Particle System

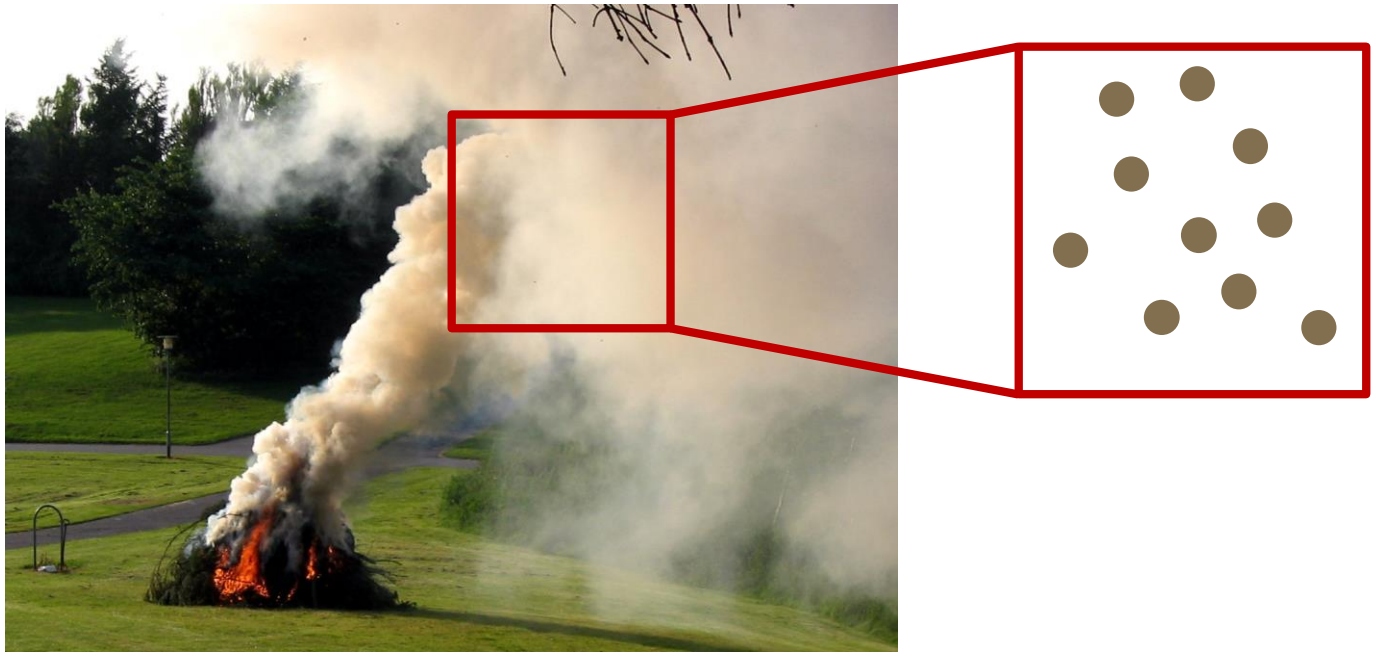
Contents

- **Introduction**
  - System Configuration
  - Basic System Example
- **Collision Detection and Response**
  - Algorithm: Collision and Response
  - Coding Example
  - Acceleration Method
- **GPU Coding with CUDA**
- **Rendering Methods**
  - Basic Shading
  - Shading with Refraction & Reflection
  - Rendering using Texture
  - Raytracing

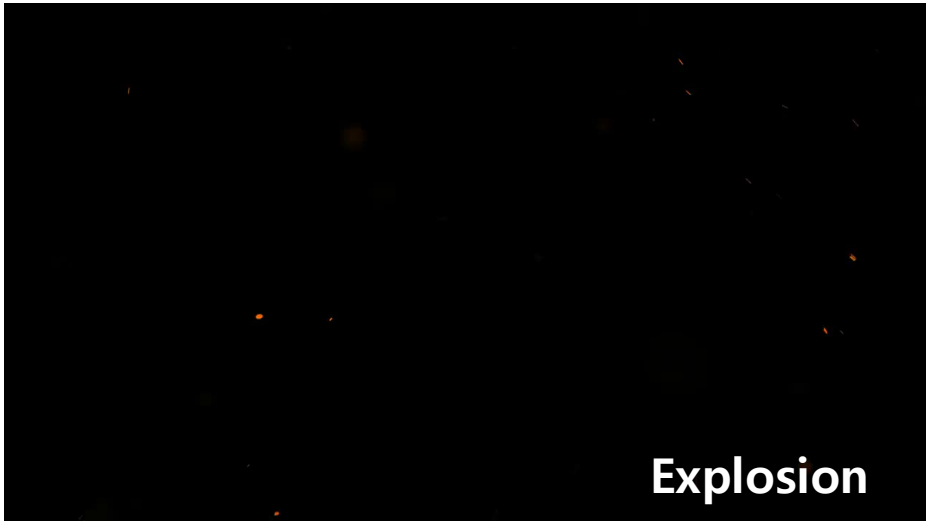
Introduction

What is Particle System?

- A **particle system** is a technique in game physics, motion graphics, and computer graphics
- A **particle system** uses a large number of very small sprites, 3D models or other graphics objects to simulate certain phenomena
  - Fire , Fireworks , Smoke , Water. etc

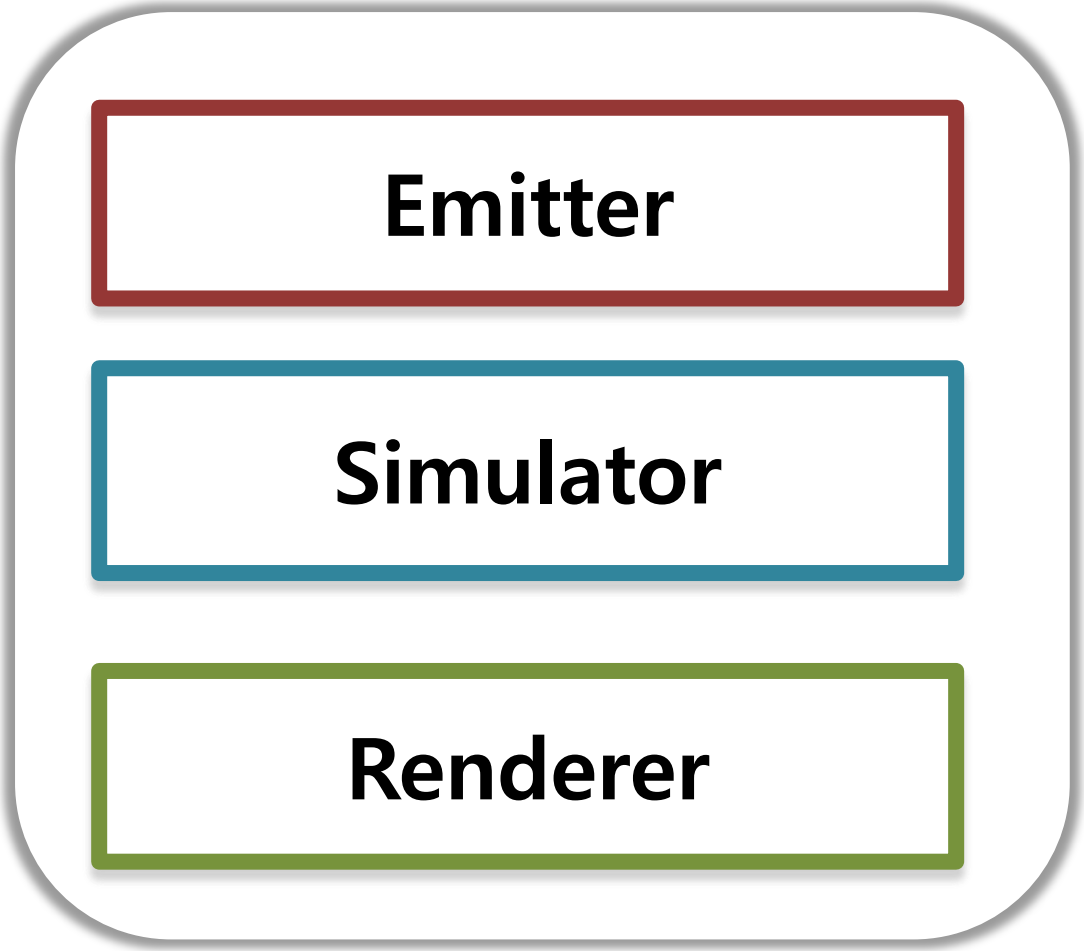


Particle System Effects



Particle Systems Configuration

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



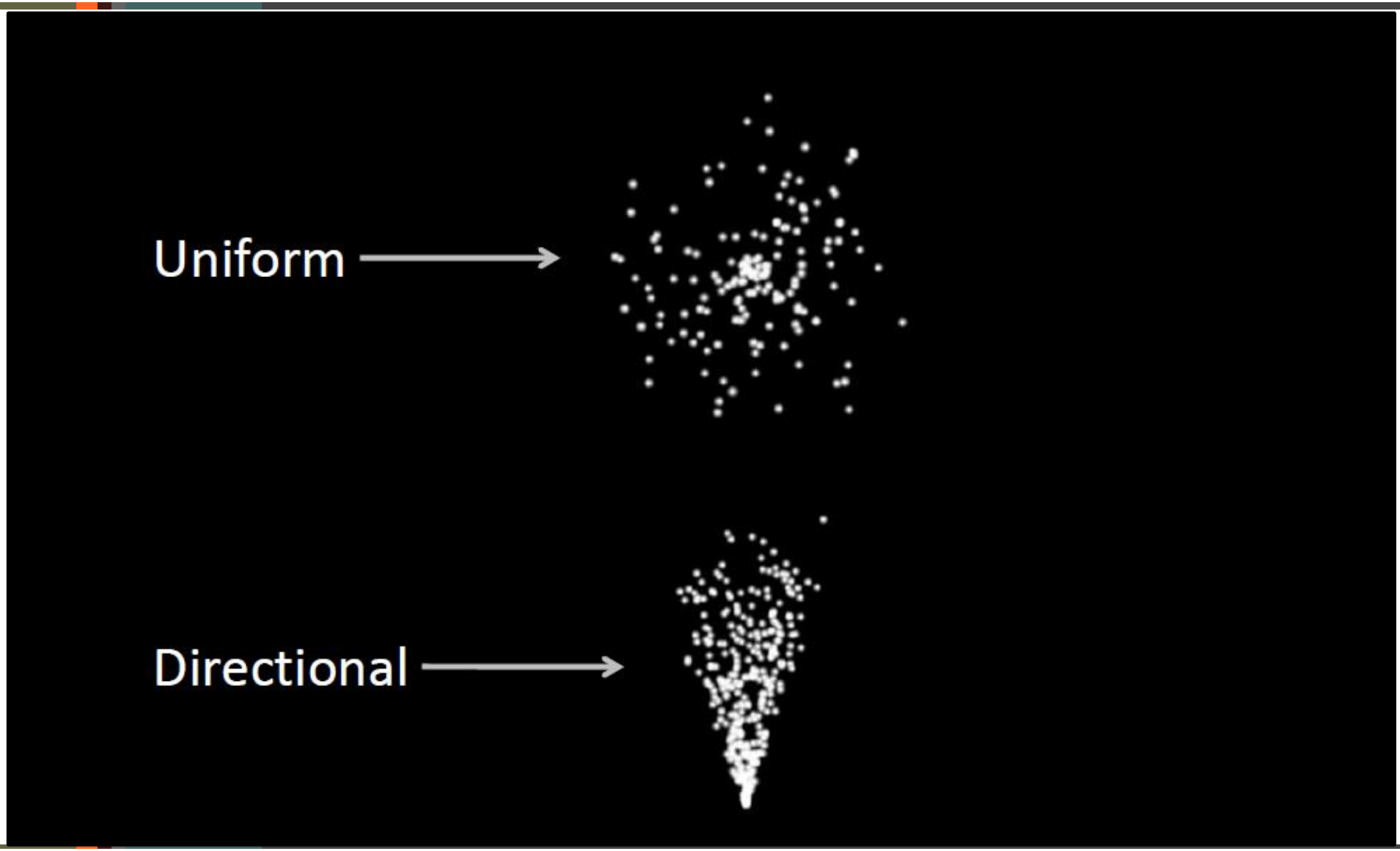
Particle system

Configuration

Emitter

- **What an Emitter do:**
  - **Emit particles**
  - **Controls how to emit particles**
    - Birthrate, velocity, direction… etc.
  - **Holds a zone to generate particles**
    - Can be a point, textfield or any bitmapdata using alpha mask etc..

Emitter Type(Control)



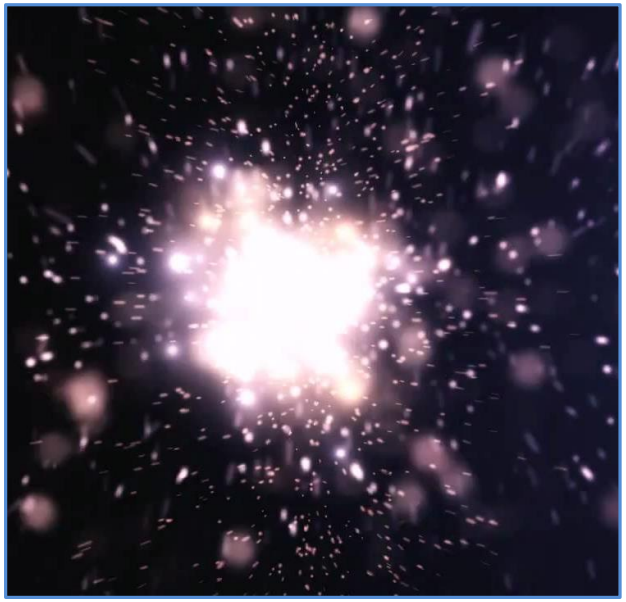
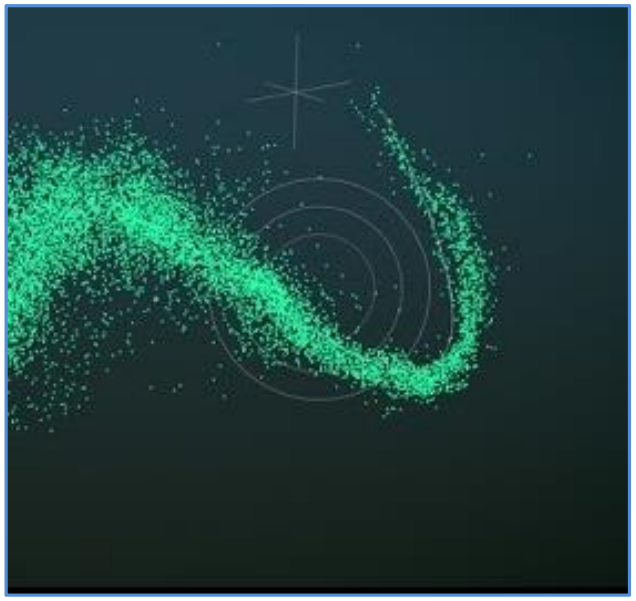
Emitter Type(Zone)

Point →

TextField →

Creating Particles

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

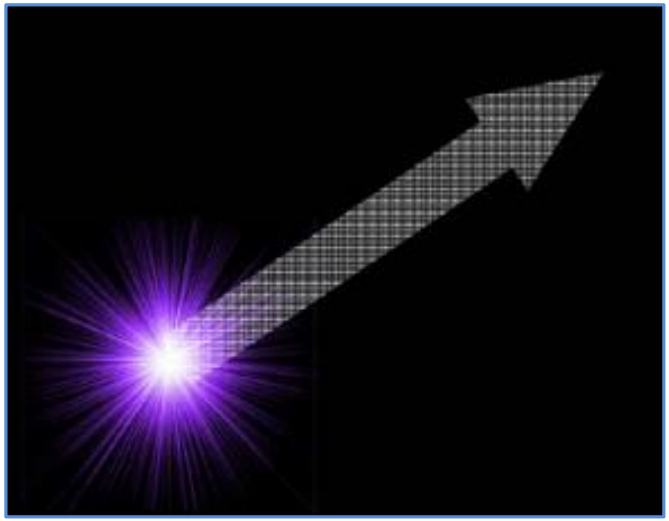




This is where user controls animation

Particle Attributes

- **Creation—number, initial conditions**
  - **Position / Velocity**
    - Surface of emitter shape
    - Vertex of polygonal object
    - Randomness
  - **Size**
  - **Color**
  - **Transparency**
  - **Shape**
  - **Lifetime**
  - **Etc.**

What control handles do we want/need?



Simulator

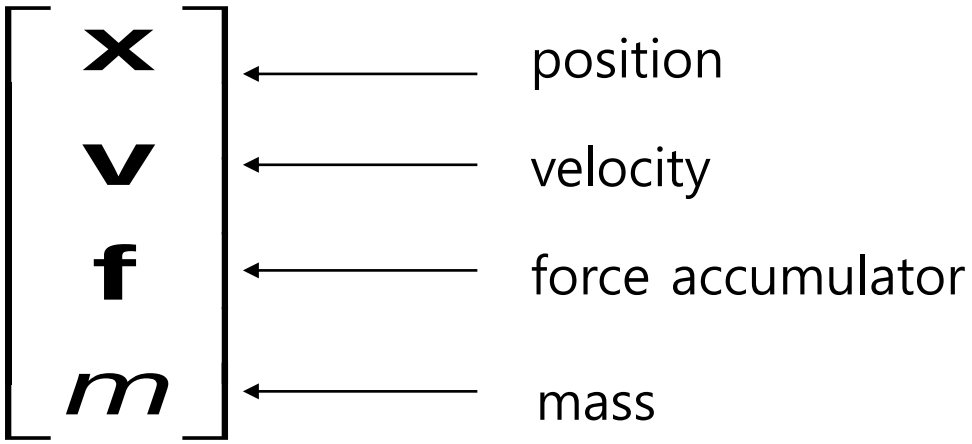
- **What a Simulator do:**
  - **Control the motion of every particle**
    - **Update particles**
      - Based on **physics** system update the particle attributes
    - **Delete particles**
      - Where particle density is high
      - Lifetime
      - Random

Physics of Particle System

- 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
  - e.g. gravity simulation

Particle Structure

- **How do we represent a particle?**



Velocity & Acceleration

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

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

  - Multiply by time

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

- **Acceleration**
  - The rate that velocity changes

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

  - Useful for gravity , spring, wind etc.

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

Update Step

- For each particle:

  {

  - Acceleration

$$a = F/m$$

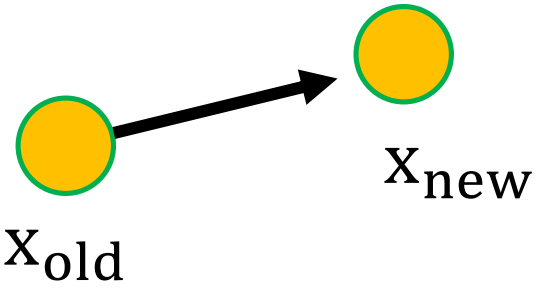
  - Velocity

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

  - Position

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

  }



Particle Systems Structure

- In general, we have a particle system consisting of n particles to be managed over time:

$$\begin{bmatrix} \mathbf{x}_1 \\ \mathbf{v}_1 \\ \mathbf{f}_1 \\ m_1 \end{bmatrix} \begin{bmatrix} \mathbf{x}_2 \\ \mathbf{v}_2 \\ \mathbf{f}_2 \\ m_2 \end{bmatrix} \cdots \begin{bmatrix} \mathbf{x}_n \\ \mathbf{v}_n \\ \mathbf{f}_n \\ m_n \end{bmatrix}$$

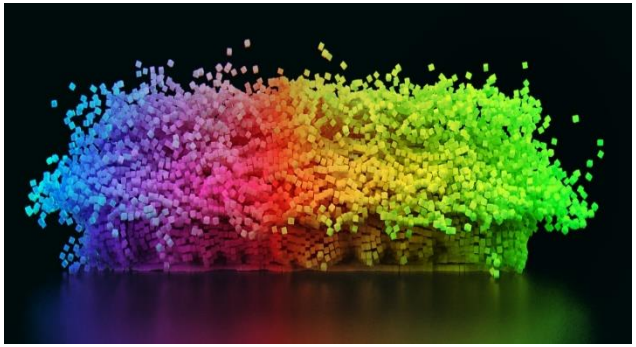
- We can solve the evolution of a particle system

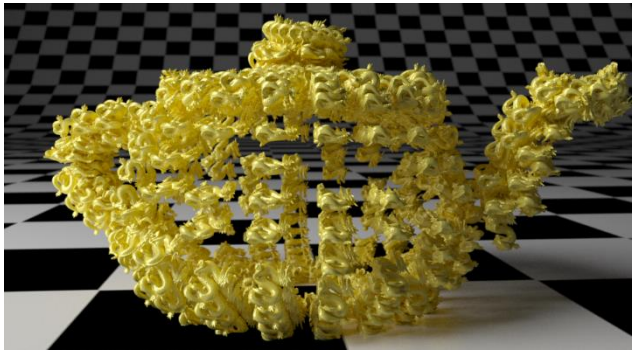
$$\begin{bmatrix} \mathbf{x}_1^{i+1} \\ \mathbf{v}_1^{i+1} \\ \vdots \\ \mathbf{x}_n^{i+1} \\ \mathbf{v}_n^{i+1} \end{bmatrix} = \begin{bmatrix} \mathbf{x}_1^{i} \\ \mathbf{v}_1^{i} \\ \vdots \\ \mathbf{x}_n^{i} \\ \mathbf{v}_n^{i} \end{bmatrix} + \Delta t \begin{bmatrix} \mathbf{v}_1^{i} \\ \mathbf{f}_1^{i} / m_1 \\ \vdots \\ \mathbf{v}_n^{i} \\ \mathbf{f}_n^{i} / m_n \end{bmatrix}$$

Renderer

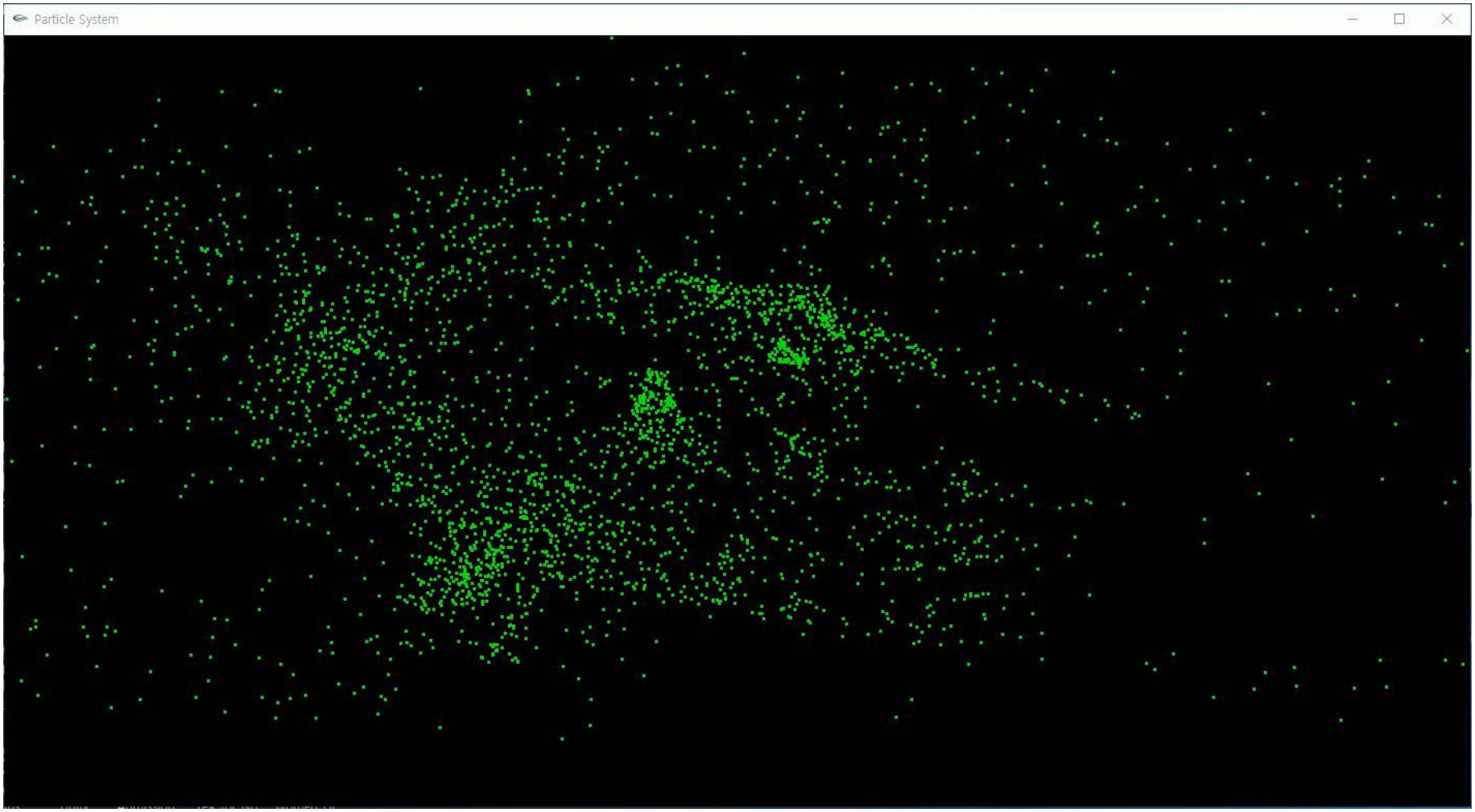
- After the simulation stage, the particle need rendering

- The particle render type of a particle object specifies the form of its particles
  - E.g. you can display as
    - Points
    - Spheres
    - Texture

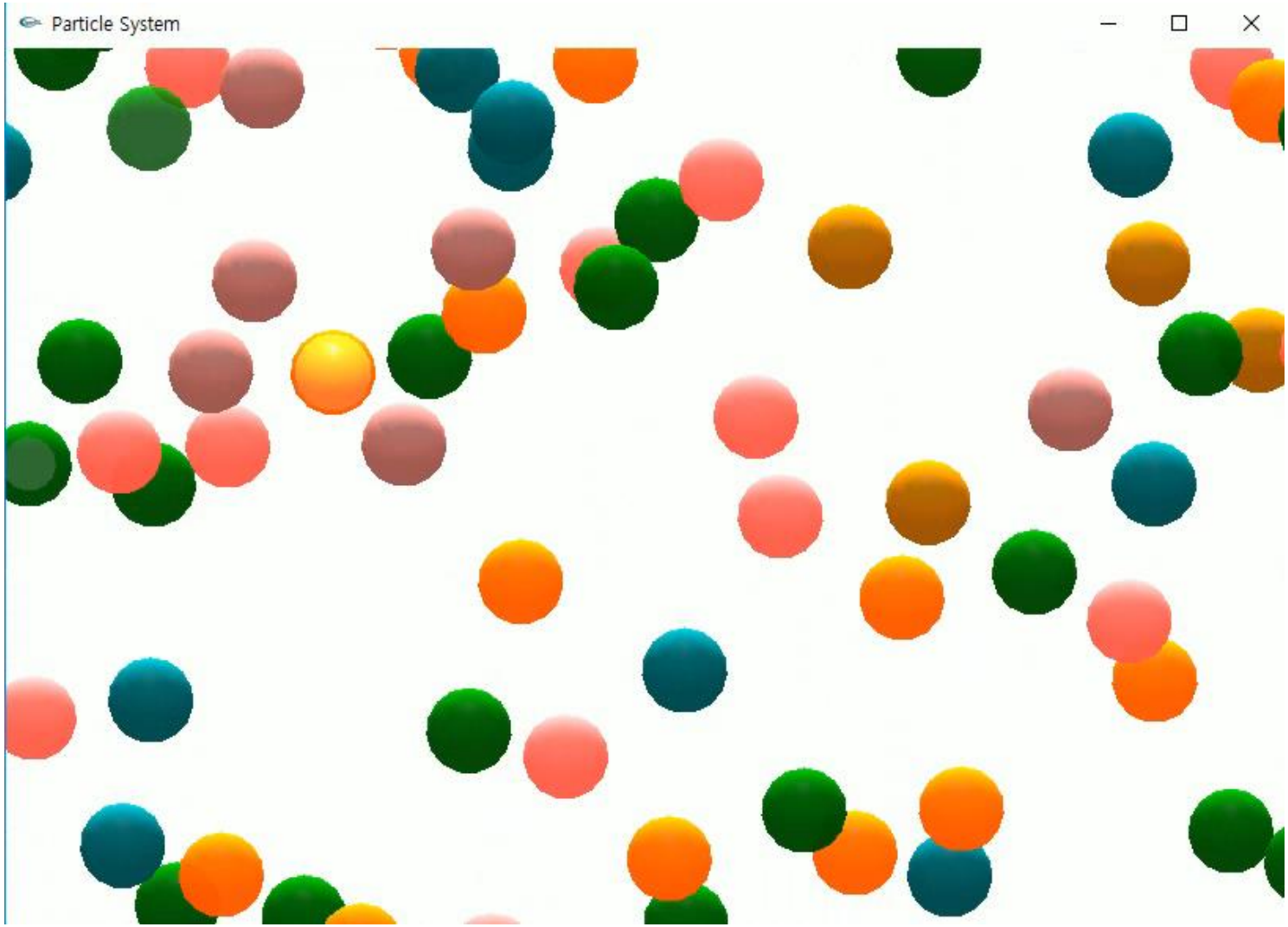




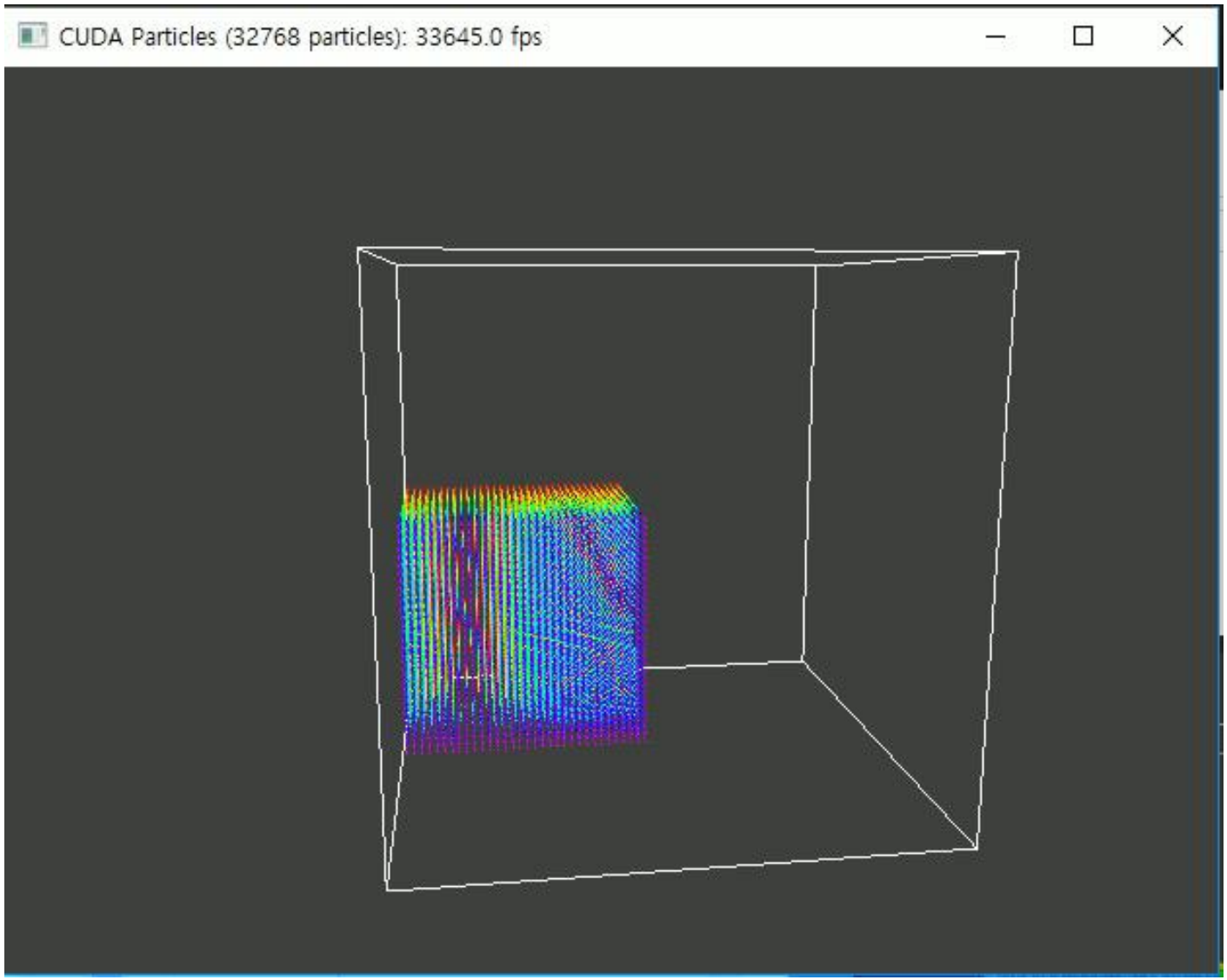
Rendering Particle using Points



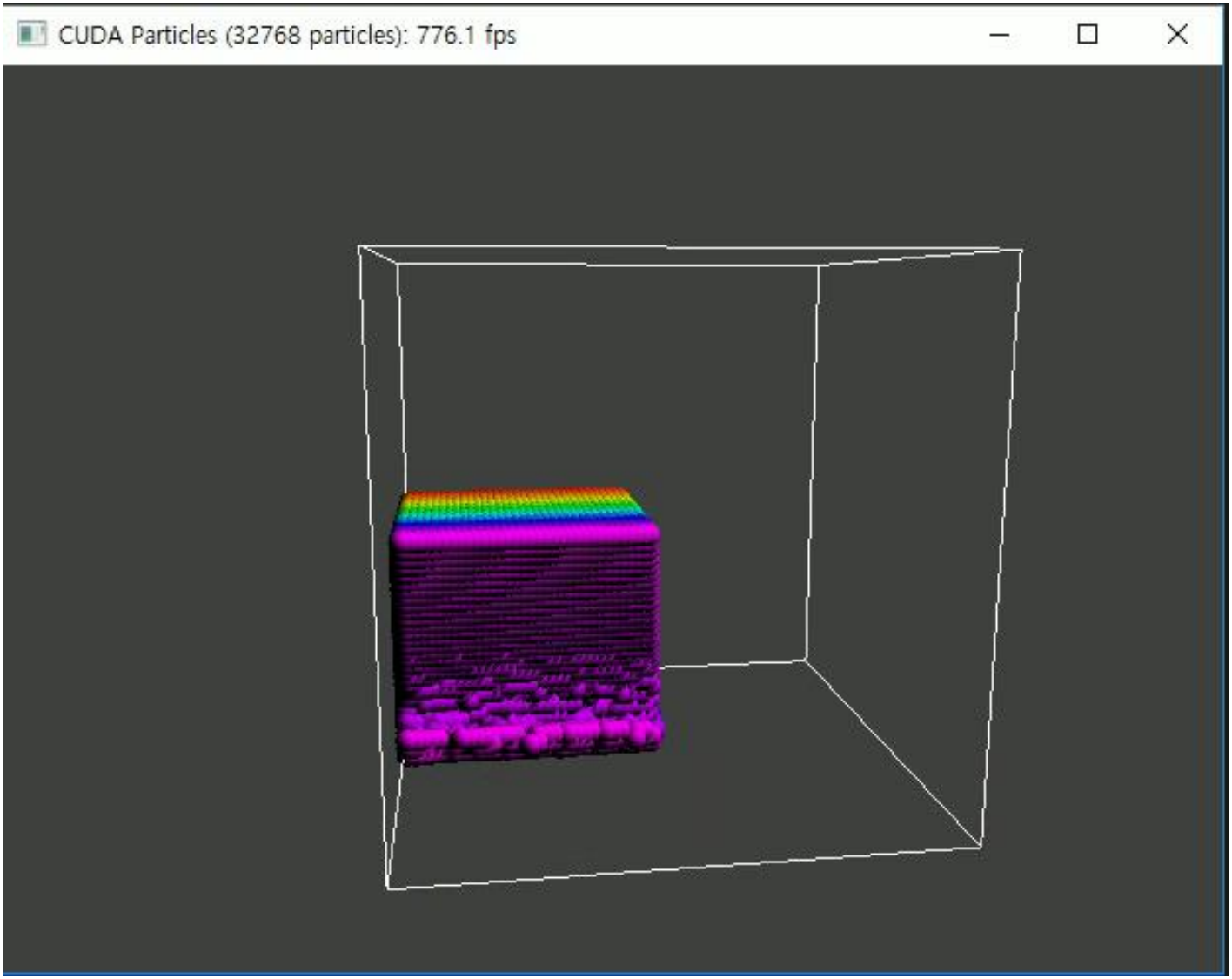
Rendering Particle using Spheres



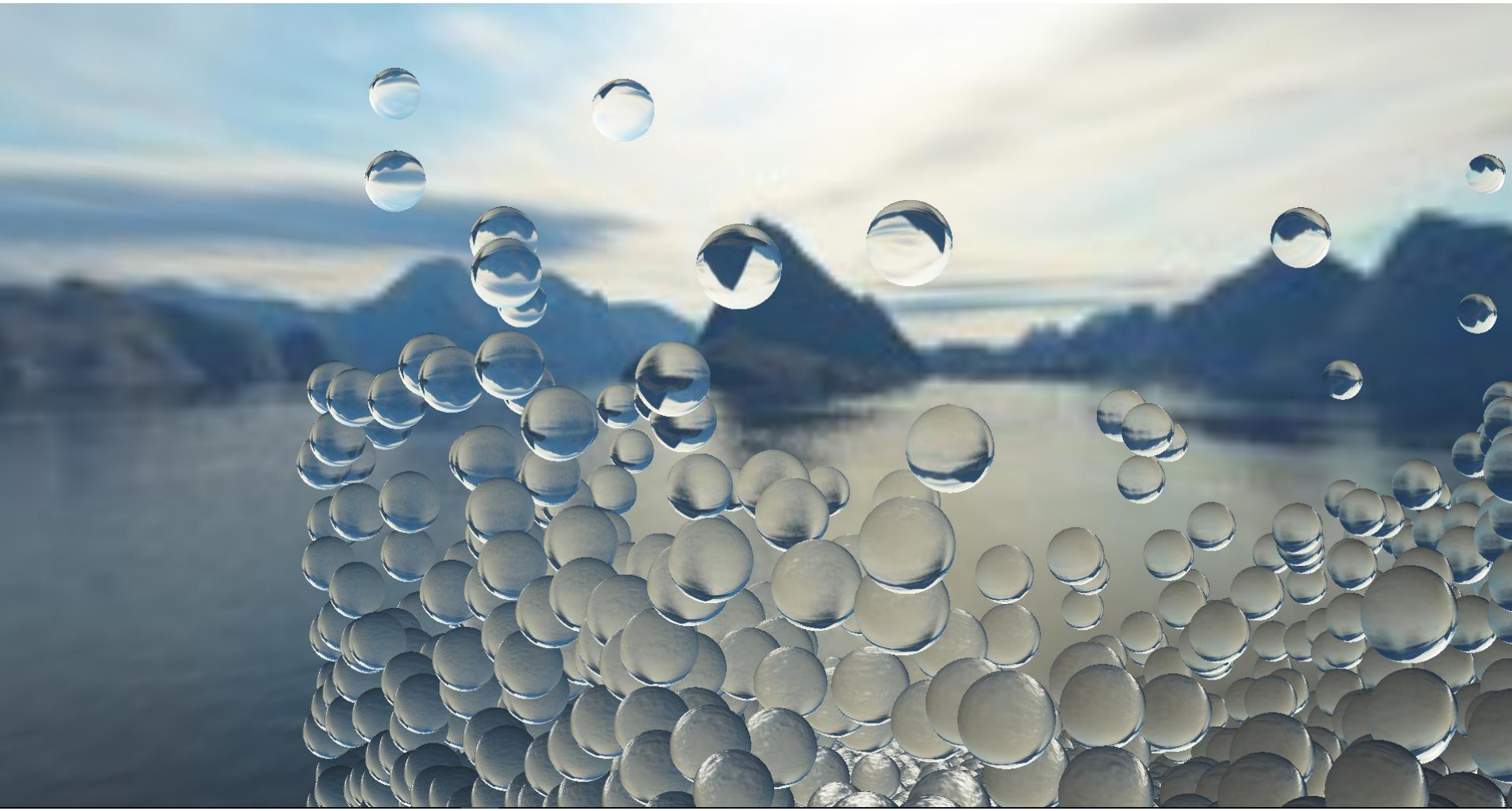
Basic Shader with Points



Basic Shader with Sphere

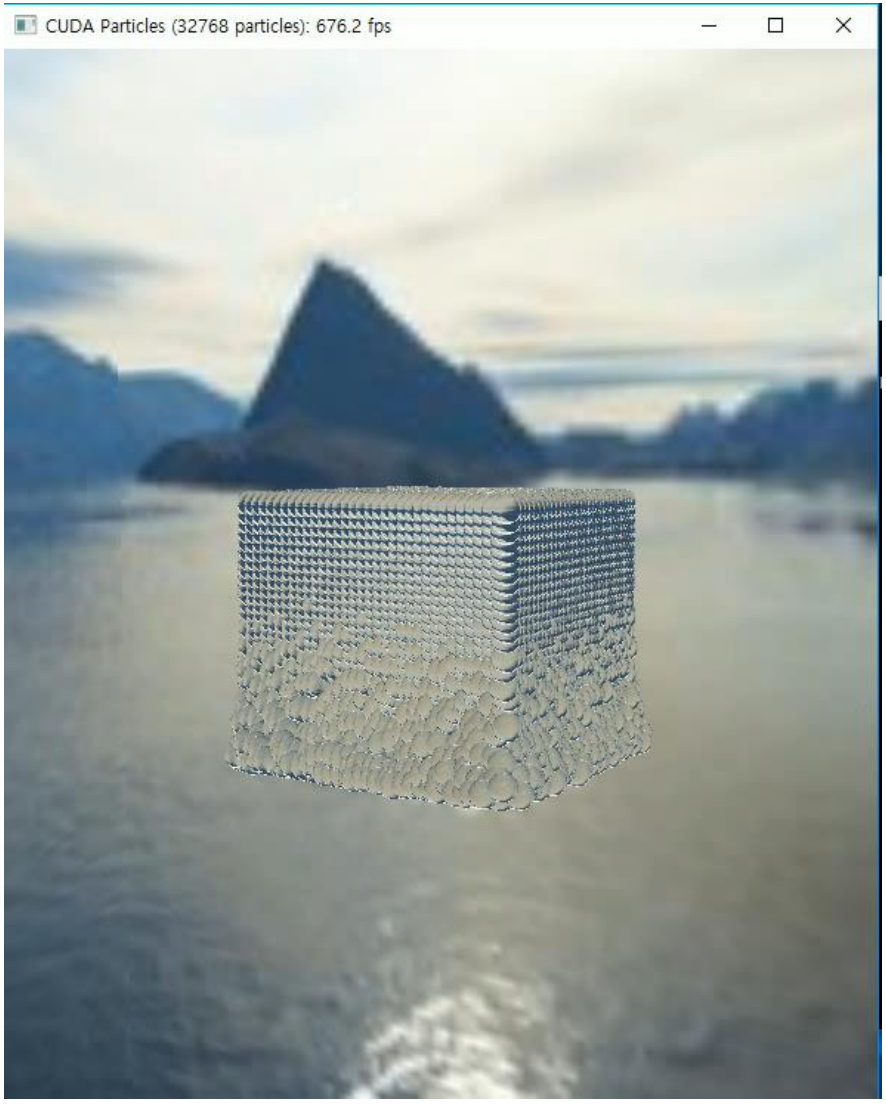


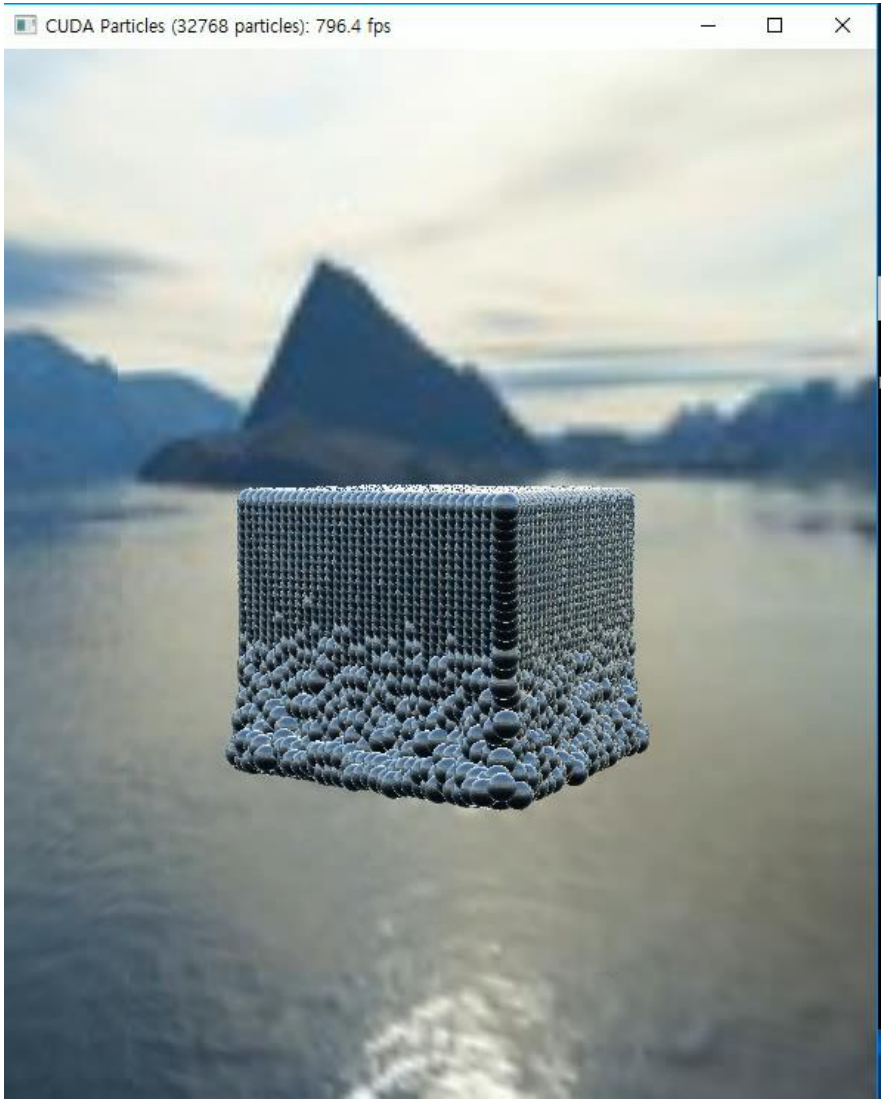
Rendering Particle with Refraction



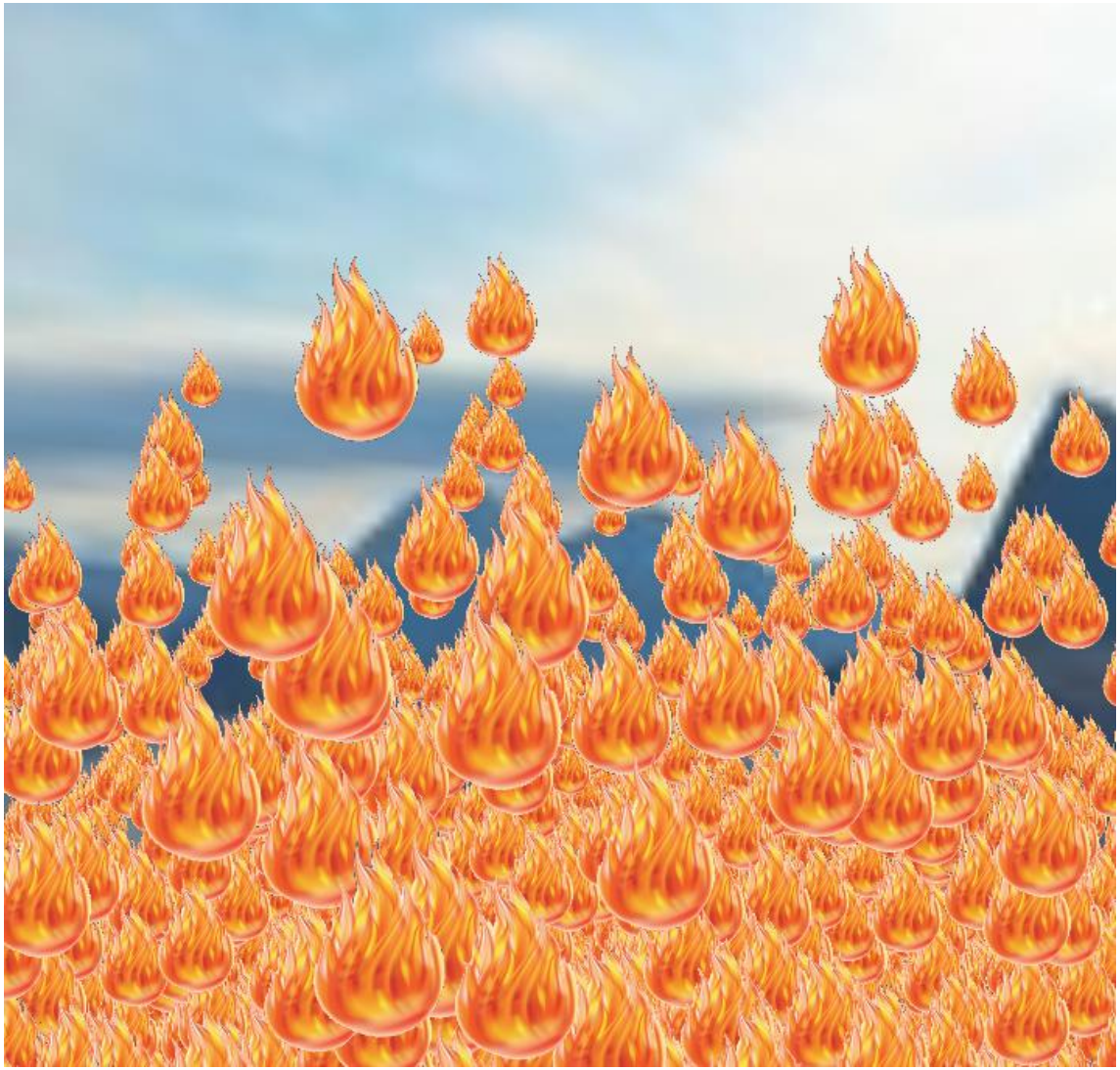
Rendering Particle with Reflection

Refraction & Reflection



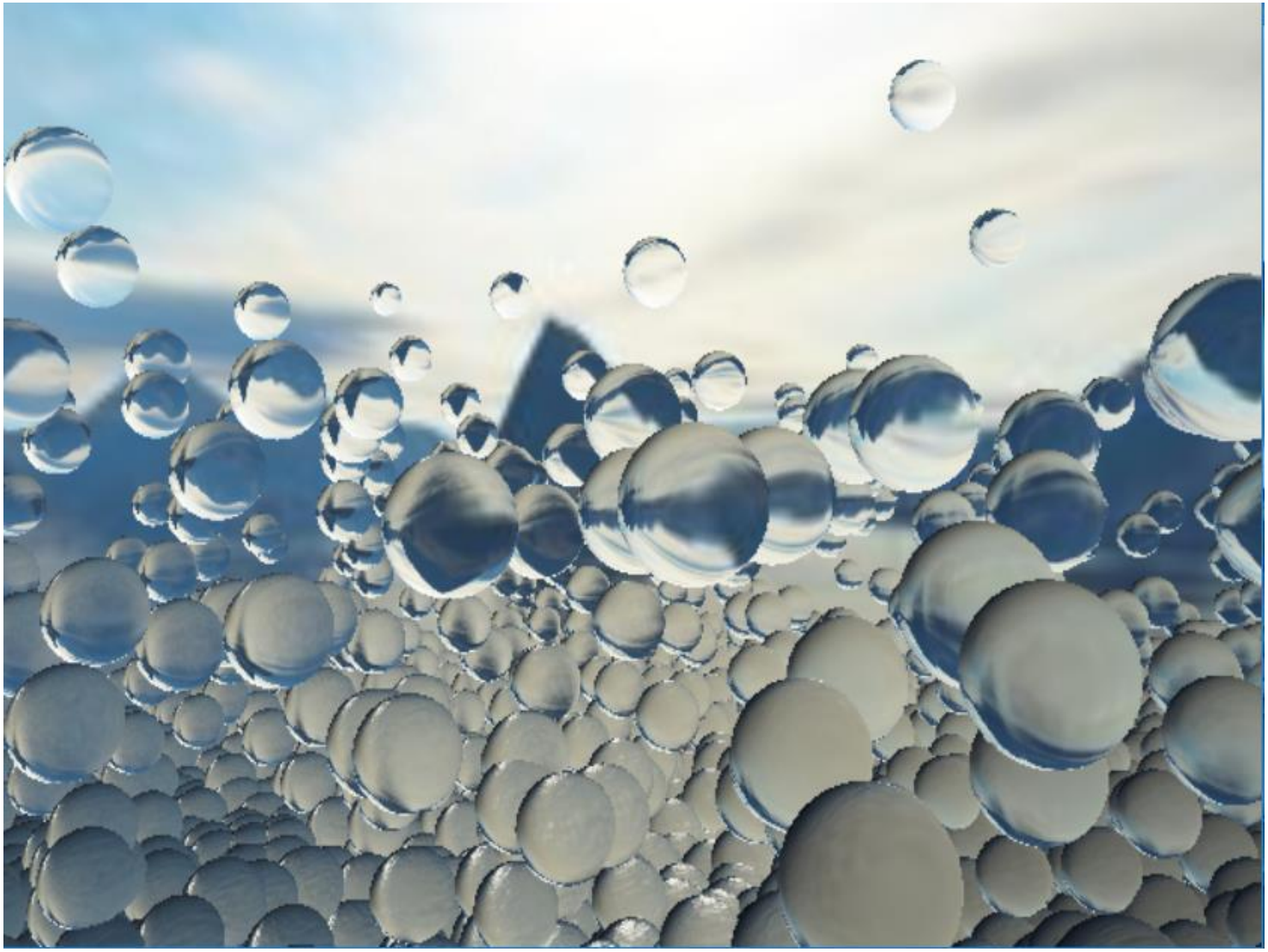


Rendering with Texture

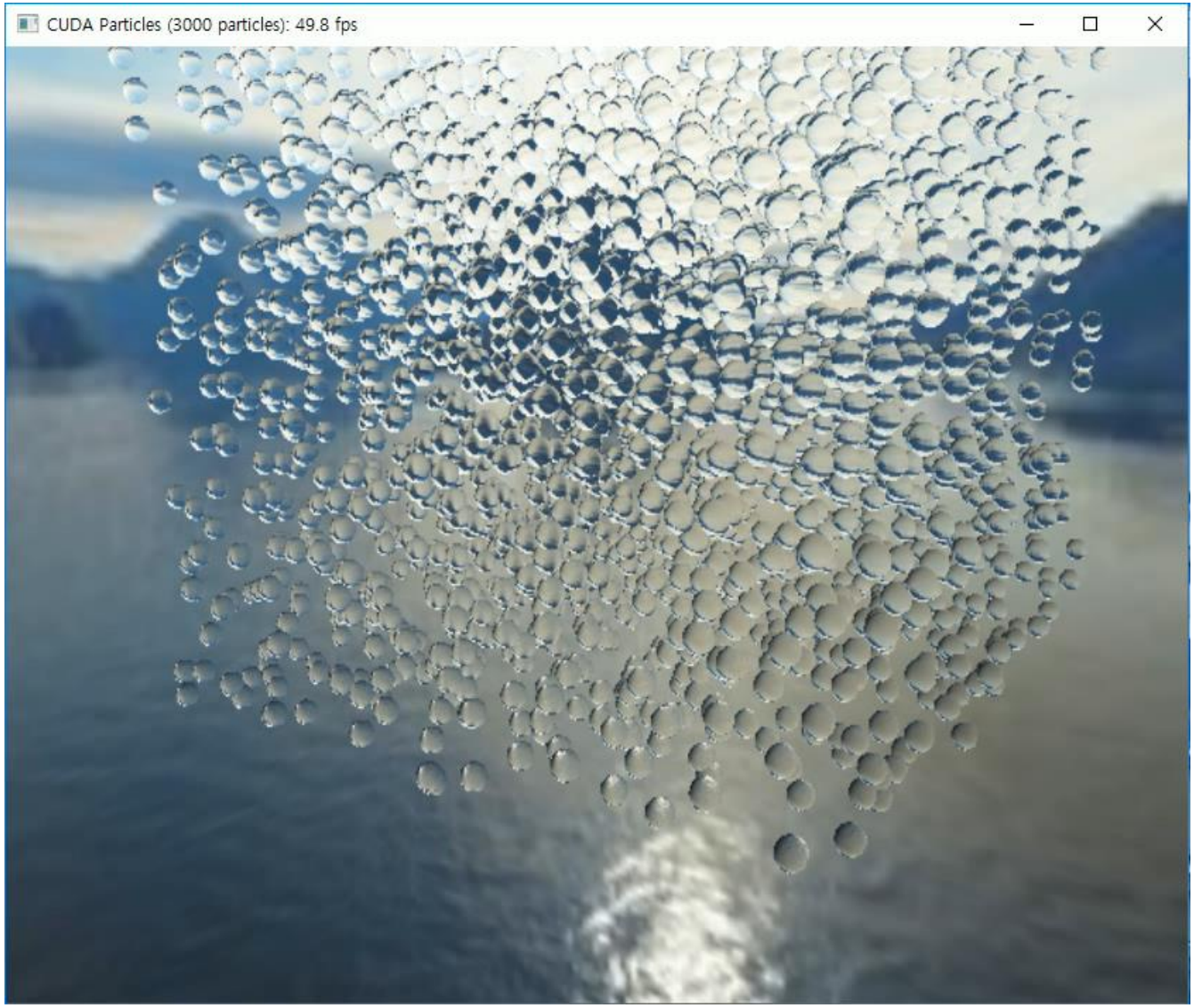


Texture image

Rendering Particle with Raytracing



Rendering Particle with Raytracing



Basic 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**

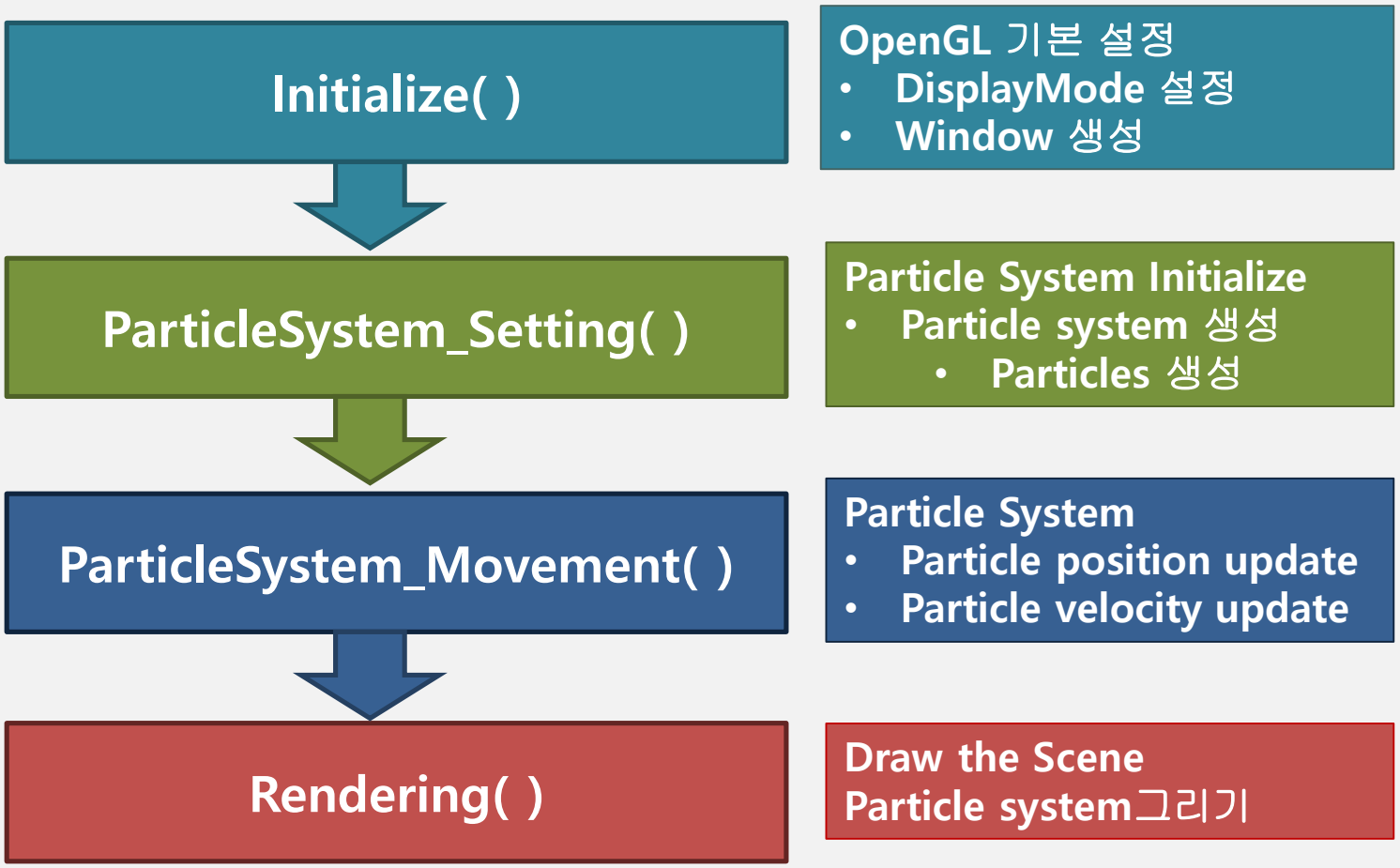
  }

Basic Particles System

Coding Example

Program Flow

Main



Main Function

```
int main(void)
{
  Initialize();                    //Initialize OpenGL
  ParticleSystem_Setting();        //Set particle system
  //Control movement of particles
  glutIdleFunc(ParticleSystem_Movement);
  glutDisplayFunc(Rendering);

  glutReshapeFunc(Reshape);
  glutPassiveMotionFunc(Mouse);
  glutMainLoop();
  return 0;
}
```

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

OpenGL window initialize

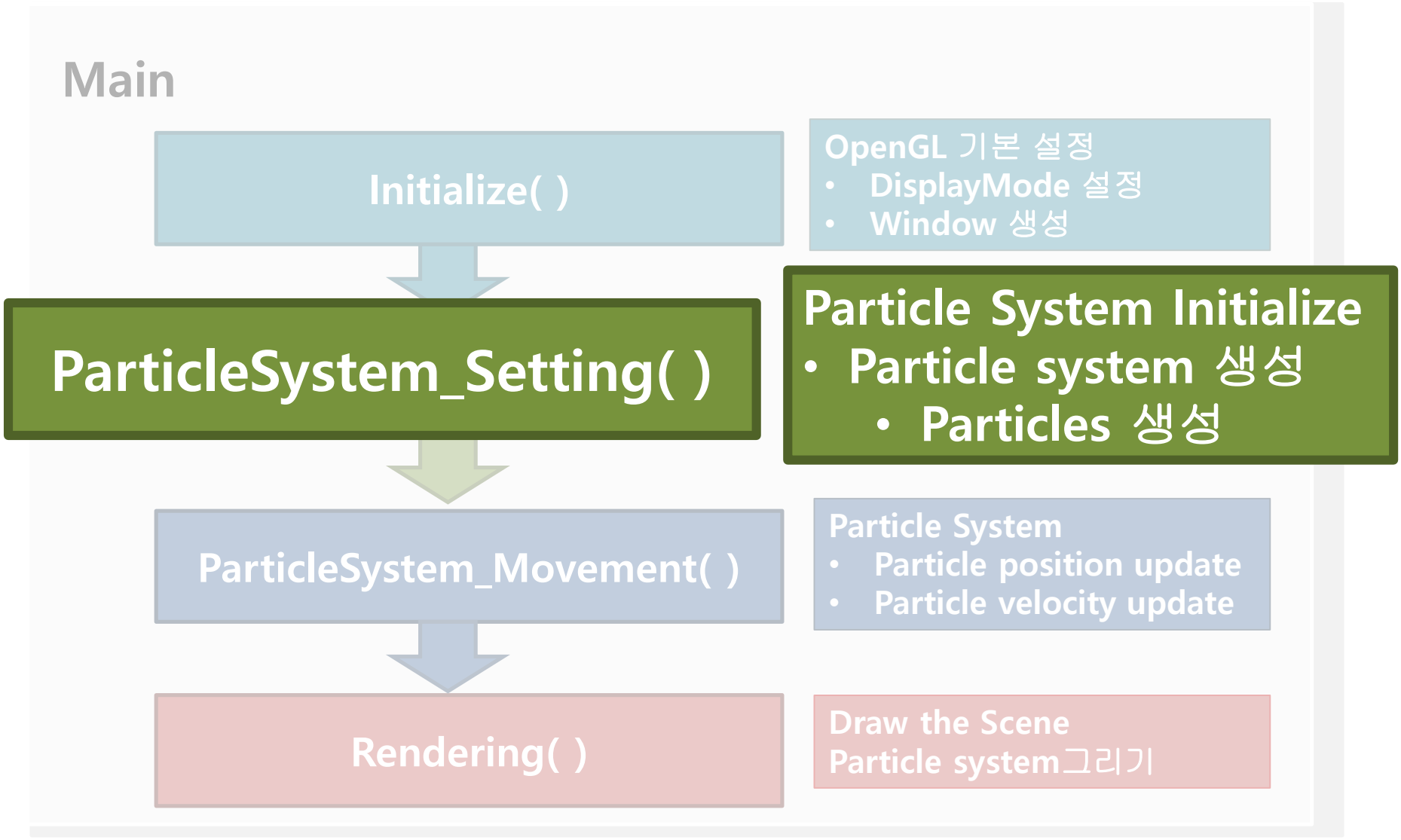
```
float win_width = 800.0, win_height = 800.0;

void Initialize()
{
  //Initialize GLUT
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA);
  glutInitWindowPosition(400, 100);
  //Set the window size
  glutInitWindowSize(win_width, win_height);
 //Create the window and initialize OpenGL
  glutCreateWindow("Particle System");
}
```

Reshape

```
//Called when the window is resized
void Reshape(int w, int h)
{
    win_width = (w==0) ? 1 : w;
    win_height = (h==0) ? 1 : h;
    //Tell OpenGL how to convert from coordinates to pixel values
    glViewport(0, 0, win_width, win_height);
    //Switch to setting the camera perspective
    glMatrixMode(GL_PROJECTION);
    //Set the camera perspective
    glLoadIdentity(); //Reset the camera
    glOrtho(-LENGTH, LENGTH, -LENGTH, LENGTH, -LENGTH, LENGTH);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}
```

Particle System Setting



ParticleSystem_Setting()

```
const int NUMBER_OF_PARTICLES=1000;

particle_system ParticleSystem;

void ParticleSystem_Setting()
{
  ParticleSystem.init(NUMBER_OF_PARTICLES);
  ParticleSystem.set_gravity(vec3d(0,0,0));
}
```

Particle Class

```
class particle
{
 public:
  float mass;
  vec3d velocity;
  vec3d position;
  //Function to Initialize particle attributes
  void init();
  //Function to advance state of particle by time t in ms and force in
  given direction
  void Movement(float, vec3d);

  particle();
 ~particle(void);
};
```

Particle attributes

Particle System Class

```
#include <vector>
#include "particle.h"
using namespace std;
const int MAX_PARTICLES=10000;
const int FORCE_MAG=2000;
class particle_system
{
  public:
  vector<particle> particles;
  vec3d gravity_point;
  void init(int);                    //Construct system given n number of particles
  //Function to advance state of particle system by time t in ms
  void Movement(float);
  void set_gravity(vec3d);       //Function to set gravity point
  particle_system();
  ~particle_system();
};
```

Particle System Initialize

```
void particle_system::init(int n)
{
  if (n > MAX_PARTICLES)
          n = MAX_PARTICLES;
  for (int i = 0; i < n; i++)
  {
    particle temp;                    //Create a particle
    particles.push_back(temp);        //Push a particle in particle_system:particles
                                        (use <vector> header file)
  }
  for (int i = 0; i < particles.size(); i++)
  {
    particles[i].init();
  }
}
//Function to set gravity point
void particle_system::set_gravity(vec3d gravity)
{
   gravity_point = gravity;
}
```

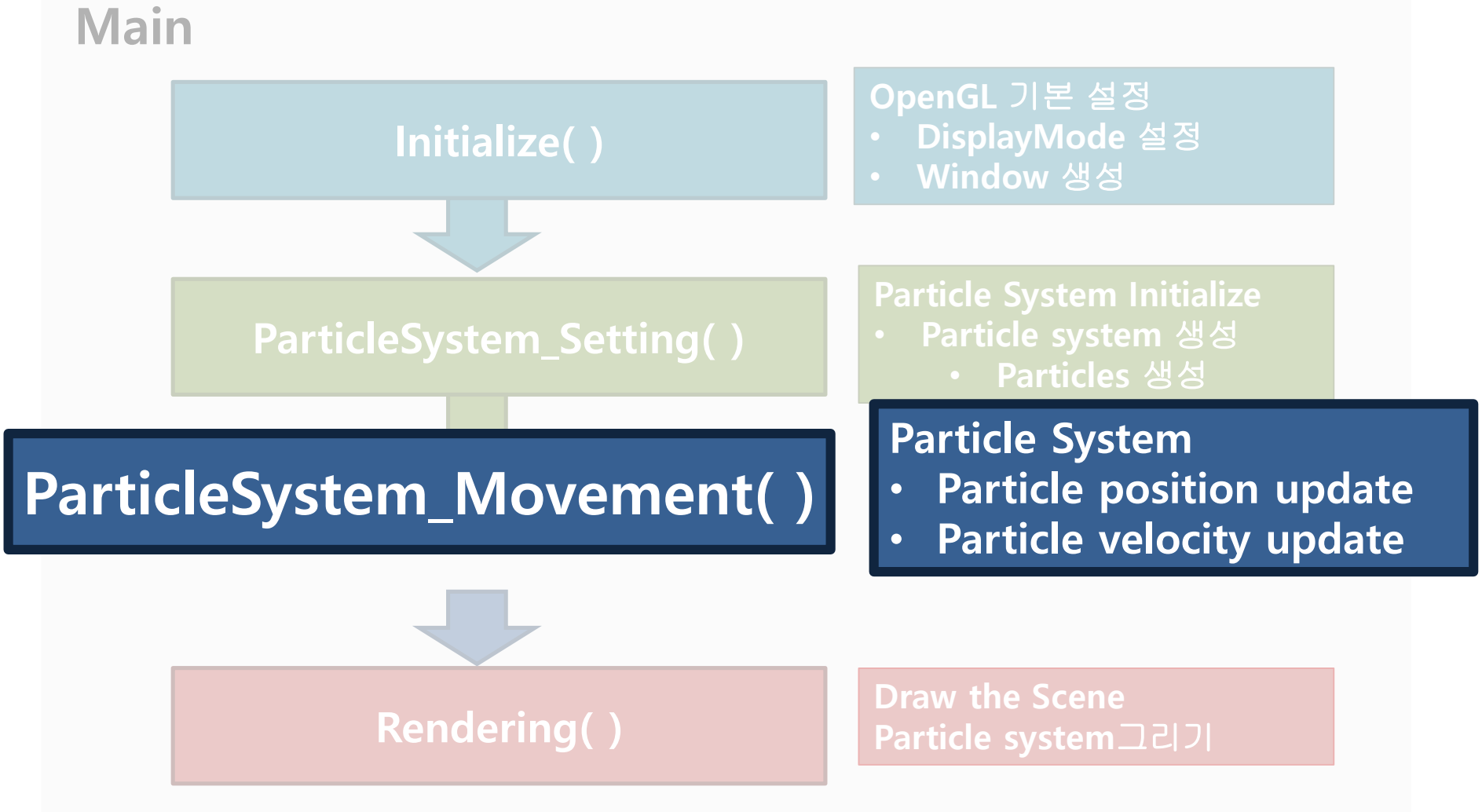
Particle Initialize

```
void particle::init( )
{
 //Initialize mass, velocity, position with random values
  mass = rand() % (MAX_MASS - MIN_MASS) + MIN_MASS;
  velocity = vec3d(rand_float(), rand_float(),rand_float());
  position = vec3d( (1 - 2 * rand_float())*LENGTH, (1 -
rand_float())*LENGTH,  (1 - 2 *rand_float())*LENGTH);
}

//Random function
float rand_float()
{
  float value = rand() / float(RAND_MAX);
  return value;
}
```

```
const int MIN_MASS=2;
const int MAX_MASS=8;
const int MIN_INIT_VELOCITY=10;
const int MAX_INIT_VELOCITY=100;
const int MAX_VELOCITY=200;
const float LENGTH=100.0;
```

Particle System Movement



ParticleSystem_Movement()

```
const float TIME_STEP = 1.0f;

void ParticleSystem_Movement()
{
  ParticleSystem.Movement(TIME_STEP);

  glutPostRedisplay();
}
```

Particle System Movement

```
//Function to advance state of particle system by time t in ms
void particle_system::Movement(float time)
{
  for (int i = 0; i < particles.size(); i++)
  {
    vec3d force = (gravity_point - particles[i].position).unit()*FORCE_MAG;
    particles[i].Movement(time, force);
  }
}
```

Particle Movement

```
void particle::Movement(float t, vec3d force)
{
  vec3d acc = force / mass;                    //Calculating acceleration
  velocity = velocity + acc*(t /1000);         //Calculating velocity
  if(velocity.mag() >= MAX_VELOCITY)
     velocity = vec3d(velocity.unit()*MAX_VELOCITY);
  position = position+velocity*(t /1000.0); //Changing position
  //Boundary
  if(position.x <= -LENGTH) position.x = LENGTH;
    else if(position.x >= LENGTH) position.x = -LENGTH;
  if(position.y <= -LENGTH) position.y = LENGTH;
    else if(position.y >= LENGTH) position.y = -LENGTH;
  if(position.z <= -LENGTH) position.z = LENGTH;
    else if(position.z >= LENGTH) position.z = -LENGTH;
}
```

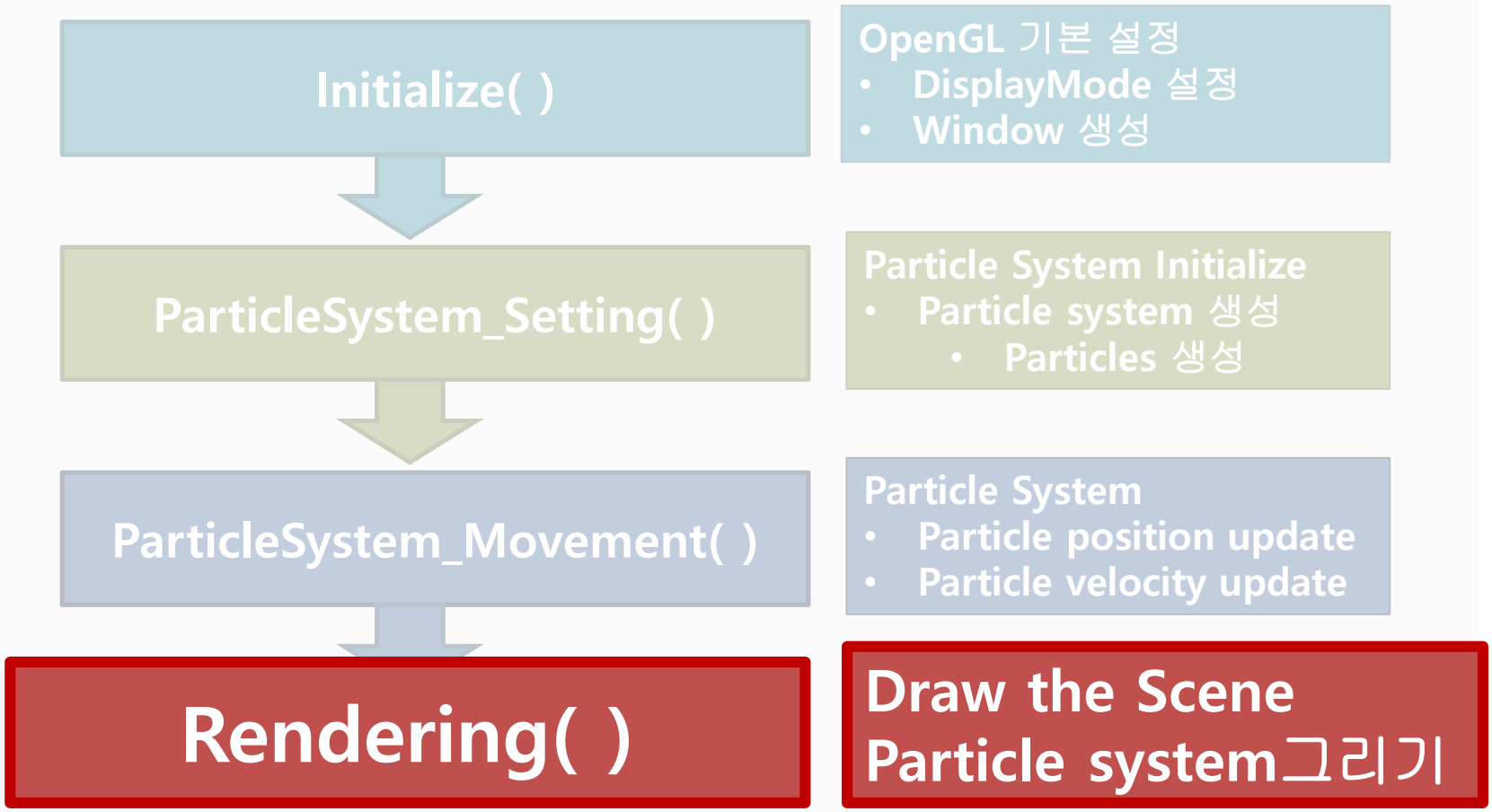
Mouse()

```
//Handle mouse movement, set force
void Mouse(int x, int y)
{
  float ww_ratio = float(x)/win_width;
  float wh_ratio = float(y)/win_height;

  ParticleSystem.set_gravity(vec3d((2 * ww_ratio - 1)*LENGTH, (1 - 2 * wh_ratio)*LENGTH, 0));
}
```

Rendering

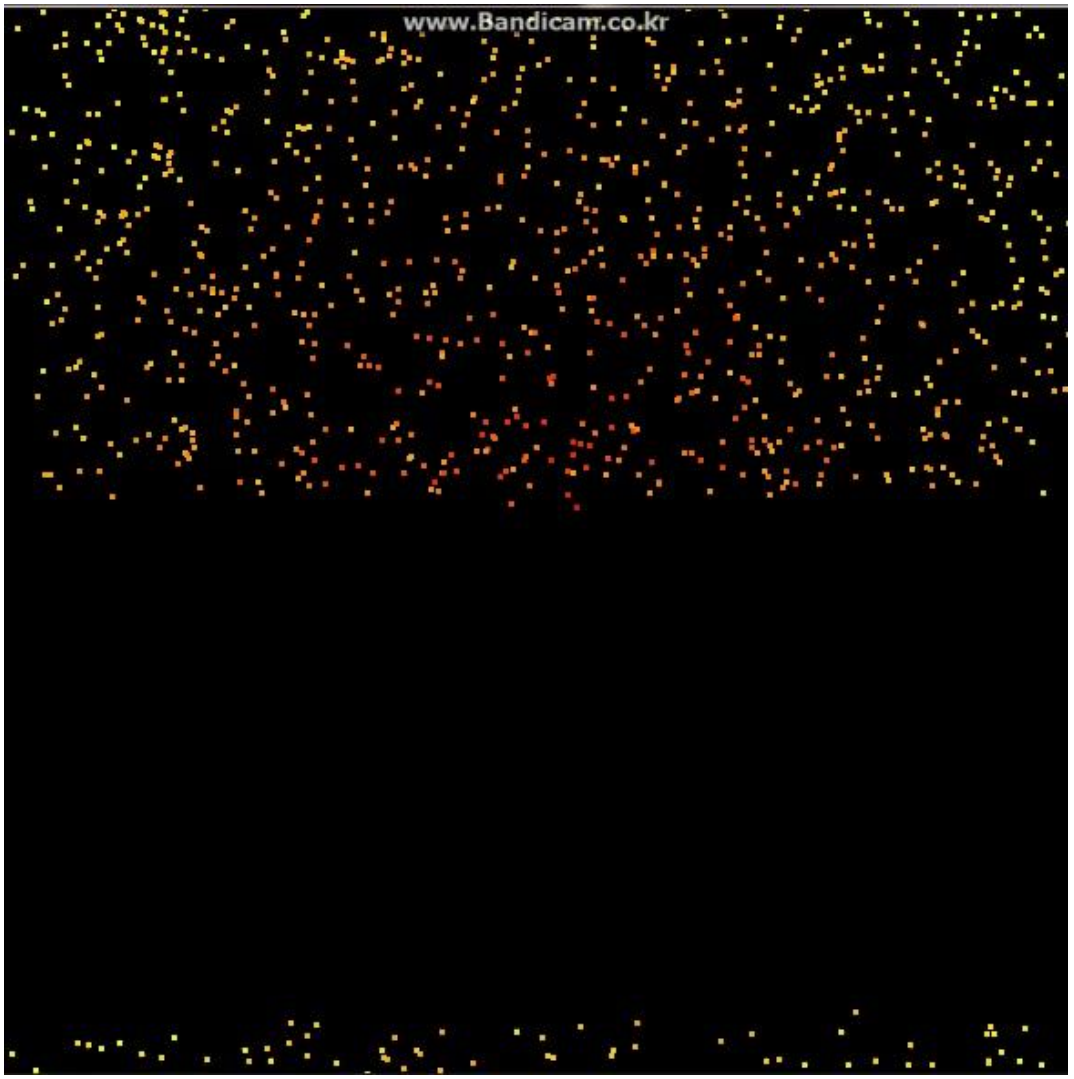
Main



Rendering()

```
void Rendering()
{
  glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
  glLoadIdentity();
  // Make big points
  glPointSize(3);
  // Draw particles
  for (int i = 0; i < ParticleSystem.particles.size(); i++)
  {
    vec3d pos = ParticleSystem.particles[i].position;
    float k = (ParticleSystem.gravity_point - pos).mag() / (1.5*LENGTH);
    glColor4f(1, k, 0, 1);    // Color setting
    glBegin(GL_POINTS);
       glVertex3f(pos.x, pos.y, pos.z);
    glEnd();
  }
  glutSwapBuffers();
  glutPostRedisplay();
}
```

Result



Practice: Example #1

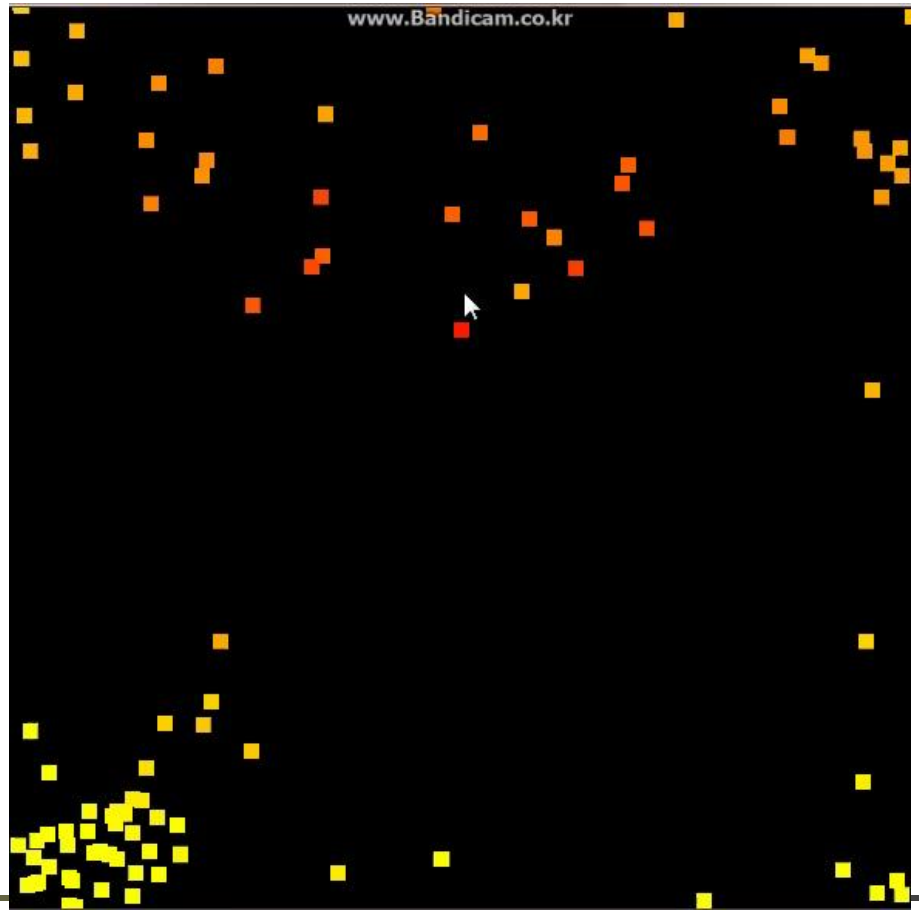
- Copy Sample Skeleton Code
  - cp –r /home/share/24_Particles ./[FolderName]
  - cd [FolderName]
- VGL Setting
  - Run Xming
  - vglconnect [id]@163.152.20.246
  - export VGL_SYNC=1
- Compile & run program
  - make
  - vglrun ./EXE
- Modify Simulation parameters
- Compile & run program again

Example #1: TODO

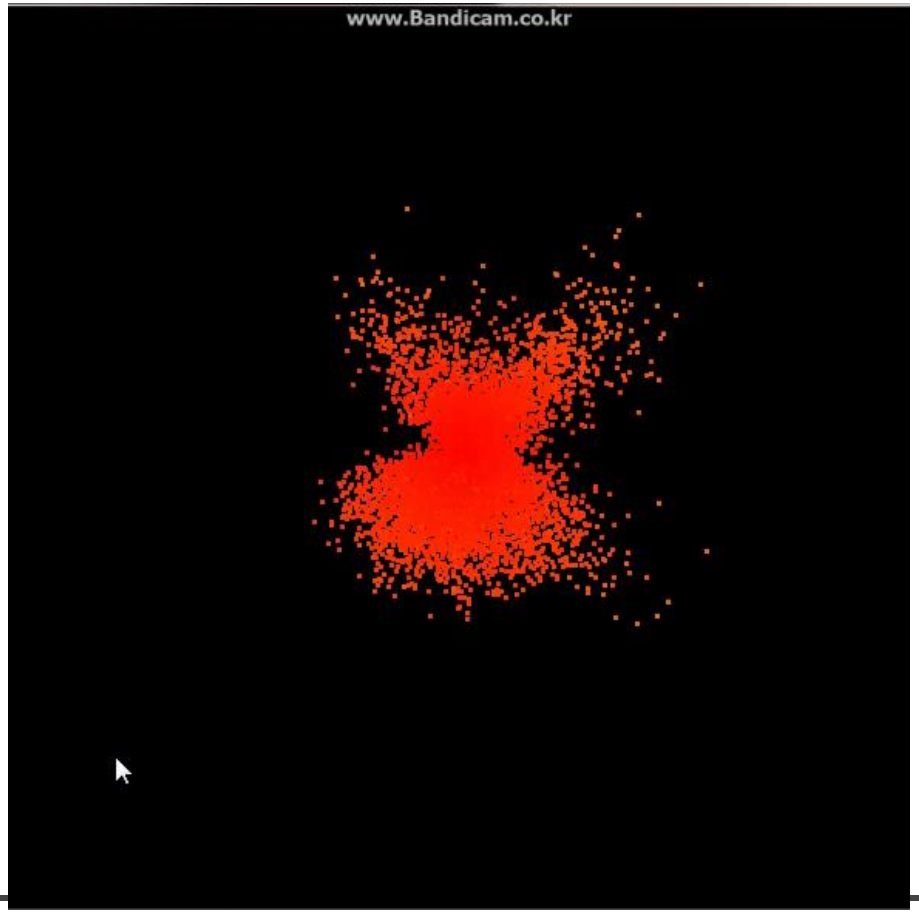
- Modify Point Size
  - glPointSize(float size) @ Display Function
- Modify Parameters for Particle System
  - const int NUMBER_OF_PARTICLES @ Main.cpp
  - const float TIME_STEP @ Main.cpp
  - const int FORCE_MAG @ particle_system.h

Example #1: Results

- Time Step: 5
- Number of Particles : 100
- FORCE_MAG: 2000
- Point Size: 10



- Time Step: 15
- Number of Particles : 10000
- FORCE_MAG: 5000
- Point Size: 3

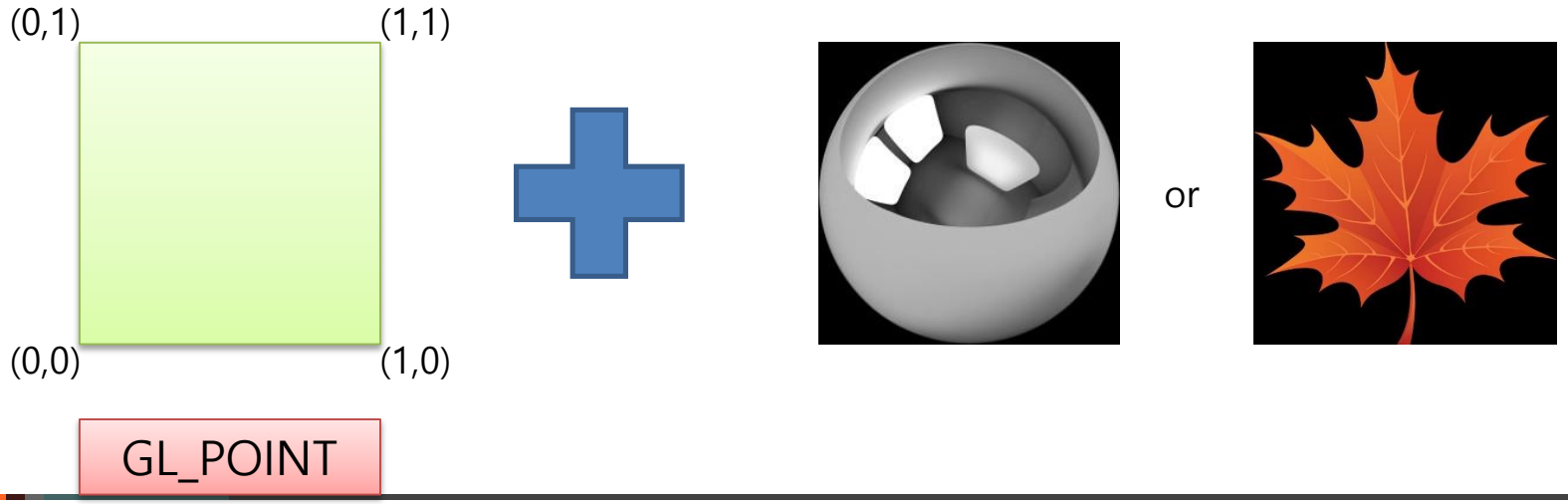


Practice: Example #2

- Copy Sample Skeleton Code
  - cp –r /home/share/25_TexturedParticles ./[FolderName]
  - cd [FolderName]
- Compile & run program
  - make
  - vglrun ./EXE
- Modify Main&Shaders
  - Main.cpp
  - Vertex.glsl
  - Fragment.glsl
- Compile & run program again

Example #2: Texture Mapping

- You can do texture mapping to Particle
- Add 2 function to make texture coordinates at GL_POINTS
  - glEnable(GL_POINT_SPRITE_ARB);
  - glTexEnvi(GL_POINT_SPRITE_ARB, GL_COORD_REPLACE_ARB, GL_TRUE);
  - They are included in skeleton code



Example #2: TODO

- Call functions proper position
  - glUseProgram(program) @ Display Function
  - glBindTexture(GL_TEXTURE_2D, textureID) @ Display Function
- Swap Texture Images@initTexture function
  - sphere.jpg
  - maple.jpg
- Coding Shader
  - Vertex.glsl
  - Fragment.glsl

Shader:: Texture Mapping

- It's not different to basic Texture Mapping to QUADS

Vertex.glsl

```
#version 130
void main() {
        gl_Position = gl_ModelViewProjectionMatrix*gl_Vertex;
}
```

Fragment.glsl

```
#version 130
uniform sampler2D tex; //set 2D Texture@Fragment Shader
void main() {
        vec2 texCoord = gl_TexCoord[0].st;
        vec3 color = texture2D(tex, texCoord.st).rgb;
        if(length(color)<0.01f) discard;
        gl_FragColor = vec4(color,1.0f);
}
```

Shader:: Texture Mapping

- It's not different to basic Texture Mapping to QUADS

Vertex.glsl

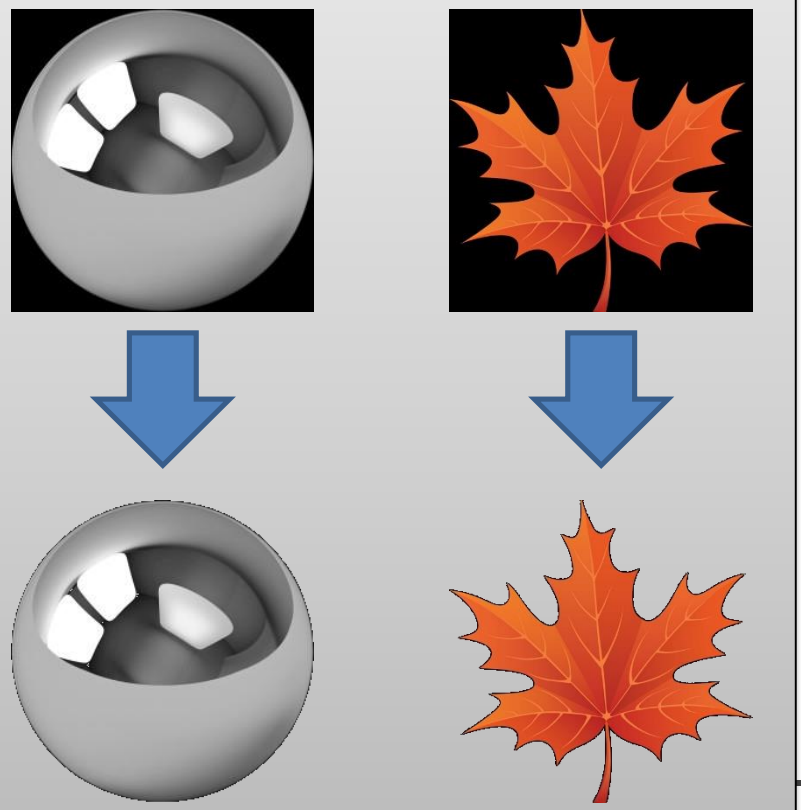
```
#version 130
void main() {
        gl_Position = gl_ModelViewProjec
}
```

Fragment.glsl

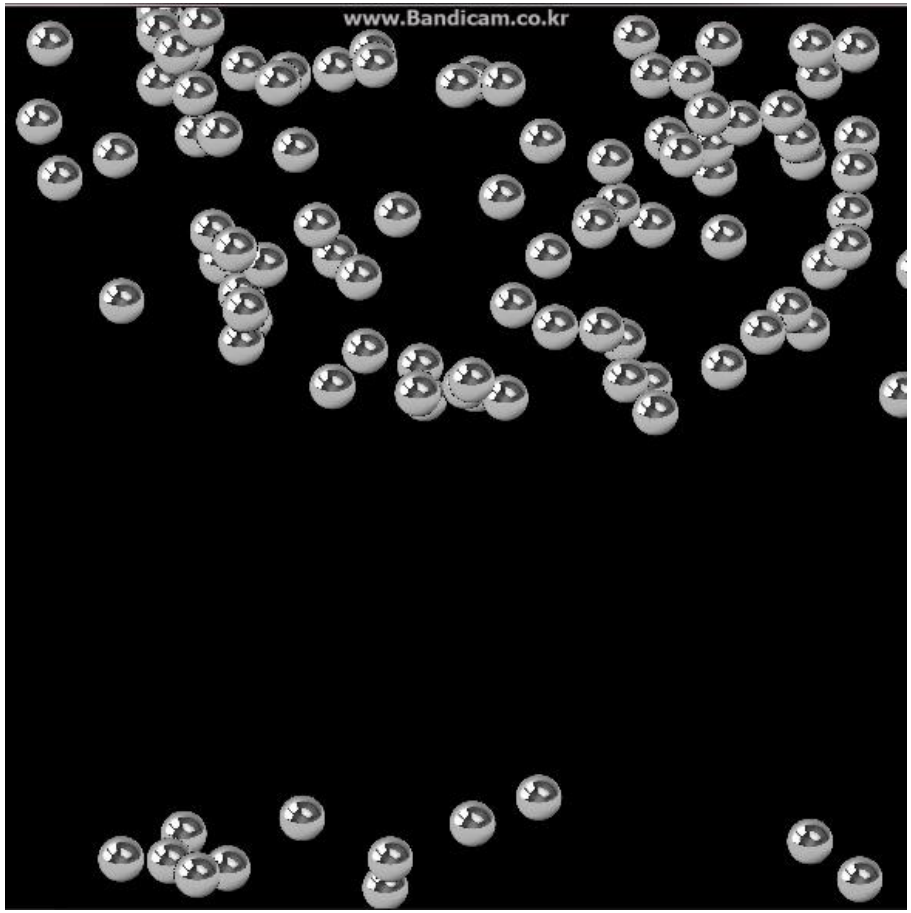
```
#version 130
uniform sampler2D tex; //set 2D Texture@Fra
void main() {
        vec2 texCoord = gl_TexCoord[0].st
        vec3 color = texture2D(tex, texCoo
        if(length(color)<0.01f) discard;
        gl_FragColor = vec4(color,1.0f);
}
```

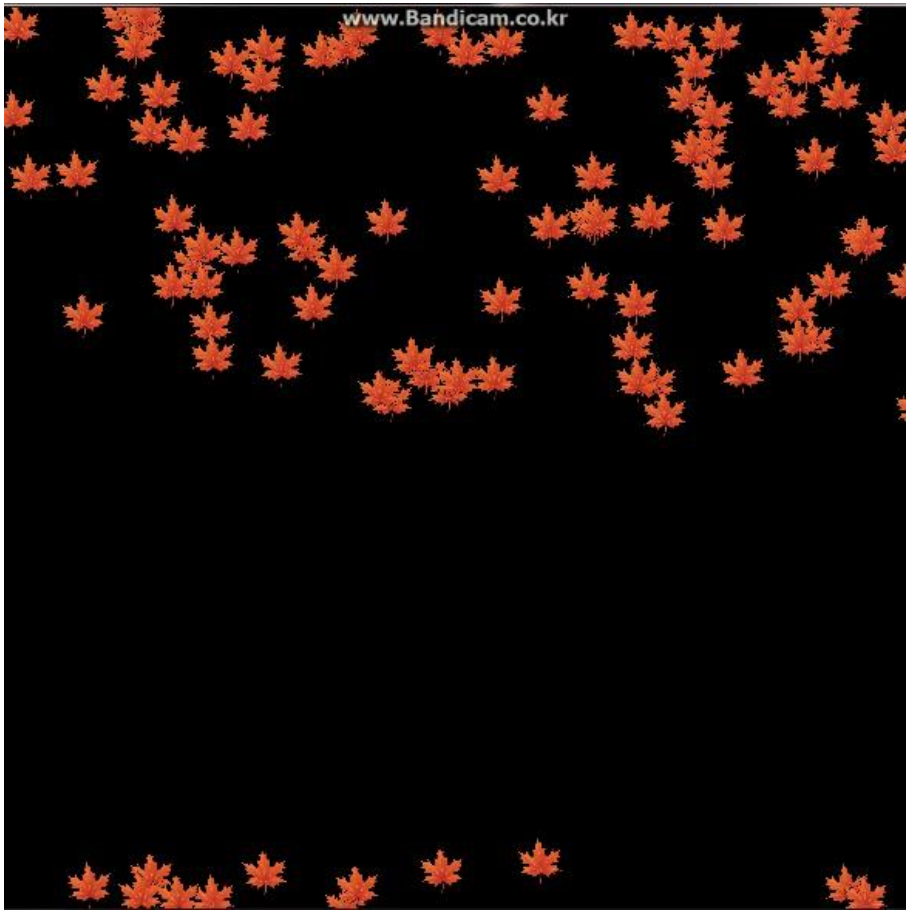
discard: do nothing current pixel

We use this to delete background(Black color)



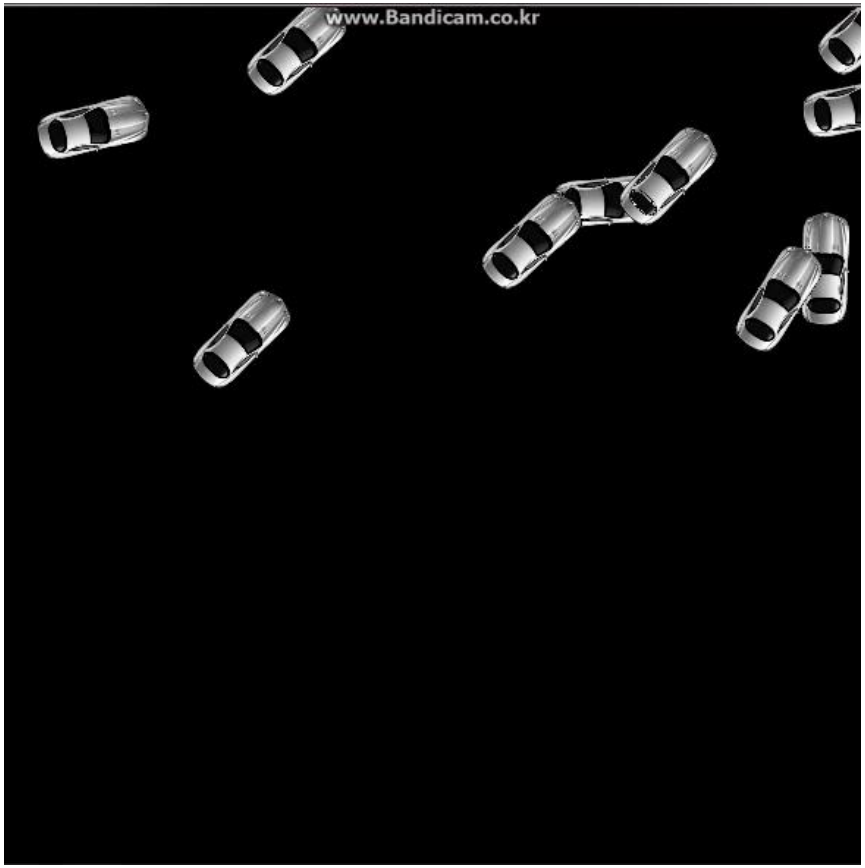
Example #2: Results

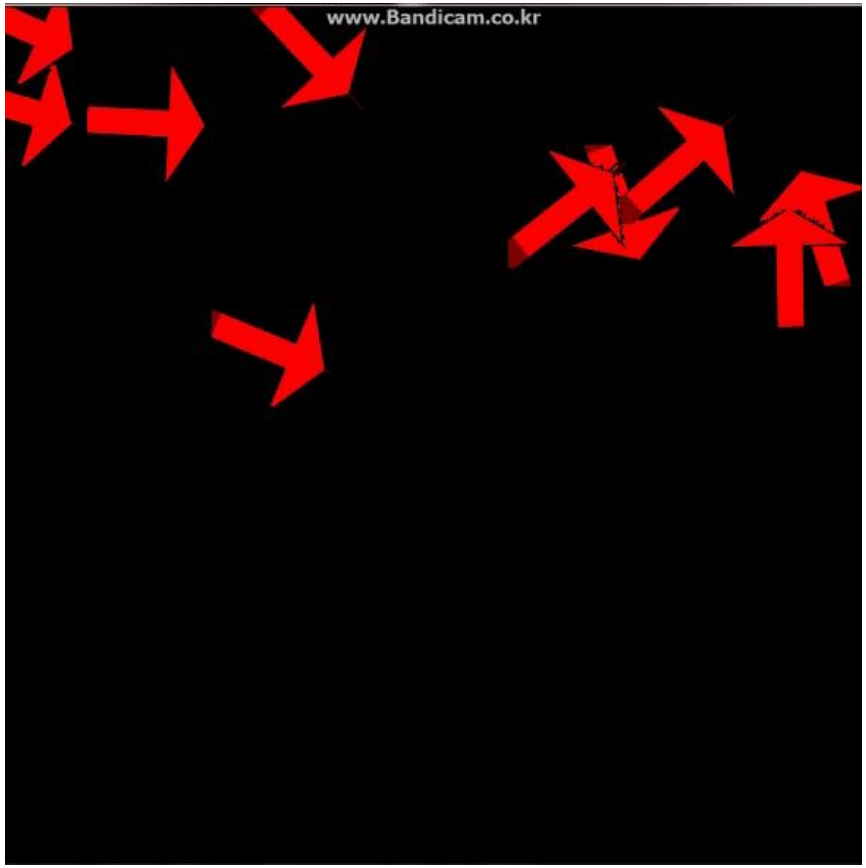




Texture Transformation

- You can make rotated Texture mapped particle





Appendix

STL (Standard Template Library)

- 위 문제를 해결하기 위해 나온 것이 STL Vector Type
  - STL(Standard Template Library)은 C++을 위한 Library
    - 알고리즘, 컨테이너, 함수, 반복자로 구성
    - 어떠한 빌트인 타입 / 사용자 정의 타입과도 함께 사용 가능

    - STL의 변수의 종류
      - Vector
      - Hash Map, Map … etc..

  - Vector는 STL에서 제공하는 동적 배열
    - 런타임에 크기 조절 가능
    - 객체 삽입 / 삭제할 때 자동으로 자신의 크기를 조절
    - 사용하기 쉽고, 구현시간을 대폭 감소 시켜줄 수 있음

STL Vector Class

- Member function
  - push_back
    - add new element at the end of vector
  - pop_back
    - delete last element from the vector
  - insert
    - insert elements
  - erase
    - erase elements
  - assign
    - assign vector content
  - clear
    - clear content
  - Etc…

STL Vector

Declare

- Header Include

```
#include <vector>
```

- 변수 선언하기
  - using namespace를 사용할 경우

```
using namespace std;
vector<float> fVector;
```

  - 사용하지 않을 경우

```
std::vector<float> fVector;
```

Example: Push_back

```
using namespace std;

int main()
{
    // float type 벡터 생성
    vector<float> fVector;

    // 값 삽입
    fVector.push_back(1.0f);
    fVector.push_back(2.0f);
    fVector.push_back(3.0f);

    // 벡터의 크기만큼 순회
    for (int i = 0; i < fVector.size(); i++)
    {
        // 각 값을 출력
        cout << i << "번째 값: " << fVector[i] << endl;
    }

    return 0;
}
```

```
0번째 값: 1
1번째 값: 2
2번째 값: 3
```