3D Graphics Library for Generating Real-time Special Effects

Eung-Kon Kim, Bong-Kil Yoo and **Seung-Heon Song**, *Member, KIMICS*

Abstract—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 graphics library for generating special effects in video games and virtual reality applications. The library is 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.

Index Terms—About four key words or phrases in alphabetical order, separated by commas such as Maritime, Information, Communication, Science etc.

I. INTRODUCTION

The modeling and animation of natural phenomena has 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.

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[1,2,3].

Commercial animation packages such as Maya and 3D Studio Max now include a particle animation package. But they are rendered offline. OpenGL and DirectX are application program interfaces necessary to potable interface to the hardware implementations of graphics functionality, but they support low-level operation of real-time rendering for particle system.

The goal of this paper is to develop particle system graphics library module for modeling special effects such as smoke, fire, explosions, rain, etc. 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 graphics library for generating special effects, and finally chapter 4 concludes and discusses future works.

II. PARTICLE SYSETM

A. Particle Generation

Particles are typically generated according to a controlled stochastic process. For each frame, a random number of particles are generated using some user-specified distribution centered at the desired average number of particles per frame(Eq. (1)). The distribution could be uniform or Gaussian or anything else the animator wants. Rand(_) returns a random number from -1.0 to +1.0 in the desired distribution, and range scales it into the desired range. If the particles are used to model a fuzzy object, then it is best to use the area of the screen covered by the object to control the number of particles generated(Eq. (2)). The features of the random number generator, such as average value and variance, can be a function of time to enable the animator to have more control over the particle system[2].

$$\#\text{ of particles} = \text{average} + \text{Rand}(\_) \cdot \text{range} \quad (1)$$

$$\#\text{ of particles} = \text{average} + \text{Rand}(\_) \cdot \text{range} \cdot \text{screenArea} \quad (2)$$

B. Particle Attributes

The attributes of a particle determine its motion status, its appearance, and its life in the particle system. Typical attributes are position, velocity, shape parameters, color, transparency and lifetime.

Each of attributes is initialized when the particle is created. Again, to avoid uniformity, the user typically randomizes values in some controlled way. The position and velocity are updated according to the particle's motion. The shape parameters, color. and transparency

Manuscript received September 10, 2004.

Eung-Kon Kim is with the Department of Computer Science, Sunchon University, Suncheon-si, Korea.

Bong-Kil Yoo is with the Department of Cyber Police Administration, Suncheon Cheongam College, Suncheon-si, Korea.

Seung-Heon Song is with the Graduate School of Computer Science, Sunchon National University, Suncheon-si, Korea.

control the particle's appearance. The lifetime attribute is a count of how many frames the particle will exist in.

C. Particle Termination

At each new frame, each particle's lifetime attribute is decremented by one. When the attribute reaches zero, the particle is removed from the system. This completes the particle's life cycle and can be used to keep the number of particles active at any one time within some desired range of values.

D. Particle Animation

Typically, each active particle in a particle system is animated throughout its life. This activation includes not only its position and velocity but also its display attributes: shape, color, and transparency. To animate the particle's position in space, the user considers forces and computes the resultant particle acceleration.

The velocity of the particle is updated from its acceleration, and then the average velocity is computed and used to update the particle's position. Gravity, other global force fields, local force fields, and collisions with objects in the environment are typical forces modeled in the environment.

The particle's color and transparency can be a function of global time, its own life span remaining, its height, and so on. The particle's shape can be a function of its velocity. For example, an ellipsoid can be used to represent a moving particle where the major axis of the ellipsoid is aligned with the direction of travel and the ratio of the ellipsoid's length to the radios of its circular cross section is related to its speed[4,5,6].

E. Particle Rendering

To simplify rendering, model each particle as a point light source so that it adds color to the pixel it covers but is not involved in visible surface algorithms (except to be hidden) or shadowing. In some application, shadowing effects have been determined to be an important visual cue. The density of particles between a position in space and a light source can be used to estimate the amount of shadowing[4].

F. Particle System Representation

A particle is represented by a tuple [x,v,f,m] which holds its position, velocity, force accumulator, and mass.

```
typedef particle_struct struct {
        vector3D p;
        vector3D v;
        vector3D f;
        float mass;
        } particle;
```

The state of a particle, [x,v], will be updated by [v,f/m] by the ODE(ordinary differential equation) solver. The solver can be considered to be a black box to the extent that the actual method used by the solver does not have to be known.

A particle system is an array of particles with a time variable: [*p,n,t].

```
typedef particleSystem_struct struct{
        particle *p;
        int n;
        float t;
        } particleSystem;
```

Typically, each particle will have a life span. The particle data structure itself can be reused in the system so that a particle system might have tens of thousands of particles over the life of the simulation but only, for example, one thousand in existence at any one time. Initial values are set pseudo randomly so that particles are spawned in a controlled manner but with some variability.

III. THE PARTICLE SYSTEM GRAPHICS LIBRARY FOR GENERATING SPECIAL EFFECTS

A. Characteristics of the Particle System Graphics Library

The particle system library is a high level graphics application interface that is based on OpenGL[7,8] graphics library to generate special effects in games and virtual reality applications. Figure 1 represents the applicability and the scalability of the particle system library.

The library was designed with specific goals and constraints.

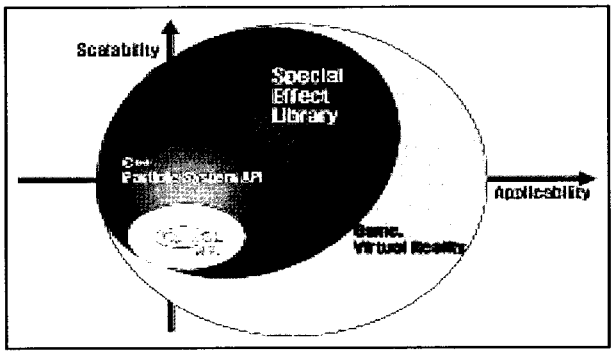


Fig. 1 Applicability and Scalability of the Particle System Library

The primary purpose of the library is to enable real-time applications to include dynamics be computed efficiently so that the CPU has enough time per frame to perform the rest of the application's computation.

The library was specifically designed to allow the user to create many effects not envisioned by the library designers. For this reason, the library consists of simple building blocks such as Gravity and Bounce.

Since the library consists of simple building blocks, the design space for implementing a particular effect can be quite large. 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 library is usable for offline animation and for real-time special effects in video games and virtual reality. The application

programmer is able to specify different accuracy needs for different effects.

The library is an abstraction of the particle system functionality. The application should work identically whether the library is implemented entirely as a library linked into the application, or as a thin layer that merely communicates the particle dynamics instructions to the graphics hardware or some other device.

A particle within the particle system library is an object with a set of attributes very similar to those of Reeves' original particle system[1]: position, velocity, color, alpha, size, age, secondary position, and secondary velocity. All attributes are three-vectors, except alpha and age, which are scalars. The secondary position is normally a destination position and is rarely used. The secondary velocity normally stores the velocity from the previous time step for computing particle orientations from their instantaneous curvature. The three-vector size is only a rendering attribute, like color. Its use is completely application-dependent. For particle dynamics purposes, particles are a unit point-mass[3].

B. The Particle System Library

The particle system library 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 library, particle action functions that act on particle groups, and particle action list functions that create and operate on action lists[3].

Library 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
```

C. Applications of the Particle System Graphics Library

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 2 represents pseudo code for calls of particle system library[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. 2 Pseudo code for calls of particle system library

The following figure 3 and figure 4 represent the part of C code and the execution screen to generate a waterfall effect respectively.

```
void Waterfall(bool first_time = true)
{    particleVelocityD(PDBlob, 0.1, 0, 0.1, 0.004);
       //set initial velocity of new particles
     particleColor(1.0,PDLine,0.8,0.9,1.0,1.0,1.0,1.0);
                    //set the color of new particles
     particleSize(1.5); //set size of new particles
     particleInitialAge(0);
        //set initial age of new particles

     if(first_time)
     {
        //action_handle
       particleGenerateActionLists(1);
        //generate empty  action lists
       particleCreateActionList(action_handle);
        //create the specific action list
     }
     particleAdd(50, PDPoint, -4, 0, 6);
        //add particles in the specific domain
     particleAcceleration(0.0, 0.0, -0.01);
        //accelerate particles
        //in the specific direction
     particleRemove(250);     //remove old particles
     particleBounce(0, 0.01, 0, PDSphere, -1, 0, 4, 1);
        //bounce particles off a domain of space
     particleBounce(0,0.01,0, PDSphere, -2.5,0,2,1);
     particleBounce(0,0.01,0, PDSphere, 0.7,-0.5,2,1);
     particleBounce(-.01,0.35,0,PDPlane,0,0,0,0,0,1);
     particleMove();
        //move particle positions based on velocity
     if(first_time)
     pEndActionList();
        //end the creation of a new action list
```

Fig. 3 The part of C code to generate a waterfall effect

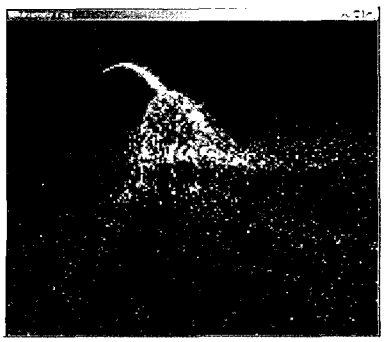


Fig. 4 Execution screen of a waterfall effect using fig. 3

The following fig. 5 and fig. 6 represent the part of C code and the execution screen to generate a moving lights effect respectively.

```
void MovingLights(bool first_time = true)
{    particleSize(1.0);   //set size of new particles
     particleVelocity(PDPoint, 0,0,0);
     //set initial velocity of new particles
     particleColor(1.0, PDSphere, .5, .4, .1, .1);
     //set the color of new particles
     particleInitialAge(0);
     //set initial age of new particles
     particleAdd(1, PDBlob, 0, 0, 2, 2);
     //add particles in the specific domain
     particleAccelerateRandom(PDSphere,0,0,0,0.02);
     //accelerate particles in random directions
     particleRemove(20);     //remove old particles
     particleMove(); }
     //move particle positions based on velocity
```

Fig. 5 The part of C code to generate a moving lights effect

Fig. 6 Execution screen of a moving lights effect using fig. 5

D. Performance

To measure numerical performance of the library, 20,000 points were simulated. Particles were rendered using GL_POINTS in a 512x512 window. The 1.2 GHz Pentium IV processor with Geforce 256 achieved 27.8fps

IV. CONCLUSIONS

This paper presents particle system graphics library for generating special effects in video games and virtual reality applications. The library is 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.

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 library is usable for offline animation and for real-time special effects in video games and 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 library functions to generate diverse effects and to apply them to games and virtual reality applications.

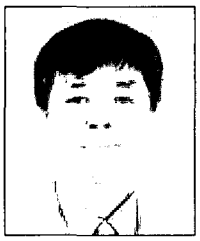
ACKNOWLEDGMENTS

This work was supported in part by Research Foundation of Engineering College, Sunchon National University and University Research Program supported by Ministry of Information & Communication in Republic of Korea

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] Reeves, W. T. and R. Blau "Approximate and Probabilistic Algorithms for Shading and Rendering Structured Particle Systems" Proc of SIGGRAPH '85, San Francisco, California, July, 1985.
[5] Leech, J. P. and R. M. Taylor. "Interactive Modeling Using Particle Systems". Proc. of the 2nd Conference on Discrete Element Methods, MIT, 1993.
[6] Allen, M. B. Flow - a particle animation application. http://www.dnai.com/~mba/sortware/flow/, 1999.
[7] http://www.opengl.org
[8] Neider, J. T. Davis, et al., OpenGL Programming Guide, Adison Wesley, 1993.

Eung-Kon Kim
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.



Bong-Kil Yoo
Received B.S. degree in department of business and administration from Chosun University, Gwangju, Korea, in 1983, the M.S. degree in department of business and administration from Chosun University, Gwangju, Korea, in 1988. He is currently a Ph.D. candidate in Department of Computer Science, Sunchon National University.
He is interested in Computer Graphics and Cyber Administration,

Seung-Heon Song
Received B.S. degree in Department of Phyiscs from Seonam University, Namwon, Korea in 1998 and M.S. degree in Major of Industrial Information Management The Graduate School of Information Science, Sunchon National University, Suncheon, Korea, in 2000.
He is currently a Ph.D. candidate in Department of Computer Science, Sunchon National University.
He is interested in Computer Graphics and Multimedia.