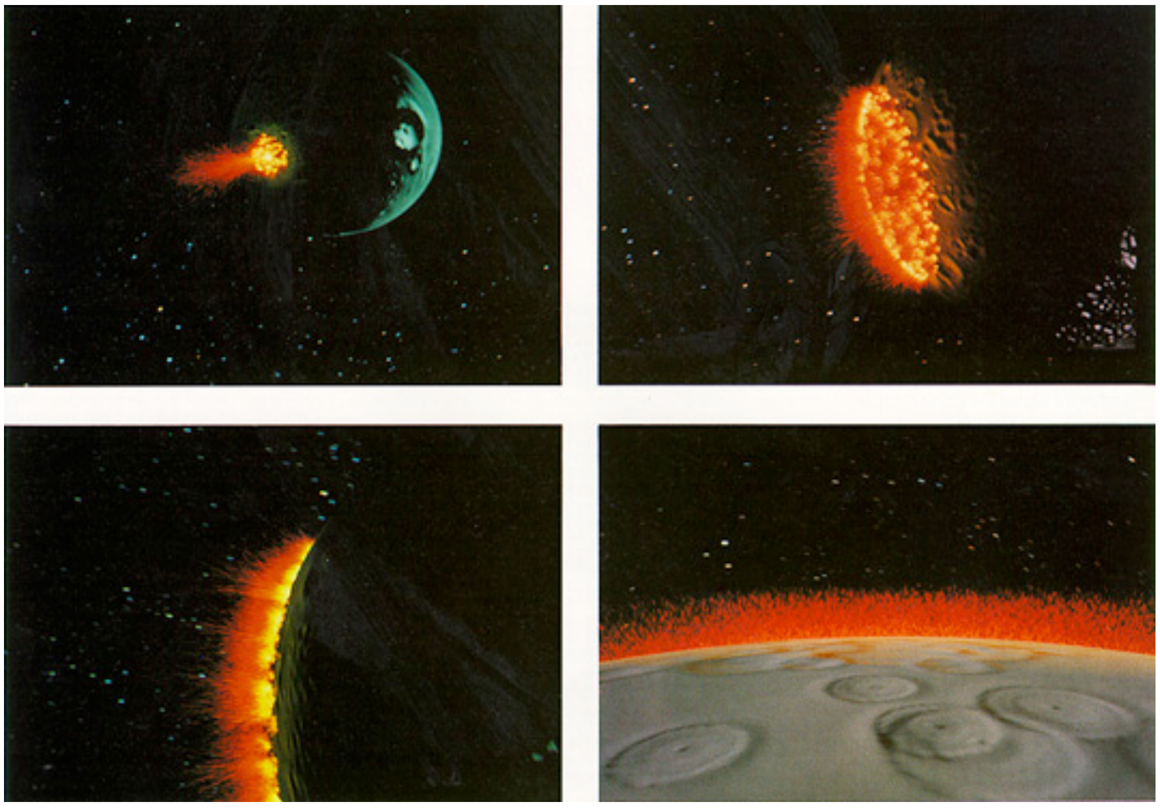
Particle Systems
©Denbigh Starkey

Major points of these notes:



Four scenes from the Genesis Effect in *Wrath of Khan*

1. History of Particle Systems

In 1982 Industrial Light and Magic (ILM) was given the job of producing some graphics sequences for the movie *Wrath of Khan*. This included what now appears to be some very crude vector graphics, a fly through of some mountains that is mainly famous for its disappearing mountain, and a fantastic short sequence showing the use of the Genesis Effect to bring new life to a dead planet.

Bill Reeves was put in charge of the Genesis Effect programming. In this sequence a Genesis torpedo is fired at the dead planet, and when it is ignited above the surface the explosion rolls around the planet, giving off sub-explosions, finally engulfing it, and when it is finished the planet has become habitable.[1] To simulate the explosions Reeves created a new graphics technique called Particle Systems. Subsequently particle systems have been used in many other graphics applications, usually related to natural objects, but in these notes I'll concentrate on the original Genesis Effect system, and briefly mention other applications in the last section.

[1] One might want to ignore the explanation in the movie about how the molecules of the planet surface are rearranged by the Genesis Effect to become life giving.

2. The Genesis Effect Particle System

Reeves' goal was to create a fiery explosion which rolled around the planet and ultimately engulfed it. So the explosion needed some average velocity, and also had to be affected by gravity so that it would move around the planet, and would also stick close to it. Other constraints were that the color should be white hot at the densest part of the explosion, ranging from orange to red as one got closer to the edge of the explosion.

To accomplish this he modeled the explosion as a very large number ($\sim 10^5$) of very simple objects that he called particles, all of which were independently tracked through time. Each particle had a number of associated properties including its location, velocity, mass, color, lifetime, age, and probability of exploding and creating a new particle system. In this first particle system all particles had the same values for these properties, apart from their locations, velocities and ages.

Rendering was easy. All of the particles were projected onto the view plane, as usual, and the sum of the colors of all of the particles that projected onto any pixel were summed to get the pixel color. Each of R, G, and B were independently restricted so that they could not overflow, and so if, for example, the sum of the R, G, and B values for a particular pixel were 1.5, 0.15, and 0.015, then the color displayed would be (1.0, 0.15, 0.015).

Whenever a particle system was created, either through the initial explosion or through a subsequent particle explosion, the average velocity of the new particles would be the average velocity of the creating object (torpedo or particle). Each of the individual particles would blow out in all directions from the object, but the originating object velocity would be added to these velocities.

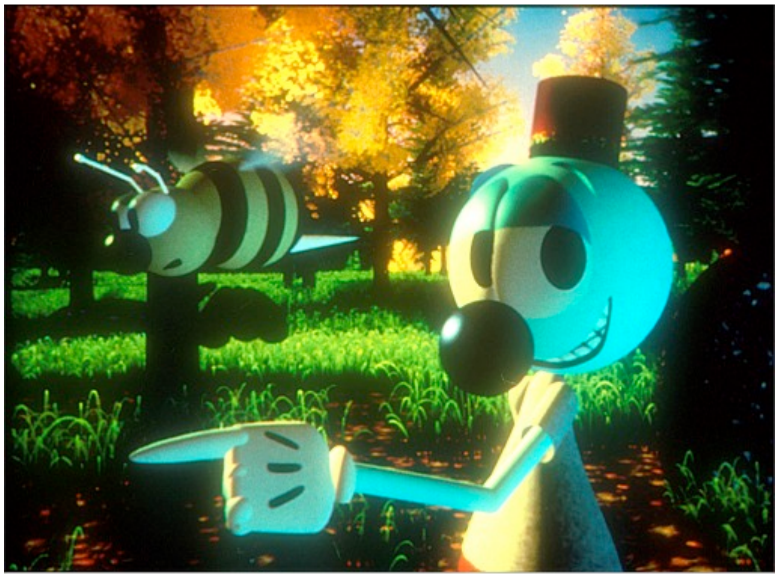
The simulation was done on a time unit basis for each individual particle. Once a time step was complete the particle's new position was computed based on the old position, the velocity, and gravity (which is why they had mass). A random number determined whether the particle would explode and generate a large number of new particles. The particle was then displayed, and if its age had reached its lifetime it was removed from the system. Otherwise it would be ready for the next time unit. This was repeated for every particle in the system.

The particle color was critical for the success of Reeves' system. The color of each particle was low in red, very low in green, and extremely low in blue. E.g., (0.1, 0.01, 0.001) would be possible RGB values. With these values a single particle projecting onto a pixel would not be visible against the black sky. A small group of, say, 10 particles projecting onto a pixel would give the value (1, .1, .01), which is bright red. 100 particles would give a value of (1, 1, 0.1), which is orange. 1000 particles would give a value of (1, 1, 1), which is white. So using these (or similar) colors meant that near the edge the pixels would be red, but as one approached the center of an explosion where there were far more particles the color would change through oranges and yellows to white hot, which is what we wanted to achieve.

3. Other Early Uses of Particle Systems

3.1. The Adventures of Wally B

Reeves next used a particle system in his short movie *The Adventures of Wally B*, (originally named My Breakfast with Andre) a Lassiter film from Lucasfilms. In this movie the trees and the grass were both double particle systems, both in terms of the distribution of the trees and grass but also in terms of the construction of the trees and the individual blades of grass. In the trees, which were more complex than the grass, the branches were a system of line particles and the leaves were a system of circle particles. Instead of the cumulative rendering that was used for the genesis effect, a traditional rendering system was used, which included self shadowing from the particles. There is a description in his paper at W. T. Reeves, "Approximate and Probabilistic Algorithms for Shading and Rendering Structured Particle Systems," *Computer Graphics*, vol. 19, no. 3, pp 313-322, 1985.



3.2. Flocking birds

The other most famous use of particle systems was by Reynolds, who used a particle system to model the behavior of flocking bird objects, or boids as he called them. Boids were complex polygonal objects whose flight behavior depended not only on the individual boid, but also on the behavior of the boid's neighbors. There were three major features to this behavior, which combined to give remarkably realistic flock behavior. Obviously the most critical was collision avoidance where each boid avoids midair collisions with other boids, or with objects in its path in later versions. This alone would not, however, give realistic flocking patterns because it would tend to lead to flocks separating apart. To avoid this tendency to separate, the boids also tried to match the velocity of their neighbors and tried to stay as close as possible to their neighbors without violating the collision constraint. Reynolds also used the system to model fish and other critters, including running dinosaurs. The effectiveness of this model, which is described at C. W. Reynolds, "Flocks, Herds, and Schools: A Distributed Behavioral Model," *Computer Graphics*, vol. 21, no. 4, pp 25-34, 1987, was so good that many now believe that natural flocking could use the same basic principles. For animation demos and other descriptions, as well as his SIGGRAPH demo and a clip from *Stanley and Stella, Breaking the Ice*, see Reynolds' boids page at http://www.red3d.com/cwr/boids/.

4. The Current View of Particle Systems

Any particle system is expected to contain a large number of small objects, but there are disagreements on whether additional constraints are needed before something can be called a particle system. Most CG people add the additional requirements that the particles must be simple objects and that they shouldn't be completely independent but should have some level of interaction. Note that Reeves' system had no interaction between particles (the rendering where there effects are summed is considered a separate stage from the particle system dynamics).

I'll look briefly at three recent (2007) particle systems, the birth of Sandman from Spiderman 3, a tree branch geometry generator, and a water wave simulator. I've chosen these three because they show three very different views of particle systems.

4.1. Birth of Sandman

Spiderman 3 has a number of spectacular graphics effects, but I'll just be looking at the birth of Sandman sequence. Surprisingly the studio assigned different teams to major Sandman sequences, and so, for example, the birth of Sandman sequence shared very little development effort with, for example, the 60' Sandman sequence later in the movie where the director wanted different special effects including sucking in a dump truck and even a kitchen sink, and showing Sandman as far more threatening and also having a much harder time holding himself together. Possibly non-intuitively there are far fewer sand particles in the 60' Sandman sequence than there are in Birth of Sandman.

The particle system in Birth of Sandman is huge, with some images having over 500,000,000 particles. I don't know whether half a billion particles is a record in an animation, but obviously it required that the developers spent a lot of time concentrating on efficiencies. The lead developer says that without this effort they could have just "started the dynamics and come back in a year." In particular they had to use different representations of sand particles based on the camera distance. For example Sandman pushes his arm forward towards the camera to pick up a locket, and as this happens the particle generator changes through three representations of the sand particles from minimal to faked out polygonal structures to hand designed and hand textured particles.

The close up views, with hand designed and textured irregular polygonal sand grains, really aren't a particle system, since full rigid body dynamics (RBD) are used on these grains. However some of the images are mixed, with both the hand drawn grains of sand and the regular particle system being used at different level of detail (LoD) locations, and so the grains must physically interact with the sand particles during collision detection.

The underlying algorithm gives a basic shape that Sandman should currently have and fills that shape with particles. These particles are subject to a number of forces such as gravity, acceleration of the body part, and collisions. in addition, to stop him just falling apart, an external force is applied from outside normal to the surface. Then a "ring" is placed outside the Sandman at a fixed distance above the surface, and any sand particles that escape this ring become subject only to gravity, which gives the effect that Sandman is always shedding sand.

The developers knew that doing the dynamics on half a billion particles was impractical, however efficient they made the particle structures. So they divided the dynamics up into two phases; first they used a relatively small number (64K) of much larger sand particles and used the flow patterns of these particles to generate a vector field for each frame. Then they went to their full particle system model and just had each particle follow the motion specified in the vector field. They called this process *boosting*.

The design of the particles was a major issue. They wanted to use eight-sided polygonal structures (think of two four-sided pyramids with their bases joined) but this took too much computation. They did, however, give very realistic lighting effects as they rotated, and so they wanted a representation that would have the same effect but with much less computation. They did this with objects that they called 4-RiPoints. These are four overlapping coplanar disks with a slight separation which are given fake normals corresponding to the normals on the four sides of the pyramid that would have been visible to the viewer. Visually this worked well, but was still too expensive for the system. So while 4-RiPoints were used for sand particles that were relatively close to the camera most particles were represented by single RiPoints which were each given a normal randomly selected from the 4-RiPoint normals. The determination of which representation to use was dynamic, and so a particle could move between hand-designed, 4-RiPoint, and RiPoint during the animation based on its distance to the camera.

The developers were intending to custom build a sand renderer, but the production schedule led them to just use an existing product (RenderMan).

This system is nice because it not only shows a good example of a particle system, but also gives a feel for what production animators must go through. The production requirements force compromises since the first requirement is to get the sequence complete on schedule, but the director requires that this happen without a loss of visual quality or artistic control.

The final birth of Sandman sequence (about 3 minutes long) is on YouTube at http://www.youtube.com/watch?v=KCV3yUYvV5Q.

4.2. Tree branch structure generator

In this project the researchers (Neubert *et al.*) had still photographs of a scene and needed to build models of individual trees[2]. To do this they needed to first determine the structure of the branches, which was mainly hidden by leaves, and then attach virtual leaves to the generated structure. Over the years there have been a number of different approaches taken to this problem, but these researchers used a particle system.

Their preprocessing approach was to get the user to roughly sketch where they thought that the major branches lay, and these locations became gravitational targets for the particles, along with another target placed at the base of the tree.

Particles were small line segments with narrow width, which were attracted towards gravitational targets, so they initially tended to head towards the closest branch. Once there they tended to want to move down the branch structure towards the base gravitational target. Obviously without any other constraints this would lead to a pile of particles at the base of the tree. So there is an additional rule that if two particles get close together they are joined rigidly together to form a larger particle. In this way branches and the trunk rapidly form in a surprisingly natural looking form, until the tree is a

[2] *Approximate Image-Based Tree Modeling Using Particle Flows*, Neubert, Franken, and Duessen, SIGGRAPH 2007.

single particle. More, however, will finish up in the trunk, and so the effect will be a wider trunk, narrower main branches, and working down in width to the twigs, which at the end are likely to just be from a single original particle.

4.3. Wave generator

My final example of recent work with particle systems is a wave system[3]. The goal of this project was to generate realistic water waves in both constrained (e.g. a swimming pool) and unconstrained (e.g. the ocean) environments. Additional goals were to get real time performance and to be able to interact with objects (e.g. dropping a bunch of boats into the water, and animating the boats).

The system is based on a set of wave particles located in a smooth water environment. Wave particles add a circular height field at their location, where the height of each particle can be positive or negative, representing a wave or a trough respectively. A simple wave is represented as a line of equally high particles. If these particles are too far apart then one will get a blobby wave, but if they overlap by half a radius they are rendered as a cylinder. If waves with different heights pass through the same space then their heights are added together.

A wave can either be generated by the animator, or can be created by an object interacting with the water surface (e.g. dropping a boat in the water and driving it across the surface). These object interaction waves are constructed instantly through approximations of the actual physics, which ignore nasty details like underwater turbulence that actually drive much of the physics of wave generation.

One complication is that the length of waves can increase as in, for example a wave shaped like a semicircle that is advancing. This leads to a potential problem that the separation between particles could become sufficiently large that they revert to a blobby look as compared to the cylindrical effect. To solve this if particles in a single wave object separate too far the system just replaces each affected particle with three equally separated particles, and the problem goes away[4].

[3] *Wave Particles*, Yuksel, House, & Keyser, SIGGRAPH 2007.
[4] Why three instead of two, I have no idea. Also it appears that the system never does the opposite and removes particles if they get sufficiently close – doing that seems an obvious thing to do and would improve efficiency.

The results of this system look very realistic, and can handle, in real time, relatively complex patterns. However it is possible to add too much complexity for real time animations. E.g. ?? shows an animation with thousands of boats thrown into the ocean, and although the boats are all moving and generating waves they are very slow.

5. A Public Domain Particle System

UNC, at http://www.cs.unc.edu/~davemc/Particle/, has a downloadable API where you can try out particle systems. The software is free for non-commercial uses and has been used in some significant projects, including the CMU Alice virtual worlds project. http://www.alice.org. The picture below is from an animation of a fountain from the UNC project page.