Improving advanced particle system by adding property milestones to particle life cycle

Miroslav Sabo[1]
University of Maribor, Faculty of Electrical Engineering and Computer Science

Abstract

This article provides information on how to create special visual effects using advanced particle systems, which may then be used in a computer game. The basic concept of an advanced particle system is expanded by introducing property milestones to a particle's life cycle in order to provide mechanism for further control over its properties with respect to particle age. This article covers the basic principles and some guidelines on how to create realistic and eye-catching visual effects using this approach.

Keywords: Property milestones, Advanced particle system, Real time visual effects.

1 Introduction

Particle systems have the ability to create realistic natural phenomena in real time. Their superiority over another computer graphic methods was realized for the very first time by William Reeves [1] back in 1982 and 1983 when working on special effects for Star Trek II: The Wrath of Khan movie. When searching for a method of creating realistic fire effect he realized that conventional modeling, which was best at creating objects with smooth, well-defined surfaces, would not do the trick. The objects that made up realistic effects such as fire were not made of easily definable surfaces. These "fuzzy" objects, as he called them, would be better modeled as a system of particles that behaved within a set of dynamic rules. Reeves realized that by applying a system of rules to particles, which at the time were only used to create simple effects such as galaxies and distant stars, he could achieve a chaotic effect while maintaining some creative control.

Basically, a particle system is just a collection of 3D points in space. Particles making up the system are non-static unlike standard geometry objects. Particles are born, they change over time, and then die off. A key point regarding particle systems is that they are chaotic. Instead of having a completely predetermined path, each particle can have a random element, called a stochastic component, which modifies its behavior. This random element is, in fact, the main reason why particle systems are so good at reproducing realistic effects.

On the other hand some kind of creative control is desirable over those stochastic elements. This aspect of the particle system is usually implemented by applying some given value to a particle's parameter, while the stochastic component is given as a variance to this base value. Some control over particle's parameters and its behavior during creating is provided using this approach.

In some cases it is desirable to have further control over particle's parameters to achieve different types of visual effects, such as change of particle color during its lifecycle. This can be achieved by adjusting the values of particle's parameters during its lifecycle. Property milestones were chosen to achieve this.

A property milestone represents a certain parameter value at a given point in time in a particle's life cycle. The values between two milestones are linearly interpolated the between values of these two milestones. Any function describing a particle's parameter value within its life cycle can be described using this approach.

An advanced particle system is just a collection of two or more single particle systems. It is possible to create more complex visual effects such as realistic fire with flame, smoke and sparks using advanced particle system, where each of them is realized as a single particle system. This can be observed in Figures 1-4.

Figure 1: Single particle system representing fire

[1] miroslav.sabo@hermes.si

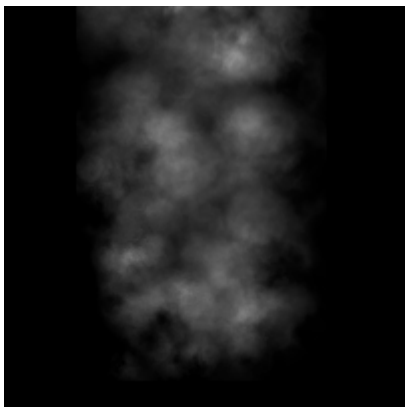

Figure 2: Single particle system representing smoke generated by the fire

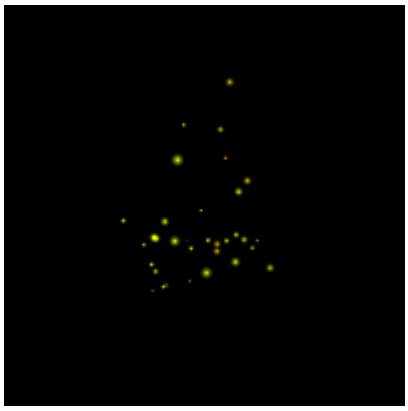

Figure 3: Single particle system representing sparkles generated by the fire

Figure 4: Advanced particle system made of particle systems in Figures 1-3 produces more realistic fire effect

2 Built advanced particle system

As mentioned above an advanced particle system is a collection of single particle systems. Each particle system behaves in a unique manner. I.e. in a fire effect a change in the wind direction vector might be desirable so that a car moving closely past a smoke system makes the smoke particles respond to the wind generated by the passing car, but not to affect other parts of the effect (fire, sparkles). Updating one particle system may differ from the method used to update another particle systems in one advanced particle system.

So what is needed is some kind of a manager class to control all particle systems, which make up an advanced particle system. Such a manager class would be in charge of creating, releasing, updating, and rendering all of the systems, although update functions may differ from one particle system to another. As such, one of the attributes of the manager class must be an array of pointers to particle systems or a vector of them.

The manager class has been implemented as an Advanced Particle System class. Figure 5 shows an overview of the built system.

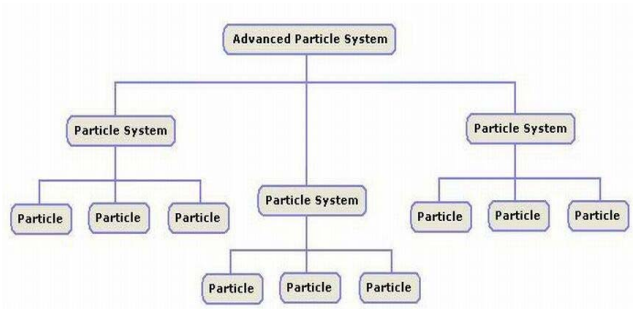

Figure 5: System overview

Let's use a bottom-up approach to describe the developed classes, beginning with the particle class.

The basic element is a particle class. Different instances of this class, with some additional functionality for creating and controlling them, are incorporated in the particle system class.

Further more two or more instances of particle system class are joined together into an advanced particle system class. Of course, additional functionality also is added.

Note that the needed attributes or building particle system itself are not in the scope of this article. More information about that topic can be found in the article "Building an Advanced Particle System" by John van der Burg [2].

The following sections present an overview of the classes used to build this advanced particle systems for generating various visual effects.

2.1 The particle class

The particle class encapsulates all the types of attributes a particle must have. Table 1 contains a list of the used attributes in a developed particle class.

Type	Name	Description
vector	m_vCurPosition	current position
vector	m_vOldPosition	previous position
vector	m_vVelocity	particle speed
color	m_clrCurColor	current color
color	m_clrOldColor	previous color
float	m_fCurPower	current power
float	m_fOldPower	previous power
float	m_fLifeTime	lifetime
float	m_fInitTime	birth time
float	m_fSpin	z-axis rotation

float	m_fSize	size
float	m_fWeight	weight

Table 1: Used particle attributes

There are some essential particle attributes, which each particle class must have. These are a particle's age or its birth time, its lifetime and position. All the others just increase the realism of the designed effect and are not essential for a particle system.

Values for its attributes are assigned, as a particle is born. They depend upon the selected base value and the defined variance range for a given attribute in a particle emitter, which are properties of the particle system class. The stochastic behavior of the developed system was achieved using this approach. During its lifetime attributes may change according to the environmental influences or influences defined by milestones.

2.2 The particle system class

The particle system class is the core of the built advanced particle system. It takes care of creating new particles, updating them according to environment parameters and according to milestones, rendering and destroying of particles.

Table 2 lists the attributes, which have been used in the developed particle system class. Note that the particle attributes shown in Table 1 and the variances for each one of them, according to which particle attributes are set at their births, are omitted from the table, although it is, in fact, also a property of the particle system class (its particle emitter, in fact).

Type	Name	Description
int	m_iMaxParticles;	maximum number of particles in system
float	m_fReleaseInterval	particle release interval
int	m_iNumToRelease	number of particles released at each release interval
float	m_fDelay	delay before starting to emit particles
vector	m_vGravity	gravity acceleration
vector	m_vWind	wind acceleration
float	m_fAirResistence	air resistance factor
float	m_fOldPower	previous power
enum	m_eEmmiterMode	emitter mode
char*	m_pchTexFile	name of particle texture file

Table 2: Used particle system attributes

Each particle system can have different kinds of particle emitter behavior.

The most common behavior is by a point emitter. If this emitter is used, all the particles are born at the same point in space, but with different speed vectors. If the speed vectors of the particles were to be almost the same in size (with very small velocity variance), the result would be something like a spreading particle sphere. Of course, the directions of the particles' speed vectors can also be limited to only two dimensions or even to different angles in different dimensions.

The next common emitter behavior is line emitting. In this case the particles are born at random points on a given line, which can also be a curve, circle, etc. They can be emitted in or out of the line in an angle range between 0°and 180°. In and out emitting can be observed in Figure 6, where a line emitter uses a circle for an emitting line. Outside emitting means that particles are emitted outside the circle.

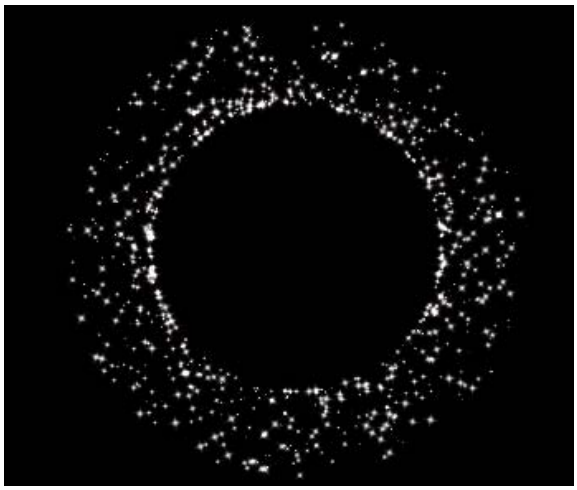

Figure 6: Line particle emitter shaped as a circle and outside emitting

Basic emitter behavior is also area emission. In this case rectangular of any other shape of the area is provided and particles are emitted within this area.

In the particle system class the possibility of choosing between different behavior has been implemented such as a point emitter with the possibility to emit only in certain angle range, a line emitter with the possibility of emitting in, out or in line both directions (also a possible emission range angle can be specified) and an area emitter, which emits within a given area. Different implemented modes of emitting particles may be observed in Figure 7.

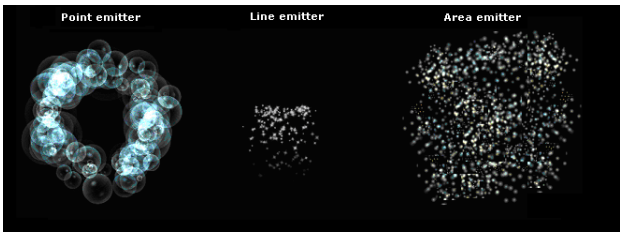

Figure 7: Supported emitter modes

The rendering of each single particle was realized using a 2D quads and billboard technique[2], due to its

[2] In this approach each particle is represented by 2D quad consisting of 4 vertices in spaces. As rendered, quad is always rotated in a way that the user always looks into it (bill-boarding technique) [9].

advantages compared to using point-spites[3], such as unlimited particles sizes.

Particle attributes are updated with respect to the environmental variables of particle system, but this does not provided enough control over particle system behavior, so it was opted to use property milestones to solve this problem. This used approach is further explained in section 3.

2.3 The advanced particle system class

An advanced particle system class has been used as a manager class to control all of our various particle systems. This class is in charge of creating, releasing, updating, and rendering all of its subsystems (particle systems). As such, one of the key attributes in the advanced particle system class is an array of pointers to the particle systems, which are managed by it.

Its basic functions are displayed in Table 3.

Name	Description
Init	Initializes the advanced particle system.
AddSystem	Adds a specified particle system.
RemoveSystem	Removes a specified particle system.
Update	Update all particle systems
UpdateSystemPosition	Update all particle system positions
Render	Render all particle systems
IsActive	Check is particle system is not active anymore (if all of its particles have died)

Table 3: Advanced particle system class function

All members of particle system class, which were inherited allowed to link developed particle systems within the hierarchy of our game engine, thus allowing the engine to affect the position of each particle system. As the position of the advanced particle system is changed by the game or any other application using it, the positions of all particle systems are also updated respectively.

This is needed to link visual effect to an object of a game. The effect of exhaust flames could be attached to a flying rocket using this approach. As the rocket moves across the screen, the game engine also updates the position of advanced particle system, which enables these particles to be emitted at the correct position. This is not needed, if you are building a stand-alone particle system.

[3] Point sprites are a new feature in Microsoft DirectX 8.1 APIs. DirectX 8.1 defines a point sprite as a textured point with texture co-ordinates 0,0 in the top left corner and 1,1 in the bottom right corner [11]. This feature exists in OpenGL graphic library as well.

3 Updating particle properties using property milestones in its lifecycle

In order to create realistic visual effects using particle systems, it was necessary to have very good control over particle properties, but the stochastic nature of particles should not be sacrificed to achieve this.

Some control over a particle's parameter such as its size is provided as the given particle is born. Base value and its variance are passed as properties of the particle system, which emits particles, as shown in the following formula.

```
pParticle->m_fSize = m_fSize + vRandomVec.x * m_fSizeVar;
```

By assigning base value and setting variance to 20%, it can be assured that particle size, which is assigned to it as it is being created, will be between 80% and 120% of the given base value.

The behavior of each particle can be further influenced during its lifecycle by using environmental parameters. These can be used for particle parameters such as its speed. Current particle speed is update during its lifecycle according to particle initial speed, wind and gravity,.

In some cases the provided control is insufficient to achieve the desired effect. Only the staring size of the particle, which would differ for particle to particle, can be set using this approach. It would still remain the same during its whole lifecycle. What if its size or its color needs to be changed according to some function during its lifecycle?

In order to achieve this goal, property milestones have been introduced to particles' lifecycles. A property milestone is basically a relative value for particle initial or current property value, which is applied at certain points in time during a particle's lifecycle. An example of property milestones for particle size is shown in Figure 8.

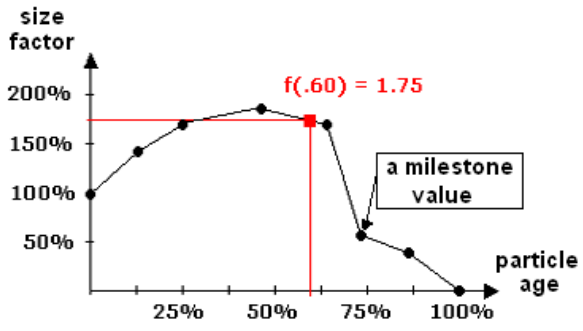

Figure 8: Property milestones for particle size

Values at certain times of particle lifecycle between two property milestones follow the linear function defined by these two milestones. Any time-dependent function which the particle's property value needs to follow during its lifecycle, can be linearly approximated by adding more property milestones

Relative value is used to preserve the stochastic nature of the particle system. It is multiplied to initial or current particle's attribute value to gain more control over particle behavior. The use of relative factor values was chosen in the range from 0% to 200%, but using different ones can be chosen too.

This approach may be applied both to particle attributes, which are unaffected by environmental forces (i.e. particle size), and those, which are effected by them (i.e. particle speed). When applying to the second group, the relative factor at a given point in time must be applied to the current particle attribute value in order to preserve all influences applied to it by environment until this point.

The following code shows how relative milestone value was applied to the size attribute of the particle.

```
pParticle->m_fCurSize = pParticle->m_fSize * GetSizeOverLife(fAge / (pParticle->m_fLifeTime));
```

Additional control over the particle properties was provided using explained approach, while the stochastic nature of the particle system remained intact.

4 Conclusion

Creating suitable visual effects by implementing an advanced particle system using property milestones is flexible, fast, extensible and quite easy. Various visual effects have been created, which are shown in figures 9 - 12.

Figure 9: Explosion with shock wave effect

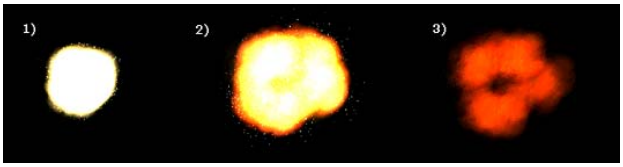

Figure 10: Big explosion effect

Figure 11: Fireworks effect

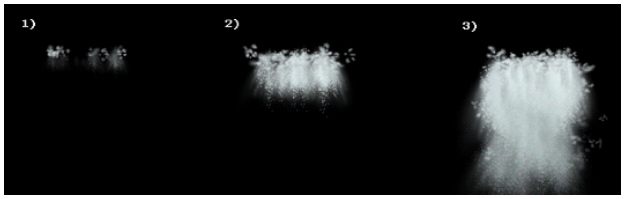

Figure 12: Waterfall effect

Some problems while trying to visualize the particles, emitted by the particle system, were encountered during implementation. The first version implemented an approach using point sprites for rendering particles. This approach proved to be insufficient for our purposes. The maximum sizes of the particles varied from one video card to another and the rotation of particles around the z-axis was impossible to implement using this approach. The old-fashioned approach using quads and bill boarding was used instead, which proved to be satisfactory.

DirectX 8.0a library was used for rendering the advanced particle system.

It very easy to give additional functionality and further enhance the visual effects using class inheritance.

All developed visual effects can be easily integrated into simple 3D video games as shown in Figure 13.

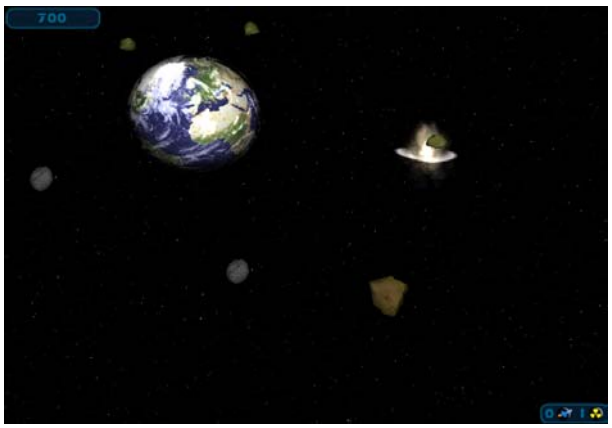

Figure 13: Created explosion with shock wave effect used in video game

Of course, the implemented advanced particle system is far from being perfect. It could be improved in many ways, which can be addressed in future work.

Particles could have additional properties such as separate sizes in the x and y directions. Achieving this would enable the stretching of particles during their life cycles, which can be very useful for providing visual effects of various light rays or effects during teleportation (often used in computer games and science fiction movies). Further improvement would be to expand the particle emitter with new emitting modes. In addition a very interesting idea would be to possibility add the change of emitter's properties during its emission time (similar to what was already done to the particle properties). For instance, doing this it would be possible

to control how many particles the same emitter would emit over different time intervals.

References

[1] W.T. Reeves, *Particle Systems - a Technique for Modelling a Class of Fuzzy Objects*. In ACM Transactions on Graphics, ACM, Academic press, April 1983

[2] W. T. Reeves, *Approximate and Probabilistic Algorithms for Shading and Rendering Structured Particle Systems*. In Computer Graphics, vol. 19, no. 3, pp 313-322, 1985.

[3] John van der Burg, *Building an Advanced Particle System*. Available from Gamasutra.com at http://www.gamasutra.com/features/20000623/vanderburg_pfv.htm

[4] Markus Hadwiger, *Real-Time Special Effects for a Computer Game Using Particle Systems*. Available from http://www.cg.tuwien.ac.at/~msh/partsys.pdf

[5] C. W. Reynolds, *Flocks, Herds, and Schools: A Distributed Behavioural Model*, Computer Graphics, vol. 21, no. 4, pp 25-34, 1987.

[6] Jing Zhong, *Particle Animation and Procedural Texture Maps*. Available from http://cs-people.bu.edu/jingzh/CS580/P2/CS580p2.htm

[7] Todd Reed and Brian Wyvill, *Visual Simulation of Lightning*. In Proceedings of SIGGEAPH '94, pp. 359-363, 1994

[8] Mel Guymon, *Pyro-Techniques: Playing with Fire*. In Game Developer, vol. 7, no. 2, pp. 23--27, Feb. 2000.

[9] Tomas Akenine-Möller, Eric Haines, *Billboarding, Excerpt From Real-Time Rendering 2E*, september 2002. Available from http://www.flipcode.com/articles/article_rtr2billboards.shtml

[10] Mason McCuskey, *Special Effects Game Programming with DirectX*, The Premier Press, 2002

[11] Microsoft, *DirectX 8.1 SDK*, Microsoft, 2002. Available from http://msdn.microsoft.com/directx