Particle System Graphics Library for Generating Special Effects

Eung-Kon Kim*
Department of Computer Science
Sunchon National University, Suncheon, Korea

ABSTRACT

The modeling and animation of natural phenomena have received much attention from the computer graphics community. Synthetic of natural phenomena are required for such diverse applications as flight simulators, special effects, video games and other virtual realty. In special effects industry there is a high demand to convincingly mimic the appearance and behavior of natural phenomena such as smoke, waterfall, rain, and fire. Particle systems are methods adequate for modeling fuzzy objects of natural phenomena. This paper presents particle system API(Application Program Interfaces) for generating special effects in virtual reality applications. The API are a set of functions that allow C++ programs to simulate the dynamics of particles for special effects in interactive and non-interactive graphics applications, not for scientific simulation.

***Keywords**: special effects, particle system, virtual reality, graphics library.*

1. INTRODUCTION

The appearance of natural phenomena has always intrigued humankind. This fascination is evident from the vast array of depictions of natural phenomena created, ranging from early cave painting to impressionistic masterworks. With the advent of photography and film, many aspects of natural phenomena can be depicted by a semi-automatic procedure. There are, however many reasons why such depictions are not always satisfying. Movies require special effects not commonly found in nature. Also, even if the real phenomenon can be found, controlling it can be a difficult and even dangerous task. It is necessary, then, to artificially create phenomena such as explosions for entertainment. In response to the need to evoke and control nature, scientists have proposed many descriptive physical models of natural phenomena.

The visual simulation of virtual environments has many practical applications. In addition to art and entertainment these include flight simulation and scientific visualization. Natural phenomena such as smoke, clouds, grass, and hair, are ubiquitous. It is therefore important to design convincing visual simulation for them.

Particle systems are a method for modeling fuzzy objects such as fire, clouds, water, smoke, and so on. Simulation of dynamic particle systems has been used in computer animation for several years and has more recently been used in real-time simulation and games to enrich the visual appearance of the virtual worlds. A particle system is composed of one or more individual particles. Each of these particles has attributes that directly or indirectly affect the behavior of the particle or how and where the particle is rendered. The other common characteristic of all particle systems is the introduction of some type of random element. This random element can be used to control the particle attributes such as position, velocity and color[1-3].

The goal of this paper is to develop particle system graphics library module for modeling special effects in virtual reality applications. The library can be based on some existing code such as particle system API[3]. The tasks would be then to integrate such code into C++ especially with its rendering module (the particles should be probably based on the shaders), and further to define some higher-level way of controlling the effects. Designer of the scene will be dealing with fuzzy object primitives such as fire or smoke and their parameters such as location, intensity, color etc., not directly with equations describing underlying particle system.

The next chapter introduces particle system for special effects, chapter 3 presents the particle system API for generating special effects in virtual reality applications, and finally chapter 4 concludes and discusses future works.

2. PARTICLE SYSTEM

The use of Particle systems is a way of modeling fuzzy objects, such as fire, clouds, smoke, water, etc. These don't have smooth well-defined surfaces and are non-rigid objects,

* *Corresponding author. E-mail: kek@sunchon.ac.kr*
Manuscript received Apr. 20, 2006 ; accepted Jun. 20, 2006

This work is financially supported by the Ministry of Education and Human Resources Development(MOE), the Ministry of Commerce, Industry and Energy(MOCIE) and the Ministry of Labor(MOLAB) through the fostering project of the Industrial-Academic Cooperation Centered University

i.e., they are dynamic and fluid. Particle systems differ in three ways from "normal" representations for image synthesis:

a. An object is not represented by a set of primitive surface elements, e.g., polygons or patches, but as clouds of primitive particles that define its volume.

b. A particle system is not a static entity, its particles change form and move. New particles are created and old particles are destroyed.

c. An object represented by a particle system is not deterministic, its shape and form is not completely specified. Stochastic processes are used to create and change an object's shape and appearance. Note that particle systems can be used in a deterministic way to create certain objects, e.g., the human head in the video "Particle Dreams" by Karl Sims[2].

Particle systems are an example of stochastic procedural modeling, similar to fractals, and have some of advantages, such as the following:

a. Complex systems can be created with little human effort.

b. The level of detail can be easily adjusted. For example, if a particle system object is in the distance, then it can be modeled in low detail (few particles), but if it is close to the camera, then it can be modeled in high detail (many particles).

2.1 Basic Model of Particle Systems

A particle system is a collection of many minute particles that model an object. For each frame of an animation sequence the following steps are performed:

a. New particles are generated

b. Each new particle is assigned its own set of attributes

c. Any particles that have existed for a predetermined time are destroyed

d. The remaining particles are transformed and moved according to their dynamic attributes

e. An image of the remaining particles is rendered

2.2 Particle Attributes

Each new particle has the following attributes:

a. initial position

b. initial velocity(speed and direction)

c. initial size

d. initial color

e. initial transparency

f. shape

g. lifetime

A particle system has several parameters that control the initial position of the particles:

a. X, Y, Z (the particle system origin)

b. Two angles of rotation that give its orientation

c. A generation shape which defines the region around the origin in which new particles are placed, e.g., a sphere of radius R. These shapes can be simple or quite complicated.

The generation shape describes the initial direction of new particles, e.g., for a sphere the particles would move away from the origin in all directions. For a planar shape, e.g. a circle in the x-y plane, the particles would move up and away from the plane (not necessarily straight up, this would be determined by the rotation angles).

The initial speed of a particle can be given by:

Initial Speed = MeanSpeed + Rand() * VarSpeed

The initial color can be:

InitialColor = Meancolor(R,G,B) + Rand() * VarColor(R,G,B)

The initial opacity can be:

InitialOpacity = MeanOpacity(R,G,B) + Rand() * VarOpacity(R,G,B)

The initial size can be:

InitialSize = MeanSize + Rand() * VarSize

There is also a parameter that specifies the shape of each particle, e.g., spherical, rectangular, or streaked spherical (for motion blur).

2.3 Particle Dynamics

A particle's position in each succeeding frame can be computed by knowing its velocity (speed and direction of movement). This can be modified by an acceleration force for more complex movement, e.g., gravity simulation.

A particle's color can be modified by a rate-of-color-change parameter, its opacity by a rate-of-opacity-change parameter, and its size by a rate-of-size-change parameter. These rates of change can be global, i.e. the same for all particles, or they can be stochastic for each particle.

2.4 Particle Extinction

When a particle is created it can be given a lifetime in frames. After each frame, this is decremented and when the Lifetime is Zero, the particle is destroyed. Another mechanism might be that when the color/opacity is below a certain threshold the particle is invisible and is destroyed. When a particle has left the region of interest, i.e., is a certain distance away from its origin, it could be destroyed.

2.5 Particle Rendering

Particles can obscure other particles behind them, can be transparent, and can cast shadows on other particles. They can also interact with other, conventionally modeled primitives. In this system the authors made two assumptions. The first was that the particle systems do not intersect with other primitives (so the rendering system only has to handle particles). The other objects in a scene are rendered separately and then compounded with the particle system images. If the particles do interact with other objects, e.g., go behind them, then the images are divided into sub-images which are compounded.

A second approximation is that the particles are light sources, that additively combine according to their color and opacity values. This eliminates the hidden surface problem since particles do not obscure each other but just add more light to a given pixel. It also eliminates shadows.

3. THE PARTICLE SYSTEM API FOR GENERATING SPECIAL EFFECTS IN VIRTUAL REALITY

3.1 The Particle System API

The particle system API consists of four sets of functions. These are particle group functions that operate on and manage particle groups, particle attribute functions that set the current state of the API, particle action functions that act on particle groups, and particle action list functions that create and operate on action lists[3].

API function names take the form particleFunctionName. Most calls are defined with default values for the lesser-used arguments to simplify the application developer's coding in the common case.

Particle attribute functions are used to set attributes of particles to be created. The followings are principle particle attribute functions.

```
void particleColor(...) //set the color of new particles
void particleSize(...) //set size of new particles
void particleInitialAge(...) //set initial age of new particles
void particleTimeStep(...) //set time step length
void particleVelocity(...) //set initial velocity of new particles
```

A particle group is a system of particles that are acted on together. The following particle group functions create and deal with particle group.

```
int particleCreateGroups(...)  //create particle groups
void particleChangeGroup(...)  //change a particle group
void particleDeleteGroups(...) //delete particle groups
void particleDrawGroup(...)    //draw a particle group
int particleGetNumberParticles() //get the number of particles in the current group
int particleChangetMaxParticles(...)
//change the maximum number of particles
```

Action functions directly manipulate particles in particle groups. They perform effects such as gravity, explosions, bouncing, etc. to all particles in the current particle group. A program typically creates and initializes one or more particle groups, then at run time it calls particle action functions to animate the particles and finally draws the group of particles on the screen.

```
void particleAccelerate(...)
    //accelerate each particle toward each other particle
void particleAcceleration(...)
    //accelerate particles in the specific direction
void particleAcceleratePoint(...)
  //accelerate particles toward the specific point
void particleAccelerateRandom(...)
    //accelerate particles in random directions
void particleAdd(...) //add particles in the specific domain
void particleBounce(...)
//bounce particles off a domain of space
void particleChangeColor(...)
//change color of all particles into the specific color
void particleChangeSize(...)
//change size of all particles into the specific size
void particleChangeVelocity(...)
//change velocity of all particles into the specific velocity
void particleDampen(...)   //dampen particle velocities
void particleExplode(...)  //explode
void particleJet(...)
//accelerate particles near the center of the jet
void particleMove()
//move particle positions based on velocity
void particleRemove(...)   //remove old particles
void particleRemoveOff(...)
//remove particles with positions off the specific domain
void particleSwirl(...)  //swirl particles around a vortex
```

Action lists are blocks of actions that are applied together to a particle group. They are conceptually similar to scripts or procedures. They can be also be thought of as similar to display lists in OpenGL. An action list abstracts the specifics of a particular effect and allows complex effects to be treated as primitives like actions. The followings are principle particle action list functions.

```
void particleApplyActionListt(...)
//apply the action list to the particle group
void particleCreateActionList(...)
//create the specific action list
void particleEndActionList()
//end the creation of a new action list
void particleGenerateActionLists(...)
//generate empty action lists
void particleRemoveActionLists(...)
//remove consecutive action lists
```

3.2 Applications of the Particle System API

Library functions are called to generate special effects. Programmer of the scene will be dealing with fuzzy object primitives such as explosion or waterfall and their parameters such as location, intensity, color etc., not directly with equations describing underlying particle system. Figure 1 represents pseudo code for calls of particle system API[3].

```
for each particle group i
    particleChangeGroup(i)   //change particle group
    for each time step per frame
        particleAdd(...)
        //add particles in the specific domain
     other actions....        //perform other actions
     particleMove()           //move particle positions
    end for
    particleDrawGroup(...)    //draw particles
  end for
 other drawing...             //draw others
```

Fig. 1. Pseudo code for calls of particle system API

The following figure 2 and figure 3 represent the part of C code and the execution screen to generate a Fountain effect respectively.

```
void Fountain(bool do_list = true)
{ pVelocityD(PDCylinder,
0.0, -0.01, 0.35, 0.0, -0.01, 0.37, 0.021, 0.019);
    pColorD(1.0, PDLine, 0.8, 0.9, 1.0, 1.0, 1.0, 1.0);
    pSize(1.5);
    static int al = -1;
    if(al == -1)
    {   al = pGenActionLists(1);
        pNewActionList(al);
        pSource (150,PDLine, 0.0,0.0,0.401,0.0,0.0,0.405);
        pEndActionList(); }
    if(do_list && action_handle<0)
        action_handle = pGenActionLists(1);
        if(do_list)
        pNewActionList(action_handle);
    pCopyVertexB(false, true);
    pCallActionList(al);
    pGravity(0.0, 0.0, -0.01);
    pSinkVelocity(true, PDSphere, 0, 0, 0, 0.01);
    pBounce(-0.05, 0.35, 0, PDDisc, 0, 0, 0,  0, 0, 1,  5);
    pSink(false, PDPlane, 0,0,-3, 0,0,1);
    pMove();
    if(do_list)
        pEndActionList(); }
```

Fig. 2. The part of C code to generate a fountain effect

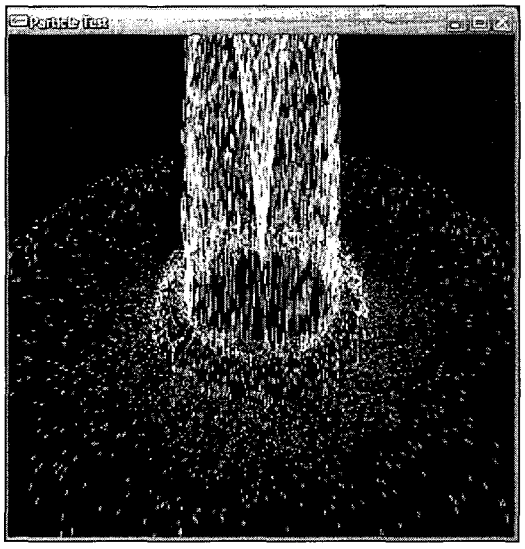

Fig. 3. Execution screen of a fountain effect using fig. 2

The following figure 4 and figure 5 represent the part of C code and the execution screen to generate a Explosion effect respectively.

```
void Explosion(bool do_list = true)
{  if(do_list && action_handle<0)
        action_handle = pGenActionLists(1);
    pVelocityD(PDSphere, 0,0,0,0.01,0.01);
    pColorD(1.0, PDSphere, 0.5, 0.7, 0.5, .3);
    pSize(1.0);
    if(do_list)
        pNewActionList(action_handle);
    pCopyVertexB(false, true);
    pDamping(0.999, 0.999, 0.999);
    static float i=0;
    if(do_list)
        i = 0;
    pOrbitPoint(0, 0, 0, .02, 0.1);
    pExplosion(0,  0,  0,  1,  2,  3,  0.1,  i+= (1.0f /
            float(numSteps)));
    pSink(false, PDSphere, 0, 0, 0, 30);
    pMove();
    if(do_list)
        pEndActionList();
}
```

Fig. 4. The part of C code to generate an explosion effect

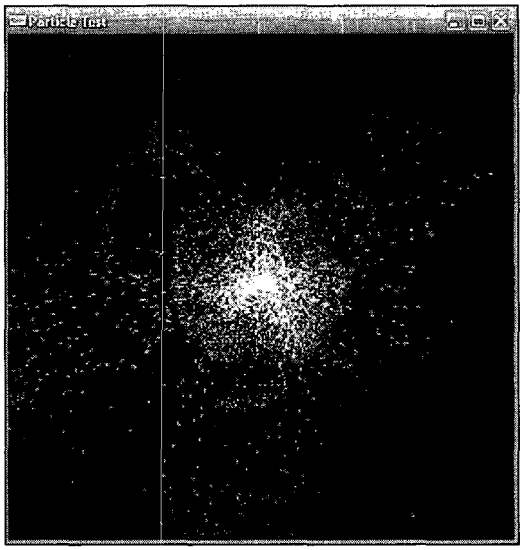

Fig. 5. Execution screen of an explosion effect using fig. 4

3.3 Performance

To measure numerical performance of the API, 30,000 points were simulated. Particles were rendered using GL_POINTS in a 640x480 window. The 2.8 GHz Pentium IV processor with RADEON 9600 achieved 46fps.

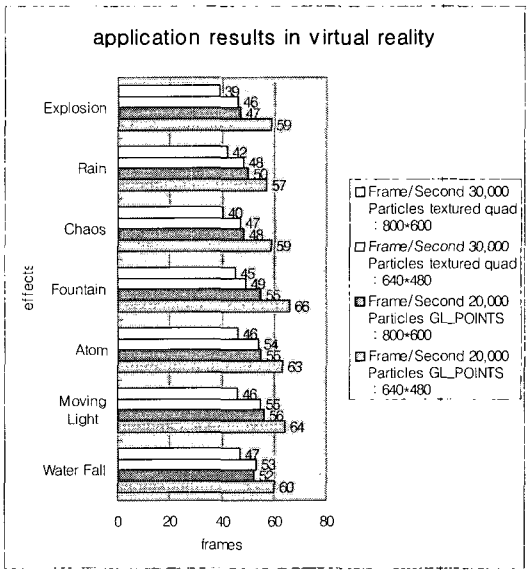

Fig. 6. Application results

4. CONCLUSIONS

This paper presents particle system graphics APIs for generating special effects in virtual reality applications. The APIs are a set of functions that allow C++ programs to simulate the dynamics of particles for special effects in interactive and non-interactive virtual reality applications, not for scientific simulation.

It is clear which parameter of which action in a particular effect should be modified for a particular visual result. The numerical accuracy of the simulation must be scalable and modifiable by the application. The API is usable for offline animation and for real-time special effects in virtual reality. The application programmer is able to specify different accuracy needs for different effects.

Programmer of the scene will be dealing with fuzzy object primitives such as fire or smoke and their parameters such as location, intensity, color etc., not directly with equations describing underlying particle system.

Future work is to add API functions to generate diverse effects and to apply them to games and other entertainment applications.

REFERENCES

[1] Reeves, W. T. **Particle Systems - A Technique for Modeling A Class of Fuzzy Objects**, Proc. of SIGGRAPH '83, Detroit, Michigan, July, 1983.

[2] Rick Parent, **Computer Animation, Algorithms and Techniques,** Morgan Kaufmann Publihers, 2002.

[3] McAllister, D. K. **The Design of an API for Particle Systems,** http://cs.unc.edu/~davemc/Particle, 1999.

[4] Leech, J. P. and R. M. Taylor. **Interactive Modeling Using Particle Systems,**. Proc. of the 2nd Conference on Discrete Element Methods, MIT, 1993.

[5] Allen, M. B. **Flow - a particle animation application,** http://www.dnai.com/~mba/sortware/flow/, 1999.

[6] William T. Reeves, **Particle Systems - A Technique for Modeling a Class of Fuzzy Objects,** Computer Graphics 17:3 pp. 359-376, 1983

Eung-Kon Kim

He received the B.S. degree in electionic engineering from Chosun University, Gwangju, Korea, in 1980, the M.S. degree in electronic engineering from Hanyang University, Seoul, Korea in 1987 and the Ph.D. degree in computer engineering from Chosun University in 1992.

He is currently a professor of Department of Computer Science, Sunchon National University.

He is interested in Computer graphics and its applications.