Particle Systems: Theory and Practice

Ciara Belle
CMSC498A
Spring 2012

Introduction:

Modeling non-deterministic, complex objects is difficult, using general techniques in computer graphics. Particles systems are used to overcome the difficulty of modeling these "fuzzy" objects, including clouds, fire, and water [1].

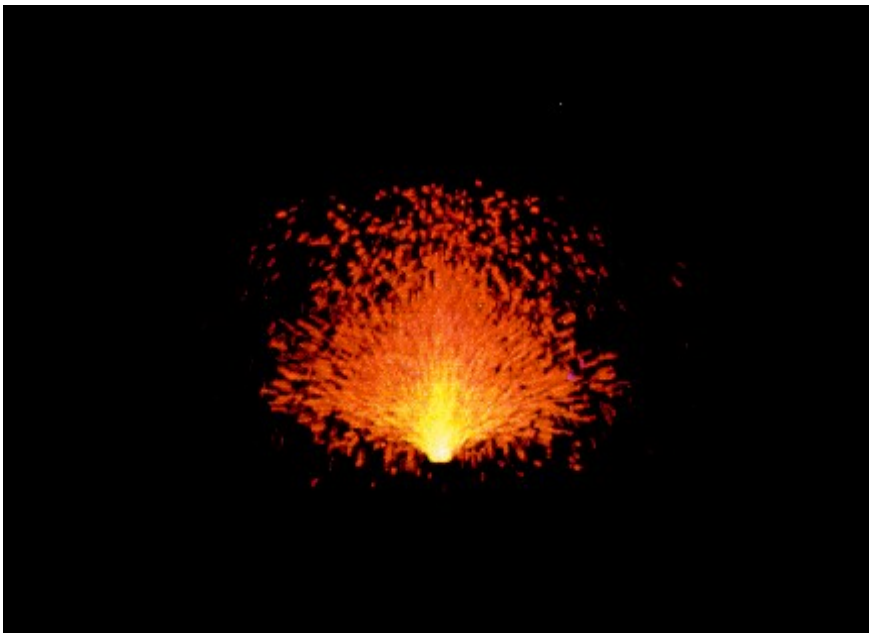

Figure 0) Two red fireworks

Particle systems differ in three ways from normal modeling techniques. The first difference is that particle systems are not sets of primitive shapes and surface elements, instead they are clouds of primitive particles that define an object's volume. Second, particle systems are dynamic and are constantly changing form with the passage of time. The last difference is that the objects that particle systems attempt to represent are generally non-deterministic [1].

There are many advantages associated with using particle systems. One advantage is that a particle is much simpler than a polygon, which helps with computation speed and flexibility. More primitives can be produced in the same amount of time as more complex objects. This advantage also makes it easier to produce blurring effect, such as motion blurring [1].

Another advantage involves the motion of the particles in a particle system. Particle motion is done procedurally, and is primarily controlled by a random process. One reason why this is an advantage is that the system requires very little human design. Another positive implication of this is that a particle system can adjust its level of detail to suit specific viewing parameters.

Perhaps the most prominent advantage of particle systems is that it makes it much easier to model objects that would be considered “alive,” and change over time [1].

The basic model of a particle system involves a collection of minute particles. All particles within the collection belong to a system. As this system animates, particles are generated into the system. Over time, these particles change within the system according to their attributes, then eventually die from the system. The steps for computing each frame of a particle system motion sequence are [1]:

1. New particles are generated and introduced into the system.
2. Each particle is assigned its individual attributes
3. Old (expired) particles are extinguished

4. Remaining particles change based on their attributes
5. Image of living particles are rendered in frame buffer

The generation of particles in a particle system is accomplished by the use of controlled non-deterministic processes. There is one process that determines the number of particles that will be entering the system during each cycle of time for a given frame. The amount of particles within a system determines the density of the fuzzy object. The designer of the model can control the number of new particles within a system in one of two ways [1]:

1. Control the average number of particles generated at a frame and its variance.
   - $Nparts = MeanParts + Rand() \ VarParts$
     - *Rand*() is procedure that returns value between -1.0 and 1.0
2. The designer of the model controls the average number of particles generated for every unit of screen area and its variance. Procedures for particles systems can determine view parameters at a particular frame, calculate the approximate screen area it covers, and set the new number of particles accordingly.
   - *Nparts* =

$$(MeanParts[sa] + Rand() \cdot VarParts[sa]) \cdot ScreenArea$$

For each new particle generated, the particle system must determine the values for each of the new particle's attributes. These attributes are its initial position, initial velocity (speed and direction), initial size, initial color, initial transparency, shape, and lifetime [1].

There are several parameters within a particle system that determine and control the initial position of the particles being generated. A particle's initial orientation is defined by two angles about the origin of a coordinate system in three-dimensional space, since a particle system resides in and has a position in three-dimensional space. The generation shape of a particle system defines the region of space that new particles will be randomly placed. Particles are neither born out of, nor leave the region of the generation shape [1].

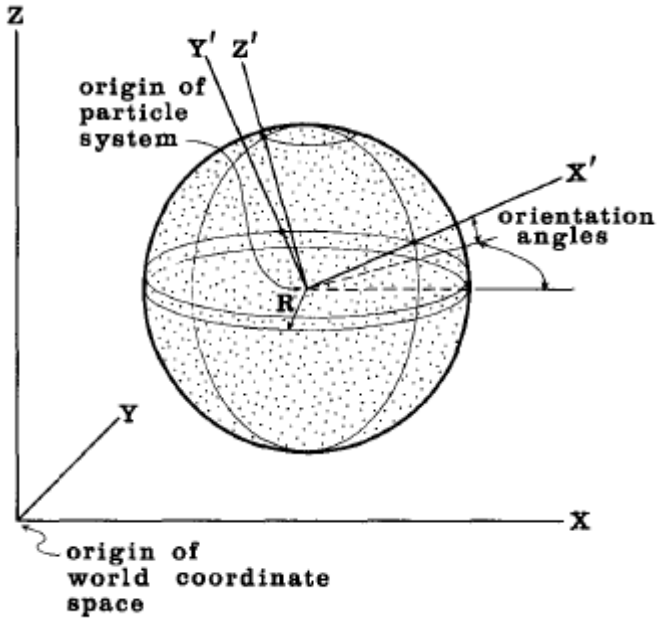

Figure 1. Particle System with a spherical generation shape.

In particle systems, the generation shape also defines the initial direction of the particle's movement. For example, in a rectangular or circular generation shape on an x-y plane, the particles in the particle system will initially move upwards from the x-y plane, however, the particles are allowed to have vertical variance. Th following equation illustrates how a particle's initial speed is determined [1]:

- InitialSpeed = $MeanSpeed + Rand() \cdot VarSpeed$

In this equation, *MeanSpeed* and *VarSpeed* are parameters set within the particle system. They define the average speed of the new particles, along with the speed's variance.

For the initial color of a particle within a particle system, the system receives information about the average color and the variance of that color. The formula for calculating the particle's color is as follows[1]:

- InitialColor = $MeanColor + Rand() \cdot VarColor$

Similar to the previous equation for computing the initial speed, *MeanColor* denotes the information of the average color given to the system, and *VarSpeed* defines how much the particle's color can deviate away from the mean color. Particle opacity and size use similar equations that utilize average values and variation. So the initial transparency formula is [1]:

- InitialTransparency = $MeanTransparency + Rand() \cdot VarTransparency$

And the equation for the initial size is [1]:

- InitialSize = $MeanSize + Rand() \cdot VarSize$

There is also a parameter within a particle system that determines the shape of each partcile that is generated. The most basic shapes for particles are spherical, rectangular, and streaked spherical. Streaked spherical is primarily used for particle motion blurring techniques [1].

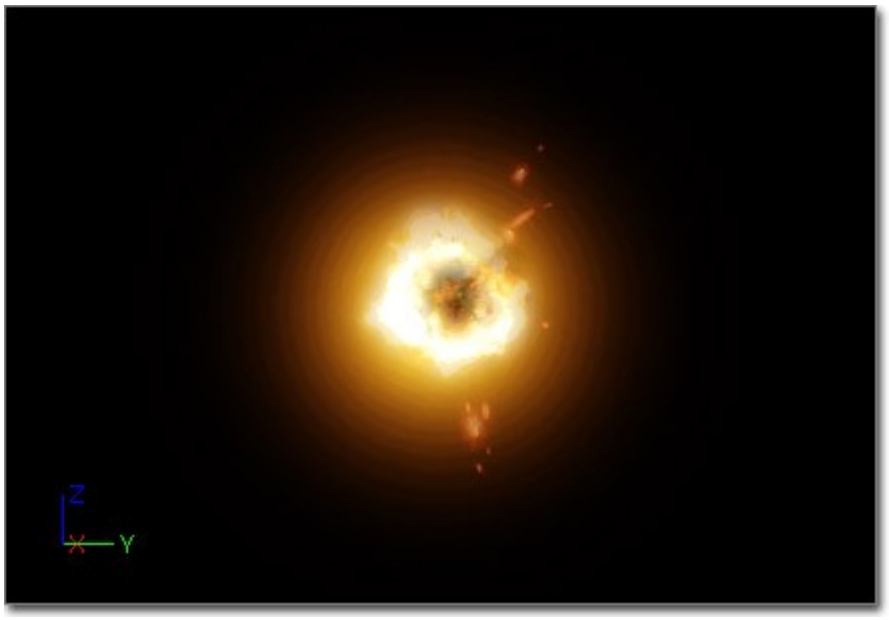

Figure 2. Example of particle blurring

Next, let us discuss particle dynamics. Over time, particles of a particle system change in attributes such as color, size, and transparency, within three-dimensional space.

It is relatively simple to move a particle from frame to frame. The way in which one would achieve this is to add the particle's velocity vector to the particle's current position vector. If the model designer desires, they could apply the particle system's acceleration element and alter a particle's velocity from one frame to the next, adding higher complexity to the system. This allows the modeler to simulate physical forces such as gravitational fields, and move along more complex functions instead of only a straight line [1].

There is also a parameter within a particle system that effects the rate at which a particle's color changes. When adjusting the transparency and size of a particle, it is achieved used the same method [1].

After a particle's birth, it is assigned a lifetime that is measured in frames. The lifetime counter is decremented after every frame, and the particle is “killed” when the lifetime runs out, or reaches zero.

Applications:

Particle systems have many applications in computer graphics. One such application includes surface modeling of free-form shapes, or shapes that do not have a definitive look. For example, particle systems are a great tool for modeling elastic objects. One way to achieve this using particle systems is to create a framework in which the particles in the system have long-range attraction forces, and short-range repulsion forces, following Newtonian dynamics. The

approach is similar to many computational models of fluids and solids. This framework was developed by Richard Szeliski and David Tonnesen. Their computational model enables the ability for particles to model surface elements instead of a point mass or volume object [3].

When modeling natural phenomena like waterfalls or fire, particles under this model have motion that is influenced by restrictions and force field, but interactions between the particles are absent. Particle systems that base their models off of molecular dynamics in order to model solids and liquids, have potential fields that are spherically symmetrical, but particles organize themselves in a way that characterize volumes instead of surfaces.

Shannon Drone wrote about particle interactions in terms of particles that react to other particles [6].

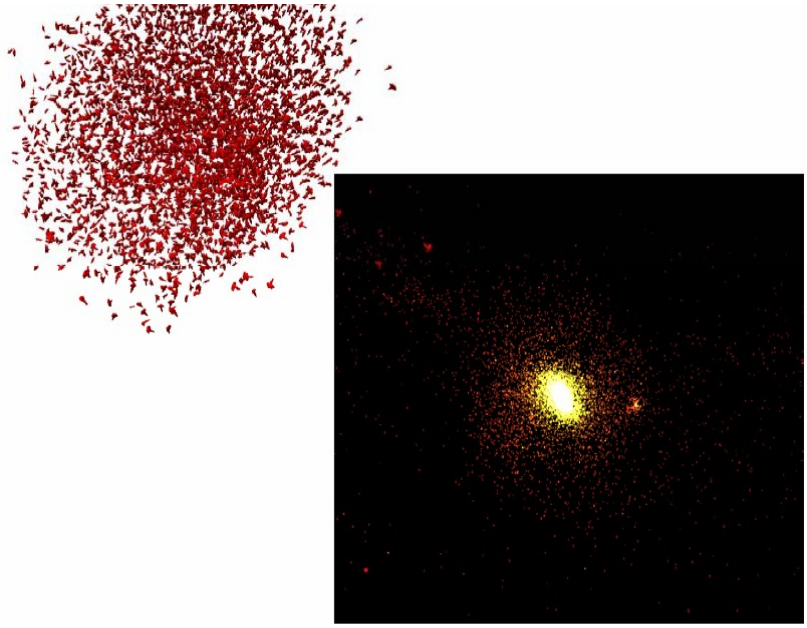

Figure 3. Flocking and gravity simulation.

Particle systems that require every particle to influence every other particle in the system, are usually categorized as N-Body problems. The N-Body problem involves predicting the motion of objects that react and interact with each other gravitationally. Force splatting is a method used for projecting a force from one particle unto every other particles of the system within a single operation [6].

It is possible to use force splatting to generate gravity interactions in which you calculate the force of gravity of every particle to all of the other particles in the system. Force splatting is used to gather all forces brought onto every particle in the system.

Game developers often used the concept of flocking particle systems, which is another system where particles interact with each other.

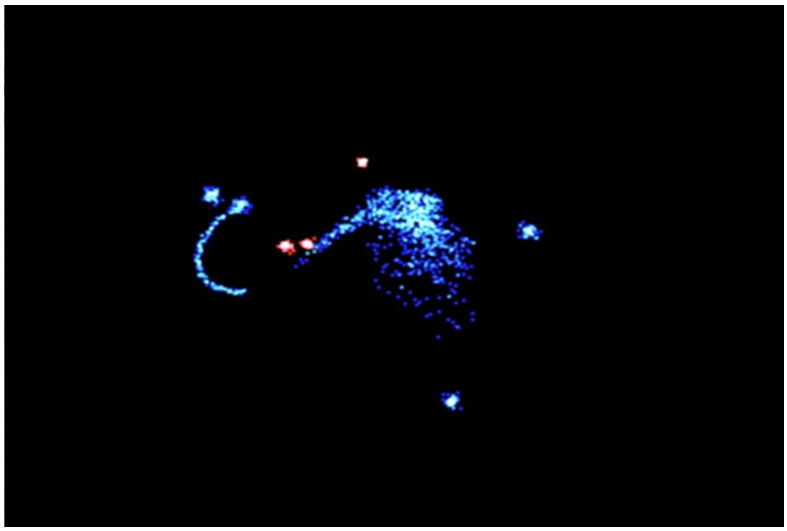

Figure 4. Snake game based of flocking particle system

It is often the case that developers will use particle systems to simulate the motion and look of a flock of birds or other animals such as flies flocking or swarming around trash or a light. Drone discussed usual flocking behaviors that should be followed if one desires a flocking illusion to look real. Adapting Reynolds concept of flocking [8], Drone describes the following rules [6]::

1. Collision avoidance
2. Separation
3. Cohesion
4. Alignment

Drone achieved collision avoidance and separation by calculating a force of repulsion for every particle that is determined by "how close the particles are to colliding" or how much distance or space in between the particles.

In order to implement behavior such as cohesion and alignment, it is necessary to know the "average position and average velocity of the particles respectively." The idea includes the creation of a force vector that reaches from the particle to the center of mass of the flock for cohesion, or the creation of a force vector that aligns a particle with the overall average velocity of every other particle in the flocking particle system.

Not only can particles in a particle system react with each other, they can also react to the surrounding environments.

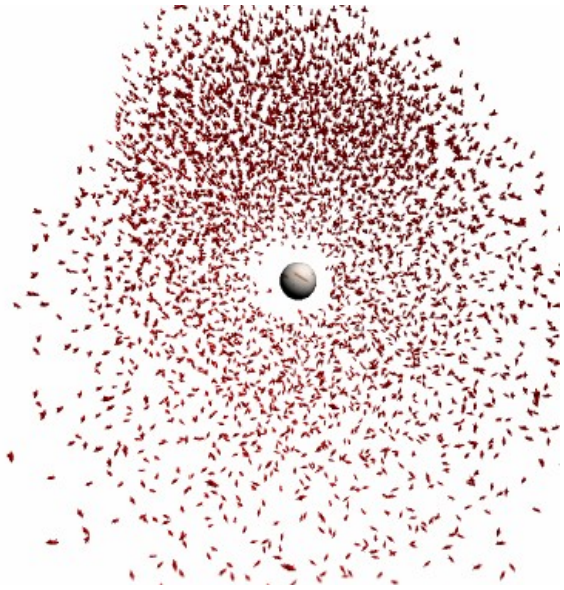

Figure 5. Image of thousands of spaceships flying away from a sphere

Drone presents what she refers to as the simplest way for an environment to interact with a particle system. The idea involves the influence of a set of "point charges." This method can be used for both seek and flee particle system behavior.

It is often the case that particles in a system need to interact with objects and shapes that are not able to be accurately defined. One way to overcome this is by partitioning the occupied space into a three-dimensional grid, where each cell of the grid has information about the area of the scene geometry that intersects that particular cell. This information includes the plane equation and velocity of the scene geometry that intersects that grid cell.

When a particle in a particle system enters the update phase, the particle fetches information about what grid cell they are located in as well as the previously mentioned plane equation and velocity of the scene geometry. The data describes whether an intersection occurred between a particle and the scene geometry, as well as the new position and velocity of the particle for when a collision actually occurs.

Theories:

There are many theories for future uses of particle systems. There is one theory that describes a programmable particle system framework for shape modeling. As previously mentioned, particle systems are great at modeling not only fire, water, smoke, clouds, and other fuzzy objects, but surfaces and shapes. Particle systems are also great for scientific visualization. They are useful and effective tool for the visualization and interactive modeling of surfaces. the framework utilizes the idea of a surface-constrained particle system, which includes the method of point repulsion. This concept is useful for concepts such as free-form modeling, morphing, cellular textures, and surface texture synthesis [7].

Surface-constrained particle systems are a very important tool in computer graphics. Its many and varied applications require that there be sometimes subtle and sometimes major

modifications of:

1. the dynamics of particles in a particle system as they “distribute across the surface”
2. the state of the particle system in its entirety as well as each individual particle in the system
3. the interactivity and appearance of the particle-user communication.

Small changes can be easily accomplished using parameter alterations within a particle system that currently exists. However, it is often the case that bigger, more major modifications have lead to the need for reconstruction of new particle system applications largely from scratch[7].

In the 1980's, there was a similar situation that surrounded the use of a utility that involved the writing of programs that followed one procedure in generating texture and other phenomena. The standardization of the articulation of procedural shading algorithms came along in 1990. Su [7] wrote a theory that aimed to standardize the articulation of particle systems, providing the opportunity for the same level of assistance. The concept, in Su's case, focused primarily on particle systems used for shape modeling and texturing.

Her system arranges a particle system into three types of objects:

1. Behavior objects, which decompose and compartmentalize the action of a collection of particles into reusable and interchangable modules
2. Shader objects, which likewise enclose the appearance and user interactivity of the particles
3. Attribute objects, which contain state information and utilities that can be easily shared and accessed among the different combinations of behavior and shader objects.

All of these objects are a collection that describes a homogeneous collection of particles in a generic *Particles* object, and multiple Particles are collected into a ParticleSystem object [7].

The importance of the modular, compartmentalized design of the framework is that it provides a useful and intuitive mental model for the rapid prototyping of particle programs from resuable high-level building blocks. The building blocks provide programmability which allows the programmer to efficiently add functionality while focusing only on the added element [7].

The *Particles* object in the Wickbert system [7], is a homogeneous set of particles that have similar behaviors, and one or more of these can be collected together in a ParticleSystem object. A *Particles* object includes :

1. a sequence of ParticleBehavior objects that, together, details the action of the particles
2. a sequence of ParticleShader objects that determine how the particle object is drawn
3. a set of ParticleAttribute objects that contain shared data used by the behaviors and shaders.

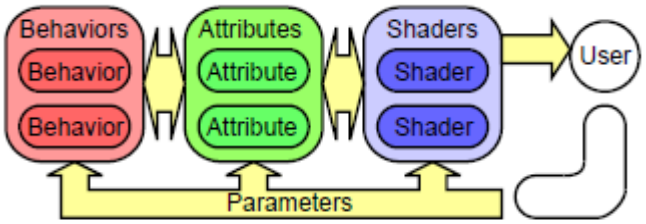

Figure 6. *Particles* object data flow. Attributes collect information shared by Behaviors and shaders. User observes particles and interactively adjusts parameters.

In the Wickbert system, individual particle and global elements and procedures of a *Particles* object are arranged in several *ParticleAttribute* objects. The attributes are stored and referenced by name in an associative array. In the initialization phase, behaviors and shaders get access to an attribute by using the *attachAttribute(attr,name)* function. This function finds the attribute associated with the name *name*. It then assigns a pointer *attr* to that attribute [7].

The movement of the particles in the particle system is represented by a set of *ParticleBehavior* objects, which are placed in a particular order. Each object represented a different element of the particle's overall behavior. The input from the *ParticleBehavior* object to the behavior the the particles is split up into four steps:

1. *applyForce*() - add this behavior's force to a force accumulator (a per-particle attribute),
2. *applyConstraint*() - alter the force that is currently applied to each particle in order to satisfy a constraint on its motion,
3. *integrate*() - update the particle (e.g. its position, velocity) based on the constrained force held in each particle's force accumulator, and
4. *cleanup*() - create and destroy particles based on population dynamics.

Particle System frameworks like these help programmers adapt varied particle system models to their applications more easily.

A more cross-disciplinary theory involving particle systems is a proposed method for simulating biological systems. Laurier Boulianne, Michel Dumontier, and Warren J. Gross [5] wrote about this theory. They discuss one of the biggest challenges and goals in the field which is the simulation of a biological cell. The two main challenges of this are:

1. the construction of accurate biological models and
2. the development of scalable simulation architectures

A D3Q27 model means that any voxel has access to any of its 27 neighboring voxels, including itself, for motion, interaction, and reaction. Anything existing outside of its 27 neighbors is ignored [5]. The simulator proposed by this paper divides the simulation volume

into a D3Q27 grid model inhabited by voxels. A voxel is a volume element that represents a value on a three-dimensional grid.

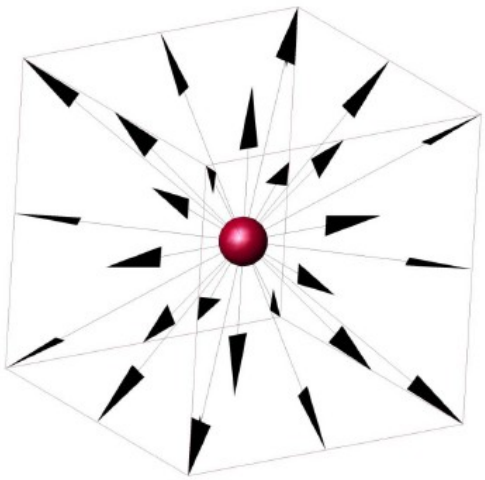

Figure 7. The 27 possible directions of a D3Q27 grid.

Where particle systems come in is that a particle represents macro-molecules such as proteins, small molecules such as ions, inert particles that contribute to molecular crowding or complex structure such as membranes. Particles are able to move only once per turn, in one of the 27 immediately surrounding locations. There is a *moving ratio* which determine a particle's diffusion speed. The probabilistic and random motion of the particles leads them to conform to a Brownian random walk. Brownian motion involves the seemingly random motion of particles in liquid or gas[5].

Conclusion:

Particle systems are tremendously versatile, and programmers are continuing to further explore the full potential of particle systems. It had its beginnings in the simplification of “fuzzy” object modeling and simulations, and found itself expanded into other disciplines such as cell biology. Their use also continues to be popular in game development and special effects for movies. The geometric simplicity and computational ease of using particle systems is what makes particle systems the primary choice for many applications that involve non-deterministic and dynamic shapes in computer graphics. Particle systems continue to evolve into more uses over time.

References

1. Reeves, William T. "Particle Systems - Technique for Modeling a Class of Fuzzy Objects." *ACM Transactions on Graphics* 2.2 (1983): 91-108. http://portal.acm.org/citation.cfm?id=357320 .

2. Smaldon, James, Jonathan Blakes, Natalio Krasnogor, and Doron Lancet. "A Multi-scaled Approach to Artificial Life Simulation With P Systems and Dissipative Particle Dynamics." www.cs.nott.ac.uk/~nxk/PAPERS/Gecco2008a.pdf .

3. Szeliski, Richard, and David Tonnesen. "Surface Modeling with Oriented Particle Systems."*ACM SIGGRAPH Computer Graphics* 26.2 (1992): 185-94. Print.

4. Volino, Pascal, and Nadia Magnenat-Thalmann. "Simple Linear Bending Stiffness in Particle Systems." *Eurographics/ ACM SIGGRAPH Symposium on Computer Animation* (2006). http://dl.acm.org/citation.cfm?id=1218078.

5. Boulianne, Laurier, Michel Dumontier, and Warren J. Gross. “A Stochastic Particle-Based Biological System Simulator.” http://iml.ece.mcgill.ca/GridCell/SCSC07.pdf

6. Drone, Shannon. “Real-Time Particle Systems on the GPU in Dynamic Environments.” *Advanced Real-Time Rendering in 3D Graphics and Games Course – SIGGRAPH 2007*

7. Su, Wen Y. and John C. Hart. “A Programmable Particle System Framework For Shape Modeling.” *SMI '05 Proceedings of the International Conference on Shape Modeling and Applications 2005:* 112-123. http://dl.acm.org/citation.cfm?id=1097876.1098463

8. Reynolds, C. W., "Flocks, Herds, and Schools: A Distributed Behavioral Model," in Computer Graphics, 21(4) (SIGGRAPH '87 Conference Proceedings), 1987, pp. 25-34.