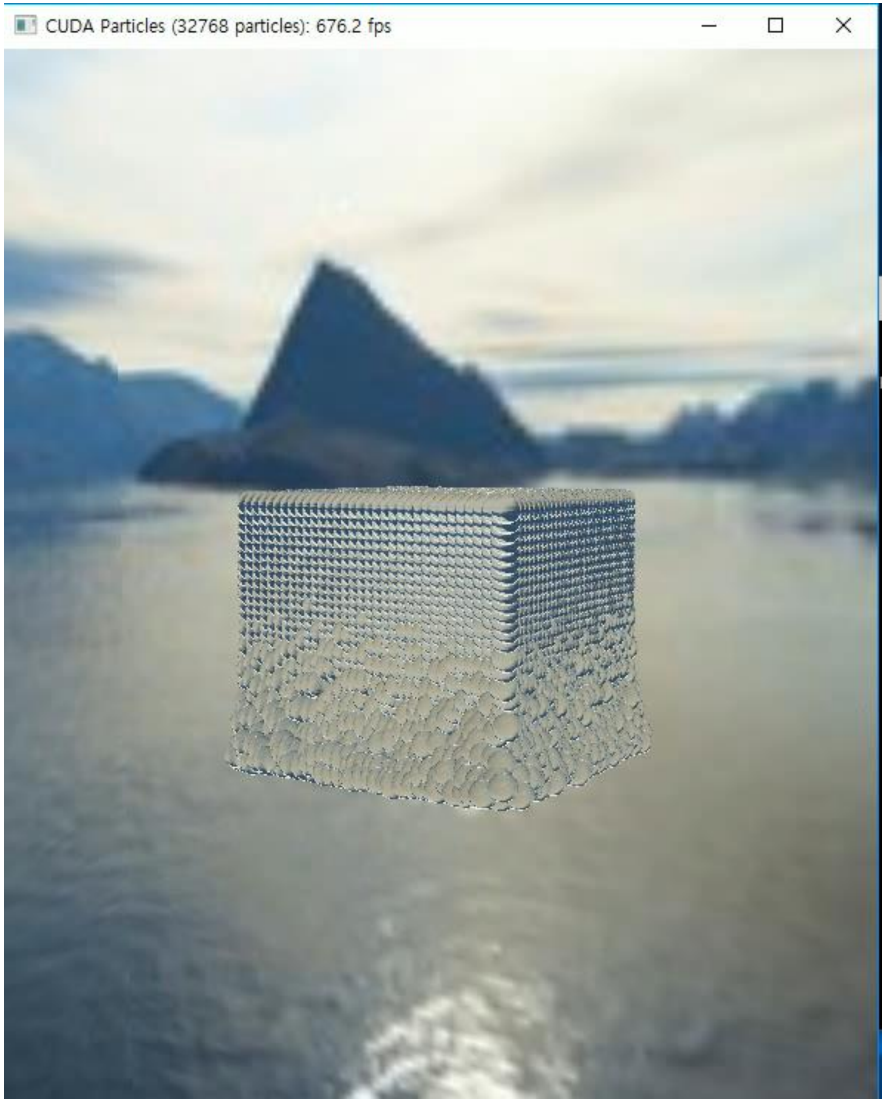
GPU Shading with Refraction

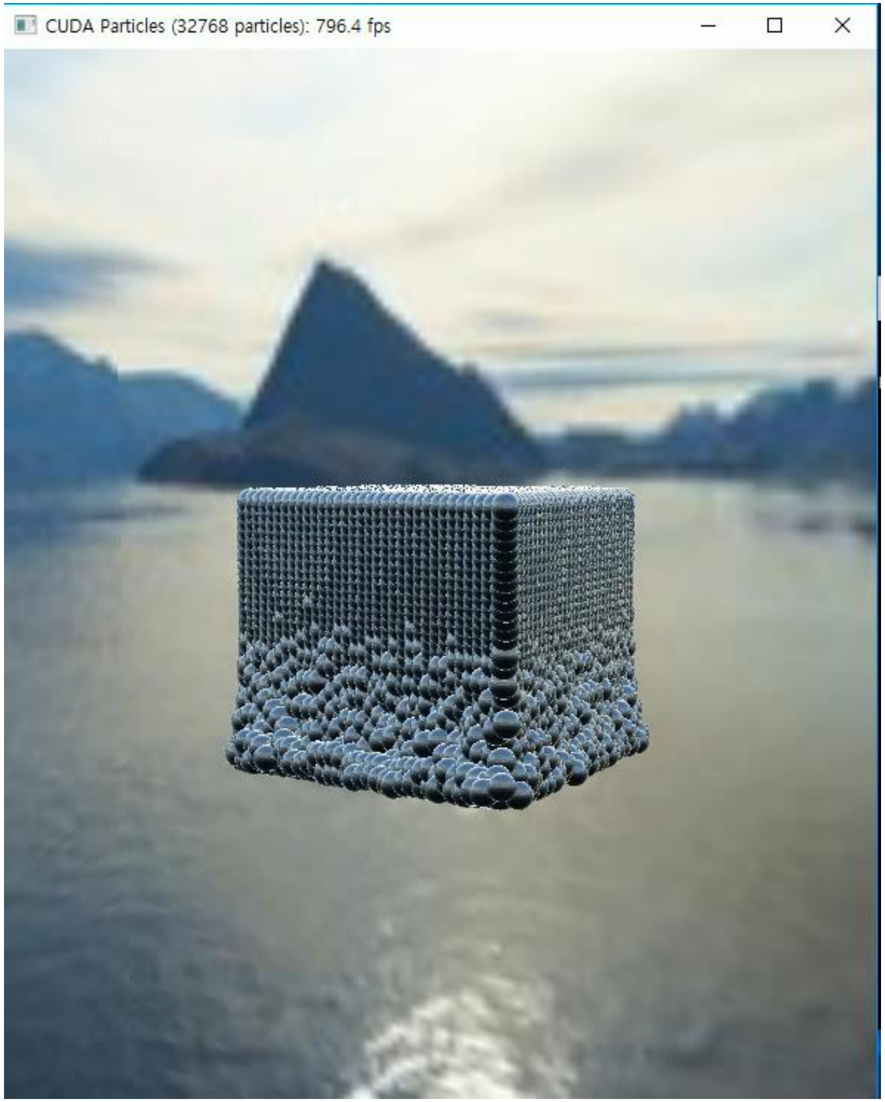


GPU Shading with Reflection

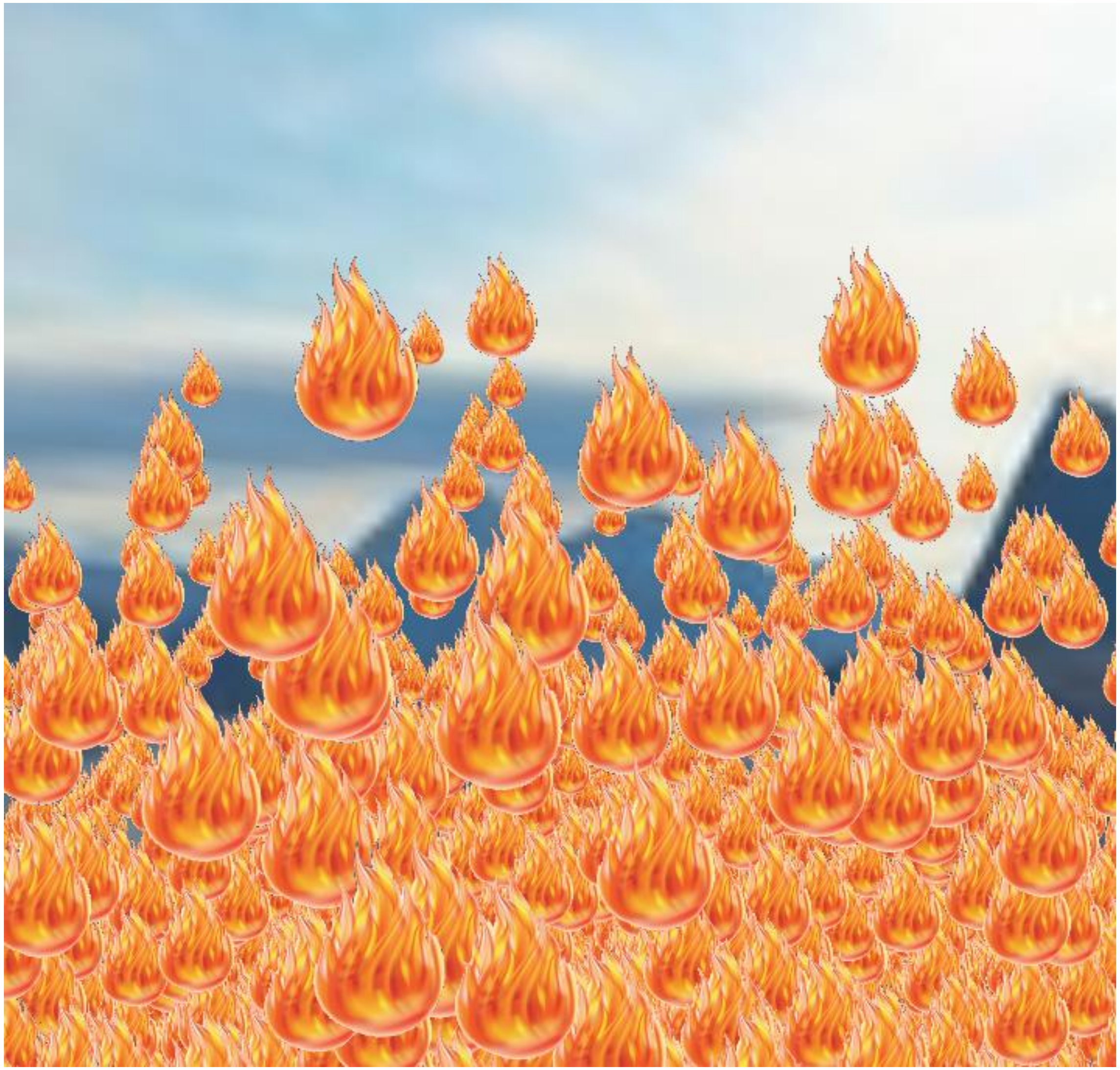


Refraction vs. Reflection


CUDA Particles (32768 particles): 676.2 fps

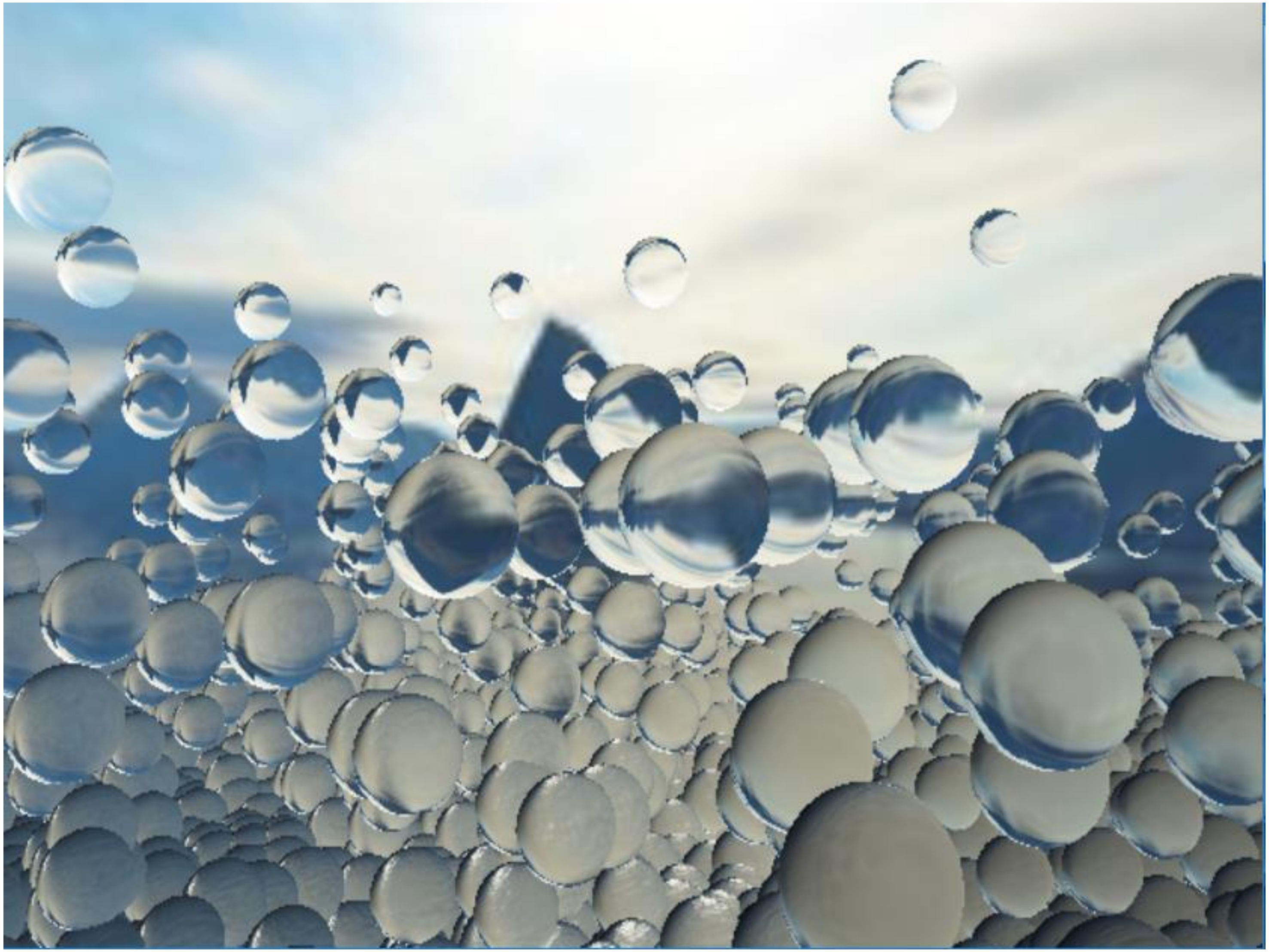

CUDA Particles (32768 particles): 796.4 fps

Rendering with Texture

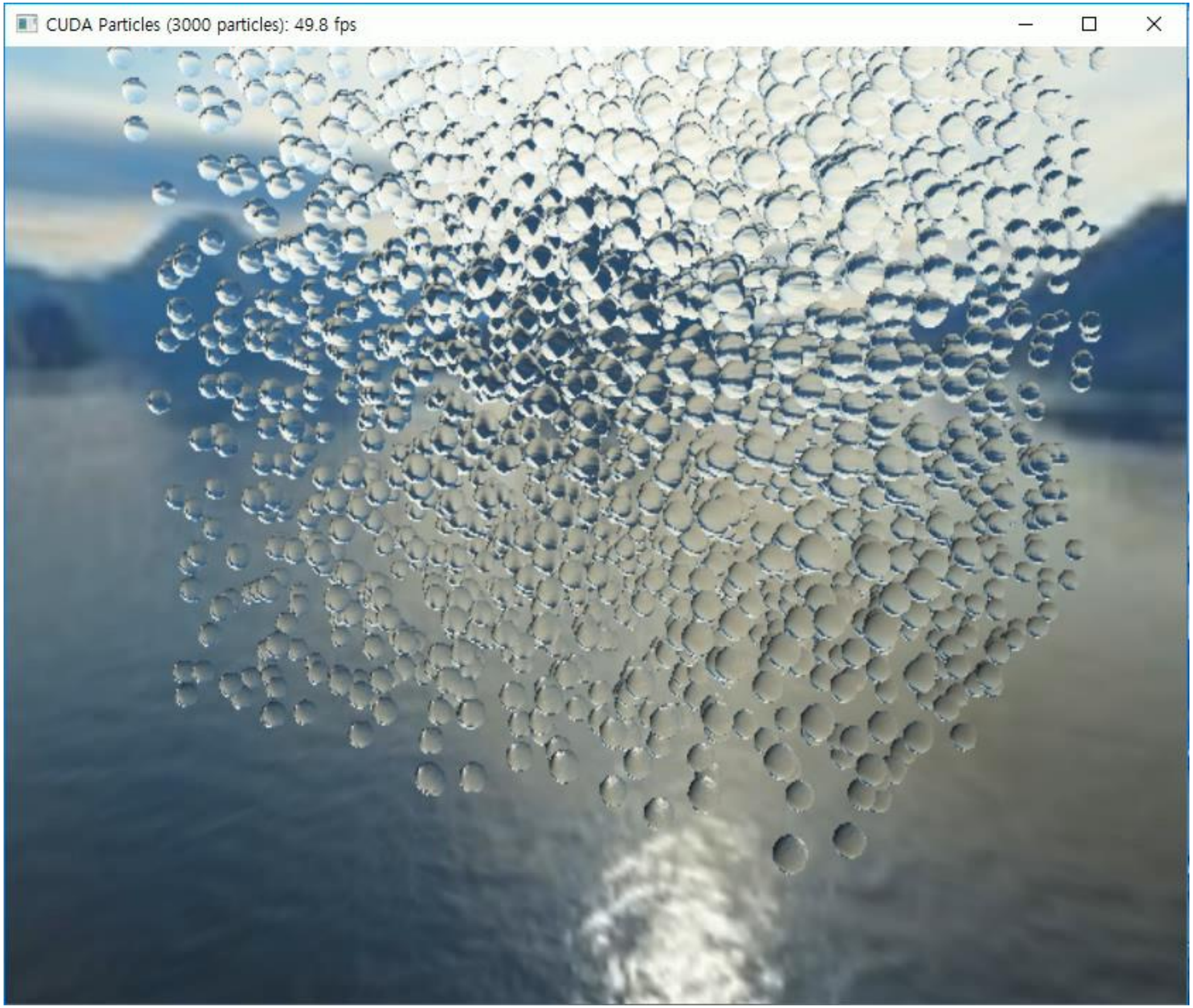


Texture image

Raytracing Particles

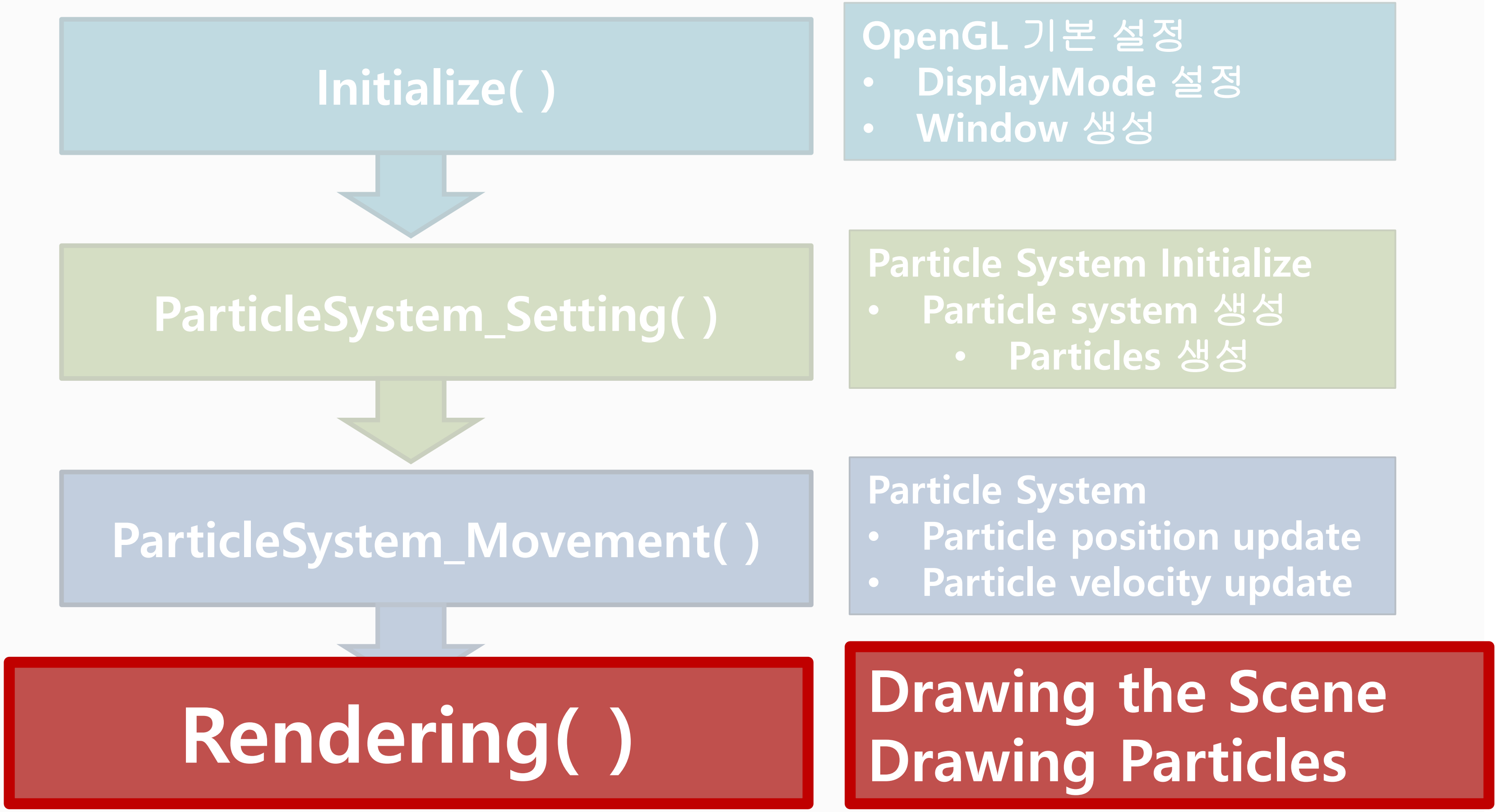


Particle Rendering with Raytracing



Code Skeleton: Rendering()

Main



Initialize( )

OpenGL 기본 설정
- DisplayMode 설정
- Window 생성

ParticleSystem_Setting( )

Particle System Initialize
- Particle system 생성
  - Particles 생성

ParticleSystem_Movement( )

Particle System
- Particle position update
- Particle velocity update

Rendering( )

Drawing the Scene
Drawing Particles