Hierarchical Spatial Hashing for Real-time Collision Detection

Mathias Eitz
Department of Computer Science
Shanghai Jiao Tong University

Gu Lixu
Department of Computer Science
Shanghai Jiao Tong University

Abstract

We present a new, efficient and easy to use collision detection scheme for real-time collision detection between highly deformable tetrahedral models. Tetrahedral models are a common representation of volumetric meshes which are often used in physically based simulations, e.g. in virtual surgery. In a deformable models environment collision detection usually is a performance bottleneck since the data structures used for efficient intersection tests need to be rebuilt or modified frequently. Our approach minimizes the time needed for building a collision detection data structure. We employ an infinite hierarchical spatial grid in which for each single tetrahedron in the scene a well fitting grid cell size is computed. A hash function is used to project occupied grid cells into a finite 1D hash table. Only primitives mapped to the same hash index indicate a possible collision and need to be checked for intersections. This results in a high performance collision detection algorithm which does not depend on user defined parameters and thus flexibly adapts to any scene setup.

1. Introduction

Deformable models based on tetrahedral meshes are commonly used in interactive real-time applications [16]. Their deformations are physically based responses to internal and external forces like gravity, collision forces and user interactions. A numerically stable, interactive simulation of those deformations requires high update rates which demands extremely efficient collision detection algorithms.

During each step of a deformable models simulation, the simulated objects change their shape slightly due to the involved forces. A collision detection algorithm needs to adapt to those changes in shape, either by modifying (often difficult) or by completely recomputing (often slow) its internal data structures. A recomputing approach has several advantages over an update approach. It is easier to implement since we do not need to consider complicated update rules. It is very general, strong deformations, cuts or fractures of the underlying meshes are handled implicitly. Finally, it returns exact results of the collision state at any time.

We propose a new collision detection algorithm comprising all those advantages that reports intersections of vertices with object primitives. It is based on a hierarchical spatial grid whose occupancy by primitives can be computed extremely efficient for any kind of scene. This method was inspired by the work of Teschner et al. [15], but improves upon speed, is completely independent of user defined parameters and most important, flexibly adapts to strong changes in the simulated meshes.

Our collision detection algorithm is made up of two passes. In the first pass, the so-called "mapping-phase", we map all tetrahedra in the scene into a hierarchical grid. Therefore, we determine a well fitting grid cell size for each single tetrahedron and then map it into that cubical tiling of space that best fits its size. In scenes with primitives of different sizes this results in a set of cubical tilings of space with differing resolutions, a hierarchical grid. This hierarchical grid is not stored explicitly, instead we map occupied grid cells to a single 1D hash table.

In the second pass, the so-called "intersection-phase" we test all vertices in the scene for intersection with the tetrahedra already stored in the hash table. For each vertex, we determine those grid cells that enclose the current vertex at all tiling resolutions. Next, we determine all tetrahedra that have been mapped to the same grid cells as the vertex by multiple lookups in the hash table. Finally, the vertex is tested for intersection with the resulting set of tetrahedra.

We analyze all parameters that influence the performance of the proposed algorithm, namely the size of the used hash table, the hash function and the narrow-phase intersection test and compare the results against the regular grid algorithm. Our proposed algorithm is not limited to tetrahedral meshes, instead it can be used with any kind of meshes, only the narrow-phase intersection test needs to be adapted to the used primitives.

IEEE International Conference on Shape Modeling and Applications(SMI'07)
0-7695-2815-5/07 $20.00 © 2007 IEEE

2. Related Work

Many efficient collision detection algorithms exist for scenes of rigid objects, i.e. scenes with models of fixed shape and size. The employed data structures are usually precomputed, attention is only turned to the efficiency of the actual collision tests. The situation is very different in scenes with dynamically deforming models. Since the shape and size of the involved models potentially changes each simulation step, the underlying collision detection algorithm needs to adapt its internal data structure accordingly, either by a complete rebuild [15] or by an update exploiting potential coherence between the simulation steps [9, 11, 2]. Typical collision detection approaches include bounding volume hierarchies [3, 6, 19, 8] and spatial subdivision methods [1, 10, 15]. For an extensive overview about the state-of-the-art in collision detection see [5, 17].

Uniform grid subdivision methods combined with hashing to reduce storage requirements have been used for a long time but have been recently made very popular by the work of Teschner et al. [15]. Turk [18] uses an uniform grid approach to determine collisions between molecular models represented by collections of spheres. He states that an uniform grid approach is perfectly suitable in this special case, since the radii of organic atoms do not differ significantly and therefore a small upper bound for the number of atoms per grid cell exists. Overmars [14] employs a uniform grid approach combined with hashing to accelerate point location in fat subdivisions of n-dimensional space (subdivisions that contain no long and skinny cells). He derives general upper bounds both on storage requirement and point location time. Mirtich [12] employs a hierarchical hash table to efficiently detect intersections of axis-aligned bounding boxes which are used to represent swept volumes of moving objects over a time interval. His approach requires a preprocessing step to determine a range of appropriate hash table resolutions. In [4], a hybrid approach is presented. Space is subdivided by a coarse regular grid where each non-empty grid cell stores a reference to an oriented bounding boxes tree of the primitives contained in it. Teschner et al. [15] use a uniform grid approach to accelerate collision detection for deformable models consisting of tetrahedra. They are the first to give a detailed analysis of all involved parameters including hash table size and optimal grid cell size.

We employ a new, general hierarchical spatial hashing approach for fast collision detection between tetrahedral deformable models. Tetrahedra are mapped into an implicit hierarchical grid where the grid cell size is specifically optimized for each single tetrahedron in the scene. Nevertheless, the only data structure that has to be stored in memory is a simple 1D hash table. We analyze all important parameters of the approach focusing on how to automatically determine a well fitting grid cell size for a tetrahedron.

3. Broad Phase Collision Detection

The aim of broad phase collision detection is to efficiently sort out all those pairs of primitives that do not possibly collide. Space partitioning achieves this goal by storing primitives in disjoint partitions of space. Only when multiple primitives occur in a single partition, a possible collision between those primitives has been detected.

We are partitioning space into a hierarchy of cubical tilings with different resolutions. Each tiling consist of a potentially infinite number of cubical axis-aligned so-called grid cells with edge length $k \in \mathbb{R}$. We associate a certain tiling with a hierarchy level by the so-called subdivision level $l \in \mathbb{Z}$ of that tiling. The subdivision level l is an unique identifier for the hierarchy level of a certain tiling. Each point in space (x, y, z) can now unambiguously be associated with a certain grid cell of subdivision level l by defining the mapping

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \rightarrow \begin{bmatrix} \lfloor x/k \rfloor \\ \lfloor y/k \rfloor \\ \lfloor z/k \rfloor \\ l \end{bmatrix} \tag{1}$$

For a tetrahedron t a tiling of subdivision level l is chosen such that the edge length k of the grid cells of that tiling "optimally" fits the size of t. We define k to be "optimal", when it is chosen such that a tetrahedron is not mapped into more than eight grid cells. Let $s = \text{size}\,(t)$ be the length of the longest edge of the axis-aligned bounding box of t. We then define the subdivision level l as:

$$l = \lceil \log_2 (s) \rceil \tag{2}$$

and the grid cell size k which is to be used for the mapping of t as:

$$k \quad = \quad 2^l \tag{3}$$

The grid cell size k is now defined such that in an one-dimensional case a primitive never occupies more than two, in a two-dimensional case never more than four and in a three-dimensional case never more than eight grid cells. This choice has been made in order to have a hard upper bound on the number of operations we need to map all tetrahedra into the hierarchical grid.

Mapping phase In a first step, we process all tetrahedra in the scene. For each tetrahedron t, its axis-aligned bounding box b and its associated subdivision level l according to Equation 2 are determined. The axis-aligned bounding box is stored and the subdivision level l inserted into the set of all subdivision levels occurred so far during this step. Tetrahedron t is then mapped into all grid cells of the tiling with

IEEE International Conference on Shape Modeling and Applications(SMI'07)
0-7695-2815-5/07 $20.00 © 2007 IEEE

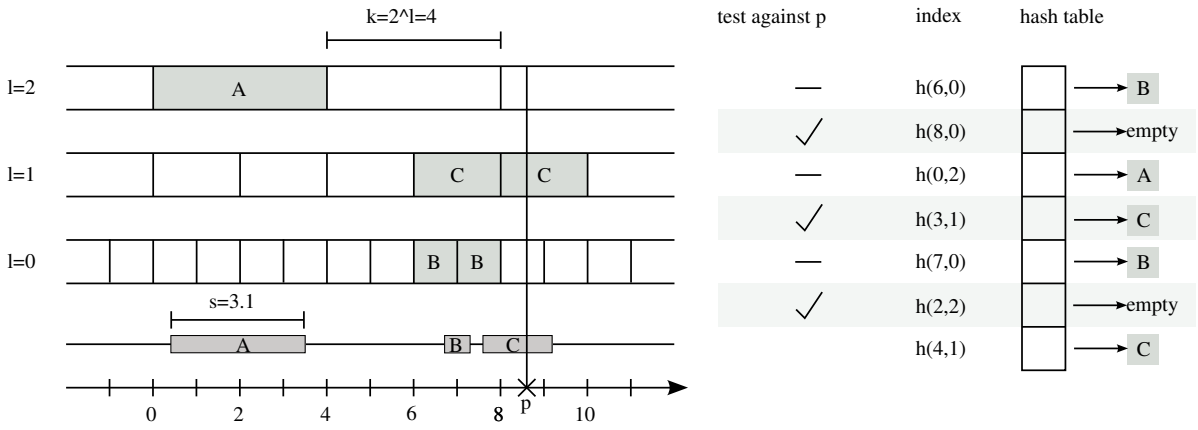

Figure 1. One-dimensional hierarchical grid with three subdivision levels on the left, corresponding hash table on the right. Line segments A to C are mapped into those grid cells that are overlapped by them and have a fitting subdivision level according to Equation 2. The hash table stores a reference to the objects occupying the grid cells. Point p is tested for intersection with the line segments by mapping it into the grid cells according to Equation 1 for all three subdivision levels.

subdivision level l that are overlapped by its axis-aligned bounding box. Each affected grid cell gets mapped into the hash table, which stores a reference to t.

Let the coordinates $(a_{min}, b_{min}, c_{min}, l)$ and $(a_{max}, b_{max}, c_{max}, l)$ be a mapping according to Equation 1 of b's corner of minimum (x, y, z) and maximum (x, y, z) coordinates, respectively. A grid cell with mapped coordinates (a, b, c, l) is overlapped by b if and only if $a_{min} \leq a \leq a_{max}$, $b_{min} \leq b \leq b_{max}$ and $c_{min} \leq c \leq c_{max}$.

The hash value i of an occupied grid cell is then determined by computing $i = hash\,(a, b, c, l)$ (refer to Section 3.1 for details on the hash function). A reference to t is stored in the hash table at index i.

Intersection phase In a second pass, we process all vertices in the scene and test them for intersection with the tetrahedra mapped in the first step. Therefore, for a certain vertex v for each used subdivision level l the mapping (a, b, c, l) of the (x, y, z) coordinate of v is determined according to Equation 1. Then, the hash function is used to determine the hash indices of v (for all l). All tetrahedra that are stored at those hash indices have to be tested for intersection with v.

If no tetrahedron is found at a certain hash index, no collision has occurred. Otherwise a potential collision has been detected and an intersection test has to be performed. For performance reasons, the intersection test is made up of two steps: First, v is tested against the axis-aligned bounding box of t. If v is inside the AABB of t, then an actual vertex/tetrahedron intersection test has to be performed. Details for this intersection test are given in Section 4. If an intersection of a vertex v with an object is detected and v is part of that object, a self intersection has been detected, but only, if v is not part of the penetrated tetrahedron itself.

Example Consider the one-dimensional hierarchical grid illustrated in Figure 1, where line segments are tested for intersection with point p. Line segment A has a size (length) of $s = 3.1$. According to Equation 2, it gets mapped into that tiling of space with subdivision level $l = \lceil \log_2 (3.1) \rceil = 2$. A occupies all grid cells of subdivision level 2, that are "overlapped" by it. Now, the "address" (a, l) of the occupied grid cell is computed by mapping its minimum x coordinate (here: $x_{min} = 0$) according to Equation 1, i.e. $(0) \xrightarrow{l=2} (\lfloor 0/4 \rfloor, 2) = (0, 2)$. Then, a reference to A is stored in the hash table at index $h\,(0, 2)$.

For an example of the intersection phase consider point p located at $x = 8.6$ in Figure 1, assuming the mapping phase has already been completed. Point p needs to be checked against all those grid cells that it could be contained in. To determine exactly those grid cells, p gets mapped for all existing subdivision levels according to Equation 1. I.e. $(8.6) \xrightarrow{l=0} (\lfloor 8.6/1 \rfloor, 0) = (8, 0)$, $(8.6) \xrightarrow{l=1} (\lfloor 8.6/2 \rfloor, 1) = (4, 1)$ and $(8.6) \xrightarrow{l=2} (\lfloor 8.6/4 \rfloor, 2) = (2, 2)$. Finally, a hash table lookup at $h\,(8, 0)$, $h\,(4, 1)$ and $h\,(2, 2)$ yields one possible intersection of p with line segment C.

IEEE International Conference on Shape Modeling and Applications(SMI'07)
0-7695-2815-5/07 $20.00 © 2007 IEEE

3.1. Hash Function

In our algorithm, we use a hash function h to map grid cell "addresses" of the form $(a, b, c, l) \in \mathbb{Z}^4$ into a hash table. Our hash function maps an infinite set of possible input keys K onto a finite set of hash values $\{0, 1, ..., m-1\}$:

$$h(a, b, c, l) \rightarrow \{0, 1, ..., m-1\} \quad (4)$$

where m is the chosen hash table size. Hash collisions are resolved by a chaining approach, i.e. primitives that occupy a common grid cell at a common subdivision level get stored in a linked list at the same hash table index.

The most desirable feature of the hash function in our application is computational speed. In only one collision detection cycle the hash function usually gets called multiple times for each tetrahedron in the scene (one tetrahedron typically gets mapped to multiple grid cells) and each vertex in the scene (penetration test over all subdivision levels). This requires a hash function that can be efficiently computed. Uniform distribution of the hash keys is also important. Due to the fact that tetrahedra in a typical scene are clustered into objects, hash keys of those tetrahedra will usually tend to be very similar to each other and differ only slightly in one single coordinate of the key. Thus, we need a hash function with good avalanching properties to avoid clustering in the hash table.

Due to the fact that $|K| \gg m$, hash collisions can not be avoided altogether but rather need to be minimized. In order to find a suitable hash function that results in the least possible amount of hash collisions, we have been testing a selection of popular hash functions against a sample of the expected input. The sample was chosen in such a way, that it only contained unique values. This is usually not the case in a typical application but allows us to measure the number of hash collisions solely due to the distribution behaviour of the hash function itself.

Balls and bins problem Even a hash function that distributes hash values truly randomly and uniformly produces a certain amount of hash collisions for a set of unique input data. This problem is well studied and better known as the balls and bins problem. Suppose that n balls are thrown into m bins such that all balls choose a bin independently and uniformly at random (see [13] for an extensive survey about this problem). The probability for a certain bin to be missed by a ball is $\left(1 - \frac{1}{m}\right)$. The probability for a certain bin to be missed by all n balls is $\left(1 - \frac{1}{m}\right)^n \approx e^{-\frac{n}{m}}$. This means we can determine the expected approximate number of collisions c (more than one ball in a bin) as follows:

$$c \approx m \cdot e^{-\frac{n}{m}} - (m - n) \quad (5)$$

The fact that this number is astonishingly high is related to the well known birthday paradox.

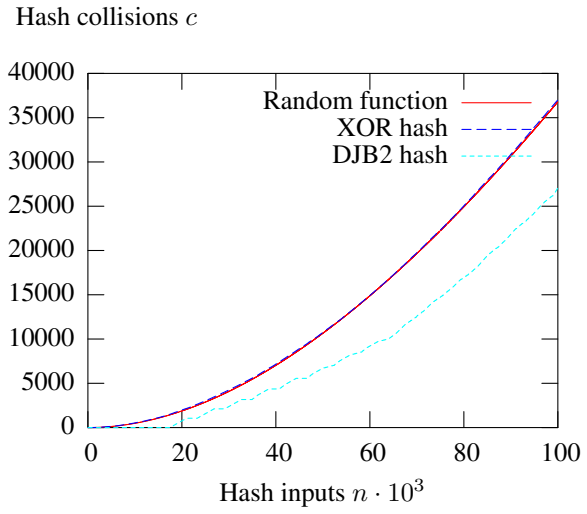

Figure 2. Expected number of hash collisions c for n unique input values hashed into a hash table of size $m = 100000$ with the XOR and the DJB2 hash function. The third curve denotes the number of expected hash collisions for a hash function, that generates perfect random and uniform hash values, see Equation 5.

Using Equation 5, we can now determine the approximate number of expected collisions for a given hash table of size m for a set of unique inputs of cardinality n. Given a hash table of size $m = 100000$ and $n = 50000$ unique inputs, we expect $c \approx 10653$ collisions for a perfect random hash function with uniform distribution. A plot for the number of expected collisions with a perfectly random and uniform hash function is given in Figure 2.

Selected hash functions We have tested various hash functions (see Algorithm 1, 2) for their applicability in spatial hashing, including the XOR hash function used in [15], among them several more sophisticated hash functions, including for example Bob Jenkins' hash function [7]. However, our experiments indicated, that simple and fast to execute hash functions are generally preferable for spatial hashing and result in the fastest possible collision detection time. Execution speed is here of greater importance than a near perfect uniform distribution. This is easily explainable. In a spatial hashing algorithm, hash collisions mainly occur because of non unique input keys, i.e. one grid cell is actually occupied by a lot of tetrahedra that clearly all get mapped to the same hash index. In this case, hash collisions due to imperfect hash functions only contribute a very small fraction to the total number of hash collisions.

While the number of hash collisions of the XOR hash function on our sample set of input keys very closely resembles that of a perfect uniform hash function, the DJB2 hash

IEEE International Conference on Shape Modeling and Applications(SMI'07)
0-7695-2815-5/07 $20.00 © 2007 IEEE

Algorithm 1 XOR hash function, the hash table size m is usually chosen as a prime number.

procedure $xorHash(x, y, z, l)$

1: $hash \leftarrow x \times 73856093$
2: $hash \leftarrow hash \oplus y \times 19349663$
3: $hash \leftarrow hash \oplus z \times 83492791$
4: $hash \leftarrow hash \oplus l \times 67867979$
5: Return $hash \bmod m$

Algorithm 2 DJB2 hash function, the hash table size m is usually chosen as a prime number.

procedure $djb2Hash(x, y, z, l)$

1: $hash \leftarrow 5381$
2: $hash \leftarrow hash \times 33 + x$
3: $hash \leftarrow hash \times 33 + y$
4: $hash \leftarrow hash \times 33 + z$
5: $hash \leftarrow hash \times 33 + l$
6: Return $hash \bmod m$

function produces unexpected results (Figure 2). On our sample set, the number of hash collisions are much lower than expected. An analysis of the hash table histogram has shown, that the DJB2 hash function has produced a slightly clustered distribution of the computed hash values. The DJB2 function performs very well on all tested input samples, but from a theoretical point of view, the XOR hash function is the better choice. All other tested hash functions produced equally good results compared to the XOR hash function, but they all used significantly more instructions to compute the hash value and therefore slowed down collision detection performance.

3.2. Grid Cell Size

In a regular grid approach, choosing the right grid cell size is the key to high performance collision detection. If the grid cell size is chosen too small, each tetrahedron in the scene overlaps a large amount of grid cells, the mapping phase gets extremely slow (penetration tests however will get very fast). If the grid cell size is chosen too big, a lot of tetrahedra are contained in a single grid cell and the penetration tests therefore get very slow.

Teschner et al. [15] suggest using a grid cell size equal to the average edge length of all tetrahedra for optimal collision detection performance. As indicated in Figure 3, this value does not necessarily lead to the minimal possible collision detection time, i.e. with a regular grid approach the grid cell size that leads to absolute minimal collision detection time can usually not be exactly predetermined.

Hierarchical spatial hashing overcomes this limitation by determining a well fitting grid cell size for each single tetrahedron automatically and outperforms a regular grid approach for the grid cell size defined as optimal in the literature. The small range of grid cell sizes where the regular grid approach is actually faster than the hierarchical grid approach (computing the hierarchy information induces a slight computational overhead) can usually not be determined by the user and may even vary during simulation. However, contrary to a regular grid approach the hierarchical grid approach adapts its employed grid cell sizes for all hierarchies according to demand during each simulation step. The hierarchical grid thus flexibly adapts to any dynamic scene setup (imagine a deformable models simulation with growing/shrinking models).

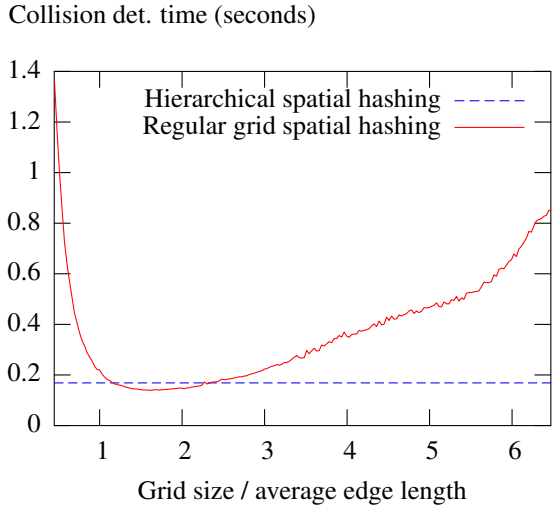

Figure 3. Collision detection time of hierarchical spatial hashing vs. regular grid spatial hashing for scene A. The collision detection performance for the regular grid algorithm is measured over a wide range of grid cell sizes. Since the hierarchical algorithm does not depend on a user selected grid cell size, its performance is shown as a constant over all grid cell sizes.

3.3. Hash Table Size

Collision detection performance is directly related to the size of the underlying hash table. If its size is chosen too small, many unnecessary hash collisions occur and the performance of collision detection suffers (see Figure 4). The size must be chosen such that the hash table provides enough space for all occupied grid cells in the scene.

Each tetrahedron in our proposed algorithm may occupy a maximum of 8 grid-cells. Let the total number of tetrahedrons in the whole scene be t_{total}. Thus, the maximum hash table size m_{max} possibly necessary to store all occupied grid cells can be determined as $m_{max} = 8 \times t_{total}$. In

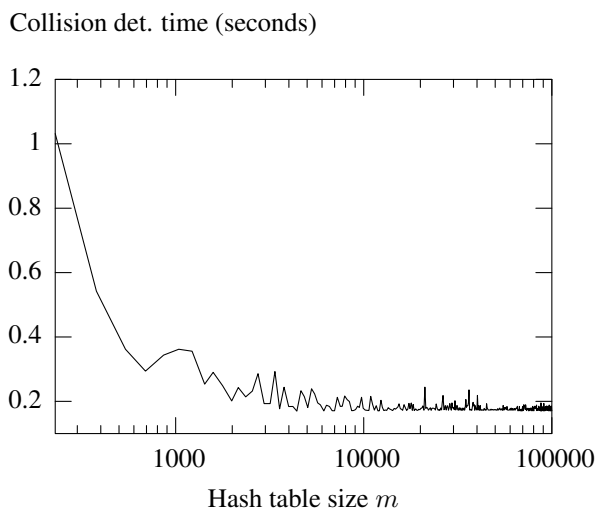

Figure 4. Collision detection time of scene A measured against various hash table sizes. Collision detection time becomes very stable for a hash table size of ≈ 50000, i.e. the number of primitives in the scene.

a typical scene, each tetrahedron will usually occupy considerably less than 8 grid-cells. Also, one grid-cell is usually "shared" by several tetrahedra. In Figure 4, collision detection time gets very stable for a hash table size m of $m \approx t_{total}$. Thus, as a more reasonable choice of the hash table size we propose to use

$$m \approx t_{total} \tag{6}$$

4. Narrow Phase Collision Detection

The result of broad-phase collision detection is a set of pairs of tetrahedra and vertices that potentially collide. Each pair needs to be tested for intersection. There are multiple methods to test if a point p penetrates a tetrahedron. We use an oriented planes approach, Teschner et al. [15] employ a barycentric coordinates approach.

The volume of a tetrahedron is enclosed by its four faces which must be oriented in such a way, that their normals show outwards of the tetrahedron. Each of those four faces lies in a unique plane in space. If p lies in the inner half-space of each of those four planes, then p lies inside the tetrahedron and a collision has been found. Otherwise, p lies outside the tetrahedron and no collision has occurred.

The signed distance D of a point x_0 from a plane specified in Hessian normal form is given by the equation $D = \hat{n} \cdot x_0 + p$. The sign of D determines, on which side of the plane x_0 lies. If $D > 0$ then x_0 lies in the half-space indicated by the direction of $\hat{n}$, if $D < 0$, then x_0 lies in the other half-space. If $D = 0$, then x_0 lies on the plane. Since we are only interested in the half-space in which the point x_0 lies but not in its the actual distance D from the plane, it is enough to determine the sign of D without calculating its actual value:

$$D = \hat{n} \cdot x_0 + p \tag{7}$$

$$\Leftrightarrow D = \frac{\vec{n}}{|\vec{n}|} \cdot x_0 + \frac{d}{|\vec{n}|} \tag{8}$$

$$\Leftrightarrow D\,|\vec{n}| = \vec{n} \cdot x_0 + d \tag{9}$$

Note that $sign\,(D\,|\vec{n}|) = sign\,(D)$ since $|\vec{n}| > 0$.

Given a triangle made up of the three vertices $A = (a_x, a_y, a_z)$, $B = (b_x, b_y, b_z)$ and $C = (c_x, c_y, c_z)$ and a point x_0, for which we want to determine in which half-space of the triangle it lies. We first compute the normal $\vec{n}$ of the triangle which is given by $\vec{n} = (C - A) \times (B - A)$. Then we compute $d = -\vec{n} \cdot A$ and finally the value for $D\,|\vec{n}|$ according to Equation 9. The sign of $D\,|\vec{n}|$ then determines in which half-space x_0 lies. To detect an intersection of a point p with a tetrahedron t, this procedure is applied to all four faces of t.

5. Results

In this section, we discuss experimental results gathered from four scenes, representing typical collision detection problems. All measurements were taken on a Intel Pentium M 1.5GHz CPU. The underlying C++ code was compiled with O2 optimization enabled. We have not yet completely integrated our new algorithm into a deformable models simulation. Therefore, collision detection times are given for static setups. Note that this is no deficiency, since the algorithm will always do exactly the same amount of computations in one simulation step and the computational complexity is independent of the strength of the deformations. We expect only very slight variations in collision detection time due to varying numbers of collisions when measuring in a simulation. Note that the implementation of the regular grid spatial hashing algorithm is actually a subset of our implementation of the hierarchical spatial hashing approach. Measured collision detection times of both algorithms are therefore directly comparable.

As recommended in [15], the grid cell size for the regular grid algorithm is for each scene set to the average edge length of the tetrahedrons in that scene.

All scene setups show a performance advantage for our new hierarchical spatial hashing approach compared to a regular grid approach. Collision detection in a scene consisting of about 16000 tetrahedrons and 100 objects (scene B) is possible at interactive rates of about 20Hz on our system. Collision detection performance is independent of the

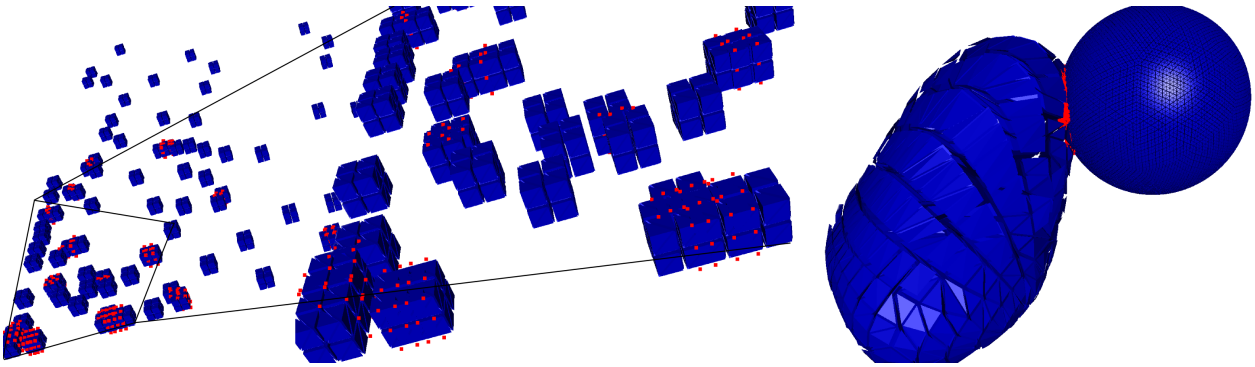

Figure 5. Experimental setup used to measure collision detection performance, red dots mark colliding vertices. Left: scene B, middle: detail of scene B, right: scene A. Scenes C and D are variations of scene B with less objects and tetrahedra.

Table 1. Experimental scene setups used for testing collision detection performance of the hierarchical spatial hashing algorithm

scene	objects	tetrahedra	vertices	coll. vertices
A	2	49082	14120	454
B	100	16200	6400	134
C	36	1728	972	304
D	9	432	243	32

number of objects in the scene, our experiments indicate that it is linearly dependent on the total number of object primitives. Note that absolute collision detection time values are pretty similar to those reported in [15] for comparable scenes.

The reader might wonder, if there are typical scenes that generally favor a regular grid approach over a hierarchical grid approach. A regular grid approach is optimal, when all primitives in a scene are of a certain fixed and regular size. However, our algorithm is actually a generalization of the regular grid algorithm. In a scene with all same sized object primitives, our approach "degrades" to a regular grid approach and thus yields the same performance as a regular grid approach.

6. Discussion and Future Work

This paper discusses an algorithm that reports collisions between tetrahedrons, approximated by intersection tests between vertices and tetrahedrons. The second possible case of the intersection of two edges of tetrahedrons is not handled by this algorithm. For regularly and densely sampled objects the vertex/tetrahedron intersection test is a good approximation of the actual tetrahedron/tetrahedron

Table 2. Collision detection performance for the setups defined in Table 1. We compare performance of the hierarchical and regular grid spatial hashing algorithm for each scene. The grid cell size for the regular grid algorithm is for each scene set to the average edge length of the tetrahedrons in that scene.

sc.	hier. grid $[ms]$	reg. grid $[ms]$	perf. inc. $[\%]$
A	174	220	22.9
B	54	65	20.3
C	5.4	6.1	12.9
D	1.3	1.5	15.4

intersection test and is computationally more efficient.

We have designed our algorithm such that a tetrahedron occupies a maximum of eight grid cells. This directly influences the performance and behaviour of our collision detection algorithm. The mapping-phase gets very fast, the performance of the intersection-phase on the other hand may suffer slightly due to the relatively coarse resolution of the spatial subdivision. We believe that this behaviour is favorable in simulation environments, where collisions are directly resolved and thus only very few collisions occur at a time.

We suggest that the most promising way to further increase the algorithm's performance is to minimize the number of primitives per grid cell while trying to keep high performance in the mapping-phase. This aim could be achieved by directly scan-converting a tetrahedron into the grid or by using tighter bounding volumes, e.g. oriented bounding boxes.

IEEE International Conference on Shape Modeling and Applications(SMI'07)
0-7695-2815-5/07 $20.00 © 2007 IEEE

7. Conclusion

In this paper, we have proposed a new algorithm for fast collision detection between tetrahedral models. We have analyzed all parameters that influence its performance. We have compared the algorithm to existing regular grid solutions and find it to be slightly faster in all cases. Its main advantage however, is its ease of implementation and especially its ease of use. It delivers high performance without requiring user defined parameters and flexibly adapts to any scene setup.

8. Acknowledgments

We would like to thank Zhang Shaoting, Jan Boehm and Marc Alexa as well as the anonymous reviewers for their insightful comments regarding this paper. This work was partially supported by the Natural Science Foundation of China, grant No. 70581171, and the Shanghai Municipal Research Fund, grant No. 045118045.

References

[1] S. Bandi and D. Thalmann. An Adaptive Spatial Subdivision of the Object Space for Fast Collision Detection of Animated Rigid Bodies. *Computer Graphics Forum*, 14(3):259–270, 1995.

[2] M. Garcia, S. Bayona, P. Toharia, and C. Mendoza. Comparing Sphere-Tree Generators and Hierarchy Updates for Deformable Objects Collision Detection. *Lecture Notes in Computer Science*, 3804:167, 2005.

[3] S. Gottschalk, MC Lin, and D. Manocha. OBB-Tree: A Hierarchical Structure for Rapid Interference Detection. *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 171–180, 1996.

[4] A. Gregory, MC Lin, S. Gottschalk, and R. Taylor. A Framework for Fast and Accurate Collision Detection for Haptic Interaction. *Virtual Reality, 1999. Proceedings., IEEE*, pages 38–45, 1999.

[5] S. Hadap, D. Eberle, P. Volino, M.C. Lin, S. Redon, and C. Ericson. Collision Detection and Proximity Queries. *Proceedings of the conference on SIGGRAPH 2004 course notes*, 2004.

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

[7] B. Jenkins. Hash functions. *Dr. Dobb's Journal*, 22(9):107–115, 1997.

[8] JT Klosowski, M. Held, JSB Mitchell, H. Sowizral, and K. Zikan. Efficient Collision Detection using Bounding Volume Hierarchies of k-DOPs. *Visualization and Computer Graphics, IEEE Transactions on*, 4(1):21–36, 1998.

[9] T. Larsson and T. Akenine-Möller. Collision Detection for Continuously Deforming Bodies. *Eurographics 2001*, pages 325–333, 2001.

[10] R.G. Luque, J.L.D. Comba, and C.M.D.S. Freitas. Broad-Phase Collision Detection Using Semi-Adjusting BSP-trees. *Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 179–186, 2005.

[11] J. Mezger, S. Kimmerle, and O. Etzmuss. Hierarchical Techniques in Collision Detection for Cloth Animation. *Journal of WSCG*, 11(2):322–329, 2003.

[12] B. Mirtich. Efficient Algorithms for Two-Phase Collision Detection. *Practical Motion Planning in Robotics: Current Approaches and Future Directions*, pages 203–223, 1998.

[13] M. Mitzenmacher, A. Richa, and R. Sitaraman. The Power of Two Random Choices: A Survey of Techniques and Results. *Handbook of Randomized Computing*, 1:255–312, 2001.

[14] M.H. Overmars. Point Location in Fat Subdivisions. *Information Processing Letters*, 44(5):261–265, 1992.

[15] M. Teschner, B. Heidelberger, M. Mueller, D. Pomeranets, and M. Gross. Optimized Spatial Hashing for Collision Detection of Deformable Objects. *Proceedings of Vision, Modeling, Visualization VMV'03*, pages 47–54, 2003.

[16] M. Teschner, B. Heidelberger, M. Muller, and M. Gross. A Versatile and Robust Model for Geometrically Complex Deformable Solids. *Computer Graphics International, 2004. Proceedings*, pages 312–319, 2004.

[17] M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrmann, M.P. Cani, F. Faure, N. Magnenat-Thalmann, and W. Strasser. Collision Detection for Deformable Objects. *Computer Graphics Forum*, 24(1):61–81, 2005.

[18] G. Turk. Interactive Collision Detection for Molecular Graphics. Master's thesis, The University of North Carolina, 1989. Tech Report TR90-014.

[19] G. van den Bergen. Efficient Collision Detection of Complex Deformable Models using AABB Trees. *Journal of Graphics Tools*, 2(4):1–13, 1998.