

Graphics Project

Implementing a Particle Engine

Prakash A Arul
arul@cs.utah.edu

Introduction

In this project, I have implemented a particle engine to allow the user to create different particle systems for simulating natural phenomenon like fire, smoke, snow and fountains. The particle engine has been compiled into a library, so that any user can link his code to it for creating particle systems.

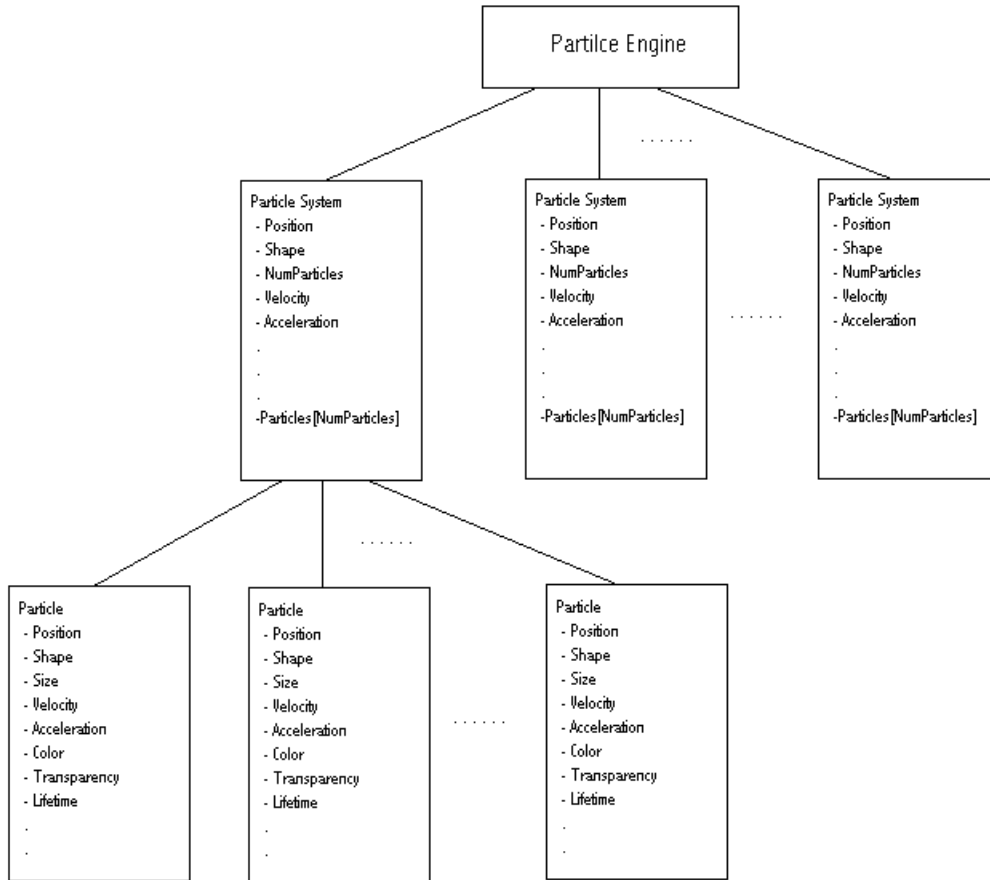
A particle system is a large set of simple primitive objects which are processed as a group to represent an object. The characteristics of these objects, such as size, position, color, and the lifetime of the particle itself, can be changed dynamically using a set of rules with the intention of modeling some effect. Particle Systems were first introduced by Reeves as a method for modeling fuzzy objects such as fire, clouds and water. Reeves describes particle systems as follows:

“Particle systems model an object as a cloud of primitive particles that define its volume. Over a period of time, particles are generated into the system, move and change form within the system, and die from the system. The resulting model is able to represent motion, changes of form, and dynamics that are not possible with classical surface-based representations. The particles can easily be motion blurred, and therefore do not exhibit temporal aliasing or strobing. Stochastic processes are used to generate and control the many particles within a particle system.”

Particle Systems are used to model objects that do not have a well-define fixed shape. Such objects cannot be represented using polygonal meshes as is generally done in computer graphics. Particle Systems differ from normal image synthesis representations in the following three ways:

- Objects are represented as a collection of points (particles) and not using a set of polygons.
- Particle systems are dynamic in nature in the sense that the attributes of particles in a system change with time. Old particles die and new particles are generated to replace them.
- An object represented by a particle system is not deterministic in nature, since its shape and appearance change over time.

System Overview:



The above figure shows an overview of the system implemented. The particle manager manages a set of particle systems. Each particle system is a collection of particles whose properties are varied over time as mentioned below. The library allows the user to access functions in the particle manager and particle system class. In the following sections, I describe each class shown above in more detail.

Particle Manager

As the name suggests, the particle manager class manages the various particle systems. The particle manager contains an array of particle systems. It provides functions for creating, removing, updating and rendering a particle system.

Functions in the Particle Manager class:

```
void Init()
void Render(int index)
void Update(int index)
bool RemoveSystem(ParticleSystem *system)
bool AddSystem(ParticleSystem *newSystem)
```

Particle Systems

This is the class where most of the actual work is done. Updation of particle attributes is done in this class. Each particle has many attributes like size, color, position, velocity, acceleration, texture etc that vary over time. However, some attributes of the particle like size, lifetime, texture etc do not change over time. So, I have kept most of these attributes (that remain fixed for a particle system) in the particle system class itself. The class also contains the initial values of attributes of particles that change over time. The attributes in the particle system are:

Attributes of the particle system:

Attribute	Description
Variance in velocity	Variance of the velocity of all particles in the system
Spread	Area over which the origin of the particle system is spread
Max Particles	Maximum number of particles in the system
NumParticles released/sec	Number of particles emitted by the particle system per sec
Plane	When particles strike these planes, they are destroyed.

Attributes common to all particles in the system

Attribute	Description
LifeTime	Time for which a particle lives in the system
Size	Size of particle
Texture	Texture that is applied to the particle

Attributes affecting position of particle:

Attribute	Description
Position	(x,y,z) coordinates of the particle
Old Position	(x,y,z) coordinates of the particle in previous frame
Velocity	Velocity of particle along the X,Y and Z axes
Acceleration	Velocity of particle along the X,Y and Z axes

Attributes affecting appearance of particle:

Attribute	Description
Color	(r,g,b) color component
Size	Size of particle
Texture	Texture that is applied to the particle
Alpha	Opacity of the particle

Functions in the Particle System class:

```
void Init()  
void Update(  
void Render()
```

In addition to these function accessor functions for each attribute have been provided. The accessor function corresponding to the attribute attr_name of type attr_type would be:

```
void Set[attr_name](attr_type attr)
```

```
attr_type Get[attr_name]()
```

For eg the accessor functions for the position attribute would be:

```
void SetPosition(Vector position)
```

```
Position GetPosition()
```

The particles in the system are updated based on their attributes as follows:

1. Particles that have outlived their life time are killed
2. Particles that collide against any of the planes are killed and the texture at that plane appropriately modified.
3. Existing particles that have not been killed are transformed according to their attributes.
4. New particles are generated if the limit on the number of particles has not been reached and also the limit on the number of particles generated per second has not been reached.

Some systems such as fog, clouds etc are not really dynamic. To accommodate these effects, I have provided functions which specify that the system is static/dynamic. By default, all systems are dynamic. In case of a static system, the particles are created and updated when the system is initialized. When the system is updated, the particles are jittered slightly about their positions.

Handling accumulation of snow using terrain rendering

In real life, snow accumulates on the ground as it falls on it. One way to handle this is to continue to draw a particle that strikes the ground at its final position. However, this solution is not practically feasible as the number of particles in the system would grow exponentially. If the ground plane is modelled as a textured polygon, then also accumulating the snow as it falls becomes difficult. Any snowflake that falls on the ground replaces the texture at that point by white color. As more and more snowflakes hit the same point on the ground they simply overwrite whatever color was there at that point. After some time, a 2D layer of snow accumulates over the ground which does not look realistic. Hence, some way of changing the texture height is needed for handling accumulation of snow. Instead of having a static texture mapped polygon, the height of the ground plane could be changed as snowflakes fall on it and the texture coordinates changed appropriately. I am detecting the collision of a particle with the ground plane and changing the height of the ground plane appropriately at the point of impact.

OpenGL Extensions Used:

- **GL_EXT_point_parameters:** Generally particle systems are used to model dynamic scenes where the distance of each particle from the viewer changes. Using constant sized points will look unrealistic. One solution would be to change the point size before rendering each particle using the `glPointSize` function. However, changing the point size for each particle in the system will reduce the performance of the system. The number of `glPointSize` calls could be minimized by sorting and grouping the particles, but then sorting itself would be an overhead. Another solution is

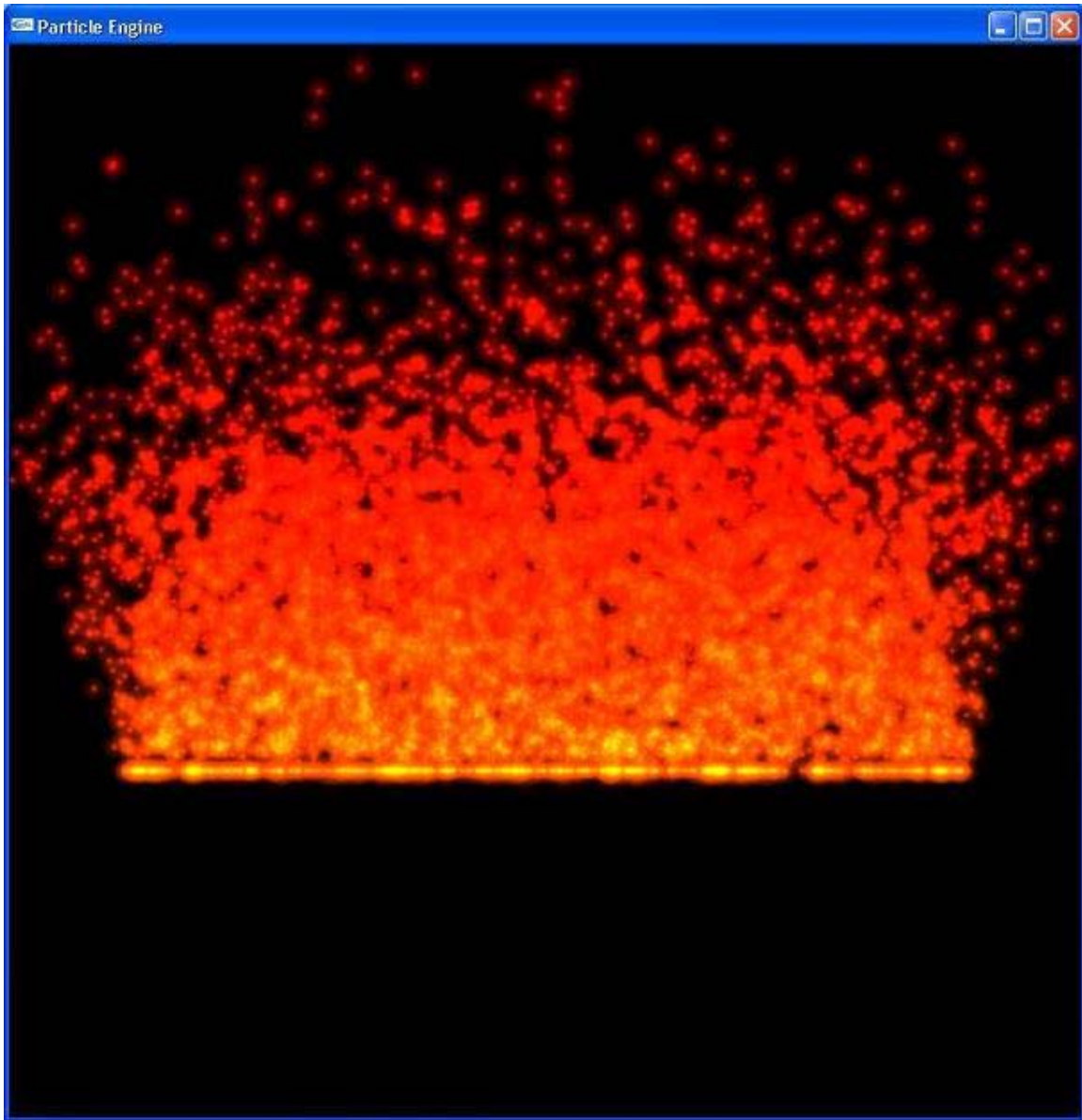
to use the `GL_EXT_point_parameters` extension to control the size of each particle as a function of its distance from the viewer.

- `GL_NV_point_sprite`: To make the particle system more realistic, I am using a texture for each particle in the scene. If the texture is mapped onto a point, each fragment in the point has the same texture coordinate. One solution would be to have quads as the basic primitive in the system instead of points. Textures could then be mapped to these quads. However, having thousands of quads in the system could make it slow. I am using nvidia's point sprite extension to replace the existing texture coordinates with point sprite texture coordinates, which are interpolated across the point.

What more could have been done

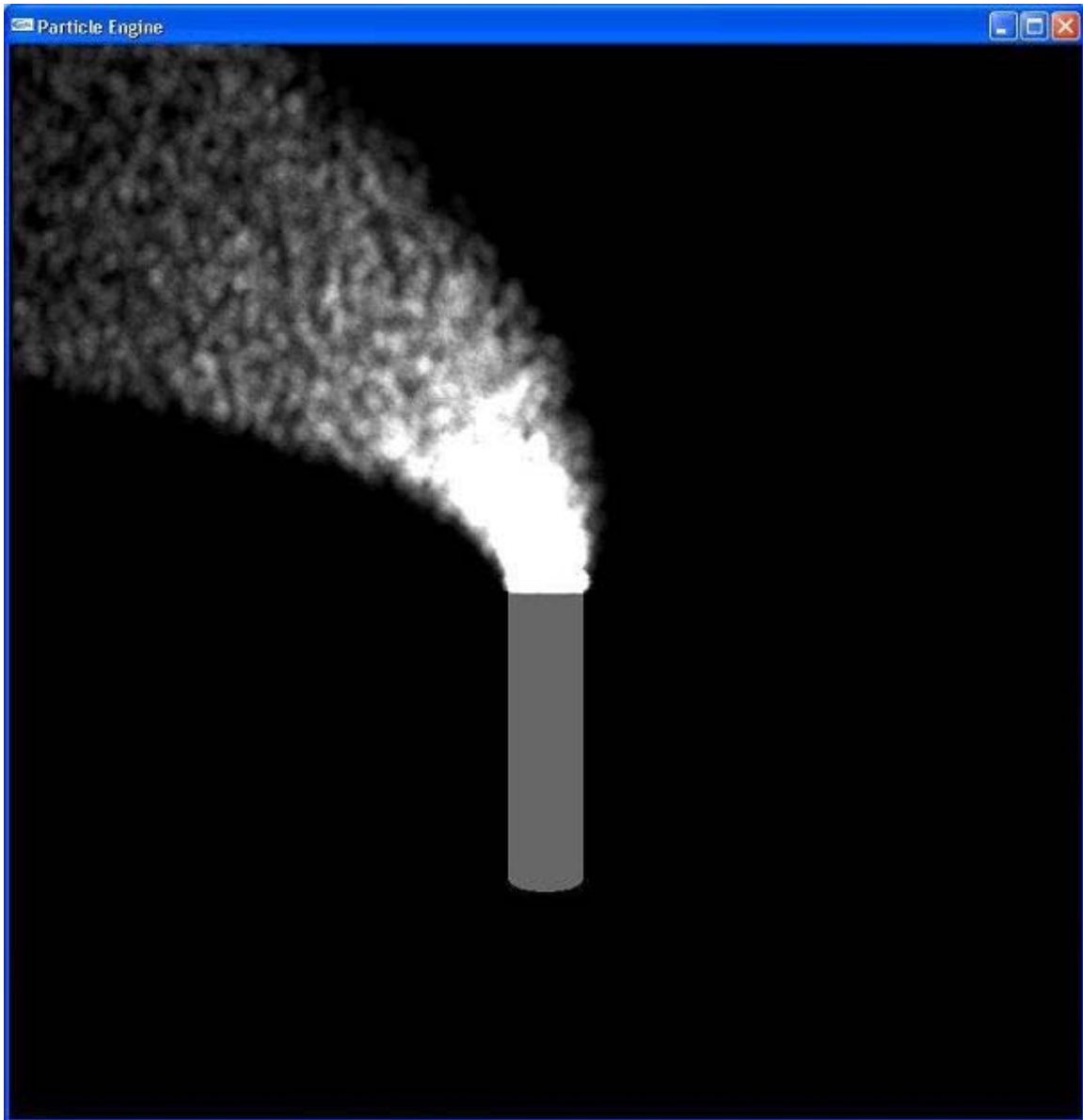
- In the current implementation the terrain for snow is randomly generated. It could be pre-generated to make it look more realistic.
- In case of rain, the ground plane should be visible.
- Lighting each particle in the system is not supported as it is a very expensive operation. Given more time, some approximate lighting model could have been tried.
- A better set of parameters for representing the particle system could have been used. The current set of parameters are good, but not very intuitive. A first time user might find it difficult to get the exact system he has in mind using these set of parameters (even though it will be possible to do so using the current set of parameters).
- The user interface for creating a new particle system could have been made better. The user could have been given the option of specifying the origin of the particle system using the mouse.

- ScreenShots



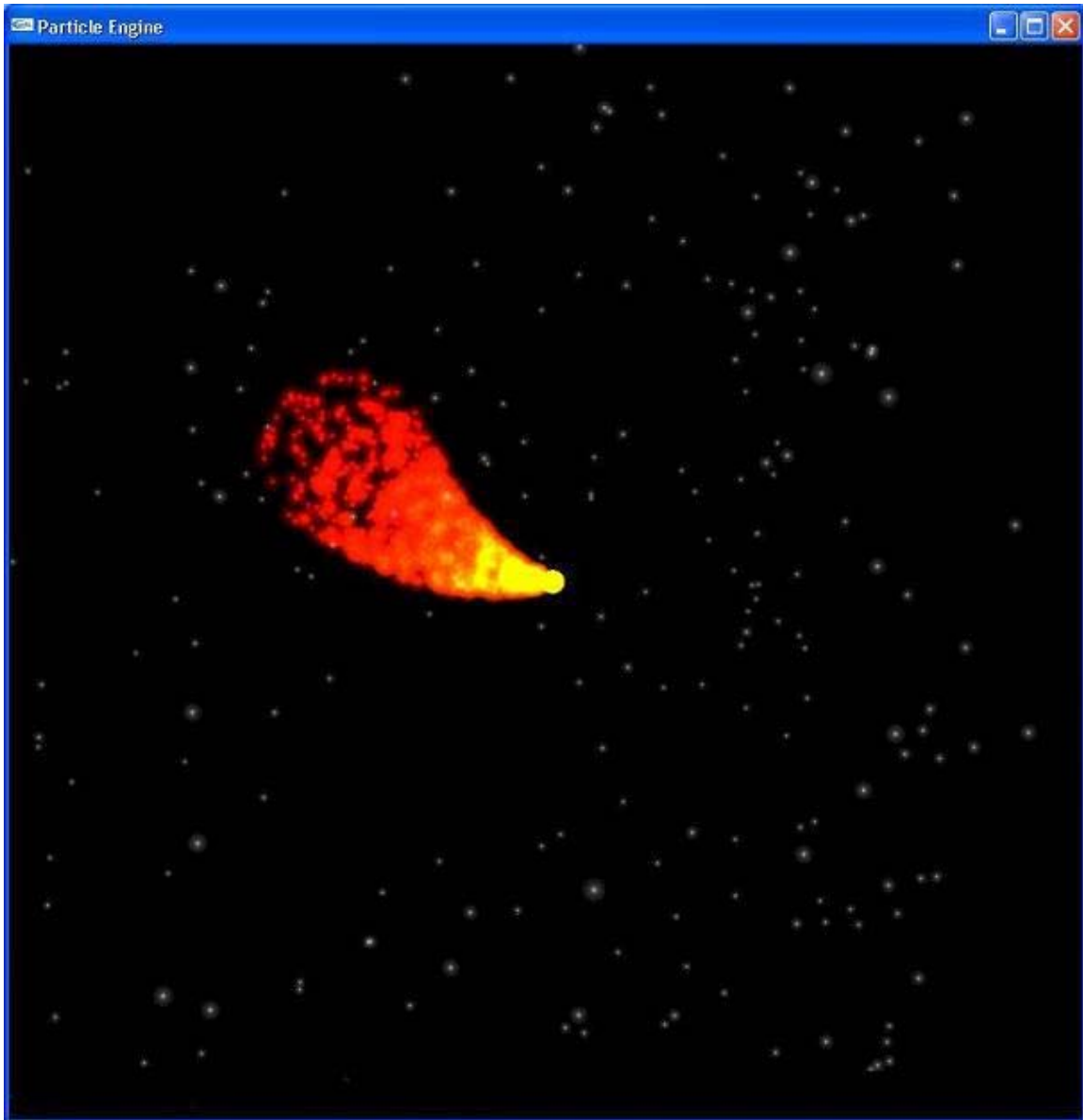
Parameter values used:

PointSize	50
Spread	5
LifeTime	18
MaxParticles	10000
NumToRelease	250
Variance in velocity	0.2f
Color(RGB)	(1.0, 0.20, 0.0)
Position	(0.0, -8.0, 0.0)
Velocity	(0.0, 0.5, 0.0)
WindVelocity	(0.0, 0.0, 0.0)
Acceleration	(0.0, 0.0, 0.0)
Alpha	1.0

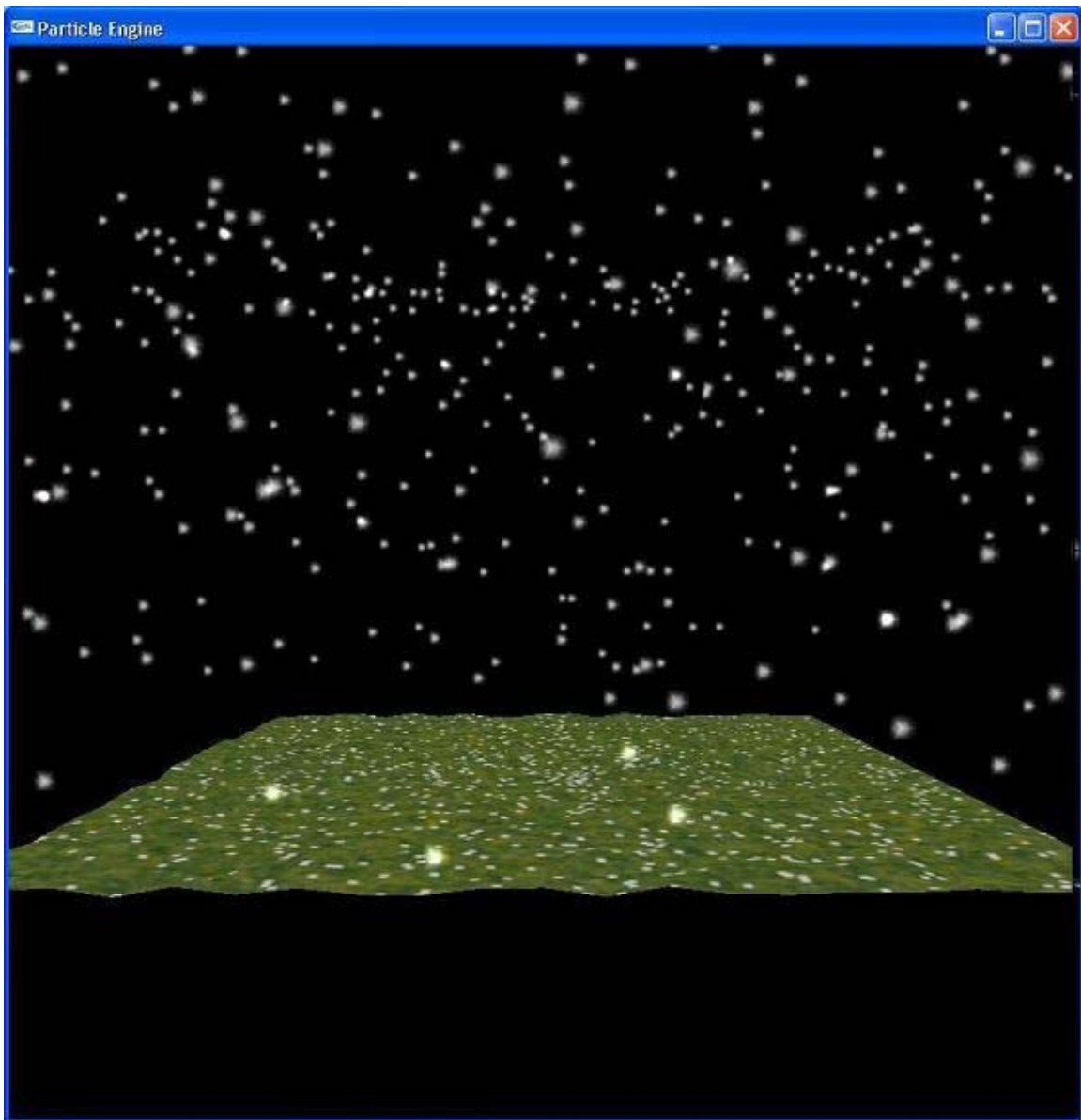


Parameter values used:

PointSize	50
Spread	1
LifeTime	25
MaxParticles	10000
NumToRelease	360
Variance in velocity	0.25f
Color(RGB)	(0.75f, 0.75f, 0.75f)
Position	(0.0f, 0.0f, 0.0f)
Velocity	(0.0f, 0.5f, 0.0f)
WindVelocity	(0.0f, 0.0f, 0.0f)
Acceleration	(0.0f, 0.0f, 0.0f)
Alpha	0.08

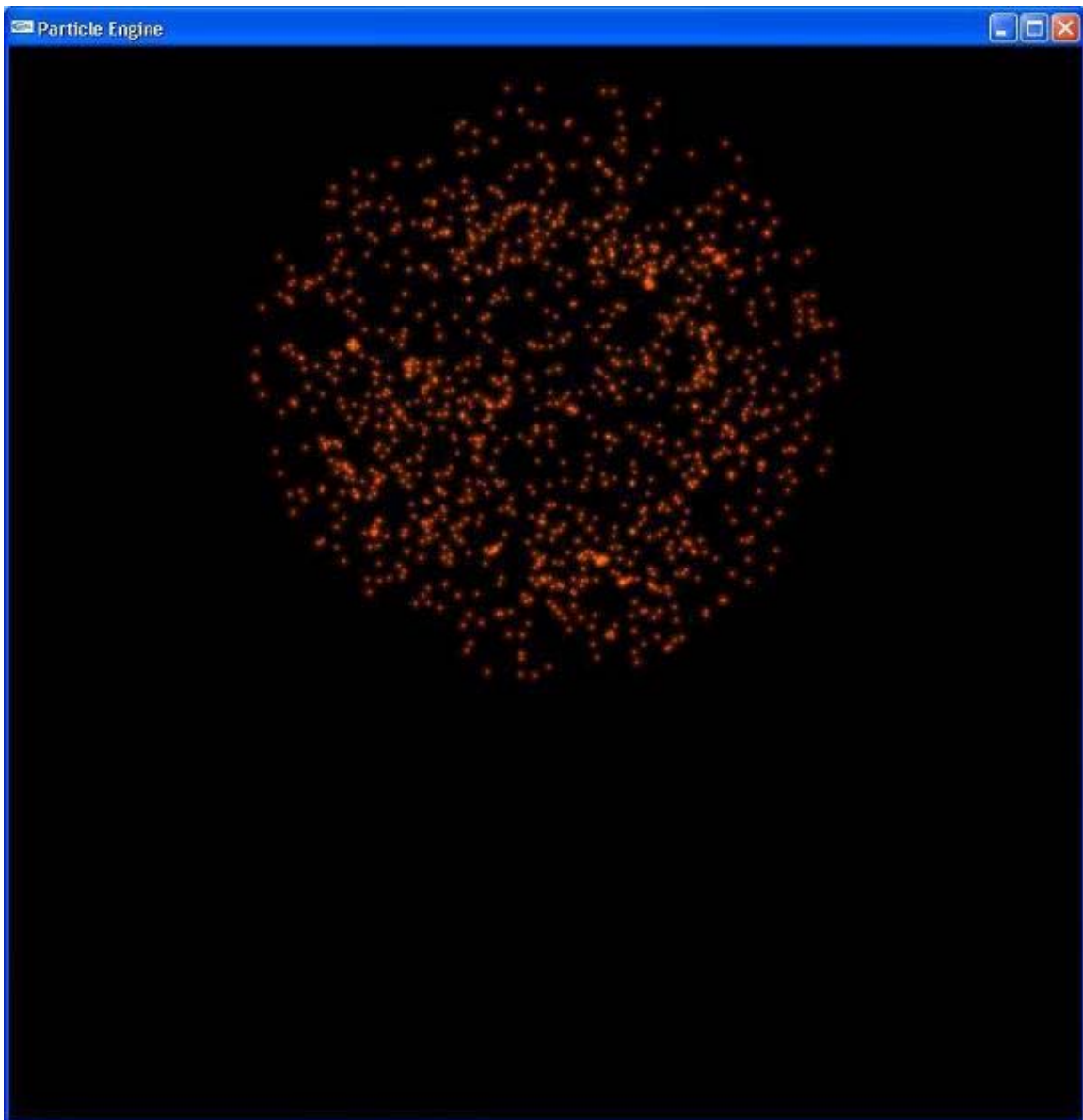


The above image contains two particle systems. One is the fire system, and the other is the snow system, except that instead of falling towards the ground the snow particles move from left to right.



Parameter values used:

PointSize	30
Spread	15
LifeTime	35
MaxParticles	1000
NumToRelease	20
Variance in velocity	0.0f
Color(RGB)	(0.75f, 0.75f, 0.75f)
Position	(0.0f, 12.0f, 0.0f)
Velocity	(0.0f, 0.0f, 0.0f)
WindVelocity	(0.0f, 0.0f, 0.0f)
Acceleration	(0.0f, -0.05f, 0.0f)



PointSize	30
Spread	0
LifeTime	50
MaxParticles	1000
NumToRelease	90
Variance in velocity	0.3f
Color(RGB)	(1.0f, 0.5f, 0.0f)
Position	(0.0f, 5.0f, 0.0f)
Velocity	(0.0f, 0.0f, 0.0f)
WindVelocity	(0.0f, 0.0f, 0.0f)
Acceleration	(0.0f, 0.0f, 0.0f)

References:

1. Particle Systems - A technique for Modeling a Class of Fuzzy Objects by William T. Reeves, Lucasfilm Ltd