Physics based systems – Particle Systems

part of “How to Make a PIXAR movie” seminary, summer term 2009, TUM

Introduction

In the previous sessions we learned about modeling and rendering of primitives with a well-defined surface such as cubes, spheres, triangle meshes, …

Using only those techniques, however it is very hard to render any sort of volumetric data such as fog and fire. Furthermore it is cumbersome to generate the datasets for phenomena such as rain and snow, which consist of many tiny particles. We cannot burden designers to place millions of drops of water in the air just to achieve the impression of rain.

A convenient solution to these problems was first introduced by W. Reeves. He worked on animating and rendering a massive explosion on a planet for the movie “Star Trek II: Wrath of the Khan”. His solution, particle systems, differ from previous techniques in three important ways: Instead of surface polygons the system is defined by a cloud of primitive particles (points in most cases). Secondly the system is not static, but it's form over time is determined by the movement of individual particles over time. And lastly particle systems are stochastic in that way that the designer only controls overall parameters but not properties of single particles. These gaps in the designer's specification are then filled through a stochastic process.

Basic: a Particle

To model a cloud of particles, we need certain properties for each particle. Let's imagine the particles as tiny balls. These balls will need a current position in 3D-space as well as their current movement vector. The movement needs to be stored if we want to accelerate the particle later (by means of some sort of force). Most applications of particle systems involve some sort of acceleration.

We will also add a few properties that control the look of each particle such as shape, size and transparency to gain flexibility in the rendering process.

In many applications we also have properties that influence the stochastic process. If you think of an explosion as a cloud of particles acceleration outwards, each particle will change over time from the bright glowing fiery one that is zooming past the viewer, to the slow moving smoke particle at the edge of the explosion that gradually becomes invisible. To achieve something like this, properties like particle age can be useful.

In practice you might even introduce more advanced properties, but the above will be sufficient for many applications.

Animation step: The Engine

The engine is the place where animation and physics are going to happen. For a basic particle systems the most important formulas are the basic laws of motion:

$$\vec{v} := \vec{v}_0 + \int \vec{a}\, dt$$
$$\vec{p} := \vec{p}_. + \int \vec{v}\, dt$$

Evaluating these integrals is no option for most applications. Assuming constant acceleration $\vec{a}$ we can simplify them:

$$\vec{v} := \vec{v}_0 + \vec{a} \cdot \Delta t$$
$$\vec{p} := \vec{p}_0 + \frac{\vec{v}_0 + \vec{v}}{2} \cdot \Delta t$$

The acceleration itself can be calculated from the sum of all forces divided by particle mass. These forces can be very simple such as gravitation, but you can also come across very complex forces in some applications.

The only basic piece still missing is where the particles come from. This is the job of the particle emitters. An emitter is the center of the stochastic process: It randomly creates particles and assigns them starting properties such as speed and color. The start position varies with different particle systems but it can usually be described with a static distribution function in 3D space. Thinking of a burning piece of wood, the particles could come from the glowing surface with an initial movement away from the surface.

Rendering step

Now that we have an animated set of points in 3D-space we need to bring them to the screen.

Easiest would be to render single pixels but the main problem is that to create a fire covering a large part of the screen you would have to simulate millions of particles witch is not (yet) feasible.

But there is a simple trick to substitute this sort of complexity. Instead of rendering single pixels we use semitransparent textures. These textures are always rendered parallel to the viewing pane, a technique called “Billboarding”. Using a few different textures we can cut the particle count massively without losing credibility for the overall scene.

With these basic design principles we are equipped to build a fairly large range of particle systems. In practice there are still quite a few problems including interaction with illumination (fire adds light, smoke would add shadows) and the combination of different particle systems.

Beyond fire and smoke

Particle systems are not only used for gaseous phenomena, but they also have applications in other areas. One example is real-time fluid physics. This application is an example, where each particle is not only influenced by external forces but also by neighboring particles. Simulating water, we would add forces that counteract differences in local pressure. A force

attracting particles nearby emulates surface tension, while another force drives particles away from areas with elevated particle density, equalizing pressure in the fluid. Such a simplistic approach is far less accurate, than traditional Navier Stokes solvers, but it is possible to run it in real-time. Particle systems with this particle-particle-interaction usually are a lot more complex to simulate efficiently than others. However, although this approach is not too realistic, it does generate very credible animation in practice.

To render particle-water we also have to change our rendering process. Billboarding, as used previously does not help here. The way water looks is mostly determined not by its volume but by its surface. To calculate a surface polygon from a set of points, you will often start with a density function that calculates local density depending on how many particles are nearby. This function can then be used with a technique called marching cubes that puts a 3D grid into the system and determines the surface by sampling the density changes between cells of this grid.

Apart from the non-scientific uses described, there are also applications in science, most notably visualization of 3D datasets. For example the vector fields generated by a Navier Stokes Simulation are hard to grasp just looking at the numbers. One way to understand the movement is to use these vectors as a forcefield for a particle system. This enables the user to “see” the whirls that result in flow resistance.

Conclusion

Particle systems are an important instrument in modern simulation and visualization:

They

- enable fast rendering of volumetric data.
- have a vast variety of applications.
- reduce modeling work through their stochastic process.
- are also useful for scientific applications.

References & further material

- [REEV83] W. T. Reeves, Particle systems - a technique for modeling a class of fuzzy objects. ACM Transactions on Graphics 2(2), pages 91–108, 1983.
- Alan Watt, Mark Watt, Advanced animation and rendering techniques, ACM, 1991
- Matthias Müller, Particle-Based Fluid Simulation for Interactive Applications, SIGGRAPH Symposium on Computer Animation, 2003

The engine shown during the session can be downloaded from
http://www.kernelz.de/projects/particle-system/
Java™ source files will be available, too.