Seventh Irish Workshop on Computer Graphics (2006)
Eurographics Irish Chapter

Towards a Field-Programmable Physics Processor (FP^3)

Muiris Woulfe and Michael Manzke

Interaction, Simulation and Graphics Lab (ISG), Department of Computer Science, Trinity College Dublin

Abstract

In this paper, we outline the design and implementation of an FPGA-based numerical integrator that will ultimately form the basis of our FPGA-based physics engine. Physics engines are considered one of the most important of a multitude of components requesting CPU time in a modern computer game, and so we propose offloading aspects of this physics computation to an FPGA-based physics engine. We aim to ameliorate the speed of the physics computation in concert with the other game components. Currently, our physics processor uses the fourth-order Runge-Kutta numerical integration algorithm to solve the ordinary differential equations used in game physics. Our analyses indicate that the performance of our physics processor should surpass that of the equivalent software executing on a CPU when several objects are simulated.

Categories and Subject Descriptors (according to ACM CCS): I.3.1 [Computer Graphics]: Hardware Architecture

1. Introduction

Physics engines are approaching ubiquity due to the ever-increasing demand for realistic computer games. These middleware solutions perform physics computations on behalf of other software, in order to simulate the behaviour of objects realistically. They are primarily used in computer games for improving the realism of on-screen objects, but are also used in 3D modelling tools to assist animators in creating realistic motion when rendering scenes.

Traditionally, physics engines have modelled rigid body dynamics, which describe the interactions between rigid bodies or solid objects. These are typically modelled by ordinary differential equations (ODEs), which are capable of expressing the dynamic behaviour of systems. ODEs are usually rewritten so that the equations are in terms of integration rather than differentiation, so that they may be evaluated using a numerical integration algorithm. Using ODEs, acceleration may be integrated to compute velocity, and velocity may be integrated to compute displacement. Recently, physics engines have expanded their abilities beyond rigid body dynamics to include related fields such as particle and cloth simulation.

Computer game physics must be computed in real-time, so that the physics computations do not delay the motion displayed on the screen. While realism is important, it is secondary to this real-time requirement and it is satisfactory if the user perceives the motion generated by the physics to be entirely correct.

Havok Physics [Havb] and AGEIA PhysX [AGE] are some prominent commercial examples of software physics engines, while the Open Dynamics Engine (ODE) [ODE] provides a non-commercial substitute. AGEIA PhysX is also available as a hardware platform. This platform has recently become commercially available, and can be used to improve the physics of games above what is achievable with a purely software solution. Physics, primarily background and effects physics, may also be computed using the shaders on either a recent ATI or NVIDIA Graphics Processing Unit (GPU) and the Havok FX physics engine [Hava].

Originally, 3D graphics had to timeshare the CPU with other game components, but the relentless desire for 3D content spurred the development of GPUs, which are used to offload complex graphical computations from the CPU. This paper proposes to mirror this concept for physics, through the creation of physics engine hardware, similar to the AGEIA PhysX. However, we additionally propose the use of Field-Programmable Gate Arrays (FPGAs), whose reconfigurability should provide unique advantages.

FPGAs are integrated circuits (ICs) consisting of programmable resources that may be reconfigured at runtime, in the field. Algorithms can often be accelerated when they are offloaded from the CPU to an FPGA, in the same way an

Application-Specific Integrated Circuit (ASIC) may accelerate an algorithm.

An FPGA physics engine solution offers advantages over the more obvious ASIC approach. By placing FPGAs inside commodity PCs, the reconfigurable fabric of the FPGAs may be utilised for purposes other than physics computation. For instance, if a game performs many physics calculations, the FPGA could be used for accelerating these calculations as discussed in this paper. If, instead, a game performs many AI computations, the FPGA could be used to accelerate these AI routines. The adaptability of FPGAs is illustrated by the way many diverse applications, such as ray tracing [SWS02, SWW*04] and MATLAB computations [NHCB01, HNCB01, HNS*01, BBH*03, BHN*04], have already been accelerated using an FPGA. Through its reconfigurability, the FPGA allows a multitude of tasks to be accelerated, unlike a less flexible ASIC.

Since ODEs describe the core of game physics, we chose an FPGA-based numerical integrator as the starting point for our physics engine. In the subsequent sections, we describe this numerical integrator, which forms the groundwork for a reconfigurable hardware physics engine.

2. Related work

Research into numerical integrators appropriate for physics engines indicates that sophisticated algorithms are redundant. Kokkevis' [Kok04] work suggests that the simple Euler integrator is adequate for the problem set he analysed. Meanwhile, Baraff [Bar95] advocates using the fourth-order Runge-Kutta integrator ordinarily, but in certain defined circumstances he proposes employing a number of simplifications. The more sophisticated Verlet [Ver67] family of integrators have recently proven popular, after they were first employed by Jakobsen [Jak01] in the game *Hitman: Codename 47*. Unlike the Euler and Runge-Kutta integrators, Verlet integrators are reversible in time, which is highly desirable for computer game applications.

Traditional numerical integration algorithms do not perform well on FPGAs as they consist of a large number of dependent operations, leaving little scope for parallelism. Myriad parallel numerical integration algorithms have been proposed [KM82, FHS89, MG97, BF01, VAQ01, BGD*02, Ram02, MOC02, MCH*04, BB05], but these concentrate on coarse grained parallelism using a cluster of computers. The emphasis, therefore, is on achieving high precision results over a lengthy period. However, the emphasis in a physics engine is on achieving results in real-time at the expense of precision, making these algorithms unsuitable for our integrator.

No FPGA-based physics engines have yet been designed, although Atay [ALB05] has researched collision detection using FPGAs. This research suggested that a speed gain of 36 is achievable over a commodity CPU. Implementation details of the aforementioned ASIC solution, the AGEIA PhysX, have not been published in academic fora although a number of patent filings provide some limited information [DHS*05a, DHS*05b, DHS*05c, MBST05, TZS05, ZTSM05a, ZTSM05b].

3. Mathematical background

The fourth-order Runge-Kutta method is a general-purpose numerical integration algorithm that achieves relatively good accuracy in relatively little time [Kiz64]. Moreover, the algorithm is robust, meaning that it is capable of withstanding irregularities in the function being integrated. For these reasons, the fourth-order Runge-Kutta method was selected as the first target for our evaluation.

The fourth-order Runge-Kutta method involves computing four approximate steps, each using the result of the former step [SB02]. These four steps are subsequently interpolated to create a result with a lower truncation error than each of the individual steps. The equations describing the fourth-order Runge-Kutta algorithm are:

$$\begin{aligned}
k_1 &= hf\left(y_n\right)\\
k_2 &= hf\left(y_n + \frac{1}{2}k_1\right)\\
k_3 &= hf\left(y_n + \frac{1}{2}k_2\right)\\
k_4 &= hf\left(y_n + k_3\right)\\
y_{n+1} &= y_n + \frac{1}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right)
\end{aligned}$$

where h is the time-step [s] and f is the function to integrate.

To demonstrate the operation of the integrator and obtain data on its performance, a suitable mathematical problem was required. We decided to model the motion of a ship when considering the various resistive forces, using a simplified mathematical model proposed by Bourg [Bou02]. It is indicative of many relevant physics equations due to its heavily interdependent sequence of operations and its average complexity. The relevant equations are:

$$\begin{aligned}
F &= T - Cv_0\\
a &= \frac{F}{m}\\
v_1 &= \int a \,\mathrm{d}t\\
s &= \int v_1 \,\mathrm{d}t
\end{aligned}$$

where F is the total force acting on the ship [N], T is the propeller thrust [N], C is the drag coefficient, m is the mass [kg], t is the time [s], a is the acceleration [m/s^2], v is the velocity [m/s] and s is the displacement [m]. The forces are illustrated in Figure 1.

When the simplified ship equations are merged with the

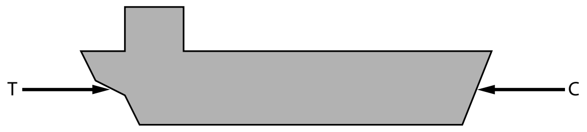

Figure 1: *Forces acting on a ship. T is the propeller thrust [N] and C is the drag coefficient.*

fourth-order Runge-Kutta algorithm, the sequence of calculations becomes:

$$\begin{aligned}
F &= T - Cv_0\\
a &= \frac{F}{m}\\
k_1 &= ha\\
\\
F &= T - C \times \left(v_0 + \frac{1}{2}k_1\right)\\
a &= \frac{F}{m}\\
k_2 &= ha\\
\\
F &= T - C \times \left(v_0 + \frac{1}{2}k_2\right)\\
a &= \frac{F}{m}\\
k_3 &= ha\\
\\
F &= T - C \times (v_0 + k_3)\\
a &= \frac{F}{m}\\
k_4 &= ha\\
\\
v_1 &= v_0 + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)\\
s &= s + hv_1 \qquad (1)
\end{aligned}$$

As can be observed from this sequence of equations, there is little scope for parallelism. Most equations are dependent on their preceding equations, preventing the computations overlapping. This significantly limits the computation speed on any modern CPU, FPGA or ASIC. Software physics engines overcome this bottleneck by simulating a multitude of objects simultaneously. In the future, we intend to use the same technique for our FPGA-based physics engine.

4. Architecture

We chose the general architecture of an instruction set processor as the starting point for our design, since our numerical integrator demanded an adaptable architecture to facilitate a potentially limitless range of equations. Our design comprises two primary modular units – a workload scheduler and an arithmetic unit – as illustrated in Figure 2. Both units are pipelined to maximise their throughput. This allows multiple instructions to be in execution simultaneously, so that all parallelism available in the instruction set may be exploited.

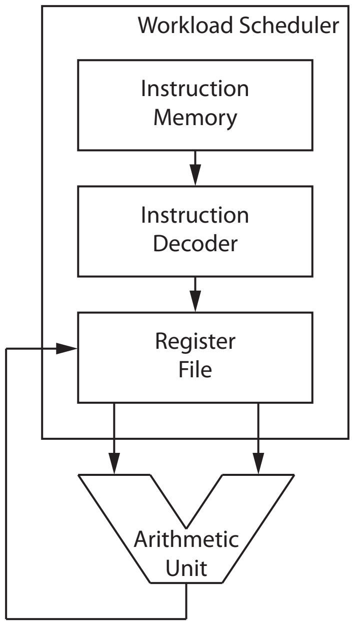

Figure 2: *Simplified architecture of the physics engine. Control logic and other extraneous details have been omitted for clarity.*

The workload scheduler's core role is to provide the instruction stream used to control the integration procedure. The first pipeline stage consists of an instruction memory. The instructions stored in this memory specify the operations that comprise the numerical integration algorithm, merged with the mathematical description of the current simulation. For the simplified ship model, the instructions would specify Equations (1). The mathematical operations that may be utilised are defined by the capabilities of the arithmetic unit.

The workload scheduler also contains a register file. This register file stores the data utilised by the instructions and is tasked with resolving data dependencies between instructions. To resolve dependencies, each register has a busy flag. The destination register for each instruction is flagged, until it is subsequently written. Any instruction requesting a flagged register is stalled until the flag is cleared. When data is written to the register file from the arithmetic unit, the pipeline is also stalled to prevent erroneous data migrating through.

The arithmetic unit's core role is to execute the operations supplied by the workload scheduler. It consists of a number of floating-point cores, which implement the required operations. For the simplified ship simulation, the required operations are addition, subtraction and multiplication.

As stated in Section 1, in a physics engine, objects need only appear realistic, without necessarily behaving with complete realism. Based on this observation, IEEE 754 single-precision floating-point cores were considered to offer sufficient precision. These cores consume significantly less logic and routing resources than the equivalent double-precision floating-point cores.

Furthermore, it was decided to replace time-consuming and logic intensive division with more efficient multiplication. For example, instead of dividing by 6, we multiply by 0.16666667. This leads to a minor loss of accuracy, but based on the aforementioned observation, this was deemed negligible.

5. Performance

To evaluate the performance of our design, we implemented the model of a ship described by Equations (1). The first integration of this model consumes 445 clock cycles while each subsequent integration consumes 418 clock cycles. The reduction in timing is primarily due to instruction overlapping between integrations. Removal of the register initialisation commands from the subsequent integrations also contributed to this reduction.

To translate these cycle counts into timing data, we implemented our integrator on a Xilinx XC2V6000 FPGA of speed grade four. The shortest period achieved was 8.888 ns, which yielded 3955.16 ns for the first integration and 3715.184 ns for the subsequent integrations. For comparison, the equivalent software algorithm executes in 56.11 ns on a 3.4 GHz Intel Pentium 4.

Since numerical integration algorithms perform a number of discrete steps and average them to compute a more accurate step, the equations describing these algorithms are heavily interdependent. This means that they do not exploit the parallelism available in hardware, so that the performance of a hardware integrator could only exhibit a modest improvement over equivalent software. However, the FPGA and CPU are executing at vastly different clock speeds, resulting in a large discrepancy between their performances.

Despite the above results, we believe we will be able to attain execution speeds greater than those achievable on a CPU, by implementing the ideas proposed in Section 6.

6. Future work

We intend to fine-tune the performance of our FPGA-based numerical integrator, to surpass the performance of the equivalent CPU algorithm. We will analyse alternative floating-point arithmetic cores, evaluate additional numerical integration algorithms, maximise the parallelism and port the system to a shared-memory graphics architecture.

6.1. Floating-point arithmetic

Initially we plan to analyse alternative floating-point arithmetic cores. Different cores have different area and timing constraints, and it is important to find the cores offering the best tradeoff between the two constraints. In particular, we intend to analyse cores that do not implement the entire IEEE 754 floating-point standard, like those used in the Cell processor [OMJ*05]. These cores would execute faster and should offer sufficient precision to allow the physics to appear realistic. Additionally, we intend to analyse logarithmic cores. These cores take only a single clock cycle to perform a multiplication or division, at the expense of more time-consuming additions and subtractions. There are, however, significant time and area overheads involved in converting the numbers between the IEEE 754 and logarithmic formats.

6.2. Numerical integrators

Next, we intend to evaluate other types of numerical integrators. In particular, we plan to analyse the Verlet numerical integration algorithm due to its many benefits and its current popularity in computer games. It is probable that certain numerical integration algorithms will be more amenable to hardware implementation than others, and consequently, it may be necessary to experiment to find the optimal algorithm. Since the current integrator was designed to accommodate extensibility, it will be easy to experiment with alternative algorithms.

6.3. Parallelism

We also intend to utilise many integrators in parallel, so that each object in a simulation is integrated simultaneously. This idea should offer the greatest acceleration, since exploiting parallelism is the primary means of accelerating algorithms on FPGAs. Each workload scheduler could share the same arithmetic unit as illustrated in Figure 3, since the multiple floating-point cores in the arithmetic unit are highly pipelined and currently underutilised. Although this will lead to some stalls as the workload schedulers are arbitrated for access to the arithmetic unit, this method should significantly improve the performance while maximising the number of integrations that may be computed on an FPGA. In this way, we will create a more complete and faster physics engine.

6.4. Shared-memory graphics architecture

Ultimately, we intend to place our physics engine on the shared-memory hybrid graphics cluster for visualisation and video processing, which has been prototyped [MBO*06],

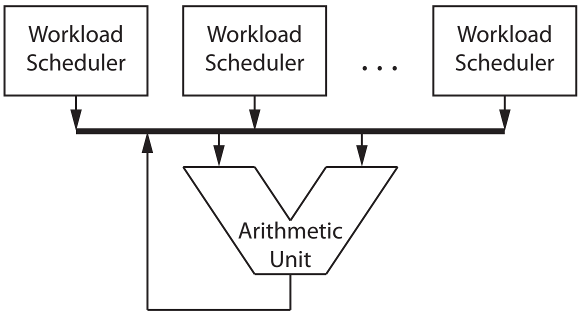

Figure 3: *A more efficient scheme where multiple workload schedulers utilise a single arithmetic unit. The arithmetic unit consists of a number of floating-point cores.*

as shown in Figure 4. The system consists of a cluster of custom-built Printed Circuit Boards (PCBs). These PCBs are connected to a cluster of commodity PCs that supply instructions and data to the boards. An Accelerated Graphics Port (AGP) interface allows commodity graphics accelerators to generate graphical output, while an FPGA provides additional reconfigurable logic that may be used to support the graphics. The boards are connected via Scalable Coherent Interconnect (SCI), which provides a high bandwidth, low latency, point to point interconnect that implements a Distributed Shared Memory (DSM) architecture.

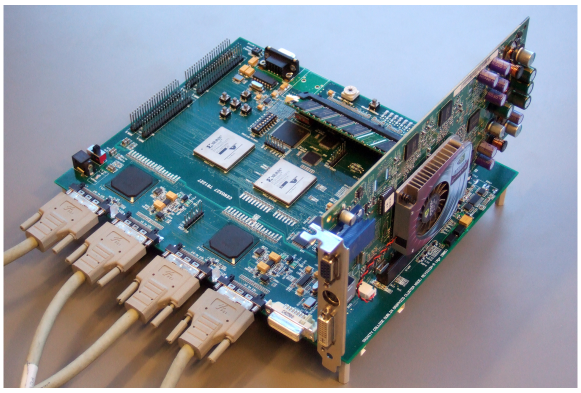

Figure 4: *The first prototype of the custom-built high-performance graphics cluster node.*

We intend to port our physics engine to the FPGAs present in this cluster. Each FPGA will simulate multiple objects in parallel, and will operate in parallel with every other FPGA, vastly increasing the parallelism of the system. Furthermore, the SCI interconnect offers extremely low communication latencies, alleviating the delay incurred when the CPU and an FPGA communicate over the PCI bus in a commodity PC. The significant parallelism and low latency interconnect should considerably increase the performance of our implementation. Moreover, the system will then facilitate large-scale physics computations.

7. Conclusions

We have presented the design of a reconfigurable hardware numerical integrator, which will ultimately form the foundations of a reconfigurable hardware physics engine for use in computer games. This physics engine will eventually perform all necessary physics processing tasks, offloading this intricate work from the CPU and leaving it free to perform other tasks such as AI. This unlocks the possibility of performing more complex and more accurate physics computations than are currently achievable. Although the performance of the numerical integrator does not currently match that attainable on a CPU, we expect to surpass this limitation by utilising many parallel integrators.

Acknowledgements

This research is supported by the Irish Research Council for Science, Engineering and Technology (IRCSET) funded by the National Development Plan (NDP).

References

[AGE] AGEIA PhysX. `http://www.ageia.com/physx/` (Last accessed: 20 September 2006).

[ALB05] Atay N., Lockwood J. W., Bayazit B.: A collision detection chip on reconfigurable hardware. In *Proceedings of the 13th Pacific Conference on Computer Graphics and Applications* (Oct. 2005).

[Bar95] Baraff D.: Interactive simulation of solid rigid bodies. *IEEE Computer Graphics and Applications 15*, 3 (1995), 63–75.

[BB05] Bailey D. H., Borwein J. M.: *Highly Parallel, High-Precision Numerical Integration*. Tech. Rep. LBNL-57491, Lawrence Berkeley National Laboratory, University of California, San Francisco, California, USA, Apr. 2005.

[BBH*03] Banerjee P., Bagchi D., Haldar M., Nayak A., Kim V., Uribe R.: Automatic conversion of floating point MATLAB programs into fixed point FPGA based hardware design. In *Proceedings of the 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines* (Napa, California, USA, 2003), IEEE Computer Society, pp. 263–264.

[BF01] Bull J. M., Freeman T. L.: Parallel algorithms for multi-dimensional integration. *Progress in Computer Research* (2001), 207–223.

[BGD*02] Bunich A. L., Ginsberg K. S., Dobrovidov A. V., Zatuliveter Y. S., Prangishvili I. V., Smolyaninov V. V., Sukhov E. G.: Parallel computation and control problems: A review. *Automation and Remote Control 63*, 12 (2002), 1867–1883.

[BHN*04] BANERJEE P., HALDAR M., NAYAK A., KIM V., SAXENA V., PARKES S., BAGCHI D., PAL S., TRIPATHI N., ZARETSKY D., ANDERSON R., URIBE J. R.: Overview of a compiler for synthesizing MATLAB programs onto FPGAs. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems 12*, 3 (Mar. 2004), 312–324.

[Bou02] BOURG D. M.: *Physics for Game Developers*. O'Reilly, Sebastopol, California, USA, 2002.

[DHS*05a] DAVIS C., HEDGE M., SCHMID O. A., MAHER M., BORDES J. P.: Method for providing physics simulation data, Apr. 2005. (International application number: PCT/US2004/030687).

[DHS*05b] DAVIS C., HEDGE M., SCHMID O. A., MAHER M., BORDES J. P.: Physics processing unit, Apr. 2005. (International application number: PCT/US2004/030686).

[DHS*05c] DAVIS C., HEDGE M., SCHMID O. A., MAHER M., BORDES J. P.: System incorporating physics processing unit, Apr. 2005. (International application number: PCT/US2004/030689).

[FHS89] FOX G. C., HIPES P., SALMON J.: Practical parallel supercomputing: Examples from chemistry and physics. In *Proceedings of the 1989 ACM/IEEE Conference on Supercomputing* (Reno, Nevada, USA, 1989), ACM Press, pp. 58–69.

[Hava] Havok FX. `http://www.havok.com/content/view/187/77/` (Last accessed: 20 September 2006).

[Havb] Havok Physics. `http://www.havok.com/content/view/17/30/` (Last accessed: 20 September 2006).

[HNCB01] HALDAR M., NAYAK A., CHOUDHARY A., BANERJEE P.: A system for synthesizing optimized FPGA hardware from MATLAB. In *Proceedings of the 2001 International Conference on Computer-Aided Design* (San José, California, USA, 2001), IEEE Computer Society, pp. 314–319.

[HNS*01] HALDAR M., NAYAK A., SHENOY N., CHOUDHARY A., BANERJEE P.: FPGA hardware synthesis from MATLAB. In *Proceedings of the Fourteenth International Conference on VLSI Design* (Bangalore, India, 2001), IEEE Computer Society, pp. 299–304.

[Jak01] JAKOBSEN T.: Advanced character physics. In *Game Developers Conference Proceedings 2001* (San José, California, USA, 2001), CMP Media, pp. 383–401.

[Kiz64] KIZNER W.: A numerical method for finding solutions of nonlinear equations. *Journal of the Society for Industrial and Applied Mathematics 12*, 2 (1964), 424–428.

[KM82] KROSEL S. M., MILNER E. J.: Application of integration algorithms in a parallel processing environment for the simulation of jet engines. In *Proceedings of the 15th Annual Symposium on Simulation* (Tampa, Florida, USA, 1982), IEEE Computer Society, pp. 121–143.

[Kok04] KOKKEVIS E.: Practical physics for articulated characters. In *Game Developers Conference Proceedings 2004* (San José, California, USA, 2004), CMP Media.

[MBO*06] MANZKE M., BRENNAN R., O'CONOR K., DINGLIANA J., O'SULLIVAN C.: A scalable and reconfigurable shared-memory graphics architecture. In *Proceedings of the SIGGRAPH 2006 Conference on Sketches & Applications* (Boston, Massachusetts, USA, 2006), ACM Press.

[MBST05] MAHER M., BORDES J. P., SEQUEIRA D., TONGE R.: Physics processing unit instruction set architecture, Nov. 2005. (International application number: PCT/US2004/030690).

[MCH*04] MATTHEY T., CICKOVSKI T., HAMPTON S., KO A., MA Q., NYERGES M., RAEDER T., SLABACH T., IZAGUIRRE J. A.: PROTOMOL, an object-oriented framework for prototyping novel algorithms for molecular dynamics. *ACM Transactions on Mathematical Software 30*, 3 (2004), 237–265.

[MG97] MOITRA S., GATSKI T. B.: *Efficient Parallel Algorithm for Direct Numerical Simulation of Turbulent Flows*. Tech. Rep. 3686, NASA Langley Research Center, Hampton, Virginia, USA, Dec. 1997.

[MOC02] MANTAS RUIZ J. M., ORTEGA LOPERA J., CARRILLO DE LA PLATA J. A.: Component-based derivation of a parallel stiff ODE solver implemented in a cluster of computers. *International Journal of Parallel Programming 30*, 2 (2002), 99–148.

[NHCB01] NAYAK A., HALDAR M., CHOUDHARY A., BANERJEE P.: Precision and error analysis of MATLAB applications during automated hardware synthesis for FPGAs. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition 2001* (Munich, Germany, 2001), IEEE Computer Society, pp. 722–728.

[ODE] Open Dynamics Engine. `http://www.ode.org/` (Last accessed: 20 September 2006).

[OMJ*05] OH H.-J., MUELLER S. M., JACOBI C., TRAN K. D., COTTIER S. R., MICHAEL B. W., NISHIKAWA H., TOTSUKA Y., NAMATAME T., YANO N., MACHIDA T., DHONG S. H.: A fully-pipelined single-precision floating point unit in the synergistic processor element of a Cell processor. In *2005 Symposium on VLSI Circuits: Digest of Technical Papers* (Kyoto, Japan, 2005), Japan Society of Applied Physics, pp. 24–27.

[Ram02] RAMA MOHAN RAO A.: A parallel mixed time integration algorithm for nonlinear dynamic analysis. *Advances in Engineering Software 33* (2002), 261–271.

[SB02] STOER J., BULIRSCH R.: *Introduction to Numerical Analysis*, third ed. Texts in Applied Mathematics. Springer, New York, USA, 2002.

[SWS02] SCHMITTLER J., WALD I., SLUSALLEK P.: SaarCOR – A hardware architecture for ray tracing. In *Graphics Hardware 2002: Eurographics Symposium Proceedings* (Saarbrücken, Germany, 2002), Eurographics Association, pp. 27–36.

[SWW*04] SCHMITTLER J., WOOP S., WAGNER D., PAUL W. J., SLUSALLEK P.: Realtime ray tracing of dynamic scenes on an FPGA chip. In *Graphics Hardware 2004: Eurographics Symposium Proceedings* (Grenoble, France, 2004), Eurographics Association, pp. 95–106.

[TZS05] TONGE R., ZHANG L., SEQUEIRA D.: Method and program solving LCPs for rigid body dynamics, Aug. 2005. (International application number: PCT/US2004/030691).

[VAQ01] VIGO-AGUIAR J., QUINTALES L. M.: Parallel ODE solver adapted to oscillatory problems. *The Journal of Supercomputing 19*, 2 (2001), 163–171.

[Ver67] VERLET L.: Computer "experiments" on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules. *Physical Review 159*, 1 (1967), 98–103.

[ZTSM05a] ZHANG L., TONGE R., SEQUEIRA D., MAHER M.: Method of operation for parallel LCP solver, Apr. 2005. (International application number: PCT/US2004/030688).

[ZTSM05b] ZHANG L., TONGE R., SEQUEIRA D., MAHER M.: Parallel LCP solver and system incorporating same, Aug. 2005. (International application number: PCT/US2004/030692).