Space-Efficient FPGA-Accelerated Collision Detection for Virtual Prototyping

Andreas Raabe, Stefan Hochgürtel,
Joachim Anlauf
Technical Computer Science
Bonn University, Germany
{raabe, hochguer,anlauf}@cs.uni-bonn.de

Gabriel Zachmann
Computer Graphics
Clausthal University, Germany
zach@in.tu-clausthal.de

Abstract

We present a space-efficient, FPGA-optimized architecture to detect collisions among virtual objects. The design consists of two main modules, one for traversing a hierarchical acceleration data structure, and one for intersecting triangles. This paper focuses on the former.

The design is based on a novel algorithm for testing discretely oriented polytopes for overlap in 3D space. In addition, we derive a new overlap test algorithm that can be implemented using fixed-point arithmetic without producing false negatives and with bounded error.

SystemC simulation results on different levels of abstraction show that real-time collision detection of complex objects at rates required by force-feedback and physically-based simulations can be obtained. In addition, synthesis results show that the design can still be fitted into a six-million gates FPGA. Furthermore, we compare our FPGA-based design with a fully parallelized ASIC-targeted architecture and a software implementation.

1. Introduction

Detecting collisions between a pair of graphical objects is a fundamental task in many areas such as physically-based simulation, automatic path finding, or tolerance checking. Applications are in games, animation systems, and virtual reality, e.g., virtual assembly simulation, or medical training and planning systems.

In most of the applications in these areas, the goal is to avoid collisions, or to enable real-time physically-based simulation. Most approaches today are reactive, i.e., they first place objects at a new trial position, check for collisions, and then compute new forces or positions, based on physical laws, so as to remove any collisions.

This approach demands very efficient collision detection, because it must perform many collision checks per simulation cycle. An emerging application area is the mobile devices market (smart phones, portable games devices). Here, the challenges, besides speed, are size and energy consumption. Another particularly demanding application is force-feedback, where updates of about 1000Hz must be done in order to achieve stable force computations.

Since collision detection is such a fundamental yet challenging task, it is highly desirable to have hardware acceleration available just like 3D graphics accelerators. The benefit is two-fold: a) the system can process objects with higher polygon counts, and b) the system's CPU can be freed from computing collisions.

In this paper, we present a novel, efficient architecture for hierarchical collision detection of two rigid objects. It is based on a novel algorithm for testing a pair of bounding volumes for overlap, which can even be implemented on fixed-point arithmetic. We present an implementation on FPGA hardware along with simulation results concerning its speed and synthesis results concerning its size. In addition, we compare these results with an earlier, parallelized ASIC-targeted architecture, and with a software implementation.

2. Related Work

Considerable work has been done on hierarchical collision detection in software [3, 4, 7, 11, 12]. Some of the bounding volumes (BVs) utilized are spheres, axis-aligned bounding boxes (AABB), oriented bounding boxes (OBB), and discretely oriented polytopes (DOP).

The first publications of work on dedicated hardware for collision detection was presented in [13, 14]. However, they presented only a functional simulation, while we present a RT level implementation along with synthesis results. [10] presented a design that was targeted on ASICs, and was optimized for speed only, and, thus, utilize a total of over 4 million gates and a 756 bits wide bus to a DDR2-RAM. Recently, a commerical hardware was announced that supposedly can do collision detection, among other things [1]. However, no details have been published, in particular, no performance results.

Most other hardware-related research has tried to utilize existing graphics accelerator boards (GPU) [2,5,6,8,9]. While earlier approaches, such as [9], can basically handle only convex objects, later algorithms, such as [2, 8], have extended these to more general cases such as unions of convex objects or closed objects. The general class of "polygon soups" can be handled by [5], but they use a hybrid approach where the graphics hardware only identifies potentially colliding sets.

All of the approaches using graphics hardware have the disadvantage that they either compete with the rendering process for the same hardware resource, or an additional graphics board must be spent for collision detection. The

3-9810801-0-6/DATE06 © 2006 EDAA

former slows down the overall frame rate considerably, while the latter would be a tremendous waste, since most of the resources of the hardware would not be utilized at all. Furthermore, most of these approaches work in image space, which reduces precision significantly.

3. The Algorithm

In this section, we will first quickly recap the algorithm of hierarchical collision detection, and of a special kind of bounding volumes, the k-DOP. Then, we will derive a new and efficient way to test DOPs for overlap, which is the heart of hierarchical collision detection. Finally, we will show how this can be done in fixed-point arithmetic such that no false negatives occur.[1]

3.1. Basics

3.1.1. Hierarchical Collision Detection and Bounding Volumes. 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 so-called *bounding volume hierarchy* (BVH), where each leave 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. Since two objects will usually intersect only in a very small number of primitives, this yields a significant speed-up in the average case.

In this work, we use k-DOPs as BVs because they were proven to yield very fast collision queries by extensive benchmarking in software [12], and performed very well in our hardware studies [10], too.

3.1.2. 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 $\mathbf{D}_{i+k/2}$.

An individual k-DOP is defined by k distances d_i, one along each vector $\mathbf{D}_i$, thus defining a half-space. These DOP coefficients $(d_1,\ldots,d_k)$ are the distances of the associated halfspaces to the origin.[2] The $k/2$ pairs of DOP coefficients $(d_i,d_{i+k/2})$ form a so-called *slab* [12].

The intersection of these slabs forms the BV:

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

[1] Here, false negatives are false reports of non-collision even though a collision does actually happening. Such false negatives would be fatal in most applications of collision detection.

[2] Note that the origin is not necessarily the center of the DOP nor even contained in it.

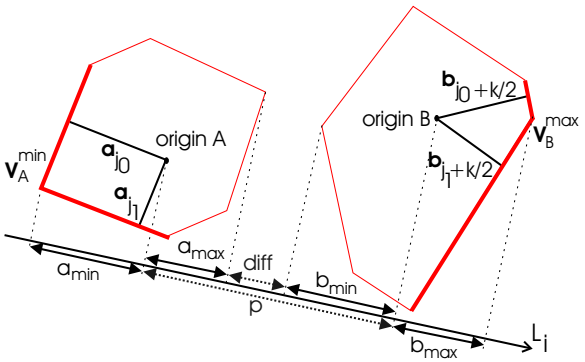

Figure 1. Two DOPs are projected onto test axis L_i. Since their images do not intersect L_i is a separating axis.

The orientation matrix D, consisting of all the vectors $\mathbf{D}_i$, is fixed and equal for all objects. This yields a very memory-efficient description for every k-DOP: only the k coefficients d_i need to be stored.

3.2. Testing DOPs For Intersection

In this subsection, we will first describe a well-known overlap test for convex objects. Then, we will transform this into a much more efficient test for DOPs in DOP-trees. Finally, we will transform this into an even more efficient algorithm utilizing fixed-point arithmetic.

3.2.1. Separating Axis Test (SAT). In this paper we use the so called Separating Axis Test (SAT) [4, 11] to check DOP-pairs for intersection, yielding a highly space-efficient collision test hardware. To our knowledge, we are the first to apply this test to DOPs and to implement it in hardware.

[4] have shown that two convex polytopes are disjoint if and only if there exists a separating axis orthogonal to a face of either polytope or orthogonal to an edge from each polytope (Separating Axis Theorem). Hence for two k-DOPs there are $N = (k/2) + (k/2) + (3k-6)^2$ potentially separating axes. If only a subset of these axes are tested, false positives might occur, i.e., the DOPs are disjoint while the (incomplete) test yields an intersection. The complete SAT is always correct.

To perform the test, both DOPs must be projected onto each of the candidate separating axes. For each axis, a pair of intervals on that axis results. If one of these pairs is disjoint, then the DOPs must be disjoint (see Fig.1).

Since with DOPs the set of vectors $\{D_1,\ldots,D_k\}$ is fixed, we can exploit that all possible face orientations of the DOPs within a DOP-tree are the same.

Assume object O is placed relatively to object Q by rotation $\mathbf{M}$ and translation $\mathbf{T}$. Let $\mathrm{DT}(O)$ and $\mathrm{DT}(Q)$ denote the DOP-trees of these objects. As described in Section 3.1.2, let $(\mathbf{A}_1,\ldots,\mathbf{A}_k)$ be the orientations of the DOPs' faces shared by all DOPs in $\mathrm{DT}(O)$ after applying rotation $\mathbf{M}$. Analogously, let $(a_1,\ldots,a_k)$ denote the DOP coefficients for DOPs in $\mathrm{DT}(O)$, let $(\mathbf{B}_1,\ldots,\mathbf{B}_k)$ denote the

vectors shared by all DOPs in $\mathrm{DT}(Q)$, and let $(b_1,\ldots,b_k)$ denote the corresponding DOP coefficients.

3.2.2. Efficient SAT. Note that everything independent of $(a_1,\ldots,a_k)$ and $(b_1,\ldots,b_k)$ is constant throughout the whole DOP-trees. Hence it can be precalculated at startup to initialize the algorithm (and, later-on, the hardware). Precomputing as much as possible significantly reduces the resulting hardware costs. Since this is done only once per pair of DOP-trees, it is not time-critical.

First, we can precompute the n test axes $\mathbf{L}_i$. All of the following is done for each L_i, so for the sake of readability we omit the index i from now on.

Second, the projection $p = \mathbf{L}\cdot\mathbf{T}$ is precomputed.

Third, for each $\mathbf{L}$ a DOP has two vertices $\mathbf{v}_A^{\min}$ and $\mathbf{v}_A^{\max}$ whose projections onto $\mathbf{L}$ have maximum distance. Each of those vertices is formed by the intersection of three faces of the DOP. The correspondences $(j_{A,0}, j_{A,1}, j_{A,2})$ of the orientations whose faces meet in $\mathbf{v}_A^{\min}$ are calculated.

Fourth, and most importantly, in the actual projection

$$\begin{aligned} a_{\min} &= \mathbf{v}_A^{\min}\cdot\mathbf{L} \\ &= \begin{pmatrix} a_{j_{A,0}} & a_{j_{A,1}} & a_{j_{A,2}} \end{pmatrix}\cdot\begin{pmatrix} \mathbf{A}_{j_{A,0}} & \mathbf{A}_{j_{A,1}} & \mathbf{A}_{j_{A,2}} \end{pmatrix}^{-1}\cdot\mathbf{L} \end{aligned}$$

we can precompute the last dot product

$$\mathbf{P}_A := \begin{pmatrix} \mathbf{A}_{j_{A,0}} & \mathbf{A}_{j_{A,1}} & \mathbf{A}_{j_{A,2}} \end{pmatrix}^{-1}\cdot\mathbf{L} \tag{2}$$

$\mathbf{P}_B$ can be precomputed analogously. The mapping vectors for $\mathbf{v}_A^{\max}$ and $\mathbf{v}_B^{\max}$ are $-\mathbf{P}_A$ and $-\mathbf{P}_B$ respectively. This exploits that $k/2$ pairs of DOP orientations are anti-parallel.

3.2.3. Intersection Testing. Using these precomputations, we can project onto the test axes very efficiently:

$$\begin{aligned} a_{\min} &= \begin{pmatrix} \mathbf{a}_{j_{A,0}} & \mathbf{a}_{j_{A,1}} & \mathbf{a}_{j_{A,2}} \end{pmatrix}\cdot\mathbf{P}_A \\ a_{\max} &= \begin{pmatrix} \mathbf{a}_{j_{A,0}+k/2} & \mathbf{a}_{j_{A,1}+k/2} & \mathbf{a}_{j_{A,2}+k/2} \end{pmatrix}\cdot(-\mathbf{P}_A) \end{aligned} \tag{3}$$

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

Now the condition for separation is straight-forward. Let

$$\begin{aligned} \mathrm{diff}_1 &= (a_{\min}+p) - b_{\max} \\ \mathrm{diff}_2 &= b_{\min} - (a_{\max}+p) \end{aligned} \tag{4}$$

$$\mathrm{diff} = \max(\mathrm{diff}_1, \mathrm{diff}_2) \tag{5}$$

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

$$(\mathrm{diff} > 0) \Rightarrow \text{separation}. \tag{6}$$

Eqs. (3)–(6) show the computations that need to be done for each DOP test (and hence cannot be precomputed).

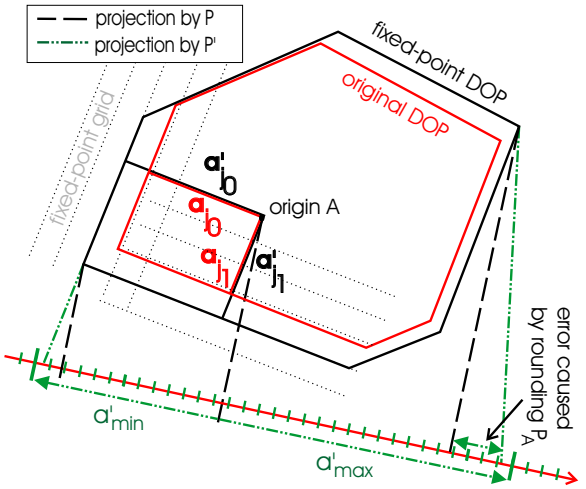

Figure 2. A floating-point DOP and its enclosing fixed-point equivalent. Both rounding the DOP to fixed-point numbers and projecting it with $\mathbf{P}'$ instead of $\mathbf{P}$ increases the DOP's image. When checked for intersection false positives can occur.

3.2.4. Fixed-Point Arithmetic. Floating-point arithmetic is very expensive with respect to size, especially in an FPGA implementation. Hence it is desirable to do fixed-point calculations.

Unfortunately, simply rounding the DOP coefficients to fixed-point numbers would result in false negatives, because the intervals on the test axes could become smaller than the projection of the enclosed object. False negatives are inacceptable, because we might miss collisions. Naïve rounding of the mapping vectors $\mathbf{P}_A$ and $\mathbf{P}_B$ would lead to even more false negatives. Hence we need to round in a manner such that each fixed-point DOP contains its floating-point DOP (see Fig. 2).

First, it is necessary to cope with the bigger scale of floating-point numbers compared with fixed-point numbers by dividing all DOP coefficients of all DOPs by the largest absolute value of the DOP coefficients in the scenario. This way, 16 bit accuracy still allows for having DOPs the size of a skyscraper and of a 6mm screw. 36 bit even allow for DOPs the size of the sun and of a screw.

Let the rounding of the DOP coefficients to b bits towards $+\infty$ be denoted by $a_i' = \lceil a_i \rceil$. Clearly, the rounded (i.e., fixed-point) DOP contains the original one. Then, $\varepsilon_i = a_i' - a_i$ is the resulting rounding error, with $0 \le \varepsilon_i < 2^{-b}$.

By ensuring that the dihedral angle between all pairs of neighboring faces of a DOP is larger than $\pi/2$, all $P_{A,i}$ are in the interval $[-1,0]$. [3] Rounding $\mathbf{P}_{A,i}$ towards $-\infty$ to c bit accuracy results in a rounding error $0 \le \eta_i = \mathbf{P}_{A,i} - \mathbf{P}'_{A,i} < 2^{-c}$.

[3] This is not really a hard restriction since every well constructed DOP does not have acute angles to improve tightness of fit.

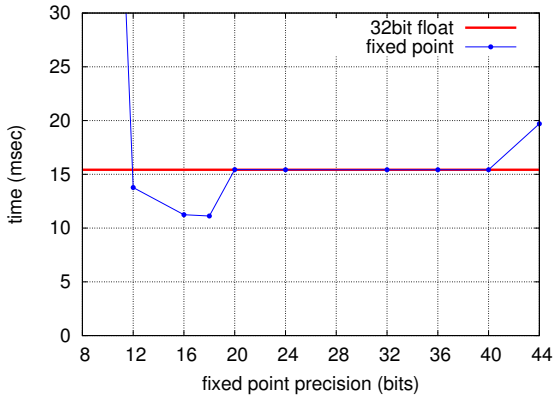

Figure 3. Speed of fixed-point arithmetic for different bit widths. Beyond 18 bits a second, and beyond 40 bits a third memory burst is needed.

By simply truncating $\mathbf{P}_{A,i}$, the resulting image would become too small in case of negative DOP coefficients, whereas always rounding up would create the same problem with positive coefficients. Fortunately, we can solve this during calculation simply by adding 2^{-c} to $\lfloor \mathbf{P}_{A,i} \rfloor$ before multiplication with negative DOP coefficients.

Let $\mathbf{a}' = (a'_{j_{A,0}} a'_{j_{A,1}} a'_{j_{A,2}})$, $\mathbf{a}'_k = (a'_{j_{A,0}+k/2} a'_{j_{A,1}+k/2} a'_{j_{A,2}+k/2})$, and let $\mathrm{sn}(\mathbf{x})$ be the sum of all $\mathbf{x}_i < 0$.

Then, correct rounding of the images amounts to:

$$\begin{aligned} a'_{\min} &= \mathbf{P}'_A \cdot \mathbf{a}' + 2^{-c}\,\mathrm{sn}(\mathbf{a}') \\ a'_{\max} &= -(\mathbf{P}'_A \cdot \mathbf{a}'_k + 2^{-c}\,\mathrm{sn}(\mathbf{a}'_k)) \end{aligned} \tag{7}$$

Finally, when computing diff_1, we can simply truncate p to z bits ($p' = \lfloor p \rfloor$). This can create only false positives, because a smaller p' only decreases the apparent distance between the two DOP images. For diff_2 we need to round p up to $\lceil p \rceil$, which, again, can be done efficiently by adding 2^{-z} to $\lfloor p \rfloor$.

Overall, calculating the distances of the fixed-point DOP images amounts to

$$\begin{aligned} \mathrm{diff}'_1 &= (\mathbf{a}'_{\min} + p') - \mathbf{b}'_{\max} \\ \mathrm{diff}'_2 &= \mathbf{b}'_{\min} - (\mathbf{a}'_{\max} + (p' + 2^{-z})) \end{aligned} \tag{8}$$

$$\mathrm{diff}' = \max(\mathrm{diff}'_1, \mathrm{diff}'_2) \tag{9}$$

Now the condition for separation can be given analogously to Eq. 6:

$$((\mathrm{diff}'_1 > 0) \text{ or } (\mathrm{diff}'_2 > 0)) \Rightarrow \text{separation}. \tag{10}$$

It can be shown by simple arithmetic calculus that

$$(\mathrm{diff}' - \mathrm{diff}) \leq (\sqrt{3} \cdot 2^{-b+1} + 6 \cdot 2^{-c} + 2^{-z}) \tag{11}$$

Simulations done early in the design process used the target hardware described in Section 5 and showed that fixed-point accuracy influences calculation time (Fig. 3). Below 18 bits accuracy, an increasing number of false positives occurs and decreases calculation speed. Above 18 bits, a second memory burst is needed to fetch DOP coefficients from DDR-RAM.

3.3. Triangle Intersection

Once the BVH traversal reaches two leaves, we need to test the enclosed triangles for intersection. Here, we utilize the same algorithm that was already proposed in [13] and implemented in VHDL in [10]. It transforms both triangles so that one of them becomes the "unit" triangle. That way, the checks to be performed on the other triangle become very simple and standardized. In this paper the triangle intersection test module remains unchanged, but it is still subject to further investigation.

4. The Architecture

4.1. The Pipeline

Combining Eqs. (7)–(10) results in the overlap condition

$$\begin{aligned} &(\mathbf{P}'_A \cdot \mathbf{a}' + 2^{-c}\,\mathrm{sn}(\mathbf{a}') + \mathbf{P}'_B \cdot \mathbf{b}'_k + 2^{-c}\,\mathrm{sn}(\mathbf{b}'_k) + p') > 0 \\ &\qquad\qquad \text{or} \\ &(\mathbf{P}'_B \cdot \mathbf{b}' + 2^{-c}\,\mathrm{sn}(\mathbf{b}') + \mathbf{P}'_A \cdot \mathbf{a}'_k + 2^{-c}\,\mathrm{sn}(\mathbf{a}'_k) + \overline{p'}) > 0 \\ &\qquad\qquad \Rightarrow \text{separation} \end{aligned} \tag{12}$$

Note that using two's complement numbers $-(p' + 2^{-z})$ can be calculated efficiently by simple bitwise negation $\overline{p'} = -(p' + 2^{-z})$.

Eq. (12) is divided into seven stages to enable pipelining.

Selection: Stage one selects the 12 out of k DOP coefficients defining the outer (maximal) vertices for a given candidate separating axis based on the correspondences (j_A, j_B).

Scalar Products and Fixed-Point Correction: Stages two to five implement the calculation of the scalar products and the fixed-point correction term. So, DOP coefficients have to be multiplied by $\mathbf{P}'$-vector entries and summed up by an adder tree. Additionally, p' ($\overline{p}'$ in case of diff'_2) is added. Concurrently, negative DOP coefficients are selected and accumulated. Stage six adds the results of both summations. Multiplying by 2^{-c} is done implicitly by shifting.

Result: Testing $\max(\mathrm{diff}'_1, \mathrm{diff}'_2) > 0$ is done by negating the conjunction of the sign bits.

4.2. Overall Design

The overall architecture is shown in Fig. 4. The calculation is initialized by the host system by sending $(\mathbf{P}'_A, \mathbf{P}'_B, p', j_A, j_B)$ and the addresses of the DOP-trees to the hardware. A controller keeps track of DOP overlap tests that must still be executed and requests the needed

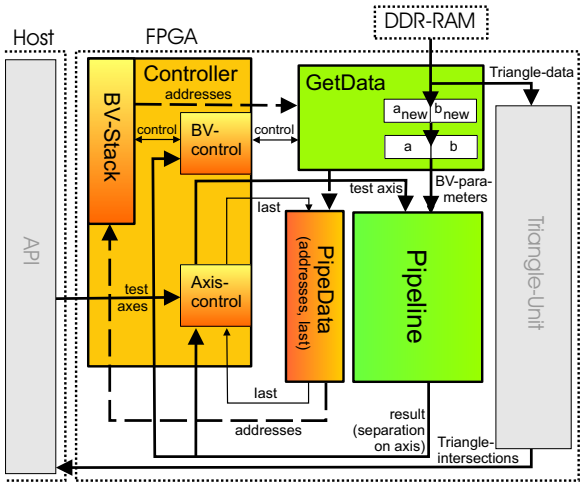

Figure 4. The complete intersection test hardware.

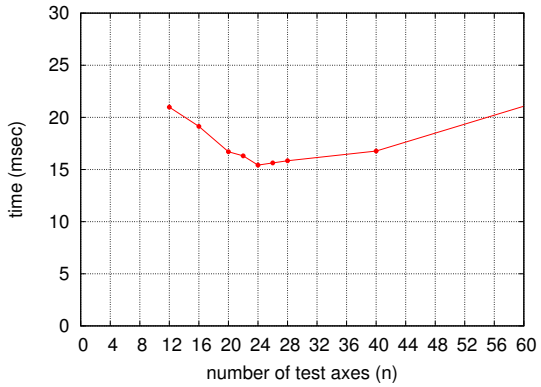

Figure 5. For a fixed $k = 24$ our design performs best using $n = 24$ on the target architecture.

DOP coefficients and triangle data. The module "GetData" reads them concurrently from memory to the current calculation. As soon as the parameters are loaded and the last calculation is finished, it feeds them into the pipeline (or the triangle-unit respectively). The pipeline receives not only the DOP coefficients but (from the controller) the data for the next axis test. For each DOP pair, n axes are tested. A shift register ("PipeData") holds additional bookkeeping information. For every pipeline stage it contains the indices of the processed DOPs and whether the contained calculation is the last axis test to be executed for the current DOP pair. If this last axis test leaves the pipeline and none of the test axes is a separating axis the controller schedules the child DOPs to be tested. If a separating axis is found, the remaining calculations belonging to the same DOP-pair are obsolete. No new axis tests are initiated and the results of the calculations that are still in the pipeline will be ignored; no new DOP tests are scheduled.

[10] showed that scheduling DOP tests in a stack is far superior to queue control with regards to memory usage. So, as soon as the stack, pipeline, and the GetData module are empty, and no intersecting triangles were found, the objects do not intersect and this is reported to the host application. On the other hand, every intersecting pair of triangles is reported to the host immediately.

4.3. Control

As mentioned in Section 3.2.1, it is not necessary to test all axes L_i whether they are separating axes. Even more, [11] has shown that it is not efficient to test all axes for OBBs since the probability that the BVs are disjoint decreases rapidly with every non-separating test axis found so far.

On the other hand, we want to eliminate disjoint branches of the DOP trees as early as possible to reduce expensive loading of DOP coefficients. Therefore, we determined which $n \leq N$ gives the best trade-off between axis-testing and parameter-loading. As Fig. 5 indicates, $n = 24$ yields the optimum performance for 24-DOPs and the given memory architecture. 24 axis-tests suffice to test all candidate separating axes generated from the 12 face-orientations of each DOP. Although this exceeds the time to load a complete set of DOP-coefficients (only 20 clock-cycles) by 4 cycles, testing 24 axes seems to reduce the number of false positives enough to yield a performance gain.

5. Results

The target architecture is a Xilinx Virtex II (XC 2V6000, speed grade -4) on an Alpha Data ADM-XRC-II board with 256 MB DDR-RAM at 100MHz. The FPGA features 144 18-bit multipliers and 6 million gate equivalents. CoCentric from Synopsys was used to compile SystemC RTL to VHDL code. Synthesis, Place, Route and Mapping were done with Xilinx ISE 6.3.

5.1. Synthesis Results

Although 18-bit accuracy performs best on our test data (Fig. 3), we decided to implement the pipeline for 35 bits fixed-point 24-DOPs to tolerate bigger differences in DOP size (see Section 3.2.4). Since the target architecture features 18-bit multipliers only, this results in two extra pipeline stages to implement 35-bit pipelined multipliers.

Overall, a total of 7278 out of 33792 slices ($21\% = 1,260,000$ million gate equivalents) is utilized by the pipeline. Maximum clock frequency is 111.117MHz.

5.2. Benchmarking

All results presented here were obtained with two identical objects (a car headlight) with 5947 triangles ([10]). They

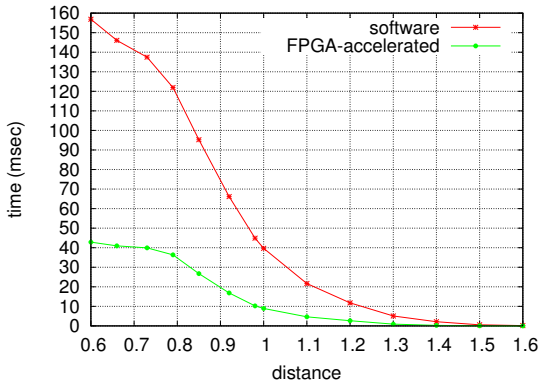

Figure 6. The presented architecture is approximately 4 times faster than a state-of-the-art software intersection test.

are placed at different distances from each other and with different rotations. For each constellation, the time to detect all intersecting triangles is determined. Fig. 6 shows the comparison of our new architecture with a state-of-the-art software intersection test running on a 1 GHz Pentium III with 512 MByte main memory. The presented acceleration hardware yields a speed up of about factor 4.

6. Conclusion and Future Work

We have presented a novel algorithm for hierarchical collision detection of pairs of virtual objects. We have also presented a highly space-efficient, FPGA-optimized architecture implementing this algorithm on an FPGA using fixed-point arithmetic. The fixed-point calculations do not produce any false negatives, and we have given bounds on the deviations from floating-point arithmetic.

Simulation results for collision queries using this architecture proved that a speed-up of 4 compared to state-of-the-art software intersection tests on a standard CPU can be obtained. Taking earlier ASIC-targeted results into account [10], we conclude that an ASIC implementation of our novel algorithm and architecture will perform by one or two orders of magnitude faster than a software implementation. Synthesis result proved the design to be highly space-efficient.

In addition, our novel DOP overlap test algorithm lends itself well to parallelization. Only a slight modification of the controller is necessary to use multiple pipelines to test multiple candidate separating axes in parallel. The pipeline utilizes only 21% of the chip area so multiple instances are possible. Here, memory bandwidth becomes the limiting factor for speed of collision queries. Possible solutions could be compression of DOP coefficients and the introduction of a cache.

Another important topic is fixed-point accuracy. Here, a lot of different ways to get smaller projections are conceivable.

Collision detection of deformable objects is another important issue. It remains an open problem, which algorithms and data structures are best suited for hardware implementation. Furthermore, we will evaluate different kinds of primitives like quadrangles and NURBS.

References

[1] Ageia. White paper, May 2005.

[2] G. Baciu, W. S.-K. Wong, and H. Sun. RECODE: an image-based collision detection algorithm. *The Journal of Visualization and Computer Animation*, 10(4):181–192, October - December 1999. ISSN 1049-8907.

[3] J. Eckstein and E. Schömer. Dynamic collision detection in virtual reality applications. In *WSCG'99*, pages 71–78, Plzen, Czech Republic, Feb. 1999. University of West Bohemia.

[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. held in New Orleans, Louisiana, 04-09 August 1996.

[5] N. K. Govindaraju, S. Redon, M. C. Lin, and D. Manocha. CULLIDE: Interactive collision detection between complex models in large environments using graphics hardware. In *Graphics Hardware 2003*, pages 25–32, July 2003.

[6] A. Gress and G. Zachmann. Object-space interference detection on programmable graphics hardware. In M. L. Lucian and M. Neamtu, editors, *SIAM Conf. on Geometric Design and Computing*, pages 311–328, Seattle, Washington, Nov.13–17 2003. Nashboro Press.

[7] P. M. Hubbard. Collision detection for interactive graphics applications. *IEEE Transactions on Visualization and Computer Graphics*, 1(3):218–230, Sept. 1995. ISSN 1077-2626.

[8] D. Knott and D. K. Pai. CInDeR: Collision and interference detection in real-time using graphics hardware. In *Proc. of Graphics Interface*, Halifax, Nova Scotia,Canada, June11–13 2003.

[9] K. Myszkowski, O. G. Okunev, and T. L. Kunii. Fast collision detection between complex solids using rasterizing graphics hardware. *The Visual Computer*, 11(9):497–512, 1995. ISSN 0178-2789.

[10] 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)*, volume 3, pages 130–135, Munich, Germany, Mar.7–11 2005.

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

[12] 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.

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

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