Simulation Modelling Practice and Theory 19 (2011) 801–816

Contents lists available at ScienceDirect

Simulation Modelling Practice and Theory

journal homepage: www.elsevier.com/locate/simpat

A real-time multigrid finite hexahedra method for elasticity simulation using CUDA

Christian Dick *, Joachim Georgii, Rüdiger Westermann

Computer Graphics and Visualization Group, Technische Universität München, Germany

ARTICLE INFO

Article history:
Received 2 August 2010
Received in revised form 8 November 2010
Accepted 9 November 2010
Available online 21 November 2010

Keywords:
Elasticity simulation
Deformable objects
Finite element methods
Multigrid
GPU
CUDA

ABSTRACT

We present a multigrid approach for simulating elastic deformable objects in real time on recent NVIDIA GPU architectures. To accurately simulate large deformations we consider the co-rotated strain formulation. Our method is based on a finite element discretization of the deformable object using hexahedra. It draws upon recent work on multigrid schemes for the efficient numerical solution of partial differential equations on such discretizations. Due to the regular shape of the numerical stencil induced by the hexahedral regime, and since we use matrix-free formulations of all multigrid steps, computations and data layout can be restructured to avoid execution divergence of parallel running threads and to enable coalescing of memory accesses into single memory transactions. This enables to effectively exploit the GPU's parallel processing units and high memory bandwidth via the CUDA parallel programming API. We demonstrate performance gains of up to a factor of 27 and 4 compared to a highly optimized CPU implementation on a single CPU core and 8 CPU cores, respectively. For hexahedral models consisting of as many as 269,000 elements our approach achieves physics-based simulation at 11 time steps per second.

1. Introduction

Over the last years, graphics processing units (GPUs) have shown a substantial performance increase on intrinsically parallel computations. Key to this evolution is the GPU's design for massively parallel tasks, with the emphasis on maximizing total throughput of all parallel units. The ability to simultaneously use many processing units and to exploit thread level parallelism to hide latency have led to impressive performance increases in a number of scientific applications.

One prominent example is NVIDIA's Fermi GPU [1], on which we have based our current developments. It consists of 15 multiprocessors, on each of which several hundreds of co-resident threads can execute integer as well as single and double precision floating point operations. Double precision operations are running at 1/2 of the speed of single precision operations. Each multiprocessor is equipped with a register file that is partitioned among the threads residing on the multiprocessor, as well as a small low-latency on-chip memory block which can be randomly accessed by these threads. Threads are further provided with direct read/write access to global off-chip video memory. These accesses are cached using a two level cache hierarchy.

The threads on each multiprocessor are executed in groups of 32 called warps, and all threads within one warp run in lock-step. Due to this reason the GPU works most efficiently if all threads within one warp follow the same execution path. Automatic hardware multithreading is used to schedule warps in such a way as to hide latency caused by memory access operations. Switching between warps is virtually at no cost, since threads are permanently resident on a multiprocessor

* Corresponding author.
E-mail addresses: dick@tum.de (C. Dick), georgii@tum.de (J. Georgii), westermann@tum.de (R. Westermann).

1569-190X/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.simpat.2010.11.005

(independently of whether they are running, blocked, or waiting). As a consequence, however, the registers are partitioned among *all* threads residing on a multiprocessor, which significantly reduces the number of registers available to each thread.

The Fermi GPU executes global memory accesses at a fix granularity of 128 bytes, i.e., the GPU reads or writes contiguous blocks of 128 bytes that are aligned at 128-byte boundaries. The hardware coalesces parallel accesses of the threads of a warp that lie in the same 128-byte segment into a single memory transaction. To effectively exploit the GPU's memory bandwidth, parallel accesses of the threads of a warp should therefore lie closely packed in memory to reduce the number of memory transactions and to avoid transferring of unnecessary data. Specifically, if the *i*th thread of a warp (half warp) accesses the *i*th 32-bit (64-bit) word of a 128-byte segment, these accesses are combined into a single memory transaction and the GPU's memory bandwidth is optimally used.

1.1. Contribution

We present a novel geometric multigrid finite element method on the GPU, and we show the potential of this method for simulating elastic material in real time on desktop PCs. To the best of our knowledge, this is the first multigrid finite element approach for solving linear elasticity problems that is realized entirely on the GPU. Since we use the co-rotational formulation of strain, even large deformations can be simulated at high physical accuracy. The CUDA API [2] is used because in contrast to graphics APIs like OpenGL or Direct3D it gives the programmer direct control over all available computing and memory resources.

To effectively exploit the GPU's massively parallel multithreading architecture, an algorithm must be restructured to expose a sufficient amount of fine-grained parallelism down or beyond one thread per data element. These threads should follow one common execution path and exhibit memory access patterns which enable coalescing of memory accesses to effectively exploit the massive memory bandwidth available on the GPU.

The particular restructuring we propose is based on a regular hexahedral discretization of the simulation domain, which provides a number of advantages for GPU-based deformable object simulation: First, a hexahedral discretization of a given object boundary surface can be generated at very high speed, including a multi-resolution representation that is required in a geometric multigrid approach. Second, the regular topology of the hexahedral grid leads to a numerical stencil of the same regular shape at each simulation vertex. This enables parallel processing of vertices using the same execution path and allows for memory layouts that support coalescing of memory access operations. Third, since all hexahedral elements have the same shape, only a single pre-computed element stiffness matrix is needed, which greatly reduces memory requirements. The stiffness matrix of a specific finite element is obtained from this matrix by scaling with the element's elastic modulus and by applying the current element rotation according to the co-rotated strain formulation.

Due to these advantages, we achieve performance gains of up to a factor of 27 compared to an optimized parallel CPU implementation running on a single CPU core. Even compared to the CPU implementation running on 8 CPU cores, our GPU implementation is a factor of up to 4 faster. This speed-up results from both the arithmetic and memory throughput on the Fermi GPU. Our CUDA implementation of the multigrid method achieves update rates of 120 time steps per second for models consisting of 12,000 hexahedral elements. For large models consisting of 269,000 elements, 11 time steps per second can still be achieved. Each time step includes the re-assembly of the system of equations, which is necessary due to the co-rotated strain formulation, as well as two multigrid V-cycles for solving this system. In combination with a high-resolution render surface, which is bound to the simulation model via pre-computed interpolation weights, a visually continuous rendering of the deformable body is achieved (see Fig. 1).

2. Related work

Over the last years, considerable effort has been spent on the efficient realization of general techniques of numerical computing on programmable GPUs [3,4]. Recent work in this field has increasingly focused on the use of the CUDA API [2],

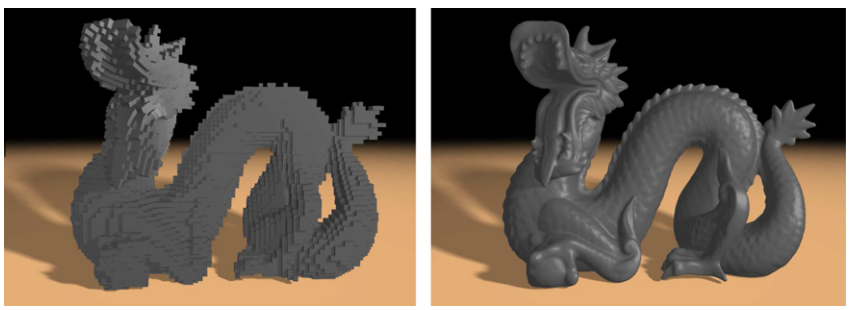

Fig. 1. Left: A deformed hexahedral object consisting of 30,000 elements is shown. Right: By using a high-resolution render surface that is bound to the deformed representation a visually continuous appearance is achieved.