The Second Order Particle System

Tommi Ilmonen
Helsinki University of Technology
Telecommunications Software and Multimedia Laboratory
P.O.Box 5400
Fin-02015 HUT, Finland
Tommi.Ilmonen@hut.fi

Janne Kontkanen
Helsinki University of Technology
Telecommunications Software and Multimedia Laboratory
P.O.Box 5400
Fin-02015 HUT, Finland
Janne.Kontkanen@hut.fi

ABSTRACT

In this paper we present an extension to the classical particle system. We unify particles, particle sources, and force generators into a second order particle system. In the second order particle system the particle sources and force generators are subject to the forces as well as visual particles. This fundamental change, along with suitable set of force classes, enables us to create better real-time simulations of fire, smoke, clouds, and explosions. We use hierarchical spatial subdivision to reduce the computational workload.

Keywords

Particle systems, physical modeling, physically based animation, fluid dynamics

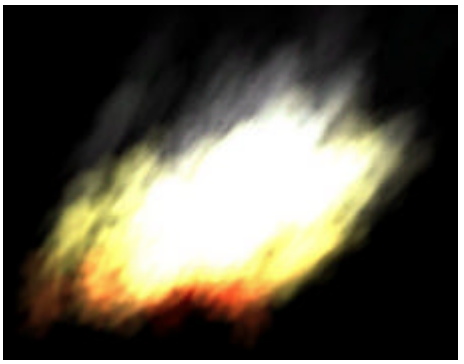
INTRODUCTION

Particle systems have been used for computer graphics for decades. During this time minor changes have been made to the fundamental paradigm laid out by Reeves [Ree83]. The classical particle system has particle sources, particles, and forces that affect the particles.

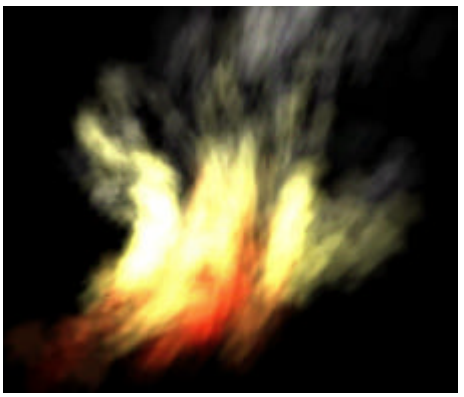
Traditional particle systems have several problems due to their overall architecture. Animations made with classical particle systems tend to be rather static – the particles fly around the system, but since the forces are stationary the trajectories of the particles do not change. Typically, this results to animations that look artificial. It is possible to build quite convincing particle effects using the traditional method, but this requires special purpose software, a lot of manual tuning, and a sufficient amount of artistic insight to overcome the limitations of the classical particle system. On the contrary, our approach generates procedural perceptually valid non-static force fields and the animator does not need to animate the behavior of each force manually.

Typically particle systems do not have a sufficient number of forces to create lively, organic animations. This problem is demonstrated in figure 1(a) that has valid forces acting on the particles, but only in the macro level. One could make the animation more realistic by adding vortices, for example, but they would have to be animated since a stationary vortex would not look realistic.

Journal of WSCG, Vol.11, No.1., ISSN 1213-6972
WSCG'2003, February 3-7, 2003, Plzen, Czech Republic.



(a) Fire with traditional particle system (88 fps)



(b) Fire effect with the second order particle system (70 fps).

Figure 1. Difference between fires created with the classical- and second order particle systems.

Completely different approach would be to use precalculated animations in place of real-time particle effects. However, this method has severe drawbacks: compromised resolution, lack of the third dimension and the inability to react to an interactive environment.

The second order particle system solves the problems discussed above. By moving the force particles with other particles we can make the trajectories time-variant in an aesthetically pleasing way.

The application areas for the second order particle systems are the same as for the classical particle systems: movie production, computer games and interactive multimedia installations. In this paper we will concentrate on systems that run in real-time with frame rate greater than 20 frames per second.

This paper is organized as follows. First we examine previous work on particle systems and fluid dynamics. Then we describe the features of the second order particle system in general. After that we show examples of how this system can be used. In addition, we also describe the earlier methods that have been used to create such effects. Finally, we describe our implementation and algorithms and give some ideas for future research.

BACKGROUND

The basic building blocks of particle systems are well known and there are articles describing how one should build a real-time particle system [Wit97] [Lan98] [McA00] [Bur00]. Particle systems are a popular tool for animation both in interactive graphics [Bur00] and in off-line graphics [Ree85]. Thus most animation applications have direct support for particle systems (including Alias|Wavefront Maya, LightWave and others). Wejchert has shown that it is important to use moving force fields when the aim is to create convincing animations [Wej91]. Sims has given comprehensive overview of the traditional particle systems [Sim90].

Particle systems have been used to create flock behavior [Rey87] and structured systems [Ree85]. O'Brien has shown a method for improving the performance of dynamics calculation [OBr01] by grouping particles to reduce the workload of force calculations.

Particle systems are often used to create animations relying on fluid dynamics that can be modeled by using simplified versions of Navier-Stokes equation. Nguyen has demonstrated fire simulation [Ngu02]. Yngve has modeled explosions [Yng00] and Foster has animated smoke and liquids [Fos97] [Fos01]. These studies show the results of using physical modeling to simulate natural phenomena. The animations are often very realistic, but the calculations are much too slow for real-time usage. In addition, the methods also use a lot of memory, preventing large-scale simulations.

THE SECOND ORDER PARTICLE SYSTEM

In this section we describe the features of the second order particle system. The system is a composition of novel ideas and more established ones.

The novel feature of the second order particle system is that forces and particle sources are also affected by the forces in the system. This fundamental change enables systems that are much more dynamic than the classical particle systems. In the second order particle system the force generators are special particles – they can be created and deleted like ordinary particles but they have the capability to affect the trajectories of other particles. In figure 1(b) we have added dynamic micro-level force generators by creating a number of small, moving vortices.

We have named our approach the second order particle system since this name reflects the relation to traditional – or first order – particle systems. In first order particle systems forces move only the visual particles. In the second order particle system the particles, the particle creators, and the force generators are all affected by the forces acting in the system.

First we specify terms that will be used. *A system* stands for "the second order particle system". A *particle* is a point object with no volume or internal structure. Visual particles are a subclass of particles that can be represented by the rendering back-end. Particles can have common properties such as area and mass. Each particle belongs to one system. A particle can have arbitrary internal logic to specify how it reacts to forces and what other behavior it has (this includes the potential to create new particles to the system and to delete existing particles). *Force generators* are special particles that exert force that moves other particles. Forces generators – being particles – are naturally affected by forces as well. *Force category* implies how the force affects the particles and *force class* specifies the behavior (or force field) of the force. Each force generator can only belong to one category (drag, gravity, electrical, magnetic etc.) but several *force classes* (vortex, blow, wind etc.) can belong to one category.

The most profound new idea in the second order particle system is that forces affect other forces. This feature enables completely new kinds of effects. It also reflects the way several natural phenomena work: For example, a powerful vortex causes new smaller vortices to appear in its edges and the smaller swirls move with flow of the liquid. In the traditional particle system the minor swirls would either be

fixed or they would need to be moved by the animator. In the second order particle system the forces move with the system, leading to more dynamic, realistic, and aesthetically pleasing motion.

By dividing forces to different categories we can get better control over the animation. Category metaphor mimics the different physical forces: gravity, electrical interaction, magnetic interaction etc. By using different force categories we can apply forces selectively. For example, we can use this feature to make fire particles float upwards with wind (particles with negative weight and large area) while the rain particles fall rapidly to the ground (heavy particles with small area).

The forces can affect particles in any way. Since we cannot foresee all the necessary force types it must be possible to create new force classes when necessary.

There are situations where particles need to access other particles in the system. This is necessary since the behavior of the particle may depend on how many particles surround it and what kind of particles they are. For example a vortex creator might emit more vortex-particles when there are a lot of ordinary particles around it. Thus particles should have access to other particles in the system.

EXAMPLE APPLICATIONS

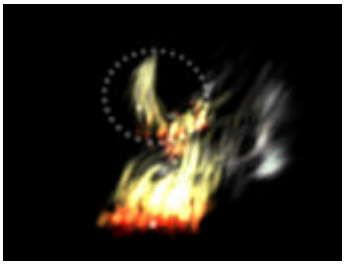
We have created a number of particle systems that display how the second order particle system can create animations not possible with classical particle systems. These examples show how the features described above relate to realistic animation tasks and also show results we have obtained with our implementation of the second order particle system.

Fire

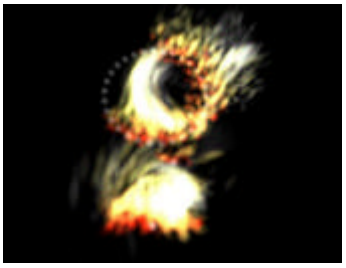
The first example is fire simulation. Fire has proven to be difficult to model accurately although there has been progress in creating visually plausible fire animations. Nguyen has described how to create highly realistic fire [Ngu02]. Lamorlette has published a way to control the behavior of physically based fire [Lam02]. However, both Nguyen's and Lamorlette's systems are too slow to be used in real-time applications.

In real-time applications fire has typically been created with a particle system or an animated billboard. The animated billboard may result in very realistic fire if one films real fire and uses that as base material. However, this kind of fire cannot react to the environment and its 2D-properties are painfully clear when viewed closely. Additionally, the video material may consume a lot of memory.

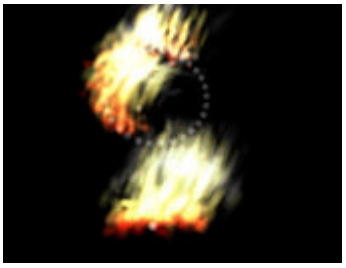
Typical fire effect that has been done with a traditional particle system is illustrated in figure 1(a). The largest problem with this and similar fire effects is that they are not dynamic – flame particles originate from some area and simply fly to the given direction. In figure 1(b) we have added vortices to the system. Vortices make the fire much more organic and one can see effects that take place in natural fire – uneven distribution of the flames, more random motion, and clustering of flames. The vortices are added procedurally and they move with the flames to make the animation look credible. While the effect is still distinguishable from real fire it is a significantly better-looking approximation than what one would get with traditional particle systems. This system demonstrates the effects we can create with moving, dynamic force objects. The forces need to interact with each other because it makes the animation more organic.



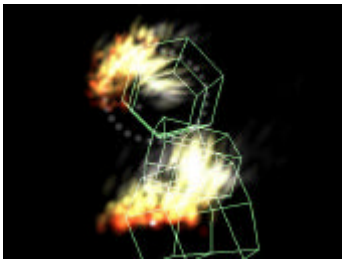
T = 4s. The fire below the ring has heated the coals at the bottom and part of the ring is burning.



T = 8s. The fire is spreading in the ring to the left and up.



T = 16s. The coals that were lit first have consumed their energy and are not longer burning.



T = 24s. Only remnants of coals remain burning. The vortices that make the animations more complex are shown as cylinders.

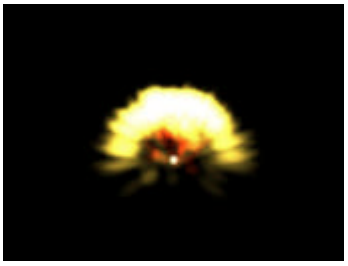
Figure 2. Ring of fire. The last frame shows the vortices as cylinders (70 fps).

The fire in figure 2 illustrates the use of particles that react to their environment. We use the basic fire effect from the previous example as the starting point. In last frame of figure 2 we have made the

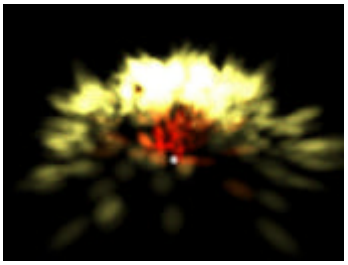
moving vortices visible (each vortex is represented by a cylinder). Then we create a ring of coal particles. The coals are initially not burning, but as the fire heats them they start to burn gradually and they burn until their fuel is consumed. In this case the coals are ignited as their internal temperature rises above a given threshold. The internal temperature follows the air temperature with some delay. The air temperature is approximated by calculating the number of fire particles in the proximity of the coal. This example shows a situation where it is necessary for the particles (in this case coals) to have access to the other objects in the system (in this case to be able to count them).

Explosions

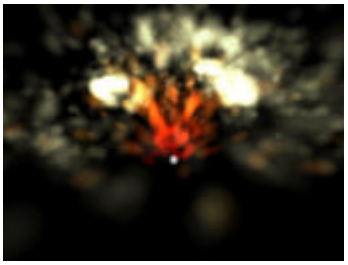
Explosions are another popular topic in computer graphics. Physical modeling of explosions for computer graphics has been done by Yngve with convincing results [Yng00]. Explosions can be presented with classical particle systems but – as with fire – the explosions can look predictable and simple. In figure 3 we have created an explosion that is composed of several elements.



T = 0.3s. A repelling force field pushes the particles away from the center of the explosion.



T = 0.6s. The repelling force field gets weaker and the vortices begin to cluster the particles



T = 1.3s. Most of the original particles have flown away, but a few clusters remain.

Figure 3. Complex explosion with thousands of particles and over 20 forces (25-40 fps)

A large number of particles are created at the center of the explosion with large random velocity. The particles have slightly randomized mass and area to make their trajectories less uniform. An explosive force pushes all particles away from the center of the explosion. Along with the explosion we send a group of vortices to the system to create more variation. After the explosion we add a small set of particles to the center of the explosion to mimic the remnants of the explosive matter. The explosions are configurable and we can create explosions ranging from simple expanding particle clouds to more complex explosions with turbulences and disintegrating particles.

Galaxy

Our third example shows a galaxy (figure 4).

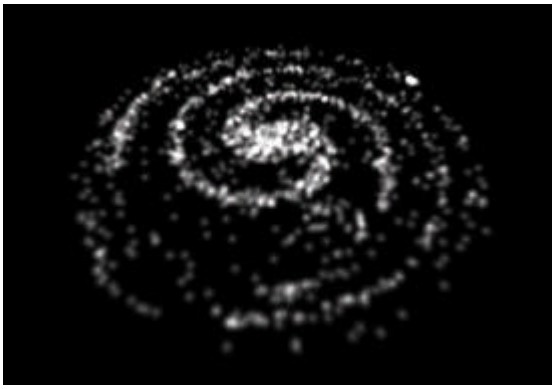


Figure 4. Galaxy (220 fps).

With slightly different rendering style this system can represent the motion of dust particles in a circular chamber. We have placed star particles into the system and we use a vortex to move the particles. We also create vortices to the edge of the main vortex. The smaller vortices create more variation to the system by disturbing the motion of the stars. Together these forces create motion that mimics the behavior of the edge vortices in fluid motion. This example demonstrates how we can mimic the effects of physics if we know how they work and what is their visible outcome. The structure of the force fields is shown in figure 5.

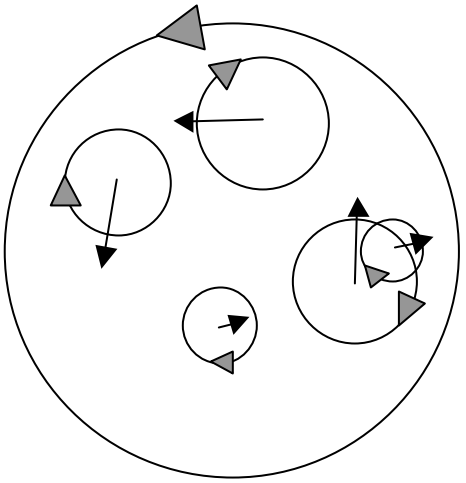


Figure 5. The forces in the galaxy.

Smoke

Figure 6 shows smoke coming from a factory chimney. We have attached three particle creators to the top of the chimney. One source creates smoke particles, another large vortices, and the third small vortices.

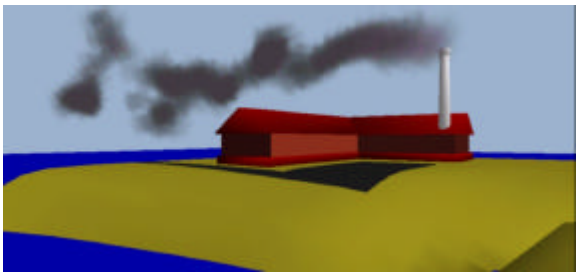


Figure 6. Smoke (160 fps).

IMPLEMENTATION

We have implemented a version of the second order particle system in C++. This implementation covers all the features that were described in the previous examples. Our system also proves that the features can be implemented with high computational efficiency.

Our implementation is a group of libraries that together form The Visual Effects Engine – VEE. These libraries provide a modular framework for developing particle systems.

We have used two force categories in our implementation: drag and gravity. Drag represents the force that depends on the size of the particle. Wind, vortex, and explosion are force classes that are intuitively understood as affecting the particle via drag. The other force category is gravity. The mass of the object specifies how strongly it reacts to gravity. These two categories do not cover all force types of real physics; instead we are seeking the smallest possible set of force categories that offer sufficient flexibility. The names of the categories could be arbitrary, but we have selected intuitive names "drag" and "gravity".

As the force generators are created procedurally it is easy to create a great number of them. This can lead to performance problems since calculating the dynamics takes more time as the number of forces increases. To counter this problem we have used octree-optimization [Gla84]: The scene is split hierarchically to boxes and the hierarchy is updated progressively (as was done by O'Brien [OBr01]). Since most of the forces have limited bounding box we can avoid calculating the forces that could not affect particles that are in another box. Whether this optimization does in fact increase performance depends entirely on the system. In some cases (with few bounded forces) the overhead of updating the octree may slow down the system by 40% whereas with systems that have lots of small-volume forces we can triple the frame rate (for example figures 4 and 6). The parameters (splitting- and collapsing thresholds and maximum recursion limit) of the octree division can be adjusted in run-time to optimize the performance.

Description of the Algorithm

The following steps describe the high-level control flow of the second order particle system:

1. Update the velocities of the particles
2. Update the particles
3. Move particles to the correct octree cells
4. Update the spatial hierarchy
5. Register force generators to proper cells
6. Render the particles

First, we update the velocities of each individual particle, including force and particle generators. For each particle the resultant acceleration is calculated by accumulating the impulses exerted into the particle by all the force generators in the current cell. In the second step we update the positions of the particles according to the velocities that were updated in the first step. In this step additional effects related to aging of particles may take place. For example, particles may change color, they may die or they may cause birth of new particles.

Next three steps (3, 4, and 5) deal with the spatial subdivision. Third stage verifies that each particle is in the correct octree cell. If not, the particle is re-inserted into the octree. In fourth step the topology of the octree is updated, so that the over-populated cells are split into eight sub-cells and under-populated ones are eliminated. When a leaf-cell is deleted the particles in the cell are pushed into the parent-cell. Correspondingly, when a cell is split into sub-cells, the particles are pushed into the leaf-cells. In the fifth step the force generators are registered to the octree. Each force generator is inserted to the cells that intersect or contain the bounding box of the force field.

Finally the particles are rendered. Various parameters may affect the visual appearance of each particle, but in general the rendering module is independent of the particle simulation system.

Rendering Back-ends

The particle system core is completely independent of the rendering method. We have created an OpenGL rendering engine that is used for real-time operation and a Renderman-file generator. The OpenGL engine can be configured in run time to use different textures, image fading, and alpha operations.

The Renderman-file generator is applied to connect VEE to external Renderman-compliant renderers. This engine outputs the particle system in Renderman Interface Bytestream (RIB) format. We have used the Blue Moon Rendering Tools (BMRT) renderer with custom Renderman shaders for this purpose. Since we can use more time in the off-line mode it is possible to have more particles and forces than in a real-time system. The real-time system can be used for interactive design with fewer particles and the final system can be rendered off-line with higher quality.

Scripting Interface

To enable rapid particle system development we have created bindings for the Python[1] programming language. Thus one can build, configure, and test particle systems interactively within the Python interpreter. This feature is practical since tuning the parameters of the particle systems takes a number of iterations. An interactive test environment helps this work.

PERFORMANCE

All of the figures have been rendered with OpenGL on a normal PC (1,5 GHz processor and NVidia GeForce4 graphics card). The captions include the rendering speed in average frames per second (fps). These test cases have been updated with maximum speed and new frames have been rendered even if the previous frame had not been shown yet. Therefore the fps count may exceed the display update frequency. These performance figures cannot be directly compared against other particle engines since all engines have a different set of functionality, flexibility, and performance. They are intended to show that our approach is useful for real-time applications.

Compared to physical modeling our approach is faster by a magnitude or two. Where rendering physically modeled fire takes seconds per frame, animations of comparable complexity (but lesser realism) can be created with the second order particle system running fast enough for interactive applications (at least 25 frames per second). The performance might be improved further by using O'Brien's dynamics simplification approach [OBr01].

CONCLUSIONS AND FUTURE WORK

We have presented a novel approach to computer animation – the second order particle system. This method is based on the traditional particle systems.

Second order particle systems can be used for simulation of fire, smoke, and other natural phenomena. Although the animations do not rival the results of physical modeling in realism this approach is computationally much more efficient, making it useful for real-time systems. The lack of physical limitations may also be a benefit when the animation should not follow physical laws. The second order particle system is not by definition a tool that must produce physically valid results but with careful selection of forces and behavioral rules it is possible to create particle systems that look like real physical fluid (or mechanics) systems.

Even though the second order particle system does not simulate the physical world it has the potential to create animations that look organic and credible (just like the special effects used by Hollywood look credible and "realistic" although they are often extremely unrealistic). By introducing force particles and programmable particles we have made the system much more flexible and capable of new kinds of animation.

We have implemented the second order particle system with C++ and it is enough to be used in real-time graphics applications with modern hardware. Our approach suits also for off-line animation (i.e. movie production). The flexibility of our approach makes it computationally heavier than a traditional particle system, but the second order systems can create animations that are much more organic than animations with traditional particle systems.

In the future we plan to make the real-time renderings better looking by using vertex- and fragment programs (i.e. shaders). Other interesting topics are the creation of effects that are expressive rather than realistic and finding better and more flexible ways to control and create particle systems, possibly with a new control language. We also plan to test how the system would work with view-dependent simplification; one could decrease the number of particles and forces when the particle system is far from the camera to improve performance.

The octree division would be useful in culling parts of the particle system in the rendering phase. Since the octree division is done in any case we only need to test the visibility of each octree-cell against the view frustum.

ACKNOWLEDGEMENTS

This work has been funded by the Helsinki Graduate School on Computer Science and Engineering of the Academy of Finland.

[1] Python is an object-oriented scripting language, see http://www.python.org

REFERENCES

[Bur00] Burg, J. v.a., Building an Advanced Particle System, Game Developer, pages 44-50, March 2000

[Fos97] Foster N., Metaxas D., Modeling the motion of a hot, turbulent gas, Proceedings of the 24th annual conference on Computer graphics and interactive techniques, pages 181-188, 1997

[Fos01] Foster N., Fedkiw R., Practical animation of liquids, Proceedings of the 28th annual conference on Computer graphics and interactive techniques, pages 23-30, 2001

[Gla84] Glassner A., Space subdivision for fast ray tracing. IEEE Computer Graphics and Applications. 4(10), pages 15-22, 1984

[Lam02] Lamorlette A., Foster N., Structural modeling of flames for a production environment, Proceedings of the 29th annual conference on Computer graphics and interactive techniques, pages 729-735, 2002

[Lan98] Lander J., Ocean Spray in Your Face, Game Developer, pages 13-9, July 1998

[McA00] McAllister, D., The Design of an API for Particle Systems, Technical report, University of North Carolina, 2000

[Ngu02] Nguyen D. Q., Fedwik R., Jensen H. W., Physically based modeling and animation of fire, Proceedings of the 29th annual conference on Computer graphics and interactive techniques, pages 721-728

[OBr01] O'Brien D., Fisher S., and Lin M., Automatic simplification of particle system dynamics, Computer Animation, pages 210-218, 2001.

[Ree83] Reeves W. T., Particle Systems - A Technique for Modeling a Class of Fuzzy Objects, Computer Graphics 17:3 pp. 359-376, 1983

[Ree85] Reeves W., Blau R., Approximate and probabilistic algorithms for shading and rendering structured particle systems, Approximate and probabilistic algorithms for shading and rendering structured particle systems, pages 313-322, 1985

[Rey87] Reynolds G., Flocks herds and schools: A distributed behavioral model, Proceedings of the 14th annual conference on Computer graphics and interactive techniques, pages 25-34, 1987

[Sim90] Sims K. Particle Animation and Rendering Using Data Parallel Computation, Proceedings of the 17th annual conference on Computer Graphics and Interactive Techniques, pages 405-413, 1990

[Wej91] Wejchert J., Haumann D., Animating Aerodynamics, SIGGRAPH Computer Graphics pages 19-22, 1991

[Wit97] Witkin A., Physically Based Modeling: Principles and Practice – Particle Systems, SIGGRAPH-97 course material, 1997

[Yng00] Yngve D., O'Brien J. F., Hodgins J. K., Animating explosions, Proceedings of the 27th annual conference on Computer graphics and interactive techniques, pages 29-36, 2000