REALTIME PARTICLE SYSTEM SIMULATION AND RENDERING IN EMBEDDED SYSTEMS

Jens Ogniewski, Ingemar Ragnemalm
Information Coding Group, Linköpings University
581 83 LINKÖPING, Sweden
{jenso||ingis}@isy.liu.se

ABSTRACT

The market for games for mobile phones/tablets is probably the fastest growing in the whole computer game industry. Although many of these games feature graphics which were out of reach for these systems not long ago, their quality is still far from what can be reached on modern PCs, and many algorithms used in PCs are a bad fit for these systems. For example, volumetric particle systems are very difficult to simulate and render in realtime on modern smartphones/tablets. This paper presents the first work on particle system simulation and rendering on embedded systems in realtime. This was achieved by approximating volumetric systems by 2D-systems and by using a novel, physically motivated yet simple particle motion model instead of a computational complex solver for e.g. Navier-Stokes equations.

KEYWORDS

Particle effects, Embedded Systems, Real-time, Computer Games

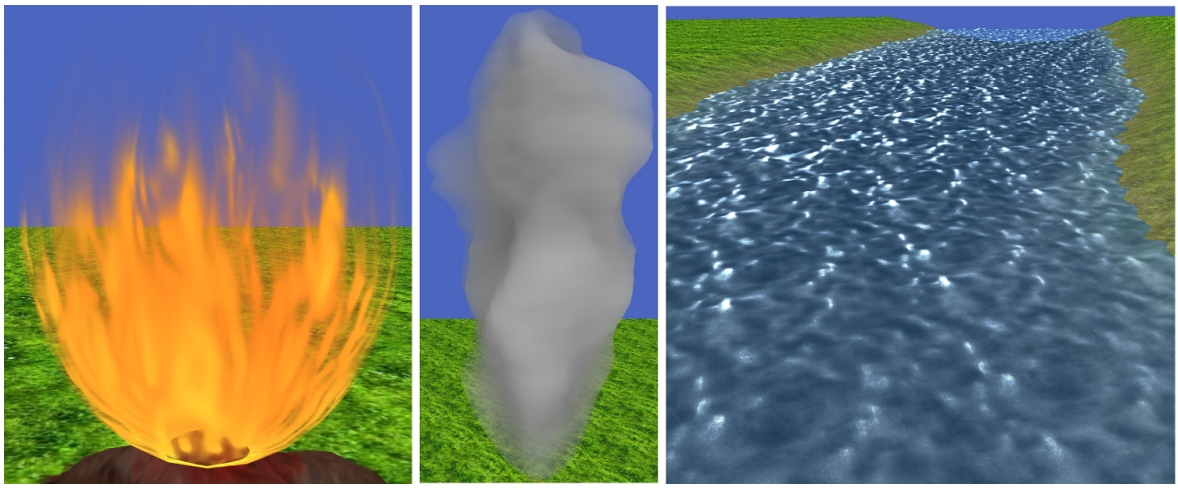

Figure 1: Example particle effects taken from the demo: 1a (left) fire, 1b (middle) smoke and 1c (right) water

1. INTRODUCTION

The visualization of particle effects has been a topic of much interest since the beginnings of computer graphics, and many approaches have been presented, most of which use volumetric particle systems, e.g. (Wrenninge et al., 2010). These are based on particle movement in a so called voxel grid, which is a discretized, closed space (realized e.g. by 3D-textures). However, these grids use a very large amount of memory, and thus several approaches have been suggested for their effective compression like octrees (Laine & Karras, 2011), trading lower memory footprint for higher runtime. To render the particles, different methods are used depending on the exact effect that should be simulated, e.g. Marching Cubes (Lorensen & Cline, 1987) for liquids or light transport for smoke/clouds , e.g. (Hadwiger et al., 2009).

Most of the recent work concentrates on reducing the runtime, for example by introducing precomputing like Kun Zhou et al. (2008) or Yubo Zhang et al. (2012), or by modification of the grid, e.g. (Horvath & Geiger, 2009) or (Selleg et al., 2005). These approaches can run in realtime on contemporary PCs, however not on embedded systems, since modern smartphones/tablets have a different architecture. Here, the CPU and the GPU, but also all other integrated systems like communication, I/O etc. share the same memory and the same bus, which therefore become a bottleneck. This has however big advantages in energy savings and cost, and is thus unlikely to change. The GPUs are highly optimized towards size and energy consumption, but often have to render to high resolution screens. This means that graphic algorithms have to be carefully optimized for runtime, but especially for memory consumption. For particle effects, most designers use so called particle systems, which in these cases mean several (often animated) billboards moving in predetermined or partly-random patterns, as described in e.g. (Harris & Lastra, 2001). A few papers have presented volumetric rendering for embedded displays, like Moser & Weiskopf 82008) or Rodriguez & Alcocer (2012), omitting however the simulation part needed for animated particle effects. Furthermore, the reported frame-rate of typically only a few frames per second is too low for real-time applications like games, and these works do not include popular effects, like lightning or advection (a method to introduce small-scale details through random noise which is offset by a turbulence field, see e.g. (Neyret, 2003) or (Qizhi Yu, 2011). Finally, Krüger & Westermann (2005) and in Guay et al. (2011) suggested the use of a 2-dimensional approximation to emulate a full volumetric system. This is based on the observation that a fairly good result can be achieved solely by knowing how many particles any possible ray from the observer through the particle system would hit. Due to its low complexity, this is the approach we use here as well. We further optimize their work by using a physically motivated model with much less complexity instead of a Navier-Stokes solver as applied by them. Furthermore, Krüger & Westermann (2005) use a 2-dimensional flow field, but move the particles in 3D, and Guay et al. (2011) introduce a fake depth during the simulation, while here no depth is used at all during simulation, thus reducing simulation time. Also, Guay et al. (2011) only described fire and it is unclear if their approach can be used for other effects as well, while we present here different particle effects proving the versatility of our approach.

To the best of our knowledge this is the first work where particle effects where generated on embedded systems by the simulation of particle movements.

2. 2D PARTICLE SIMULATION

As already pointed out, memory usage should be minimized as much as possible in embedded systems. Simulating the particle movement in 2D leads to a low memory footprint since only two 2D arrays are needed to save the data, one for the particles and one for the pressure field. We suggest here to remove the pressure field as well, so that the whole system uses only one single array, which will be called a *particle-field* in the following. Each cell of the field can contain 0 to 255 particles, which was chosen so that the whole field can be saved in a single color-channel of a texture. This also means that the simulation can be done in one single step, instead of integrating the pressure field first and then moving the particles accordingly in a second step. However, since no pressure field exist the particles have to be moved in a different way, and for that we chose a force-based approach in this work.

In the real world, the movement of particles are governed by a number of different forces. The most prominent include inertia (i.e. along the current trajectory), diffusion (from places where a lot of particles reside to places where fewer particles are), and external forces (e.g. gravity). Apart from these 3 forces, we added a random force as well to emulate other and small-scale effects. We found that this approach has the additional advantage that choosing the blend-weights of these forces helps to control the simulation and thus makes it easy for the designer to create the desired effect.

Of course, each simulated particle represents a high number of real particles. Thus, the movement of each particle in the simulated system can be seen as the average movement of all particles it presents. Therefore, a more accurate simulation would be received if several particles, that travel along the same movement vector, would be allowed to travel in slightly different paths. This could be described by e.g. a gauss distribution. Here, we suggest to use a cosine function instead as an approximation, since all forces (except for diffusion) can be represented by vectors. Thus, the dot-product between the force vector and a candidate direction can be used for the force-calculation.

This has the additional advantages that it can be computed fast in GPUs and that it guarantees that particles will be moved even if the force vectors do not align exactly with any of the candidate directions.

For the diffusion, simply the difference is used between the number of particles in the starting cell of the candidate direction and the number contained in the finishing cell of the candidate direction.

To simplify calculations, all particles are allowed to move only to neighboring cells. Also, during simulation the forces between a cell and its 8 neighbors are calculated only once for the each cell, not for each particle that the cell contains. This also means that an average direction vector is used for the inertia calculation, which can be saved comfortably in 2 color-channels of the particle field texture.

Since the forces are calculated separately for each direction, particles will be moved to a number of different neighbors, based on the strength of the calculated forces. In traditional approaches however only few directions would be used, and therefore the particles spread out more quickly using our approach.

An overview of the different forces and the resulting movements is given in figure 2b-2f. The particle field used has a size of 32x32. 2a shows the initial state, the others the system after 15 simulation steps.

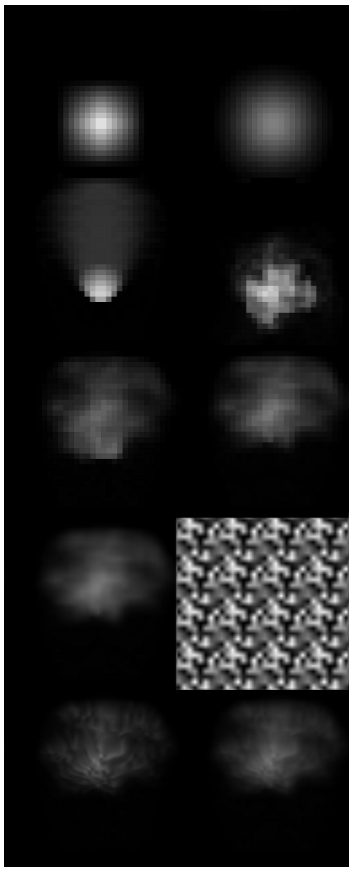

Figure 2: Example to illustrate the different steps of the suggested approach:
1st row: 2a (left) input particle field, 2b (right) movement according to entropy,
2nd row: 2c (left) movement along a common direction, 2d (right) random movement,
3rd row, 2e (left) all three forces combined, 2f (right) with additional inertia,
4th row, 2g (left) 2f drawn using linear interpolation, 2h (right) noise in a 8x8 texture repeated 4x4 times,
5th row, 2i (left) calculated advection, 2j (right) 2g and 2i blended together (1:1)

3. EVALUATION

To test the method it was implemented on a Nexus 10, which is a tablet running Android on a Samsung Exynos 5 Dual processor. The GPU included in this chip, a Mali T-604, is by the time of this writing a better middle class GPU for embedded systems. Comparing it to concurrent PC graphics-cards, its limitation can be clearly seen. While current PC graphics-cards can have more than 3000 cores running at more than 1 GHz, as well as up to 4 Gbyte dedicated memory, the Mali T-604 has 4 cores running at 533 Mhz, and has to share the main memory (2Gbyte in this case) with the CPU and all other circuits integrated in the chip. Thus, it is not surprising that the Nexus 10 reaches only 8006 at the Icestorm benchmark, which puts it in the middle class of mobile device (the current maximum reached by a mobile device is 11346). By comparison, the middle class graphic card NVIDIA GeForce GTX 660 reaches 137246, or more than 17 times the performance of the Nexus 10. The highest value reached on a PC is (to the knowledge of the authors) 167203, but it should be pointed out that the benchmark is slightly biased towards low performance systems since it does not take advantage of many features that high-end cards offer.

Three different systems are included in the demo: *fire*, *smoke* and *water* (see figure 1 for example pictures). The sizes of the used particle-fields were 64x64 for the water and the smoke and 32x32 for the fire.

Although for some cases a 2-dimensional particle effect might be enough, e.g. a fire in a fireplace, in most cases depth needs to be introduced during rendering. This can be done e.g. by displacement mapping, which has the additional advantage that the designer can choose roughly which shape the object should have. The water and the smoke in the demo are rendered using this method, the latter one using a spheroid as basic shape. The fire was rendered purely as a texture on an otherwise unmodified spheroid. 1922 triangles were used for the fire, 7938 for the smoke, and 9660 for the water.

The particle-fields are also used to generate the textures projected onto the particle objects, and advection was added in case of the fire and the water. The average movement vectors included in the particle field were used as turbulence field for the advection. For the random noise, it was chosen to use a noise texture, which

is a texture that contains random values and is a common solution for advection. This has the additional advantage that the noise can be custom tailored for the effect that should be reached, e.g. in the case of the fire it proved to be advantageous to have many high values concentrated in parts of the noise-texture and lower values in the rest, since this leads to more flame-like structures. Finally, to reduce memory consumption, noise textures of very small sizes are used, and repeated several times instead. The advection process is illustrated in figure 2g-2j.

The rendering was done in two different resolutions, 2560x1600 (which is the currently highest resolution available in tablets), as well as 1280x800, which is a high-end resolution of smaller tablets and smartphones at the time of this writing. For comparison, the iPhone 5 has 1136x640. The timing results are summarized in table 1. The time needed to render the ground is given as comparison; it uses 2386 triangles and 3 different textures depending on its height (with linear interpolation at the borders), but no other effects. These values include overhead like sending variables from the main program to the shader, and were taken from the viewpoint which (on average) lead to the worst result.

Looking at the numbers it is noticeable how close the values for the different simulations are, which is especially interesting since the particle-field of the fire is only 1/4th of the size of the other ones. Not surprisingly the smoke was rendered fastest, since its shader does not include much more than the actual geometry calculation, while the water fared worst due to its complicated advection scheme and its high number of triangles.

Table 1: Average simulation and rendering times, as well as the theoretical frame-rates of each system, which were calculated as $(1s-10*t_s)/t_r$, with t_s the simulation and t_r the rendering time, since the simulations run at a constant 10 fps.

	Simulation	Rendering 2560x1600	**Theo. fps 2560x1600**	Rendering 1280x800	**Theo. fps 1280x800**
Water	3.83 ms	46.1 ms	**20.9**	17.2 ms	**55.9**
Fire	3.42 ms	28.2 ms	**34.2**	10.5 ms	**92**
Smoke	4.12 ms	11.8 ms	**81.3**	6.92 ms	**139**
Ground	-	10.6 ms	**94.3**	7.14 ms	**140**

4. CONCLUSION & FUTURE WORK

A method was presented that shows how particle systems can be simulated and rendered in realtime on embedded systems, by approximating a volumetric system with a 2-dimensional one, and by using a novel highly efficient model for the particle movement. This method could prove very useful to include advanced particle effects in games for smartphones, tablets and similar systems. Performance is adequate for the target systems, but further optimizations are possible. We also aim for a unified system, where a designer would be given the possibility to control the look of the simulation and of the rendering by setting only a couple of parameters. Although this is already the case for the *smoke* and *water* simulations, we need to find further generalizations to eliminate the differences between the various cases to achieve this.

REFERENCES

Guay, M., Colin, F., Egli, R., 2011. Screen Space Animation of Fire. *Proceeding of SIGGRAPH Asia 2011 Sketches*

Hadwiger, M., Patric Ljung, P., Salama, C.R., Ropinski,T, 2009. Advanced illumination techniques for GPU-based volume raycasting. *Proceedings of ACM SIGGRAPH 2009 Courses*, Article No. 2

Harris, M.J., Lastra, A., 2001. Real-Time Cloud Rendering. *Proceedings of EUROGRAPHICS 2001*, vol. 20, no. 3

Horvath, C., Geiger, W., 2009.: Directable, high Resolution Simulation of Fire on the GPU. *ACM Transactions on Graphics 28*, 3, Article 41

Krüger, J., Westermann, R., 2005. GPU simulation and rendering of volumetric effects for computer games and virtual environments. *Proceedings of Eurographics*

Kun Zhou, Zhong Ren, Lin, S., Hujun Bao, Baining Guo, Heung-Yeung Shum, 2008. Real-Time Smoke Rendering Using Compensated Ray Marching. *ACM Transactions on Graphics 27*, 3, Article 36

Laine, S., Karras, T., 2011. Efficient Sparse Voxel Octrees. *IEEE Transactions on Visualization and Computer Graphics*, vol. 17 , iss. 8, pp. 1048-1059

Lorensen, W.E., Cline, H.E., 1987. Marching cubes: A high resolution 3D surface construction algorithm. *Proceedings of the 14th annual conference on Computer graphics and interactive techniques (SIGGRAPH)*, pp. 163-169

Moser, M., Weiskopf, D., 2008. Interactive volume rendering on mobile devices. *Vision, Modeling, and Visualization*

Neyret, F., 2003. Advected Textures. *Eurographics / Siggraph Symposium of Computer Animation*, pp. 147-153

Qizhi Yu, Neyret, F., Bruneton, E., Holzschuch, N., 2011. Lagrangian Texture Advection: Preserving both Spectrum and Velocity Field. *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 11 pp. 1612-1623

Rodriguez, M. B., Alcocer, P.P.V., 2012. Practical Volume Rendering in Mobile Devices. *Advances in Visual Computing*

Selle, A., Rasmussen, N., Fedkiw, R., 2005. A Vortex Particle Method for Smoke, Water and Explosions. *SIGGRAPH 2005, ACM TOG 24*, pp. 910-914

Wrenninge, M., Bin Zafar, N., Clifford, J., Graham, G., Penney, D., Kontkanen, J., Tessendorf, J., Clinton, A., 2010. Volumetric Methods in Visual Effects. *SIGGRAPH 2010 Course Notes*

Yubo Zhang, Zhao Dong, Kwan-Liu Ma, 2012. Realtime volume rendering using precomputed photon mapping. *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D '12)*, p. 217