Optimized Spatial Hashing for Collision Detection of Deformable Objects

Matthias Teschner Bruno Heidelberger Matthias Müller Danat Pomeranets Markus Gross

Computer Graphics Laboratory
ETH Zurich

Abstract

We propose a new approach to collision and self–collision detection of dynamically deforming objects that consist of tetrahedrons. Tetrahedral meshes are commonly used to represent volumetric deformable models and the presented algorithm is integrated in a physically–based environment, which can be used in game engines and surgical simulators. The proposed algorithm employs a hash function for compressing a potentially infinite regular spatial grid. Although the hash function does not always provide a unique mapping of grid cells, it can be generated very efficiently and does not require complex data structures, such as octrees or BSPs. We have investigated and optimized the parameters of the collision detection algorithm, such as hash function, hash table size and spatial cell size. The algorithm can detect collisions and self–collisions in environments of up to 20k tetrahedrons in real–time. Although the algorithm works with tetrahedral meshes, it can be easily adapted to other object primitives, such as triangles.

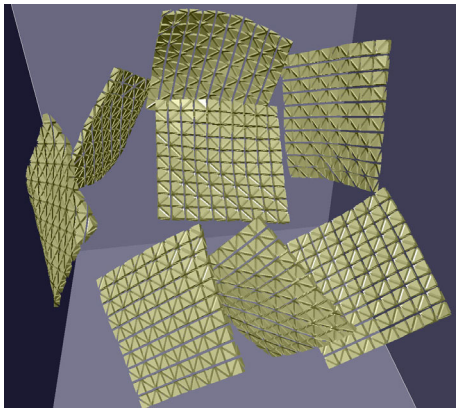

Figure 1: Environment with dynamically deforming objects, that consist of tetrahedrons.

1 Introduction

Physically–based simulations and animations of deformable objects play an important role in various research areas, such as cloth modeling, games, and computational surgery. In order to realistically process the interaction between deformable objects, very efficient collision detection algorithms are required. Further, the information provided by the collision detection approach should allow for an efficient and physically–correct collision response.

This paper describes a new algorithm for the detection of collisions and self–collisions of deformable objects based on spatial hashing. The algorithm classifies all object primitives, i. e. vertices and tetrahedrons, with respect to small axis–aligned bounding boxes AABB. Therefore, a hash function maps the 3D boxes (cells) to a 1D hash table index. As a result, each hash table index contains a small number of object primitives, that have to be checked against each other for intersection. Since a hash table index can contain more than one primitive from the same object as well as primitives from different objects, self–collisions and collisions of different objects can be detected. The actual collision detection test computes barycentric coordinates of a vertex with respect to a penetrated tetrahedron. This information can be employed to estimate the penetration depth for a pair of colliding tetrahedrons. The penetration depth can be used for further processing, such as collision response.

Using a hash function for spatial subdivision is very efficient. While spatial subdivision usually requires one pre–processing pass through all object primitives to estimate the global bounding box and the cell size, this pass can be omitted in our approach. On the other hand, the hash mechanism does not always provide a unique mapping of grid cells to hash table entries. If different 3D grid cells are mapped to the same index, the performance of the proposed algorithm decreases. In

order to reduce the number of index collisions, we have optimized the parameters of the collision detection algorithm that have an impact on this problem, namely the characteristics of the hash function, the hash table size, and the 3D cell size. The paper investigates these factors.

Further, the paper presents experimental results, that have been obtained using physically-based environments for deformable objects with varying geometrical complexity (see Fig. 1). Environments with up to 20000 tetrahedrons can be tested for collisions and self–collisions in real–time on a PC.

The remainder of the paper is organized as follows. Sec. 2 discusses the state-of-the-art in collision detection for rigid and deformable objects. Sec. 3 presents the proposed algorithm. Sec. 4 focuses on the relevant parameters of the algorithm and investigates their influence on the performance. In Sec. 6, results and experiments are described. Sec. 7 discusses limitations of our approach, followed by directions for ongoing research and a conclusion.

2 Related Work

Efficient collision detection is an essential component in physically–based simulation or animation [2], [6], including cloth modeling [4], [24], [28]. Further applications can be found in robotics, computer animation, medical simulations, computational biology, and games.

Collision detection algorithms based on bounding–volume (BV) hierarchies have proven to be very efficient and many types of BVs have been investigated. Among the acceleration structures we find spheres [13], [25], [26], axis–aligned bounding boxes [14], [3], oriented bounding boxes [9], and discrete–oriented polytopes [17]. In [29], various optimizations to BV hierarchies are presented.

Initially, BV approaches have been designed for rigid objects. In this context, the hierarchy is computed in a pre–processing step. In the case of deforming objects, however, this hierarchy must be updated at run time. While effort has been spent to optimize BV hierarchies for deformable objects [19], they still pose a substantial computational burden and storage overhead for complex objects. As an additional limitation for physically–based applications, BV approaches typically detect intersecting surfaces. The computation of the penetration depth for collision response requires an additional step.

As an alternative to object partitioning, other approaches employ discretized distance fields as volumetric object representation for collision detection. The presented results, however, suggest that this data structure is less suitable for real–time processing of geometrically complex objects [11]. In [10], a hybrid approach is presented, which uses BVs and distance fields.

Recently, various approaches have been introduced that employ graphics hardware for collision detection. In [1], a multi–pass rendering method is proposed for collision detection. However, this algorithm is restricted to convex objects. In [20], the interaction of a cylindrical tool with deformable tissue is accelerated by graphics hardware. [12] proposes a multi–pass rendering approach for collision detection of 2–D objects, while [15] and [16] perform closest–point queries using bounding–volume hierarchies along with a multipass–rendering approach. The aforementioned approaches decompose the objects into convex polytopes.

There exist various approaches that propose spatial subdivision for collision detection. These algorithms employ uniform grids [27], [8], [30] or BSPs [21]. In [27], spatial hashing for collision detection is mentioned, but no details are given. [22] presents a hierarchical spatial hashing approach as part of a robot motion planning algorithm, which is restricted to rigid bodies.

We employ spatial hashing for the collision detection of deformable tetrahedral meshes. $\mathbb{R}^3$ is implicitly subdivided into small grid cells. Information about the global bounding box of our environment is not required and 3D data structures, such as grids or BSPs are avoided. Further, our approach inherently detects collisions and self–collisions. The parameters of the algorithm have been investigated and optimized.

3 Collision Detection Algorithm

The collision detection algorithm implicitly subdivides $\mathbb{R}^3$ into small AABBs. In a first pass, all vertices of all objects are classified with respect to these small 3D cells. In a second pass, all tetrahedrons are classified with respect to the same 3D cells. If a tetrahedron intersects with a cell, all vertices, that have been associated with this cell in the first pass, are checked for interference with this

tetrahedron. The actual intersection test computes barycentric coordinates of a vertex with respect to a tetrahedron in order to estimate, whether a vertex penetrates a tetrahedron.

The consistent processing of all object primitives enables the detection of collisions and self–collisions. If a vertex penetrates a tetrahedron, a collision is detected. If the vertex and the tetrahedron belong to the same object, a self–collision is detected. If a vertex is part of a tetrahedron, the intersection test is omitted.

Spatial Hashing of Vertices: In the first pass, the positions of all vertices are discretized with respect to a user–defined cell size. Therefore, the coordinates of the vertex position (x, y, z) are divided by the given grid cell size l and rounded down to the next integer (i, j, k): $i = \lfloor x/l \rfloor$, $j = \lfloor y/l \rfloor$, $k = \lfloor z/l \rfloor$.

The hash function $hash$ maps the discretized 3D position (i, j, k) to a 1D index h and the vertex and object information is stored in a hash table at this index h: $h = hash(i, j, k)$.

Refer to Sec. 4.1 for details on the hash function. Sec. 4.2 and Sec. 4.3 describe how to find the optimal hash table size and cell size, respectively.

In addition to generating a hash value for each vertex, this first pass also computes the AABBs of all tetrahedrons based on their current deformed state.

Spatial Hashing of Tetrahedrons: While the first pass has considered all vertices to build the hash table and to update the AABBs of the tetrahedrons, the second pass of the algorithm traverses all tetrahedrons.

First, the minimum and maximum values describing the AABB of a tetrahedron, are discretized. Again, these values are divided by the user–defined cell size and rounded down to the next integer. Second, hash values are computed for all cells affected by the AABB of a tetrahedron. Therefore, all cells are traversed from the discretized minimum to the discretized maximum of the AABB (see Fig. 2). All vertices found at the according hash table index are tested for intersection.

Intersection Test: If a vertex $\mathbf{p}$ and a tetrahedron t are mapped to the same hash index and $\mathbf{p}$ is not part of t, a penetration test has to be performed.

The actual intersection test consists of two steps. First, $\mathbf{p}$ is checked against the AABB of t, which has been updated in the first pass. If $\mathbf{p}$ penetrates the AABB of t, the second step actually tests, whether $\mathbf{p}$ is inside t. This test computes barycentric coordinates of $\mathbf{p}$ with respect to a vertex of t. Details are given in Sec. 4.4.

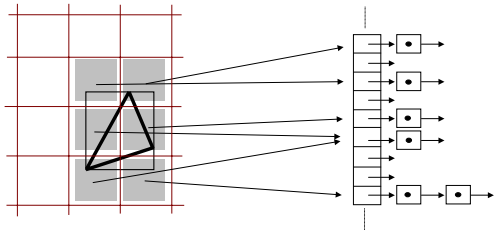

Figure 2: Hash values are computed for all grid cells covered by the AABB of a tetrahedron. The tetrahedron is checked for intersection with all vertices found at these hash indices.

4 Parameters

In this section, we investigate the parameters of the presented algorithm. The characteristics of the hash function, the size of the hash table, the size of a 3D cell for spatial subdivision, and the actual intersection test influence the performance of the algorithm. We have optimized all these aspects of the algorithm.

4.1 Hash Function

In the first pass of the algorithm, hash values are computed for all discretized vertex positions. These hash values should be uniformly distributed to guarantee an adequate performance of the algorithm. The hash function has to work with vertices of the same object, that are close to each other, and with vertices of different objects, that are farther away.

We have tested several hash functions in our implementation, basically variants of additive and rotating hash functions. However, we have not systematically investigated the characteristics of hash functions. It gets three values, describing a vertex position (x, y, z), and returns a hash value:

hash(x,y,z) = (x p1 **xor** *y p2* **xor** *z p3)* **mod** *n*

where $p1, p2, p3$ are large prime numbers, in our case $73856093, 19349663, 83492791$, respectively. The value n is the hash table size.

The function can be evaluated very efficiently and produces a comparatively small number of hash collisions for small hash tables. As described in Sec. 4.2, the quality of the hash function is less important for larger hash tables.

4.2 Hash Table Size

The size of the hash table significantly influences the performance of the collision detection algorithm. Experiments indicate, that larger hash tables reduce the risk of mapping different 3D positions to the same hash index. Therefore, the algorithm generally works faster with larger hash tables. On the other hand, the performance slightly decreases for larger hash tables due to memory management. Fig. 3 and Fig. 4 show the performance of our algorithm for two test scenarios with a varying hash table size. If the hash table is significantly larger than the number of object primitives, the risk of hash collisions is minimal. Although it is known, that hash functions work most efficiently if the hash table size is a prime number [5], Fig. 3 and Fig. 4 show performance measurements with hash table sizes of 99, 199, 299 and so on.

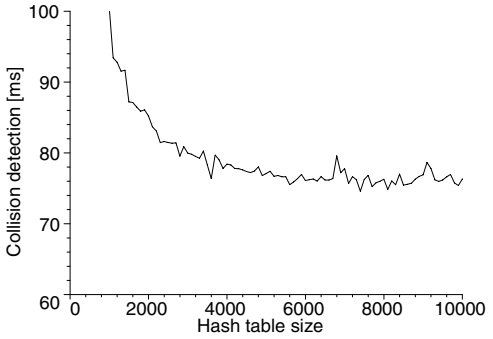

Figure 3: Performance of the collision detection algorithm for two deformable vessels (see Fig. 9) with an overall number of 5898 vertices and 20514 tetrahedrons with varying hash table size of 99, 199, 299 and so on. The grid cell size is set to the average edge length of all tetrahedrons (see Sec. 4.3).

Our implementation of the hash table does not require a re–initialization in each simulation step, which would reduce the efficiency in case of larger tables. To avoid this problem, each simulation step is labeled with a unique time stamp. If the first pass stores vertices in a hash table cell with outdated time stamp, the time stamp is updated and the cell is reset before new vertices are inserted. If the time stamp is up to date, new vertices are appended to the hash table cell. When the second pass generates hash indices for the tetrahedrons, the current time stamp is compared to the time stamp found in the hash table entry. If the time stamps differ, the information in the hash table is outdated and no intersection tests have to be performed. Therefore, no re–initialization of the hash table has to be performed during the simulation, which would be comparatively costly for larger hash tables.

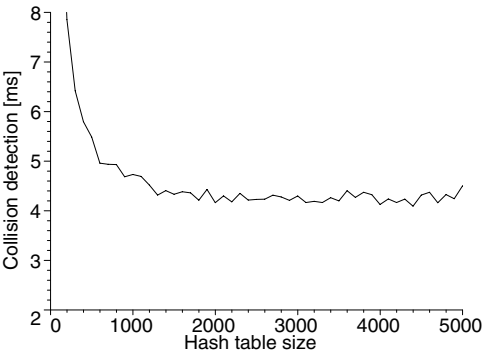

Figure 4: Performance of the collision detection algorithm for 100 deformable objects (see Fig. 8) with an overall number of 1200 vertices and 1000 tetrahedrons with varying hash table size of 99, 199, 299 and so on. The grid cell size is set to the average edge length of all tetrahedrons (see Sec. 4.3).

4.3 Grid Cell Size

The grid cell size, which is used for spatial hashing, influences the number of object primitives, that are mapped to the same hash index. In case of larger cells, the number of primitives per hash index increases and the intersection test slows down. If the cell size is significantly smaller than a tetrahedron, the tetrahedron covers a larger number of cells and has to be checked against vertices in a larger number of hash entries. The measurements in Fig. 5 and 6 indicate, that a grid cell should have the size of the average edge length of all tetrahedrons to achieve optimal performance. The graphs illustrate, that the grid cell size has a more significant impact on the performance than hash table size or hash function.

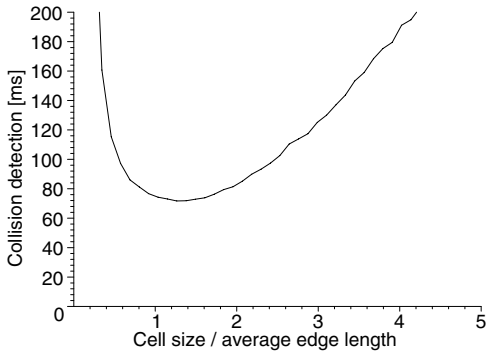

Figure 5: Performance of the collision detection algorithm for two deformable vessels (see Fig. 9) with an overall number of 5898 vertices and 20514 tetrahedrons with varying grid cell size. Hash table size is 9973.

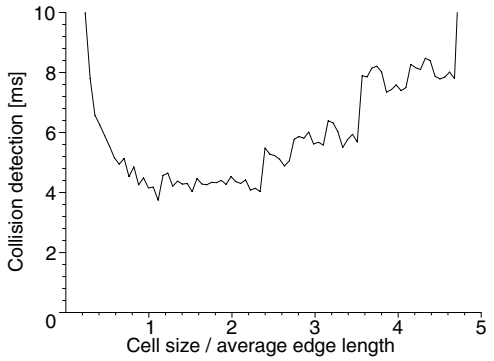

Figure 6: Performance of the collision detection algorithm for 100 deformable objects (see Fig. 8) with an overall number of 1200 vertices and 1000 tetrahedrons with varying grid cell size. Hash table size is 4999.

4.4 Intersection Test

We have compared two tests for detecting, whether a vertex $\mathbf{p}$ penetrates a tetrahedron t, whose vertices are at positions $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$,. The first test computes barycentric coordinates of a vertex with respect to a vertex of a tetrahedron. The second test considers oriented faces of a tetrahedron and checks, whether a vertex is in the positive or negative half–space of these faces.

Since the barycentric–coordinate test is slightly faster than the half–space test, we have performed our experiments in Sec. 6 using this vertex-in-tetrahedron test:

Barycentric coordinates with respect to $\mathbf{x}_0$*:* We express $\mathbf{p}$ in new coordinates $\beta = (\beta_1, \beta_2, \beta_3)^T$ with respect to a coordinate frame, whose origin coincides with $\mathbf{x}_0$ and whose axis coincide with the edges of t adjacent to $\mathbf{x}_0$: $\mathbf{p} = \mathbf{x}_0 + \mathbf{A}\beta$,, where $\mathbf{A} = [\mathbf{x}_1 - \mathbf{x}_0, \mathbf{x}_2 - \mathbf{x}_0, \mathbf{x}_3 - \mathbf{x}_0]$ is a 3 by 3 dimensional matrix. The coordinates β of $\mathbf{p}$ in this new coordinate frame are: $\beta = \mathbf{A}^{-1}(\mathbf{p} - \mathbf{x}_0)$.

Now the point $\mathbf{p}$ lies inside tetrahedron t, if $\beta_1 \geq 0, \beta_2 \geq 0, \beta_3 \geq 0$ and $\beta_1 + \beta_2 + \beta_3 \leq 1$.

5 Time Complexity

Let n be the number of primitives (vertices and tetrahedrons). To find all intersecting vertex–tetrahedron pairs a naive approach would test all vertices against all tetrahedrons resulting in a time complexity of the order of $O(n^2)$. The goal of our approach is to reduce this complexity to $O(n)$. Since a deformation algorithm needs to process all the primitives at each time step, linear time complexity for collision detection does not decrease its performance significantly.

During the first pass, all vertices are inserted into the hash table. This pass takes $O(n)$ time. First, the hash table does not need to be initialized, so the time is independent of the hash table size. Second, for each vertex, the hash function can be evaluated and a vertex reference can be added to the hash cell in $O(1)$ time.

In the second pass, for all tetrahedrons all vertices in a local neighborhood are tested for collision. The time complexity of this pass is of the order of $O(n \cdot p \cdot q)$ where p is the average number of cells intersected by a tetrahedron and q is the average number of vertices per cell. If the cell size is chosen to be proportional to the average tetrahedron size, p is a constant. If there are no hash collisions, the average number of tetrahedrons per cell is constant too and so is q, since there are at most four times as many vertices as tetrahedra in a cell. With both, p and q being constant, the time complexity of the algorithm turns out to be linearly dependent on the number of primitives.

6 Results

We have performed experiments with various setups of deformable objects (see Tab. 1 and Tab. 2). Setups A, B and D are illustrated in Fig. 7, 8, 9, respectively. Fig. 10 illustrates the tetrahedral meshes of the patient–individual models of two vessels used in setup D. Setup C and E use the same deformable objects like B.

Table 1: The following setups with dynamically deforming objects have been tested in our physically–based environment.

setup	objects	tetras	vertices
A	100	1000	1200
B	8	4000	1936
C	20	10000	4840
D	2	20514	5898
E	100	50000	24200

Table 2: Performance of the collision detection algorithm for setups in Tab. 1. Average collision detection time, minimum, maximum, and standard deviation for 1000 simulation step are given.

setup	ave [ms]	min [ms]	max [ms]	dev [ms]
A	4.3	4.1	6.5	0.24
B	12.6	11.3	15.0	0.59
C	30.4	28.9	34.4	1.25
D	70.0	68.5	72.1	0.86
E	172.5	170.5	174.6	1.08

Experiments indicate, that the detection of all collisions and self–collisions for dynamically deforming objects can be performed with 15 Hz with up to 20k tetrahedrons and 6k vertices on a PC, Intel Pentium 4, 1.8GHz. The performance is independent from the number of objects. It only depends on the number of object primitives. The performance varies slightly during simulations due to the changing number of hash collisions and due to a varying distribution of hash table elements.

7 Discussion

The proposed algorithm detects, whether a vertex penetrates a tetrahedron, but it does not detect, whether an edge intersects with a tetrahedron. This is due to two reasons. First, the performance of the algorithm would decrease significantly, while the relevance of an edge test is unclear in case of densely sampled objects. Second, the collision detection is intended to be a component in physically–based environments with integrated collision response. While collision response can be easily realized for penetrating vertices, it is rather uncommon and costly to implement collision response in case of penetrating edges.

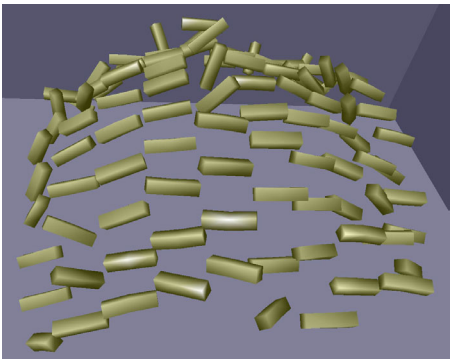

Figure 7: Test setup A.

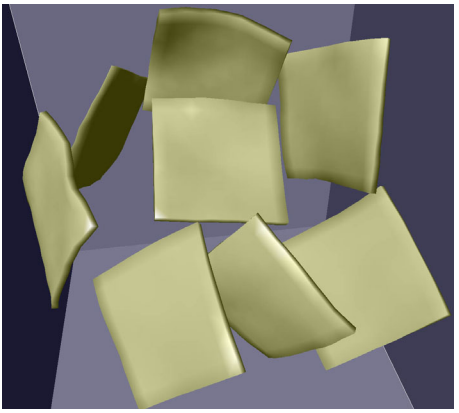

Figure 8: Test setup B.

The presented algorithm performs two passes on the objects, even though it would be sufficient to only perform one pass, which computes hash values for all tetrahedrons. In this case, each hash table entry contains tetrahedrons, that could intersect. We have implemented this approach, but found it less efficient compared to the two–pass approach.

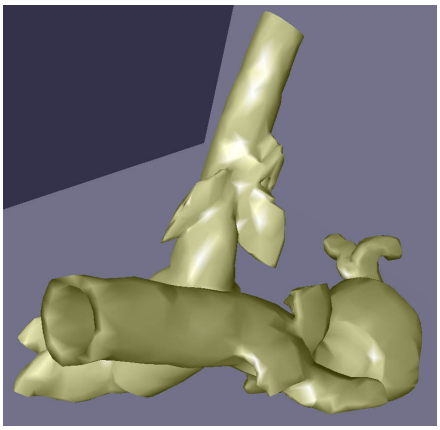

Figure 9: Test setup D.

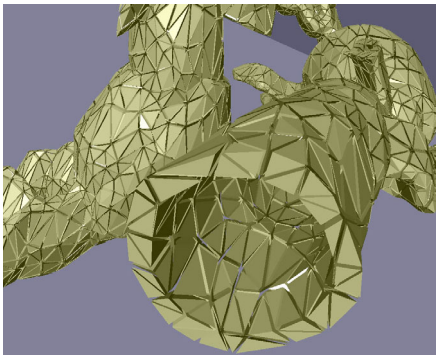

Figure 10: Tetrahedral mesh of test setup D.

While vertex positions are mapped to the hash table exactly once, tetrahedrons are usually mapped to several hash indices, which leads to a larger number of elements in the hash table, thus decreasing the performance of the algorithm.

The comparison of the performance with other existing collision detection approaches is difficult. There exist numerous public domain libraries, such as RAPID [9], PQP [18], and SWIFT [7]. However, these approaches are not optimized for deformable objects. They work with data structures, that can be pre–computed for rigid bodies, but have to be updated in case of deformable objects.

Our approach has been implemented based on tetrahedrons. However, it is not restricted to tetrahedrons and could handle other object primitives as well by simply replacing the intersection test.

8 Ongoing Work

Efficient collision detection is an essential component in real–time simulation environments for deformable objects. However, a realistic interaction between dynamically deforming objects also requires a correct collision response. Since our algorithm provides the exact position of a vertex inside a penetrated tetrahedron, we can employ this information for collision response. If we assume, that a face or a vertex of the penetrated tetrahedron is part of the object surface, we can easily derive the penetration depth, which allows for a correct collision response.

Further, we are about to accomplish a modular framework for real–time simulation of deformable objects, which can be used in game engines or surgical simulators. We have integrated efficient deformable models based on linear finite elements [23] and the proposed collision detection approach. Completed with the above mentioned collision response, the framework will handle interacting deformable models of up to several thousand tetrahedrons in real–time.

9 Conclusion

We have introduced a method for detecting collisions and self–collisions of dynamically deforming objects. Instead of computing the global bounding box of all objects and explicitly performing a spatial subdivision, we propose to use a hash function that maps 3D cells to a hash table, thus realizing a very efficient, implicit spatial subdivision. The actual vertex-in-tetrahedron test is based on barycentric coordinates. It provides information, that can be used for physically–based collision response. We have investigated and optimized the parameters of our approach. Experiments, performed with various test scenarios, show that environment of up to 20k tetrahedrons can be processed in real–time, independent from the number of objects.

10 Acknowledgement

This research is supported by the Swiss National Science Foundation. The project is part of the Swiss National Center of Competence in Research on Computer Aided and Image Guided Medical Interventions (NCCR Co–Me).

References

[1] G. Baciu, W. Wong, H. Sun, "RECODE: an image–based collision detection algorithm," *The Journal of Visualization and Computer Animation,* vol. 10, pp. 181–192, 1999.

[2] D. Baraff, "*Collision and contact*," SIGGRAPH '01 Course Notes, 2001.

[3] G. van den Bergen, "Efficient collision detection of complex deformable models using AABB trees," *Journal of Graphics Tools,* vol. 2, no. 4, pp. 1–13, 1997.

[4] R. Bridson, R. Fedkiw, J. Anderson, "Robust treatment of collisions, contact and friction for cloth animation," *Proceedings of SIGGRAPH '02,* pp. 594–603, 2002.

[5] T. Cormen, C. Leiserson, R. Rivest, "*Introduction to Algorithms*," ISBN 0-262-03141-8, The MIT Press, Cambridge, Massachusetts, 1990.

[6] G. Debunne, M. Desbrun, M.-P. Cani, A. Barr, "Dynamic real–time deformations using space & time adaptive sampling," *Proceedings of SIGGRAPH '01,* pp. 31–36, 2001.

[7] S. Ehmann, M. Lin, "*SWIFT: Accelerated proximity queries between convex polyhedra by multi–level voronoi marching*," Technical Report TR00-026, Univ. of North Carolina at Chapel Hill, 2000.

[8] F. Ganovelli, J. Dingliana, C. O'Sullivan, "BucketTree: Improving collision detection between deformable objects," *Proceedings Spring Conference on Computer Graphics SCCG '00*, Budmerice Castle, Bratislava, 2000.

[9] S. Gottschalk, M. Lin, D. Manocha, "OBB–tree: A hierarchrical structure for rapid interference detection," *Proceedings of SIGGRAPH '96*, pp. 171–180, 1996.

[10] T. He, A. Kaufman, "Collision detection for volumetric objects," *Proceedings of IEEE Visualization '97*, pp. 27–34, 1997.

[11] G. Hirota, S. Fisher, M. Lin, *"Simulation of non–penetrating elastic bodies using distance fields,"* Technical Report TR00-018, University of North Carolina at Chapel Hill, 2000.

[12] K. Hoff, A. Zaferakis, M. Lin, D. Manocha, "Fast and simple 2D geometric proximity queries using graphics hardware," *Proceedings of Symposium on Interactive 3D Graphics '01,* pp. 145–148, 2001.

[13] P. Hubbard, "Interactive collision detection," *Proceedings of IEEE Symposium on Research Frontiers in Virtual Reality '93,* pp. 24–31, 1993.

[14] M. Hughes, C. DiMattia, M. Lin, D. Manocha, "Efficient and accurate interference detection for polynomial deformation and soft object animation," *Proceedings of Computer Animation '96,* pp. 155–166, 1996.

[15] Y. Kim, M. Otaduy, M. Lin, D. Manocha, "Fast penetration depth computation for physically–based animation," *Proceedings of ACM SIGGRAPH Symposium on Computer Animation '02,* pp. 23–31, 2002.

[16] Y. Kim, K. Hoff, M. Lin, D. Manocha, "Closest point query among the union of convex polytopes using rasterization hardware," *Journal of Graphics Tools*, 2003, to appear.

[17] J. Klosowski, M. Held, J. Mitchell, H. Sowizral, K. Zikan, "Efficient collision detection using bounding volume hierarchies of k–DOPs," *Proceedings of SIGGRAPH '96,* pp. 171–180, 1996.

[18] E. Larsen, S. Gottschalk, M. Lin, D. Manocha, "*Fast proximity queries with swept sphere volumes*," Technical Report TR99-018, University of North Carolina at Chapel Hill, 1999.

[19] T. Larsson, T. Akenine–Moeller, "Collision detection for continuously deforming bodies," *Proceedings of Eurographics '01,* pp. 325–333, 2001.

[20] J. Lombardo, M.-P. Cani, F. Neyret, "Real–time collision detection for virtual surgery," *Proceedings of Computer Animation '99,* pp. 33–39, 1999.

[21] S. Melax, "Dynamic plane shifting BSP traversal," *Proceedings of Graphics Interface '00*, pp. 213–220, 2000.

[22] B. Mirtich, "*Efficient algorithms for two–phase collision detection,*" Technical Report TR-97-23, Mitsubishi Electric Research Laboratory, 1997.

[23] M. Müller, J. Dorsey, L. McMillan, R. Jagnow, B. Cutler, "Stable Real–Time Deformations," *Proceedings of ACM SIGGRAPH Symposium on Computer Animation SCA '02*, pp. 49–54, 2002.

[24] X. Provot, "Collision and self–collision handling in cloth model dedicated to design garment," *Proceedings of Graphics Interface '97*, pp. 177–189, 1997.

[25] S. Quinlan, "Efficient distance computation between non–convex objects," *Proceedings of IEEE Int. Conf. on Robotics and Automation,* pp. 3324-3329, 1994.

[26] G. Bradshaw, C. O'Sullivan, "Sphere–tree construction using medial–axis approximation," *Proceedings of the ACM SIGGRAPH Symposium on Computer Animation SCA '02*, pp. 33–40, 2002.

[27] G. Turk, "*Interactive collision detection for molecular graphics*," Technical Report TR90-014, University of North Carolina at Chapel Hill, 1990.

[28] P. Volino, M. Courchesne, N. Magnenat–Thalmann, "Versatile and efficient techniques for simulating cloth and other deformable objects," *Proceedings of SIGGRAPH '95*, pp. 137–144, 1995.

[29] G. Zachmann, "Minimal hierarchical collision detection," *Proceedings of ACM Symposium on Virtual Reality Software and Technology VRST '02*, pp. 121–128, 2002.

[30] D. Zhang, M. Yuen, "Collision detection for clothed human animation," *Proceedings of Pacific Graphics '00*, pp. 328–337, 2000.