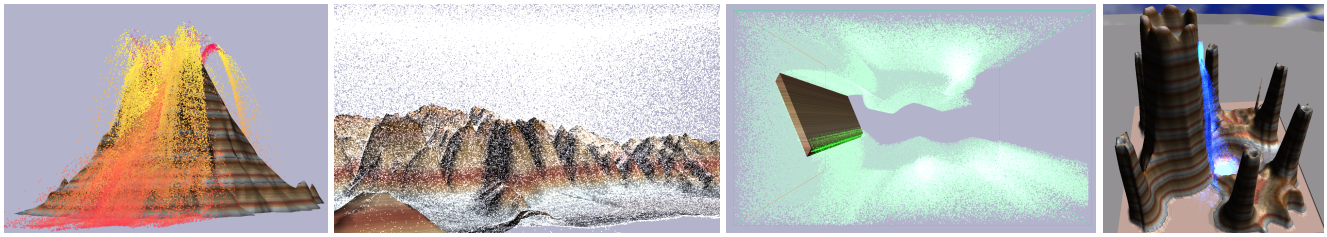
UberFlow: A GPU-Based Particle Engine

Peter Kipfer, Mark Segal, Rüdiger Westermann
Computer Graphics & Visualization, Technische Universität München*, ATI Research†



Abstract

We present a system for real-time animation of large particle sets using GPU computation and *memory objects* in OpenGL. Our system implements a versatile particle engine, including inter-particle collisions and visibility sorting. By combining memory objects with floating-point fragment programs, we have implemented a particle engine that entirely avoids the transfer of particle data at runtime.

1 Introduction and Previous Work

In current particle engines, both the dynamics module and the collision module are run on the CPU. In [Govindaraju et al. 2003], the GPU has been used to implement specific parts of the collision module while [Knott et al. 2003] confined the collision to a particular object type. This conventional assignment of functional units to processing units reveals the limitations of current graphics processors. In this work, we demonstrate how to overcome this assignment, yet providing a particle engine amenable to physics based simulations.

2 Exposition

For high resolution particle sets the transfer of these sets for rendering purposes has to be avoided. Therefore, simulation of particle dynamics must be performed on the GPU. In this sketch, we present novel approaches to carry out particle simulation in the fragment units of programmable graphics hardware. Built upon an improved GPU implementation of a sorting network, we have implemented inter-particle collision detection and visibility sorting of hundreds of thousands of primitives. In combination with OpenGL memory objects, we present the first particle engine that entirely runs on the GPU *and* includes such effects.

2.1 Sorting

Our engine either resolves inter-particle collisions and renders the particles as opaque primitives *or* it sorts particles according to their distance to the viewer and renders the particles as semi-transparent primitives. Both scenarios rely on the sorting of particles.

We present a novel vertex and fragment program sorting routine that is far faster than previous implementations [Purcell et al. 2003]. A two-stage sorting pipeline consisting of parallel geometry and fragment units sorts about 3 million keys and identifiers per second. Our routine exploits fragment and geometry processors, and it avoids expensive on-chip computations in the comparator stages by using pre-computed per-vertex attributes. We further pack consecutive sorting keys and identifiers into RGBA texture values, thus exploiting internal parallelism and bandwidth.

*kipfer@in.tum.de, westermann@in.tum.de
†segal@ati.com

2.2 Collision Detection

The GPU collision detection module simultaneously computes for each particle an approximate set of potential collision partners in the fragment units. Only the closest one is kept and used as input for the collision response module, where collision impulses are applied in parallel to every particle. In situations involving multiple collision events this can lead to incorrect dynamics, because collisions that are caused by the current update can not be resolved in the same time step. In this case, time-sequential or simultaneous processing of collision events gives more plausible results, but both techniques can not easily be mapped to graphics hardware.

3 Rendering

Our method relies on computing intermediate results in the GPU, saving these results in graphics memory, and then sending them through the GPU again to obtain images in the frame buffer. Our implementation exploits a feature of recent ATI graphics hardware that allows graphics memory to be treated as a render target, a texture, or vertex data. This feature is presented to the application through an extension to OpenGL called *SuperBuffers*. Updated particle attributes like position and velocity are simultaneously rendered to separate SuperBuffer render targets using the `ATI_draw_buffer` extension. These targets are then directly used as input to the geometry units.

4 Conclusion

In this sketch, we present the first GPU particle engine that features inter-particle collision detection and visibility sorting, and does not require any transfer of particle data at run-time. With regard to performance, the proposed implementation significantly improves previous work thus enabling realistic animation of large particle sets. Our system can move up to 120 million particles per second and processes up to 1 million particles per second with full inter-particle collision detection.

References

GOVINDARAJU, N., REDON, S., LIN, M., AND MANOCHA, D. 2003. Cullide: Interactive collision detection between complex models in large environments using graphics hardware. In *Proceedings ACM SIGGRAPH/Eurographics Conference on Graphics Hardware*.

KNOTT, D., VAN DEN DOEL, K., AND PAI, D. K. 2003. Particle system collision detection using graphics hardware. In *SIGGRAPH 2003 Sketch*.

PURCELL, T., DONNER, C., CAMMARANO, M., JENSEN, H., AND HANRAHAN, P. 2003. Photon mapping on programmable graphics hardware. In *Proceedings SIGGRAPH/Eurographics Workshop on Graphics Hardware*, ACM, 41–50.