Project Report: EECS 499

SPRING 2012

BUILDING PARTICLE SYSTEM USING XNA FRAMEWORK

Mippun Kangkan Pegu

Abhishek Sood

MS: Computer Science

Contents

I. Introduction

1. Definition

What is particle system?

Particle system is defined as "a collection of many minute particles that together represent a fuzzy object".

A particle is literally a point in three dimensional space. They are usually displayed (or rendered) as billboards (two dimensional objects which always face the viewer) because the more primitive the object, the less processing the computer has to perform. They contain a number of attributes such as life, color, velocity and direction and have a very basic life cycle. They are generated into the system, change throughout the course of their lifetime and then die from the system. Depending on the type of system, these dead particles can either be recycled back into the system or discarded.

All particles are generated from (and belong to) an "emitter". An emitter is a central point in the world from which all particles originate. The emitter is characterized by a set of behavioral attributes which govern all particles. Throughout the life of the particle it is the emitter which sets the particles attributes. The emitter, like the particle, also contains a set of attributes such as the rate at which particles are emitted, the standard life of a particle and the total number of particles in the emitter.

A degree of randomization is allowed in the attributes, even preferred if a realistic effect is required. This randomization is provided by a stochastic process which allows the value of the attribute to vary. There may be a life attribute, for example, which determines how long the particles will live for. This could be set to 100 frames. There could also be a life variation attribute which was set to 10 frames. Using the stochastic process, the particles emitted would have lives of between 90 and 110 frames.

Throughout each frame a number of steps are performed by the emitter:

1. Any particles that have died are discarded.
2. New particles are generated and assigned attributes from the emitter.
3. The remaining particles are changed according to their attributes.

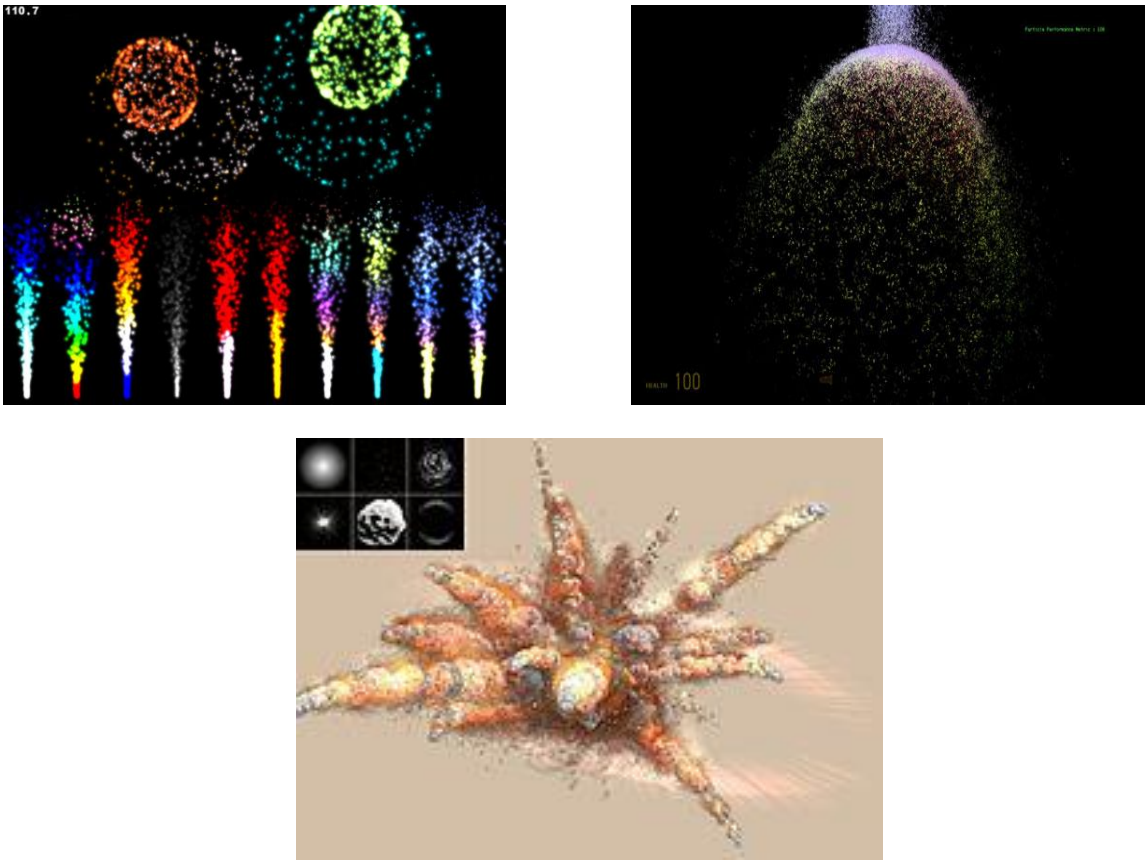


Fig: particle system examples

2. Project Goal

The goal for our project was to generalize the effect of particle system to produce various effects such as smoke, rain, snow, fire and so on.

II. Abstract

As technology advances, demand for a realistic experience in video games and movies is ever increasing and while many effects can be reproduced efficiently, there are still a wide number of effects categorized as “fuzzy” which are difficult to recreate.

Fuzzy effects are phenomena which do not have a well-defined shape. They are often chaotic and impossible to predict. Examples of these effects are fire, smoke, explosions and flowing water. Reproducing these effects can prove to be an arduous task. There are a number of solutions, arguably the most viable of which is to use a particle system.

This report will detail what particle systems are and how effective they are at reproducing fuzzy effects by looking at the alternative methods. Various existing particle systems will also be researched to view their functionality and methods used.

There are a number of methods and techniques that need to be considered when building a particle system. An implementation based on the methods and techniques researched is proposed which will allow developers to reproduce fuzzy effects in their applications and demonstrate the effectiveness of particle systems.

III. Applications of particle system.

Particle system has various applications especially in the entertainment field. At the beginning of its era, it was mainly use to create various natural effects such as explosion, rain, snow, storm etc. in animated movies. But due to the number of particles involved, the simulations of such effects were not efficient. In today’s world, the evolution of processors and graphics cards allows one to run a considerable amount of particles, without any jitter. Such progress in computation has allowed many game developers to implement particle systems in many games. They are mainly used to reproduce the effect of some natural and some un-natural events. Some of the events include explosions, rain, fire, smoke, dust.

Its application is not just limited to games or animation movies. Nowadays, there is an increasing demand of particle system in real movies as well. The applications remain the same as in games and animation movie. In the movie spider-man 3, they used particle system to produce the effect of flocking for the sandman (the main villain).

Other applications of particle system include:

1. Windows 8 startup animation.
2. Fireworks, waterfall, simulating our galaxy

IV. Implementing particle system:

1. Initializing the particles

In order to draw the particles, we have to first initialize the forces and other physics that are applied to the particles.

To create the effect of rain, we just need to initialize the position of every particle. As every particle is supposed to move in the downward direction, we multiple velocity.Y with -1. This will just create a simple rain effect. Suppose we want to add wind to the rain. In such case we have to apply wind forces to the particles. In our implementation, instead of creating a different wind force, we have just multiplied velocity.X with some positive integer, depending on the force of the wind.

```
Position = new Vector3((float)(-2 + 4 * randi.NextDouble()),
                       (float)(-2 + 4 * randi.NextDouble()),
                       (float)(-2 + 4 * randi.NextDouble()));
```

Where randi is some random value.

The above function sets the size of the emitter. For the rain effect the size of the emitter should be same as the size of the screen. For effects which requires small emitter, we just need to scale down the X,Y,Z vectors of the position. For example, in order to simulate the effect of fireworks, the particles are emitted from a single point. In such case we set the position value as following:

```
Position = new Vector3(0,-1,0);
```

We have also assigned each particle with some random age. We decrease the age of each particle at every iteration. Once the age reaches zero, we kill those particle and redraw new particle from the initial position.

2. Calculating the forces.

There are many ways to update the physics applied to the particles. We can either use solver function or simple mathematics equation to update the physics.

a. Euler solver:

The euler solver uses the following steps:

1. A=A+B*C

Where A is the current state and B is the new state. C is some constant.

*Position = A[i].Position + B[i].Position * C;*

2. Swap

In this step, it swaps the current state with the new state.

a.Position = from[i].Position;
b.Position = to[i].Position;

3. Dot Product

In this final step, it calculates all the physics applied to the particles. It first calculate the position with respect to velocity and then calculate the velocity with respect to the force.

b. Simple solver:

This method uses the mathematical relationship between acceleration, velocity, force and position to update the physics.

First step involves setting the acceleration to **9.8** which is the default value of acceleration due to gravity.

Then we equate the velocity(v) with acceleration(a) using the following equation:

V+=a * timestep, timestep is the speed with which the particles move.

Once we have the velocity, we have to update the position of every particle. This is done by the following equations:

Position+= v * timestep

3. *Drawing the particles.*

Once we have calculated all the physics, the only thing left is drawing the particles. In order to draw the particles, which are just 2D sprites, we have to first set the projection, world and view matrix of the camera. We have made the world matrix dependent on the position of each particle.

```
world *= Matrix.CreateTranslation(new Vector3((float)mipSys.pS0[i].Position.X,
                                              (float)mipSys.pS0[i].Position.Y,
                                              (float)mipSys.pS0[i].Position.Z));
Matrix projection = Matrix.CreatePerspectiveFieldOfView(1, aspect, 1, 10);
//aspect is the aspect ratio of our screen.

Matrix view = Matrix.CreateLookAt(cameraPosition, Vector3.Zero, Vector3.Up);
```

After we have set the camera we have to draw the sprites which is done by the following functions.

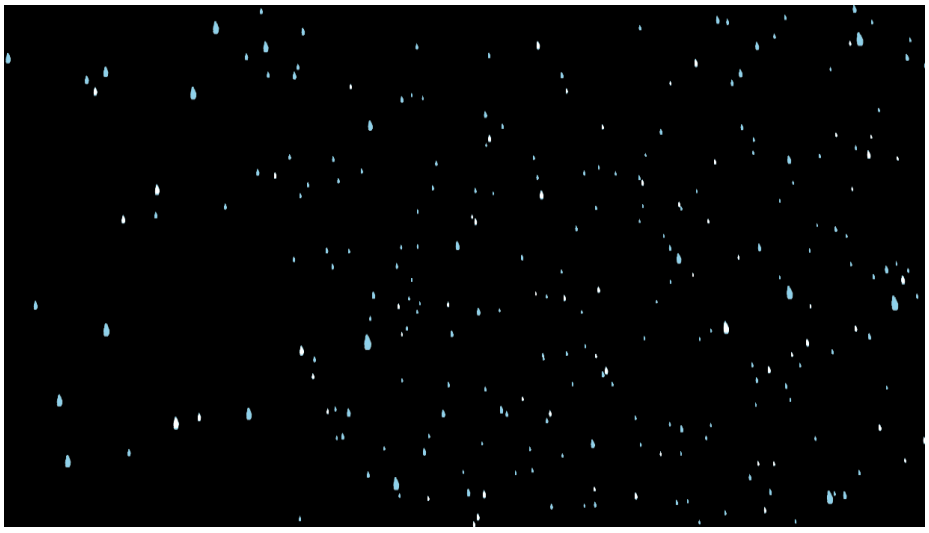
```
spriteBatch.Begin(0, null, null, null, RasterizerState.CullNone, particleEffect);
mipSys.Draw(spriteBatch);//calls the draw function in cPartSys, which then draws the particles
spriteBatch.End();
```

In case of some effects such as smoke, fire, where we need to blend the sprites, we add another effect to the above function:

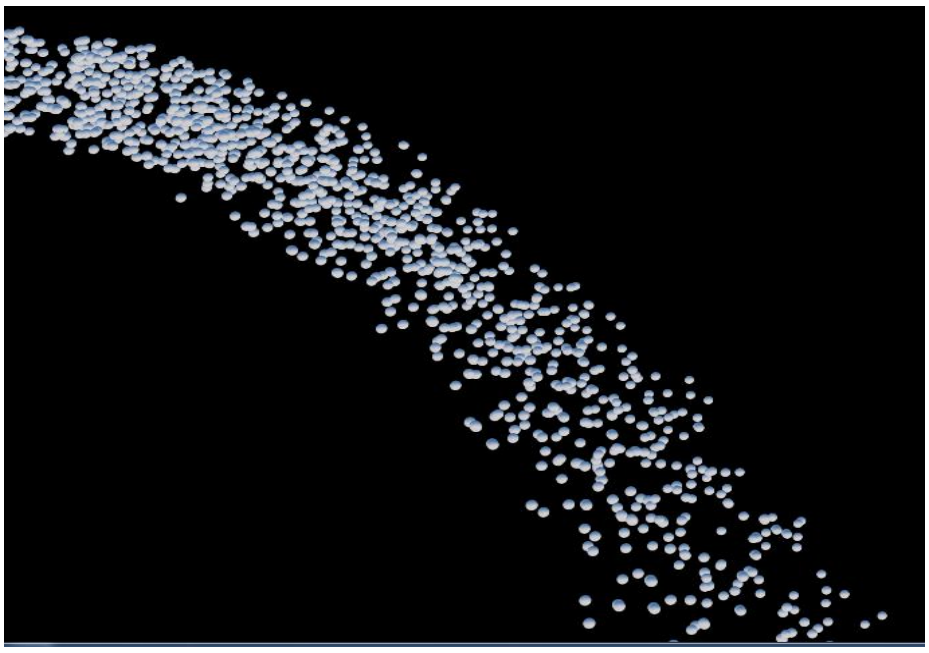
```
spriteBatch.Begin(0, BlendState.Additive, null, DepthStencilState.None,
                  RasterizerState.CullNone, particleEffect);
mipSys.Draw(spriteBatch);
spriteBatch.End();
```

V. Results

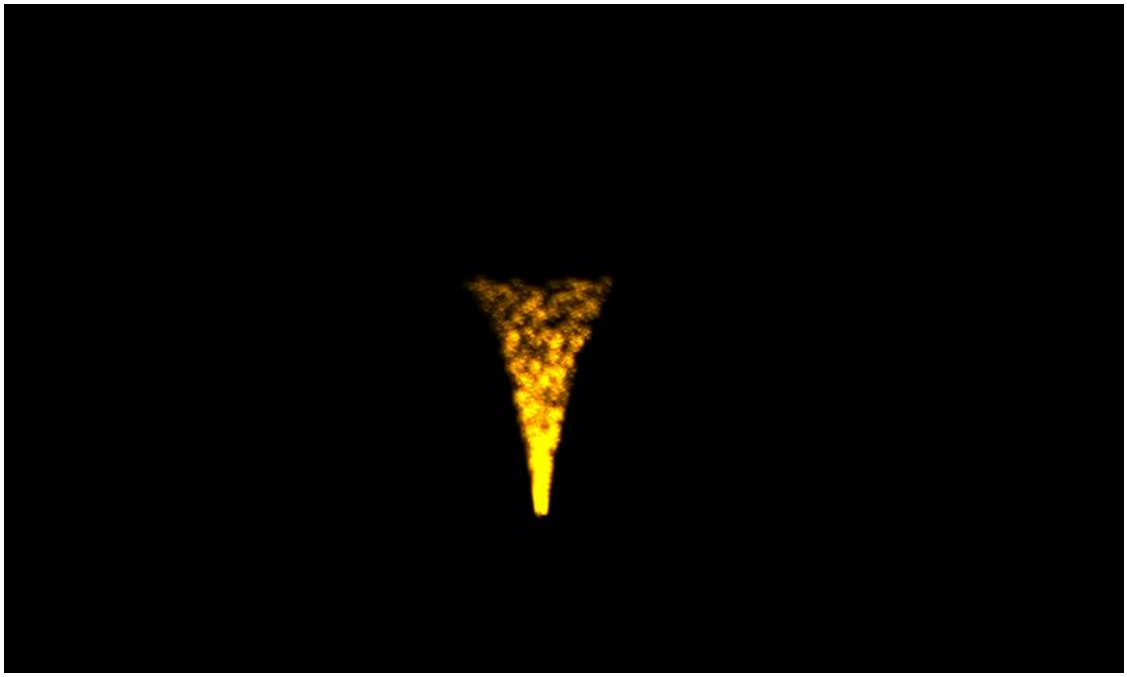
1. Rain



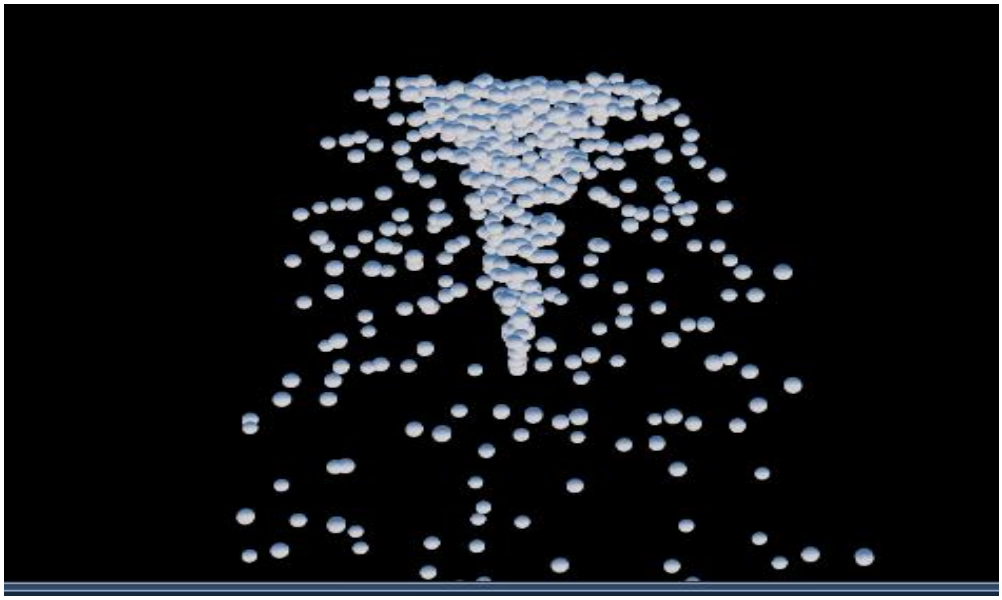
2. Fall



3. *Fire*



4. *Fountain*



VI. References

1. Gamasutra- particle systems
2. Catalin's XNA blog (XNA 3.0)
3. Intermediate graghics Course
4. Reimers tutorials