Defying the Memory Bottleneck in Hardware Accelerated Collision Detection

Andreas Raabe Frank Zavelberg
University of Bonn
Technical Computer Science
Römerstr. 164, 53117 Bonn, Germany
{raabe,zavelber}@cs.uni-bonn.de

ABSTRACT

A novel approach for hardware-accelerated high-speed collision detection is presented in this article. It focuses on dedicated hardware for collision detection queries and its interaction with the memory interface. A specialised tree-traversal algorithm is presented that exploits arbitrary memory interfaces optimally to minimise delay of collision queries. Along with this a novel caching technique is introduced that combines high-speed access to the bounding-volume hierarchy with minimal resource consumption. Simulation and synthesis results are presented that prove the conjunction of both techniques to enable real-time collision queries at rates required by force-feedback while fitting onto a standard field-programmable gate array.

1 INTRODUCTION

Collision detection between graphical objects is a fundamental task in many applications, such as gaming, virtual prototyping and physically based simulation. As [13] report it is also a major bottleneck in all areas of physically based simulation, since up to 95% of the overall effort is spent on collision detection. This is because most approaches are reactive in a way that they first place objects at certain positions, check for collisions and then calculate forces and positions that remove the collisions. This demands collision queries at tremendous rates, because of the hard real-time constraint to complete all collision checks within a simulation cycle. Another highly demanding application is force feedback, where updates of about 1000Hz must be achieved in order to yield stable force computations. Since collision detection is such a fundamental and yet performance hungry task it is highly desirable to have hardware acceleration available. The benefit is an increased number of objects that can be processed per simulation cycle and therefore can populate the simulated environment. Additionally, the CPU is liberated from processing collision queries.

This is underlined by the fact that a growing problem in high-performance computing is the increasing energy consumption of state-of-the-art processing devices. This ongoing trend favours specialised hardware in performance hungry areas of application, since these are in general far more energy efficient than calculations done on general purpose CPUs.

This article presents a dedicated hardware for collision detection queries and investigates on its interaction with the memory interface. As will be shown in the following the major bottleneck in hardware accelerated collision detection are memory accesses. Hence special effort needs to be spent to minimise the number of memory accesses and to maximise performance of the memory hierarchy. To the authors' knowledge this article is the first to systematically investigate on this. A specialised tree-traversal algorithm is presented that exploits arbitrary memory interfaces optimally to minimise delay of collision queries. Along with this a novel caching technique is introduced that combines high-speed access to the bounding-volume hierarchy with minimal resource consumption. A very fast, yet hardware efficient collision detection hardware results.

2 RELATED WORK

Considerable work has been done on hierarchical collision detection in software [9, 4, 22, 11, 23]. A great variety of bounding volumes are utilised, such as spheres, axis-aligned bounding boxes (AABB), oriented bounding boxes (OBB), and discretely oriented polytopes (DOP).

In [20, 21] cache-efficient layouts of bounding volume hierarchies for standard CPU and GPU caches were presented. A significant speed-up was yielded, but the impact of different caching techniques was not investigated.

Recently many collision detection algorithms running standard graphics hardware have been presented [3, 8, 10, 6, 5]. Those have the disadvantage that either the collision detection has to compete for resources with the rendering process or an additional graphics card must be spend. The latter is a tremendous waste of resources since the vast majority of resources is not utilised at all.

In [24, 25] a first architecture of a collision hardware was proposed, but only a functional simulation was provided. [1] presents an FPGA implementation of the well known triangle intersection test proposed in [12] that is detailed in [2]. The approach is fast, but very limited in the number of tested triangles, since it demands that all triangles to be tested reside in the fast, but small Block-RAM (BRAM) of the FPGA. [14] presents a design that is optimised for speed only. Simulation results show that speed-up factors of several orders of magnitude compared to software implementations can be achieved, if a sufficiently dimensioned memory interface is available. The approach utilises a total of over 4 million gates only for the collision detection unit itself and a 756 bits wide bus to a DDR2-RAM. This is very expensive for realisation in any technology and prohibitively expensive for FPGA implementation. A highly hardware efficient, fixed-point based architecture along with a correctness proof and error-bounds were presented in [18, 17], along with a register-transfer accurate implementation and synthesis results. On a standard FPGA prototyping board a speed-up of 4 compared to a standard PC with a comparable DDR-RAM interface was achieved.

This article combines hardware efficiency and utilisation of standard FPGA boards with high speed-up factors by implementing a specialised cache to exploit temporal and optimally exploit spacial data locality. A special traversal scheme is presented that optimally interacts with the new memory hierarchy.

3 THE ALGORITHM

In this section we will quickly recap the algorithm of hierarchical collision detection, and a special kind of bounding volume, the *k*-DOP .

3.1 Hierarchical Collision Detection using *k*-DOPs and SAT

In this paper we use hierarchical collision detection. This avoids checking every triangle of an object O for intersection with all triangles of object Q. The acceleration data structure is a bounding volume hierarchy (BVH), where each leaf corresponds to one triangle and inner nodes correspond to groups of triangles. Each node has a bounding volume (BV) attached that bounds all triangles associated with it. In order to achieve a feasible hardware design, we use a binary tree here, but n-ary trees could be considered as well. If two objects are checked for intersection, both hierarchies are traversed simultaneously. If their BVs intersect, the next level of BVs is checked. In this work, we use k-DOPs as BVs because they were proven to yield very fast collision queries by extensive benchmarking in software ([22]), and performed very well in our hardware studies ([14, 7, 18, 17]), too.

k-DOPs

All k-DOPs are defined over a fixed set $\{\mathbf{D}_1, \ldots, \mathbf{D}_{k/2}, \mathbf{D}_{k/2+1}, \ldots, \mathbf{D}_k\}$ of vectors in $\mathbb{R}^3$. Each vector $\mathbf{D}_i$ is antiparallel to $\mathbf{D}_{i+k/2}$.

A single k-DOP is defined by k distances d_i, one along each vector $\mathbf{D}_i$, thus defining a half-space. The $(d_1, \ldots, d_k)$ define the distances of the associated halfspaces to the origin.

The intersection of these halfspaces forms the BV:

$$\mathrm{DOP} = \bigcap_{i=1,\ldots,k} H_i, \quad H_i : \mathbf{D}_i \mathbf{x} - d_i \leq 0 \tag{1}$$

The orientation matrix D, of all vectors $\mathbf{D}_i$, is fixed and equal for all objects. Hence k-DOPs are very memory-efficient: only k coefficients d_i need to be stored.

SAT for *k*-DOPs

In this article we use the Separating Axis Test (SAT) [4, 19] to check DOP-pairs for intersection. This test is applied to k-DOPs and implemented in hardware.

Applying the separating axis theorem of [4] to k-DOPs results that testing $N = (k/2) + (k/2) + (3k-6)^2$ axes for being separating axes suffices. These test axes are: the normals of the faces of both DOPs and the axes orthogonal to an edge from each polytope. The test projects both k-DOPs onto each of the test axes. If the resulting pair of intervals on an individual axis is disjoint, the DOPs must be disjoint.

If only a subset of these axes is tested, disjoint k-DOPs might be reported as intersecting. This is called a false positive.

Assume object O is placed relatively to object Q by rotation $\mathbf{M}$ and translation $\mathbf{T}$. Now let $(\mathbf{A}_1, \ldots, \mathbf{A}_k)$ be the orientations of the DOPs' faces shared by all DOPs in O's bounding volume hierarchy after applying rotation $\mathbf{M}$. Analogously, let $(a_1, \ldots, a_k)$ denote the DOP coefficients for DOPs of O's BVH, let $(\mathbf{B}_1, \ldots, \mathbf{B}_k)$ denote the vectors shared by all DOPs in Q's BVH, and let $(b_1, \ldots, b_k)$ denote the corresponding DOP coefficients. Now the test axes, the projection $p = \mathbf{L} \cdot \mathbf{T}$ and a correspondence vector $(j_{A,0}, j_{A,1}, j_{A,2})$ of the orientations whose faces meet in the point that maps onto the minimal interval border with respect to $\mathbf{L}$ can be precomputed. The mapping vectors $\mathbf{P}_A$ and $\mathbf{P}_B$ that map $(a_{j_{A,0}}, a_{j_{A,1}}, a_{j_{A,2}})$ onto the minimal interval border can also be precomputed from that.

Calculating the actual projection is now reduced to calculating

$$\begin{aligned} a_{\min} &= \begin{pmatrix} \mathbf{a}_{j_{A,0}} & \mathbf{a}_{j_{A,1}} & \mathbf{a}_{j_{A,2}} \end{pmatrix} \cdot \mathbf{P}_A \\ a_{\max} &= \begin{pmatrix} \mathbf{a}_{j_{A,0}+k/2} & \mathbf{a}_{j_{A,1}+k/2} & \mathbf{a}_{j_{A,2}+k/2} \end{pmatrix} \cdot (-\mathbf{P}_A) \end{aligned} \tag{2}$$

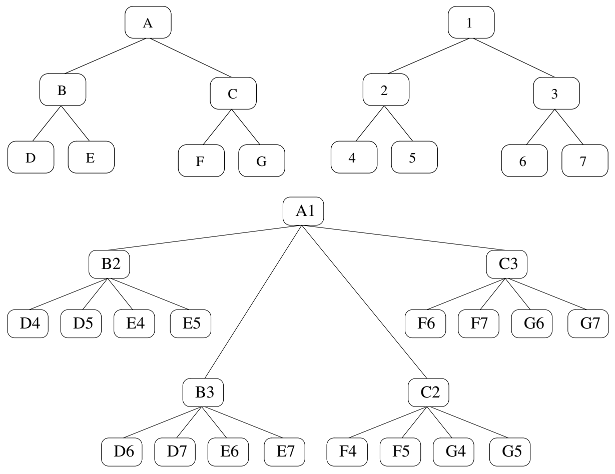

Figure 1: Bounding-volume hierarchies of two objects and the resulting test tree.

This is done for $b_{\min}$ and $b_{\max}$ analogously.

Now the condition for separation is straight-forward. Let

$$\begin{aligned} \text{diff}_1 &= (a_{\min} + p) - b_{\max} \\ \text{diff}_2 &= b_{\min} - (a_{\max} + p) \end{aligned} \tag{3}$$

$$\text{diff} = \max(\text{diff}_1, \text{diff}_2) \tag{4}$$

then the intervals $[a_{\min}, a_{\max}]$ and $[b_{\min}, b_{\max}]$ are disjoint if and only if diff > 0. And from the Separating Axis Theorem we know that

$$(\text{diff} > 0) \Rightarrow \text{separation}. \tag{5}$$

Computations shown in Eqs. (2)–(5) need to be done for each DOP test, and hence cannot be precomputed.

These calculations were implemented for 24-DOPs in SystemC in abstract style for early performance evaluation as well as in VHDL for implementation on a Xilinx Virtex II FPGA (see Sec. 5.3). They were implemented as a nine-staged pipeline. A formal correctness proof that no false positives can occur and calculation of bounds on the number of false positives can be found in [7].

3.2 Control

Fig. 1 shows two BVHs and the resulting test tree. As discussed in [15] the test tree is traversed in the following ”optimised brotherhood” order:
A1 - B2 B3 C3 C2 - D4 D5 E5 E4 - D6 D7 E7 E6 - F4 F5 G5 G4 - F6 F7 G7 G6.
This way it is possible to mostly "keep" one DOP for the next calculation and the loading order is:
A,1,B,2,3,C,2,D,4,5,E,4,D,6,7,E,6,F,4,5,G,4,F,6,7,G,6.
It can easily be seen that this still is not optimal, since almost half of the DOP loadings could still be saved. The optimal order would be: A,1,B,2,3,C,D,4,5,E,6,7,F,G.

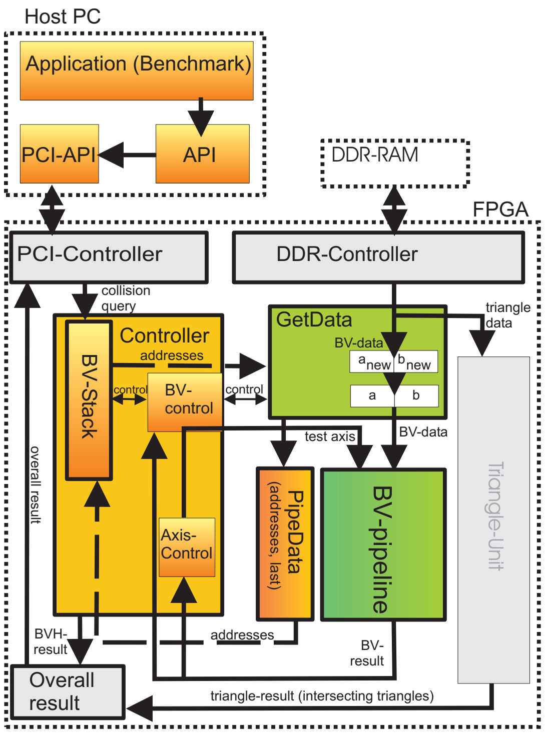

Figure 2: The overall collision detection architecture with a single BV test pipeline.

Hard wiring an order like that would demand tremendous effort, hence this potential can only be exploited using caching techniques.

As discussed in Sec. 3.1 not testing all potentially separating axes may lead to false positives. Since in case of a BV collision the algorithm descends into both BVHs and tests the sons pairwise for intersection, false positives do not alter the outcome of the calculation. The separation will still be detected later. In the extreme, if no axes are tested at all, the test proceeds to the leaves of the BVH and all primitives of one object are checked for intersection against all primitives of the other. Hence, a sufficient number of axes needs to be tested for the hierarchical collision detection to take effect.

The most straightforward form of control would be to test a fixed number of axes and abort if a separating axis is found. For the presented approach testing 24 axes was empirically found to be optimal. Concurrently the next pair of DOPs is fetched from memory into some temporary registers. When the intersection test is finished, proceed to the next test. This could be called a “pull architecture” since the next pair of DOPs is read (pulled) from the registers when the test of the preceding pair is finished. This leads to stalling the memory infrastructure whenever a test needs more time than loading the next data. Another solution would be to test only as many axes as it takes to load the next data. This on the other hand would lead to pipeline

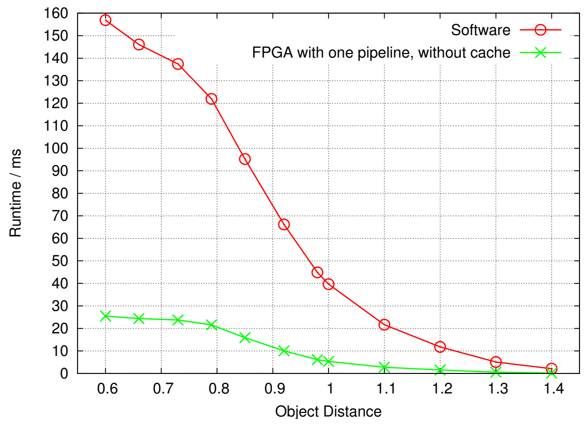

Figure 3: The architecture shown in Fig. 2 is approximately 4 times faster than a state-of-the-art software implementation running on a standard PC with comparable memory bandwidth. The runtime for each distance is an average over all orientations.

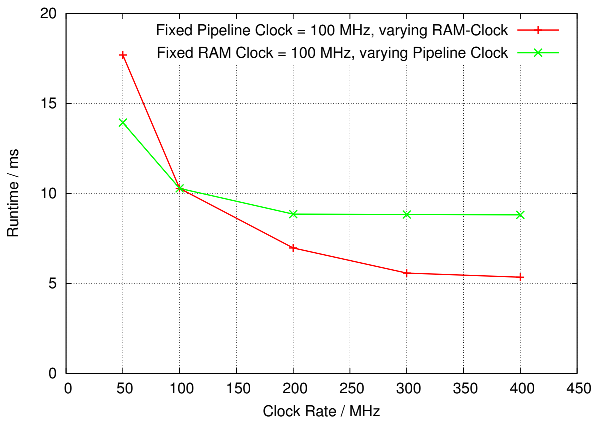

Figure 4: Influence of memory bandwidth and pipeline clock on the performance of the collision detection system. Speeding up the pipeline would not result in a significant speed-up of the overall calculation. However, increasing the memory bandwidth (done here by increasing the memory clock rate) has a much larger effect.

stalls whenever more than one DOP needs to be loaded for the next test or if triangles need to be fetched from memory.

Both, stalling the pipeline and stalling the memory can be avoided by starting a new calculation whenever a new pair of DOPs is ready to be tested. In the meantime further axes can be tested for intersection. Since finding a separating axis prevents descending deeper into the test tree the number of bounding-volume tests to be scheduled is decreased. This new approach will be called “push architecture” in the following. Here a minimum number of tests to be performed needs to be defined. Otherwise the test could run down the test tree without considering enough (or worse any) axes and this way reduce performance instead of speeding-up the calculation.

4 THE ARCHITECTURE

The complete architecture is implemented on a Xilinx Virtex II FPGA running on an Alpha Data ADMXRC board with DDR-SDRAM memory. The latter offers a maximum overall bandwidth of 2.1 GB/s. The FPGA board is connected via PCI interface to a host PC where the calling application is running.

Fig. 2 shows a block diagram of the architecture. The application can schedule tests via an `API`, which sends the request via `PCI` communication to the test `controller`. The `controller` schedules the test of the root bounding-volumes of the BVHs by pushing their addresses onto its internal `BV-Stack`. The most recently pushed BV-pair addresses are popped from the stack and the DOP coefficients are fetched by the `DDR-Controller` from the DDR-RAM. The `GetData` module decides whether triangle or DOP data was loaded and routes them to the according test pipeline. In case of DOP coefficients the current BV test is finished and a new one is started. The `Axis-Control` module provides the pipeline with the vector of the first test axis. It requests testing of further axes as long as no new DOP data is present. The `PipeData` module contains bookkeeping information of the tests within the `BV pipeline` that enables the controller to decide whether an axis test that leaves the pipeline is the last one of a DOP-pair. Additionally, it contains the son addresses of the DOPs that were tested, so they can be scheduled pairwise for intersection testing into the `BV-Stack`, if necessary. If no more BV-pairs need to be tested and all requested triangle tests are finished an `Overall result` module joins the results of the BVH test and the triangle test and reports them back to the host via the PCI-Controller. As will be discussed in the next section memory bandwidth is the main bottleneck in the design. Additionally, space is a vital resource in FPGA development. Therefore a SAT-based triangle intersection test was implemented in `Triangle-Unit`, that uses minimal bandwidth and can be implemented very resource efficiently.

5 THE MEMORY HIERARCHY

The benchmark used in the following places two different objects at different distances of their centers and with different rotations. For each constellation, the time to detect all intersecting triangles is determined. The objects intersected in nearly all object configurations used.

As shown in Fig. 3 the presented architecture is approximately 4 times faster than the state-of-the-art software implementation proposed in [23] running on a standard PC with comparable memory bandwidth. Using massively parallel calculations done on the FPGA

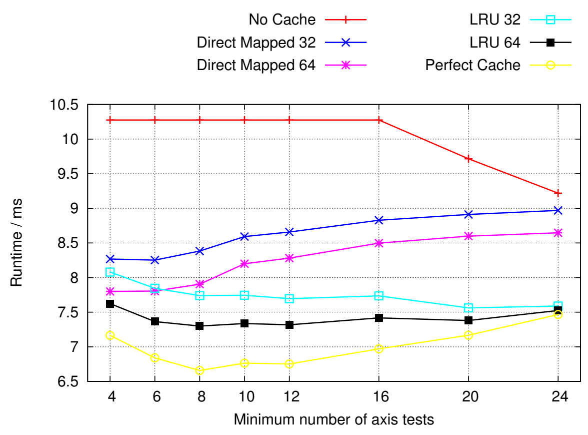

Figure 5: Comparison of the influence of different caching techniques on the performance of a single DOP pipeline. Numbers mark the cache size in number of cacheable 24-DOPs.

this is not a satisfying result. It results from the fact, that this comparison is still unfair, since the PC is equipped with a standard memory hierarchy of several caches. This indicates that a memory bottleneck exists.

The results of a systematic investigation on the influence of the memory bandwidth are shown in Fig. 4. It shows that speeding up the pipeline (or implementing multiple pipelines) without increasing memory bandwidth would not result in a significant speed-up of the calculation. However, increasing the memory bandwidth (which is done by increasing the memory clock here) has a much larger effect. Since when using a standard FPGA board the memory interface to the DDR-SDRAM itself is fixed, using caches is the only remaining solution. As discussed in Section 3.2 there still exists potential for optimising the loading order of DOPs from DDR-RAM that can be exploited using caching techniques.

5.1 Comparing Caching Techniques

Spacial Locality For cache implementation the main challenge is to determine how to exploit spacial and temporal locality of data for the problem at hand. For spacial locality it suffices to determine the optimal block size. In case of the presented collision detection hardware this is easy, since only complete DOPs are used. Additionally, the memory infrastructure never runs idle in case of a push architecture. Hence, it is obvious that loading complete bounding-volumes in one burst and storing them as a cache block is the optimal way to exploit spacial data locality.

Temporal Locality To most efficiently exploit temporal locality it is necessary to determine the caching strategy best suited for the problem at hand.

Fig. 5 shows a comparison of the influence of different caching techniques on the performance of a single DOP pipeline obtained in simulation. The span between *No Cache* and *Perfect Cache* indicates the amount of theoretically achievable savings measured in clock cycles. The perfect cache saves any data it has ever seen and never needs to replace any of it. Implementing such a structure is somewhat pointless, since it would require a tremendously large cache. It serves as a reference only. The figure shows that for the presented collision detection hardware with only a single collision detection pipeline over 35% speed-up can theoretically be achieved.

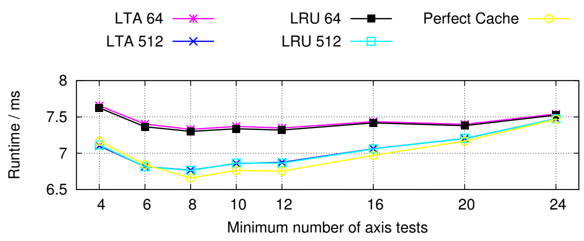

Figure 6: Comparison of a fully associative LRU cache and the lockable two-way associative cache (LTA). LTA and LRU with 64 cache entries each perform nearly equally well. LTA 512 performs close to the theoretical optimum. An LRU implementation of this size would be prohibitively expensive.

Also shown are two more realistic caching strategies. Firstly, a fully associative cache with least recently used (LRU) strategy is investigated. It qualitatively behaves like the perfect cache and yields good performance. But, to determine if the currently requested data block is present comparing all cache entries in parallel is necessary. Therefore, it consumes a lot of chip space. Secondly, the very simple direct mapped cache that maps each cache block to a unique entry is investigated. Though very hardware efficient it performs far below the optimum.

5.2 LTA Cache

To achieve good performance as well as space efficiency, this article presents a new caching strategy for bounding-volume hierarchies. The basic structure is shown in Fig. 7.

It implements a two-way set associative caching strategy. This is the minimum necessary to avoid two bounding-volumes to be tested for intersection to request the same cache entry. Additionally, it implements a FIFO that feeds data into the pipeline. This input FIFO is filled with data while the pipeline is processing the minimum number of axes necessary to avoid running down the test tree to quickly. These test requests can now be processed while the triangle intersection tests loads its data from the DDR-RAM. This way the cache can load data that will be processed at some point in the future. This is done until some test data which is still scheduled in the FIFO and needs to be processed by the pipeline first has to be replaced. To avoid double bookkeeping the FIFO only needs to contain pointers to the cache entries, not the actual data.

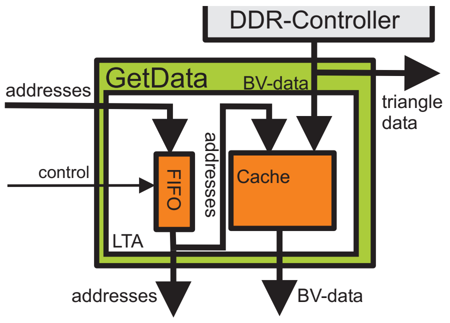

Figure 7: Basic structure of the LTA cache. To avoid double bookkeeping a FIFO contains pointers to the cache entries only. The cache itself is two-way set-associative. References to cache entries are not shown in the figure.

The only additional overhead are counters of references to the cache entries. If an entry's counter drops to zero the entry can be replaced, otherwise it is "locked". If a new entry needs to replace a locked one, the memory controller still has to be stalled. But this occurs in less than 0.0008% of the cases. Hence, no further speed-up is to be expected by increasing the associativity. Fig. 6 shows that, though very simple and yet hardware efficient, this lockable two-way set-associative cache (LTA) performs nearly as well as the fully associative approach when providing the same number of entries. Increasing the number of entries to 512 shows that the LTA caches performance is very close to the theoretical optimum. Implementing a fully associative cache of this size would be prohibitively expensive. The LTA cache's hardware consumption remains very modest, as will be shown in Sec. 5.3.

The presented caching strategy obviously collaborates efficiently with the push architecture introduced earlier in this article. As long as the FIFO is empty, more test axes are processed by the pipeline. This way the probability to find a separating axis is increased in case of a low bandwidth to the DDR-RAM (possibly temporarily caused by high emergence of triangle tests). This decreases the number of requested DOP data, since in case of separation this subtree of the test tree is not descended any further. This way the memory bottleneck is further relaxed.

5.3 Performance Evaluation and Synthesis Results

The LTA-cache was implemented in VHDL and synthesised for the target FPGA architecture. Integrating it into the collision detection architecture described in Sec. 4 results in an additional resource consumption of only 7% of the slices and 5% of the slice flip flops. Despite performing close to the theoretical optimum it consumes only 33 out of 144 Block RAMs.

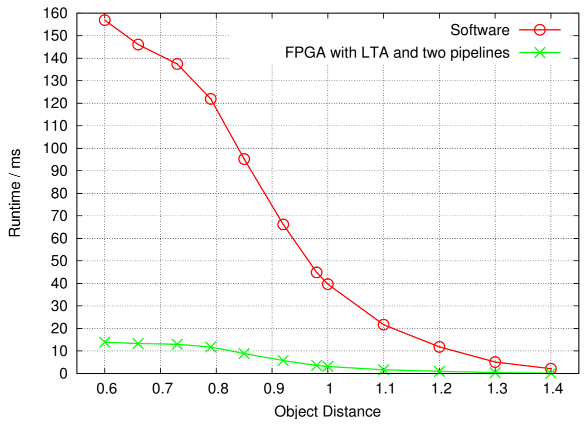

Figure 8: A design with two pipelines and a DDR-RAM, both running at only 100MHz, and implementing an LTA cache in conjunction with the push architecture is more than 10 times faster than a standard PC with a comparable memory bandwidth.

Using the LTA cache in conjunction with the push architecture to effectively decrease the number of memory accesses during hierarchical collision detection not only speeds-up calculation by a factor of about 35% when using a single pipeline. It enables parallel implementation of several pipelines in parallel. Fig. 8 shows that a design with two pipelines and a DDR-RAM, both running at only 100MHz, is over 10 times faster than a standard PC with a comparable memory bandwidth, while still fitting onto a single FPGA.

6 CONCLUSION

A novel approach for hardware-accelerated high-speed collision detection was presented in this article. It focuses on dedicated hardware for collision detection queries and its interaction with the memory interface. A specialised tree-traversal algorithm, the push architecture, was presented that exploits arbitrary memory interfaces optimally to avoid both pipeline and memory stalls during the calculation. Along with this the novel LTA cache was introduced that combines high-speed access to the bounding-volume hierarchy with minimal resource consumption. It was shown that the newly proposed caching strategy performs close to the theoretical optimum and enables concurrent use of multiple BV test pipelines. Its full functionality was tested on-chip. Simulation and synthesis results were presented that prove the conjunction of LTA cache and push architecture to enable real-time collision queries at rates required by force-feedback while fitting onto a standard field-programmable gate array. It outperforms state-of-the art software algorithms running on a standard PC with a comparable memory interface by an order of magnitude.

7 FUTURE WORK

Among the next steps will be the integration of the collision detection hardware API into a scene graph library. To further speed-up the processing of collision queries we plan on developing a custom FPGA board providing a multi-stage memory hierarchy and a state-of-the-art DDR2-RAM. Additionally, we will investigate on the effect of caching primitive data as well, to further increase effective memory bandwidth. Along the same lines additional performance could be gained by compressing both, bounding-volume and primitive data.

Another important topic is collision detection of deformable objects. It remains an open problem which data structures and algorithms are suited best for hardware implementation.

Runtime exchange of parts of the algorithm as provided by the dynamic reconfiguration ability of current FPGAs, is currently under investigation [16] to further speed-up calculation and reduce circuit size.

REFERENCES

[1] N. Atay and B. B. John W. Lockwood. A Collision Detection Chip on Reconfigurable Hardware. In *13th Pacific Conference on Computer Graphics and Application*, 2005.

[2] N. Atay and B. B. John W. Lockwood. A Collision Detection Chip on Reconfigurable Hardware. Technical report, Washington University in St. Louis, 2005.

[3] G. Baciu and W. S.-K. Wong. Hardware-assisted self-collision for deformable surfaces. In *Proceedings of ACM Symposium on Virtual Reality Software and Technology (VRST)*, pages 129–136. ACM Press, 2002.

[4] S. Gottschalk, M. Lin, and D. Manocha. OBB-Tree: A hierarchical structure for rapid interference detection. In H. Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, pages 171–180. ACM SIGGRAPH, Addison Wesley, Aug. 1996.

[5] N. Govindaraju, M. Lin, and D. Manocha. Quick-cullide: Efficient inter- and intra- object collision culling using gpus. In *Proc. of VR 2005*, 2005.

[6] B. Heidelberger, M. Teschner, and M. Gross. Detection of collisions and self-collisions using image-space techniques. In *Proc. of WSCG'04*, pages 145–152, 2004.

[7] S. Hochgürtel, A. Raabe, G. Zachmann, and J. K. Anlauf. Collision Detection for k-DOPs using SAT with Error Bounded Fixed-Point Arithmetic. Technical report, University of Bonn, Sept. 2005.

[8] K. E. Hoff III, A. Zaferakis, M. C. Lin, and D. Manocha. Fast and simple 2d geometric proximity queries using graphics hardware. In *Symposium on Interactive 3D Graphics*, pages 145–148, 2001.

[9] P. M. Hubbard. Approximating polyhedra with spheres for time-critical collision detection. *ACM Transactions on Graphics*, 15(3):179–210, July 1996.

[10] D. Knott and D. Pai. Cinder: Collision and interference detection in real–time using graphics hardware. In *Proc. of Graphics Interface '03*, 2003.

[11] S. Krishnan, M. Gopi, M. Lin, D. Manocha, and A. Pattekar. Rapid and accurate contact determination between spline models using ShellTrees. *Computer Graphics Forum*, 17(3), Sept. 1998.

[12] T. Möller. A Fast Triangle-Triangle Intersection Test. *journal of graphics tools*, 2(2):25–30, 1997.

[13] E. Plante, M.-P. Cani, and P. Poulin. A layered wisp model for simulating interactions inside long hair. In N. M.-T. Marie-Paule Cani, Daniel Thalmann, editor, *Computer Animation and Simulation 2001Proceeding*, Computer Science. EUROGRAPHICS, Springer, Sept. 2001. Proceedings of the EG workshop of Animation and Simulation.

[14] A. Raabe, B. Bartyzel, J. K. Anlauf, and G. Zachmann. Hardware Accelerated Collision Detection — An Architecture and Simulation Results. In *Design Automation and Test (DATE)*, pages 130–135, Munich, Germany, Mar.7–11 2005. IEEE Computer Society.

[15] A. Raabe, B. Bartyzel, G. Zachmann, and J. K. Anlauf. Hardware Accelerated Collision Detection – An Architecture and Simulation Results. In *8th WSEAS International Conf. on SYSTEMS*, pages 487–321, Vouliagmeni, Athens, Greece, July12–14 2004.

[16] A. Raabe, P. A. Hartmann, and J. K. Anlauf. Rechannel: Describing and Simulating Reconfigurable Hardware in SystemC. *ACM Transactions on Design Automation of Electronic Systems (accepted)*.

[17] A. Raabe, S. Hochgürtel, G. Zachmann, and J. K. Anlauf. Hardware-Accelerated Collision Detection using Bounded-Error Fixed-Point Arithmetic. *Journal of WSCG '2006*, pages 17–24, 2006.

[18] A. Raabe, S. Hochgürtel, G. Zachmann, and J. K. Anlauf. Space-Efficient FPGA-Accelerated Collision Detection for Virtual Prototyping. In *Design Automation and Test (DATE)*, pages 206–211, Munich, Germany, 2006.

[19] G. J. A. van den Bergen. *Collision Detection in Interactive 3D Computer Animation*. PhD dissertation, Eindhoven University of Technology, 1999.

[20] S.-E. Yoon, P. Lindstrom, V. Pascucci, and D. Manocha. Cache-oblivious mesh layouts. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 886–893, New York, NY, USA, 2005. ACM Press.

[21] S.-E. Yoon, P. Lindstrom, V. Pascucci, and D. Manocha. Cache-efficient layouts of bounding volume hierarchies. In *Computer graphics forum (Eurographics)*, pages 507–516, 2006.

[22] G. Zachmann. Rapid collision detection by dynamically aligned DOP-trees. In *Proc. of IEEE Virtual Reality Annual International Symposium; VRAIS '98*, pages 90–97, Atlanta, Georgia, Mar. 1998.

[23] G. Zachmann. Minimal hierarchical collision detection. In *Proc. ACM Symposium on Virtual Reality Software and Technology (VRST)*, pages 121–128, Hong Kong, China, Nov.11–13 2002.

[24] G. Zachmann and G. Knittel. An architecture for hierarchical collision detection. In *Journal of WSCG '2003*, pages 149–156, University of West Bohemia, Plzeň, Czech Republic, Feb.3–7 2003.

[25] G. Zachmann and G. Knittel. High-performance collision detection hardware. Technical Report CG-2003-3, University Bonn, Informatik II, Bonn, Germany, Aug. 2003.