Programming for Graphics

Particle System initial design report

Hannes Ricklefs
d1132332@bmth.ac.uk
University of Bournemouth

ABSTRACT

This report is structured into three different sections. The first section gives a brief introduction into particle systems. The second provides a description of each of the components that combine the particle system. The third gives a brief summary of the report and the main focus when implementing the particle system.

Keywords

Particle System, Particles, Emitter, Solver, Data Structures

1. INTRODUCTION 1
2. PARTICLE SYSTEM 1
2.1 Overview 1
2.2 Particle System Manager 1
2.3 Particle System 2
2.4 Emitter 2
2.5 Particles 2
2.6 Solver 3
2.7 Data Structure 3
3. SUMMARY 4
4. APPENDIX A 4
5. REFERENCES 4

1. INTRODUCTION

Since their introduction in 1983 particle systems have become one of the fundamental parts of computer animation [1]. The pioneer in this area is William T. Reeves, in his initial paper he describes the use of particle systems to model "fuzzy" objects such as fire and clouds. Since then particle systems have been used in scientific simulations, feature films [1,8] and games [10] to model almost any kind of natural phenomena including smoke [3,6,8], fire [1,3,4,8], water [4,6,9], explosions [1,5,7], plant growth [2] and galaxies [3].

2. PARTICLE SYSTEM

2.1 Overview

As described by Reeves [1] a particle system is defined by a collection of particles that evolve over time. The evolution is determined through applying certain rules to the particles: creation of new particles, changing their attributes during their lifetime and finally the destruction ("death") of old particles. The main essence of a particle system is that all of the afore mentioned rules are executed automatically. In general, particle systems are constructed using the following components: particles, emitters and solvers [1,2,3,4,5,10,11,12,13,14].

The proposed design will evolve its particles according to these five steps [1,10,11].

1) New particles are generated and injected into the current system.
2) Each new particle is assigned its individual attributes.
3) Any particles that have exceeded their lifetime are extinguished.
4) The current particles are moved according to their scripts.
5) The current particles are rendered.

The main aim of this proposed design is to have a particle system that is easy to extend for all the components in the system. The components are **Particle System Manager**, **Particle System, Emitter, Particle, Solver**, and **Data Structure**.

2.2 Particle System Manager

In his article J. Burg mentions a Particle System Manager, which controls all the various particle systems [5]. As the proposed design supports multiple particle system a manager class becomes important. The manager class is in charge of creating, releasing, updating, and rendering all of its particle systems. The particle system manager needs to have a timer in order to function according to their sequence in time.

2.3 Particle System

The particle system class is responsible for holding all the initialization values for its solver, particles, and emitter. One aspect of the proposed design is to be able to combine particle systems in such a way that a particle system itself can include other particle systems in order to create images as in Figure.1 [1].

Figure.1 from [1]

Furthermore, the particle system is able to change over time, which will allow effects like a glowing cigarette particle that turn into smoke. Therefore the particle system class needs to hold all the relevant attributes, solvers and emitters to make the changes to its particles. Evermore, the particle system counts the number of alive particles to determine whether a system is ready to be deleted or not.

The class diagram in Appendix A shows two subclasses of the particle system class: Behavior Particle System and Effect Particle System. The Behavior Particle System enables interaction between particles in order to model flocking behavior [15,16].

The Effect Particle System enables quick instantiation of a system to create effects such as fire, explosions, smoke, etc. It could become as a super class and have special Fire Particle Systems, Smoke Particle System as its sub classes. This would allow other developers to use the particle system as an application programmer interface (API) [13].

2.4 Emitter

For a particle system to evolve and change its volume over time new particles need to be added to the system. This is the main work of the emitter, who generates new particles by means of controlled stochastic processes [1,11]. For each frame new particles are randomly emitted into the system according to some user specified distribution [11]. Both [1,11] propose to use two different equations in order to calculate the number of new particles.

(Eq. 1) NParts = MeanParts + Rand(_) * VarParts

(Eq. 2) NParts = (MeanParts +Rand(_) * VarParts) * ScreenArea

Where Rand(_) returns a random number between -1.0 and +1.0, MeanParts states the mean number of particles, and VarParts its variance [1,11]. In the second formula (Eq. 2) the screen size of the object determines the number of new particles in the system.

In the proposed design all particle behavior is assigned from the emitter: **Emitting direction, Initial velocity, Bounce factor, Friction, Color, Transparency, Radius, Gravity** and **Texture Map.** All the initial values will be set at random at the beginning of a particle's life.

The class diagram in Appendix A shows that the Emitter class is an abstract class. In order to have the most flexibility when creating new particles the following Emitter types are proposed:

1) Spherical Emitter: Particles are emitted from the surface of a sphere. The Emitter has a radius and a sweep angel for both the top and the bottom in order to allow particles to be emitted from a ring shape.
2) Cubical Emitter: Particles are emitted from a cubical shape. The Emitter has length, width, and depth.
3) Shape Emitter: Particles will be emitted from the shape/geometry passed into the constructor. This will allow converting shapes/geometries into particles. For example a head could dissolve into smoke.
4) Planar Emitter: Particles are emitted from a plane. The Emitter has a width, depth and direction.

2.5 Particles

The particles determine the motion, volume, color and general appearance of the particle system. Typical Attributes according to [1,5,11] are: **position, velocity, color, transparency,** and **lifetime**. In [5] J. Burg adds two attributes **oldPosition, texture** and **dead**. OldPosition enables to stretch a particle between its old and new position, which can be useful when creating sparks. Texture enables particles to have a texture assigned to them. The attribute dead is particular effect introduced by J. Burgs article. He states that instead of deleting and creating new particles, a dead particle should be flagged and then reinitialized when new particles are created. This reduces the amount of memory access and is hence faster.

In [4] Sims adds jet another attribute called **bounce,** which specifies what happens to a particle when colliding with other particles or objects in the scene.

The class diagram in Appendix A shows that the particle class has three subclasses:

1) Point Particle: A Particle is just a point in space.
2) Plane Particle: A Particle consists of two triangles that form a rectangle [5] as shown in Figure.2.

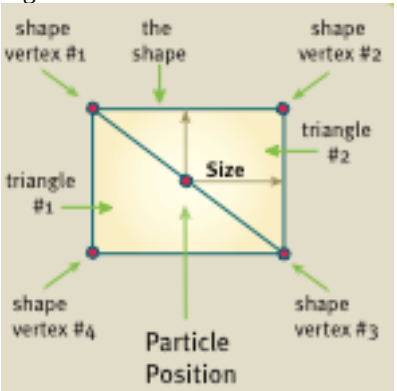


Figure 2 from [5]

For this particle the author ensures that the plane will always face the camera (bill-boarding).

3) Volume Particle: A Particle consists of a shape/geometry passed to the constructor.
4) Force Particle: A Particle can be used to be a force for other Particles [3].
5) Changing Particle: A Particle can change its Solver over time. This allows a particle to behave as a fire particle and then change its behavior to smoke after a certain duration.

2.6 Solver and Forces

In order to create any of the effects mentioned in section 1 the particles need to be animated. Although many effects need specialized algorithms, this particle system is designed to incorporate new algorithms very easily. By specifying a Solver interface newly created algorithms can be "plugged" into the particle system. The literature presents all kinds of different solvers from basic linear solvers to advanced solvers using Navier-Stokes equations [6,8,9]. In [3,4,14] the authors are introducing Forces into the movement of the particles. To give more flexibility this design separates the forces into classes in order for the Solver to choose one or many correct forces. In [14] A. Witkin groups the forces into three broad categories, which form the three subclasses of the Force class:

1. Unary forces, such as gravity and drag, that act independently on each particle, either exerting a constant force, or one that depends on one or more particle positions, particle velocity, and time.
2. n-ary forces, such as springs, that apply forces to a fixed set of particles.
3. Forces of spatial interaction, such as attraction and repulsion, which may act on any or all pairs of particles, depending on their positions.

The class diagram in Appendix A. shows the Solvers the author is proposing to implement:

1. Basic Solver: Solves particle movement according to a linear equation. Particles fly through space at constant time.
2. Gravity Solver: Solves particle movement according using gravity. Particles fly through space and the direction and speed varies according to the laws of gravity.
3. Navier-Stokes Solver: This is the most advanced Solver. In order for the Solver to work properly a 3D data structure is needed [6,18,19].

2.7 Data Structure

Through the ever-increasing speed of processing power in computer hardware, particle systems are reaching a stage at which they can be used to create effects close to reality in real-time [3]. In the special effects industry this is still not achieved [8]. In order to create satisfying results in real-time, the number of particles used within a particle system needs to be within thousands [5]. Therefore an efficient data structure is needed to store the particles.

Whenever complex or efficient storage requirements are needed, the literature refers to Octrees [6,12,18,19]. Octrees are a hierarchical construct for spatially managing large amounts of three-dimensional data [12,17]. Their construction is well documented and researched [12,17,18]. The usual approach is to start with a three-dimensional cube and continually subdivide it into eight cubes until either a specified cube size is reached or there are no more objects that could be subdivided. See figure 2.

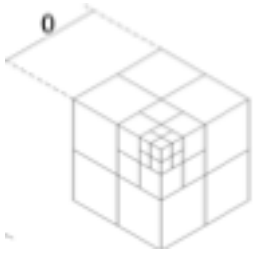


Figure 2 from [19]

When implementing an Octree data structure for particle systems, additional aspects need to taken into consideration. The main aspects are finding neighbors on the faces, edges and vertices of the corners, the constant subdividing of the sub-cubes, and changing particles from one cube to another.

The main benefit of Octrees is the ability to store extra information within nodes that can be applied when solving the animation of the particles using the Navier-Stokes solver.

The class diagram in Appendix A. shows additional data structures the author proposes:

1. Grid: A three-dimensional grid structure with a gird size of 30 * 30 * 30.
2. List: A data structure that uses a list container class from the standard C++ library. Here some special sorting algorithm (bubble or quick sort) is going to store the particles in an order relating to their position in the particle space. Another approach would be to cluster particles according to their location, as stated in section 2.5.
3. Octree: The aforementioned data structure.

It is the data structures responsibility to do collision detection between the particles and objects from within the scene. Additionally, the data structure is responsible for creating a pool of deleted "dead" particles that can be reused when new particles are created. Furthermore, the data structure will have a method to only select a subset of particles in order to have high numbers of particles within the particle system and still be able to perform simulations in real-time [4].

3. SUMMARY

This report outlined the components the author is planning to implement for the assignment. His main focus is on the extendibility of any of the components and the combination of particle systems with in particle systems.

The outline of the algorithm for evolving a particle system is as follows:

Create particles

Initialize particles

For each frame:

For each simulation time increment:

Select particles

Perform operations

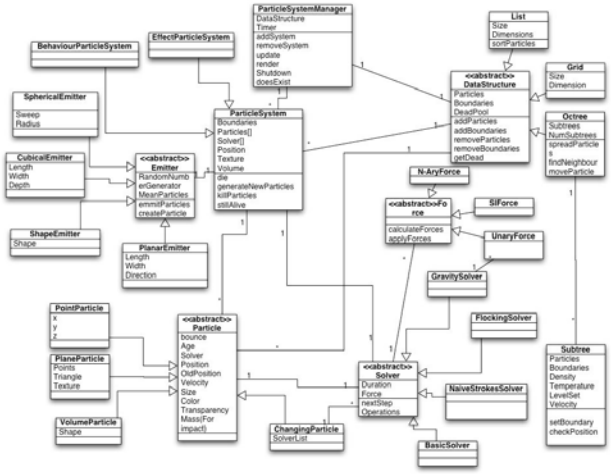
Update position and other attributes

Evolve age remove dead particles

Render

The resulting demonstrations of the particle system will include: grass growth, explosion, and a waterfall.

4. APPENDIX A



5. REFERENCES

[1] REEVES, W. T., 1983. Particle Systems – A Technique for Modeling a Class of Fuzzy Objects, *ACM Transaction on Graphics*, 2 (2), 359-367.

[2] REEVES, W. T., AND BLAU R., 1985. Approximate and Probabilistic Algorithm for Shading and Rendering Structured Particle Systems, *Computer Graphics*, 19(3), 313-322.

[3] ILMONEN, T., AND KONTKANEN J., 2003. The Second Order Particle System, *Journal of WSCG*, 11(1)

[4] SIMS, K., 1990. Particle Animation and Rendering Using Data Parallel Computation, *Computer Graphics*, 24(4), 405-413.

[5] BURG, J., 2000. Building an Advanced Particle System, *GAME DEVELOPER*, 44-50.

[6] FEDKIW, R., GIBOU, F., AND LOSASSO, F., 2004. Simulating Water and Smoke with an Octree Data Structure, *SIGGRAPH 2004 ACM TOG*, 23, 457-462.

[7] FELDMAN, E. B., O`BRIAN F. J., AND ARIKAN, O., 2003. Animating Suspended Particle Explosions, *ACM SIGGRAPH 2003, San Diego, CA, July 27-31*, 1-8.

[8] FEDKIW, R., ENRIGHT, D., AND NGUYEN, D., 2003. Simulation and Animation of Fire and Other Natural Phenomena in the Visual Effects Industry, *Western States Section, Combustion Institute, Fall Meeting, UCLA, 2003.*

[9] MÜLLER, M., CHARYPAR, D., AND GROSS, M., 2003. Particle-Based Fluid Simulation for Interactive Applications, *Eurographics/SIGGRAPH Symposium on Computer Animation (2003).*

[10] WATT, A. AND POLICARPO F., 2001. *3D GAMES Real-time Rendering and Software Technology*, New York: ADDISON-WESLEY

[11] PARENT, R., 2002. *Computer Animation Algorithms and Techniques*, London:Morgen-Kaufmann.

[12] FOLEY D. J., VAN DAM, A., FEINER, K. S., AND HUGES, F. F., 1997. *Computer Graphics PRINCIPLES AND PRACTICE,* 2nd ed. New York: Addison-Wesley.

[13] MCALLISTER, K. D., 2000. The Design of an API for Particle Systems, *Technical report, University of North Carolina*, 2000.

[14] WITKIN, A., 1999. Physically Based Modeling Principles and Practice – Particle Systems, *SIGGRAPH-99 Course Notes.*

[15] REYNOLDS, W. C., 1987. Flocks, Herds, and Schools: A Distributed Behavioral Model, *Computer Graphics*, 24(4), 25-34.

[16] LOREK, H. AND WHITE, M., 1993. Parallel Bird Flocking Simulation,

[17] DELOURA, M., 2001. *GAME PROGRAMMING Gems 2*, Hingham: CHARLES RIVER MEDIA, INC.

[18] VEMURI, C. B., CAO, Y., AND CHEN, L., 1998. Fast Collision Detection Algorithms with Applications to Particle Flow, *Computer Graphics forum*, 17(2), 121-134.

[19] TISNOVSKY, P., AND HEROUT, A., 2002. Adaptive Algorithm for Vector Field Interpolation Based on Octree Structure, *International Conference on Computer Graphics and Interactive Techniques Proceedings of the 18th spring conference on Computer graphics*, 151-156