Vis Comput (2014) 30:729–738
DOI 10.1007/s00371-014-0954-1

ORIGINAL ARTICLE

An adaptive octree grid for GPU-based collision detection of deformable objects

Tsz Ho Wong · Geoff Leach · Fabio Zambetta

Published online: 10 May 2014
© Springer-Verlag Berlin Heidelberg 2014

Abstract In spatial subdivision-based collision detection methods on GPUs, uniform subdivision works well for even triangle spatial distributions, whilst for uneven cases non-uniform subdivision works better. Non-uniform subdivision techniques mainly include hierarchical grids and octrees. Hierarchical grids have been adopted for previous GPU-based approaches, due to their suitability for GPUs. However, octrees offer a better adaptation to distributions. One contribution of this paper is the use of an octree grid that takes a middle path between these two structures, and accelerates collision detection by significantly reducing the number of broad-phase tests which, due to their large quantity, are generally the main bottleneck in performance. Another contribution is to achieve further reduction in the number of tests in the broad phase using a two-stage scheme to improve octree subdivision. The octree grid approach is also able to address the issue of uneven triangle sizes, another common difficulty for spatial subdivision techniques. Compared to the virtual subdivision method which reports the fastest results among existing methods, speedups between 1.0× and 1.5× are observed for most standard benchmarks where triangle sizes and spatial distributions are uneven.

Keywords Collision detection · Deformable objects · Octree grid · GPU-based

Electronic supplementary material The online version of this article (doi:10.1007/s00371-014-0954-1) contains supplementary material, which is available to authorized users.

T. H. Wong (✉) · G. Leach · F. Zambetta
School of Computer Science and IT, RMIT University,
GPO Box 2476, Melbourne, VIC 3001, Australia
e-mail: itszwong@gmail.com

1 Introduction

Collision detection (CD) has been a core research area in computer graphics for decades. The main challenge that arises for highly deformable meshes, such as cloth, is the large number of primitives which may collide with each other in any given time step. Although a number of approaches have been investigated, performance of CD is still a major bottleneck.

GPU computing has demonstrated its potential benefit to perform physics calculations, including CD. Recent work has been on designing fast algorithms based on bounding volume hierarchies (BVHs) [15,19,20,31] and spatial subdivision schemes [6,9,25,36]. This paper focuses on the latter that exhibits very high data parallelism and lends itself well to the GPU's parallel architecture.

Generally spatial subdivision methods suffer from two common difficulties: uneven triangle sizes and uneven triangle spatial distributions, both of which can easily impair the subdivisions efficiency, leading to a large number of broad-phase tests, which usually dominate the total running time. The first problem can be addressed by a virtual subdivision scheme (VSS) with a uniform grid [36]. However, the issue of uneven spatial distributions is difficult for uniform grids, because a uniform cell size can be too small for low-density areas, and too large for high density areas, giving an inefficient subdivision (Fig. 1). Hence it is worth optimising subdivision, if the cost to achieve it is low.

Some authors have presented approaches to address this issue. Fan et al. [6] partition dense cells, but their hierarchical grid only has two levels, so that the problem is just partially solved. Other work [17] subdivides space into a hierarchy of multiple cell levels, then each primitive is assigned to a level whose cell size fits well. The hierarchical grid adopted by these methods can be seen as a full octree, which lacks the

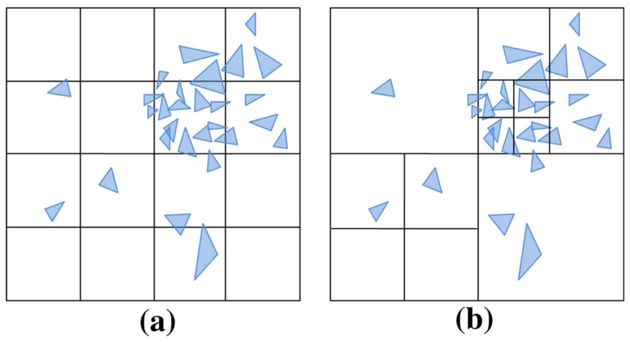


Fig. 1 A scene with uneven triangle distribution is partitioned by **a** a uniform grid and **b** a hierarchical grid according to the distribution density

adaptability to reduce the number of tests as much as possible. Ideally, a more adaptive octree is a beneficial approach to this problem, and in fact is commonly used in CPU-based algorithms. Unfortunately, parallel construction, update and traversal of traditional trees on GPUs require complicated operations [19,20], giving relatively low occupancy.

Motivated by these limitations, we investigate an adaptive octree grid (OTG) to accelerate CD for deformable objects on a GPU. The OTG takes advantages of both octrees and hierarchical grids. The octree that adaptively subdivides space based on triangle distribution is efficiently represented on a GPU by a grid which is actually an array of a length equal to the number of cells of the hierarchical grid (full octree). Using the OTG yields a considerable decrease in the number of broad-phase tests. Thus, a two-stage scheme consisting of a bottom-up stage and a top-down stage is used to cheaply improve octree subdivision, minimising the number of tests. Additionally, the number of levels in the OTG is dynamically adjusted according to triangle spatial distribution instead of being fixed. Another benefit arising intrinsically from this method is that uneven triangle sizes are also handled well, because uneven triangle sizes can be seen as uneven spatial distributions most of the time. For example, in a given scene, the triangle density of areas having refined meshes is very likely higher than that of areas having coarse meshes.

We also combine a number of techniques including newly available GPU shuffle instructions to improve performance of the parallel prefix sum (scan) [11], which is important in many parallel applications, including CD.

We use the CUDA toolkit 5.5 and have conducted experiments with a set of scenes using an NVIDIA GTX 780 GPU to investigate and analyse performance. The method can perform discrete (DCD) and continuous collision detection (CCD) of objects consisting of tens of thousands arbitrarily distributed triangles in a few milliseconds. Results show that the approach efficiently addresses issues of uneven triangle sizes and uneven triangle spatial distributions.

2 Related work

There is extensive literature on CD, for which we refer the reader to the survey in [35]. This section briefly reviews previous work most directly related to our work.

Bounding volume (BV) techniques are widely used to quickly prune unnecessary tests. These include spheres [26], axis-aligned bounding boxes (AABBs) [2], k-discrete oriented polytopes (K-DOPs) [16] and oriented bounding boxes (OBBs) [7]. BVHs are constructed based on these BVs, and have proven to be a very efficient technique to accelerate collision queries [3,21,27]. Efforts have been directed at optimizing update of BVHs for deformable models [18,24]. Spatial subdivision-based CD methods for deformable objects are discussed in [5,34,38]. Many authors present techniques to remove duplicate elementary tests [4,29,37] and reduce the number of false-positive [30,32].

The focus in recent years has shifted to GPU computing for CD. Earlier GPU-based CD algorithms use graphics shading languages [8,10,12,13,28]. Later, in order to more easily take advantages of GPUs' parallelism, specialised GPU computing languages and environments were designed, such as CUDA and OpenCL. A hybrid CPU/GPU method using CUDA is proposed in [15]. A complete GPU-based framework to construct and traverse BVHs is discussed in [19]. Based on this method, [20] present an improved CD algorithm. Tang et al. [31] abstract graphics hardware as a stream processor to develop a CD algorithm. Grand [9] describes how to use a uniform grid to perform broad-phase CD on GPU with CUDA for particle systems. Later, Pabst et al. [25] extend this method to handle CD for deformable triangular meshes. Other spatial subdivision-based CD approaches on GPUs include [1,6,17,36].

3 Approach

Like many existing efficient CD algorithms, our CD pipeline consists of a broad phase and a narrow phase. The task of the former is to cheaply prune unnecessary tests for triangle and other elementary pairs, such as vertex-triangle (VT) and edge–edge (EE) pairs that are far away from each other. In the narrow phase, proximity tests are performed to compute exact intersection information of elementary pairs, which are also referred to as elementary tests. AABBs and K-DOPs (specifically 18-DOPs) are used as the BV for the broad phase and narrow phase, respectively, in our work. Contributions of this paper relate primarily to the broad phase.

3.1 Adaptive space partition

As mentioned earlier, uneven triangle distributions can make spatial subdivision very inefficient. Therefore, adaptively partitioning space according to the distribution densities can offer advantages. We use an OTG to represent the adaptive subdivision, with triangles ultimately assigned to the leaf cells determined not to be subdivided. The top (coarsest) level is just a single cell which is defined as the AABB of the scene at the current time step. Cell sizes of lower levels are computed by halving the cell size of the parent level in x, y and z directions. Now, criteria to determine whether a cell should be subdivided is needed.

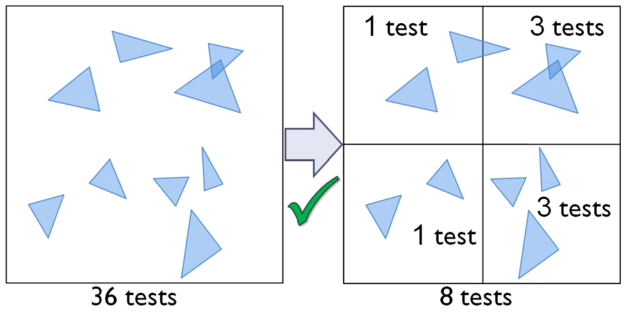


Fig. 2 The subdivision reduces the number of tests

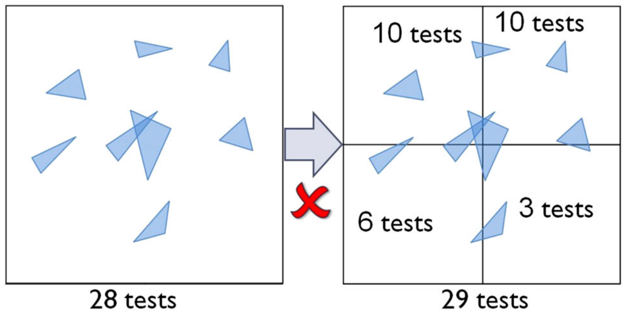


Fig. 3 The subdivision increases the number of tests

Since CD tests are performed only for triangle pairs in the same cell, the number of tests of a cell containing n triangles is $n(n-1)/2$. It is reasonable to define simple criteria that a cell needs to be subdivided if the number of tests decreases after subdivision. For example, for the 2D case shown in Fig. 2, subdivision reduces the number of tests from 36 to 8. In other words, a cell should be subdivided if subdivision reduces the number of tests (Fig. 3). With these criteria, the OTG can be built using a single-stage scheme. For example, it can be constructed top-down from the root to leaves to subdivide cells recursively until the criteria of reducing the number of tests fail. Alternatively, a bottom-up scheme can be used to construct the OTG. The OTG constructed by a single-stage scheme is referred to as OTG$_s$.

Although an OTG$_s$ offers a much better adaptation to spatial distributions compared to previous CD methods on GPUs, the efficiency of octree subdivision can be further improved. In the example shown in Fig. 4, the first subdivision increases the number of tests, so subdivision would stop with the simple criteria, although further subdividing three of the cells significantly reduces the number of tests. Hence deciding when to stop requires recording and comparing to other alternative subdivisions. This can dramatically increase the complexity since the number of subdivision combinations grows exponentially as the number of levels increases. A bottom-up scheme that determines if sub-cells need to be merged suffers the same limitations (Fig. 5). Hence a scheme that cheaply generates more efficient partitioning is required. We achieve this using a two-stage scheme to construct the OTG, which is referred to as OTG$_t$.

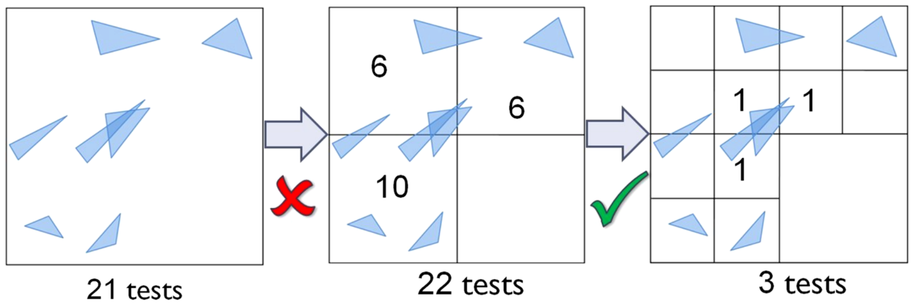


Fig. 4 The limitation of a simple top-down scheme

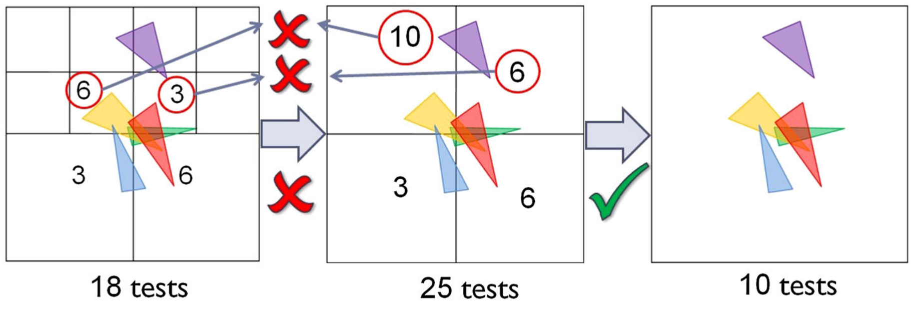


Fig. 5 A simple bottom-up scheme suffers a similar limitation

The OTG$_t$ is built as follows. At first, a counter octree and a type octree are built to record the triangle distribution information and cell types (internal, leaf or inactive). The counter tree is actually a hierarchy table stored as an array of a length equal to the number of the hierarchical grid (full octree) cells, in which each element corresponds to a cell in the hierarchy and records the number of triangles overlapped by this cell. Grid cells are marked as internal, leaf or inactive cells to represent the octree; therefore, traditional operations on tress such as adding, deleting and pruning are not needed, and thus the array length does not change during simulation. The type tree is treated as the same to store cell types. In fact on GPUs, maintaining and managing trees in this way are significantly more efficient in time than the traditional way of operating a tree, albeit at the cost of memory, which we found not to be a constraint in this work.

Next, all triangles are temporarily assigned to the base (finest) subdivision level cells only. Since triangles are very unlikely to overlap similar number of cells, this assignment step is performed using the workload distribution scheme proposed by [6]. During this process, the number of triangles assigned to each base level cell is obtained using atomic operations. For the number of triangles overlapped by higher subdivision level cells, we only cheaply record the number in the counter tree by mapping triangles to higher cells instead of expensively assigning them to higher cells. This is done by employing a token position technique [6] which efficiently removes the redundancy caused by the fact that a triangle assigned to multiple base cells can map to the same higher cell. Once the counter tree has distribution information, empty cells are marked as inactive cells, and the number of triangles overlapping each cell is converted to the number of tests.

Then, a two-stage scheme is used to obtain an improved octree subdivision. At first a bottom-up stage, using the simple criteria, checks level by level if all eight child cells of each parent cell should be merged or not, starting from the base level. If they should the only thing required to do is to mark the parent cell as a leaf cell. Otherwise, the eight child cells are marked as leaf cells and the parent cell are marked as an internal cell. Unlike the simple single-stage scheme, the bottom-up stage does not stop until the top level is reached. Therefore, in the counter tree, the number of tests of the internal parent cell should be replaced with the sum of numbers of tests of its eight child cells. After this stage is done, the highest level leaf cell in a branch is taken as the real leaf cell.

In the top-down stage, the first step is for all leaf cells to replace the number of tests stored in the counter tree with the cell identifier (ID). Then, IDs of the highest leaf cells are propagated down to elements corresponding to base level cells (excluding inactive cells), so the cells that triangles should be finally assigned to can be easily obtained by accessing base level elements of the counter tree. Now the OTG$_t$ that better subdivides space is built.

These two stages are implemented in two separate CUDA kernels that traverse the hierarchy level by level. The number of invoked threads in each iteration equals the number of cells in the corresponding level.

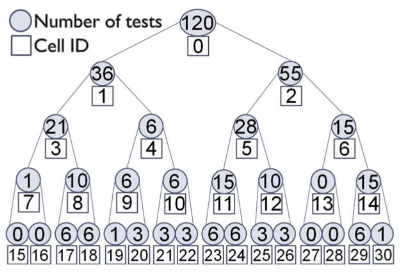
A 1D example is given in Fig. 6. The two-stage scheme easily removes the limitation of single-stage schemes. Although cells 19, 20 and 10 are marked as leaf cells in pass 1, the bottom-up stage does not stop and cell 4 is marked as a leaf in pass 2 as well. The top-down stage brings down IDs of the highest leaf cells to their child cells, which corrects the determination made in pass 1. An optimised subdivision that reduces the number of tests from 120 to 41 is obtained.

Intuitively, it seems that incrementally changing the octree by directly assigning triangles to leaf cells of previous step rather than rebuilding the tree from the hierarchical grid by assigning them to the base level of previous step might be more efficient. Unfortunately, a triangle can belong to multiple levels, so levels to which each triangle belongs to have to be recorded for the next step, and for each triangle the assignment step may need to be performed multiple times depending on how many levels the triangle belong to, leading to an increase in costs for incremental update. Moreover, since one thread handles the assignment of one triangle, uneven workloads can further harm the performance because how many times the assignment step is performed in every thread can be different. Therefore, we do not assign triangles to leaf cells directly.

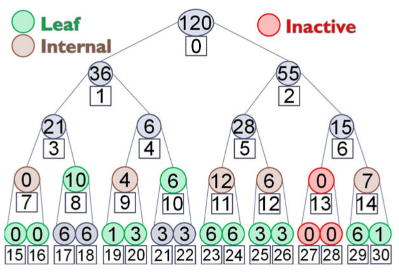
3.2 The number of cell levels

Since building the OTG starts with a bottom-up stage, the number of levels needs to be determined. Instead of using a hierarchy with a fixed number of levels, we dynamically adjust it during the simulation after the first time step. We start with a grid of nine levels in the first time step. After the two-stage scheme is done, how many triangles are assigned to each level is known. In terms of efficiency, it would be better to avoid calculations for inactive levels which are levels without any triangle assigned. Therefore, the base and top levels in the next time step can be determined from the levels in the current time step.

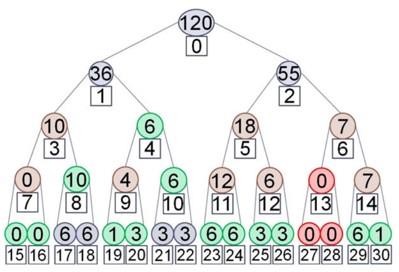
Assuming that the highest and lowest active levels in the current step are the ith and jth level ($i < j$), respectively, the top level in the next step is set as the $(i - 1)$th level if triangles assigned to the highest active level in the current step are more than a customised threshold ϵ_- (empirically set to 0.1 %), or it is set as the $(i + 1)$th level if triangles are less than a threshold ϵ_+ (0.001 %), otherwise it remains as the ith level. Similarly, the base level in the next step is set as the $(j + 1)$th level if triangles assigned to the lowest active level in the current step are more than a threshold λ_+ (97 %),



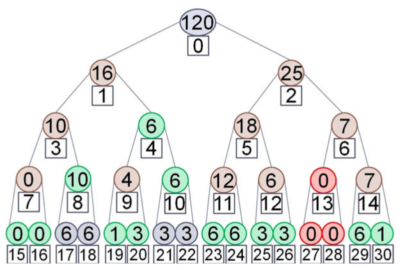
(a) a counter octree



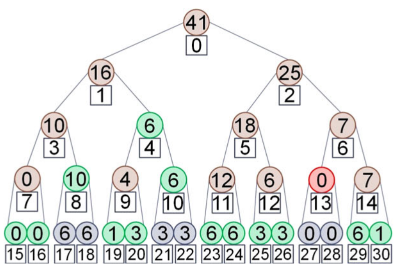
(b) bottom-up phase: pass 1



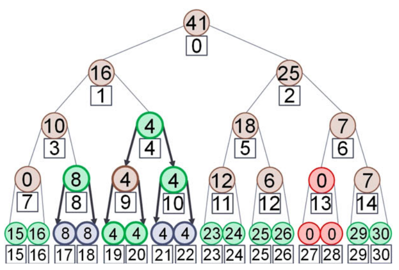
(c) bottom-up phase: pass 2



(d) bottom-up phase: pass 3



(e) bottom-upphase:pass4



(f) top-down phase

Fig. 6 A 1D example that each cell can be subdivided into two child cells is shown. In this counter octree, the number in a *box* represents a cell ID, and the number in a *circle* indicates the number of tests in that cell. **a** After the counter, octree which records the distribution information is built. The bottom-up stage is performed starting from the base level to check if all child cells (two cells in the 1D case) of every parent cell should be merged or not, according to the simple criteria. **b** During pass 1: cell 8, 10, 15, 16, 19, 20, 23, 24, 25, 26, 29 and 30 marked as leaf (*green*) cells, cell 7, 9, 11, 12 and 14 are marked as normal cells. For those parent cells that are normal, the number of tests is replaced with the sum of numbers of tests of its child cells. **c–e** This step is performed iteratively on upper levels until the top level is processed. Note that the first iteration processes two levels. **f** Finally, an top-down stage brings IDs highest leaf cells to their child cells level by level until the base level is reached, which corrects determinations made in previous passes

or it is set as the $(j-1)$th level if triangles are less than a threshold λ_{-} (3 %), otherwise it remains as the jth level. The two-stage scheme is only performed for levels between the top and base levels.

This dynamic adjustment mechanism works well in our benchmarks. Active levels are between the fourth and seventh levels for most of the benchmarks. In fact, levels higher than the third level and levels lower than the eighth level are inactive for all benchmarks.

3.3 Remainder of the CD pipeline

Once the space subdivision step is completed, a cell-triangle pair array is easily built, and then sorted by cell ID using the parallel radix sort algorithm [11]. Next, a workload distribution scheme [6] that assigns one broad-phase test to a thread is adopted to balance computation load among concurrent threads. In each thread, the token position technique is employed to determine if the test of this thread is redundant, since a triangle pair can overlap multiple cells. If not, a BV test for the triangle pair is performed. If the BVs intersect, R-triangle tests that remove redundant elementary tests are carried out. The last step of the broad phase is to perform a BV test for each elementary pair to further improve the culling efficiency. The collision pipeline up to here yields a considerable reduction in the number of elementary tests.

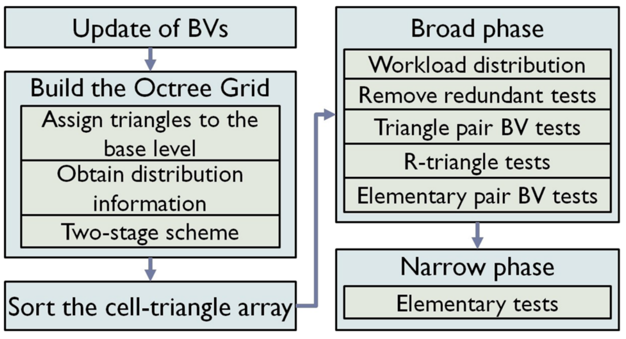


Fig. 7 The pipeline of the CD algorithm

In the narrow phase which is the same to the VSS method, proximity is checked for VT and EE pairs' output by the broad phase to compute exact intersection separately by launching two different kernels to avoid low occupancy. Before the proximity test, a cubic equation needs to be solved to compute the first contact time in the case of CCD [27]. Figure 7 shows the overview of the collision pipeline.

4 Scan with shuffle operations

Parallel scan is an important primitive in implementing many GPU algorithms. Its function is to return an output array with each element being a cumulative result of a binary associative operator applied on all previous elements of an input array. Additionally, two other useful parallel primitives, stream compaction and radix sort, also use parallel scan. These parallel primitives are called several times in every step in our algorithms. These operations are available in external libraries, such as CUDPP. We make a small change in CUDPP 2.1 [23] to improve performance of scan by taking advantage of newly available GPU shuffle operations.

The shuffle instruction is available on NVIDIA GPUs with compute capabilities 3.0 or above. It enables threads to directly read data from other threads within a warp instead of using shared memory, leading to performance gain, because only one instruction is required, whereas using shared memory requires three instructions: write, synchronize and read. Additionally, it frees up shared memory.

The small change is simple, and like that discussed in [22]. Data in a block are broken into several small warp sized groups (32 threads per warp), and a scan method proposed by [14] is used to scan the data of each warp using shuffle instructions. Because instructions within a warp are executed synchronously, explicit synchronization is not required. After the scan inside each warp is done, the partial result of all warps is stored in shared memory, then a single warp of threads loads the data and performs a scan on the partial results. Results computed by this single warp are correspondingly added to threads of each warp. Since [22] is designed for small size arrays, existing techniques are used to make it practical for larger arrays. Hence a pyramid type approach [11] is employed when multiple blocks are needed. Additionally, as [11] suggests, eight elements are processed by each thread to gain further optimisation.

Although the small change improves overall performance of our CD algorithm by just around 2 %, a reduction of computational time of scan itself of up to 16 % is observed (Fig. 8). We believe that it is worth mentioning this small change as it would likely benefit many other parallel algorithms.

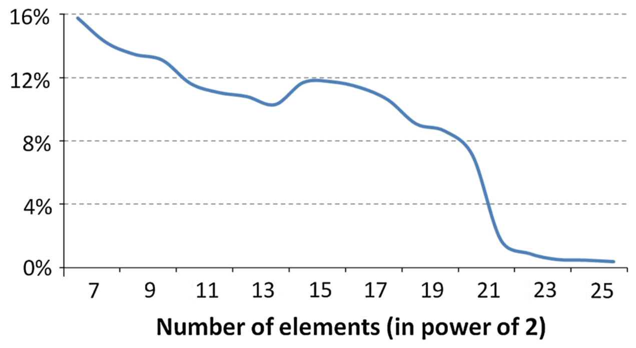


Fig. 8 The reduction of computational time of scan using shuffle instructions

5 Implementation and results

The method is implemented using the CUDA toolkit 5.5 with experiments run on a computer with an NVIDIA GTX 780 GPU which supports shuffle operations. The method is purely GPU-based, and data transfer between CPU and GPU is performed only once at the simulation beginning, so that performance completely depends on the GPU. We also run experiments on a GTX 470 GPU for comparison with previously reported methods, in which case CUDPP 2.1 is used to perform parallel scans.

5.1 Benchmarks

To demonstrate the performance of the method, a set of standard benchmarks with different scenarios is used. Refer to Online Resource 1 for the animation.

1. Funnel (20 K triangles): A piece of cloth falls into a narrow funnel and goes through it (Fig. 9a).
2. Cloth on ball (92 K triangles): A cloth with a large number of triangles drapes on a sphere, which starts rotating (Fig. 9b).
3. Flamenco (49 K triangles): A dancing character wearing a skirt with multiple layers of cloth (Fig. 9c).
4. N-body (146 K triangles): Hundreds of spheres and five cones are colliding with each other (Fig. 9d).
5. Reef knot (20 K triangles): Two pieces of ribbon are tied into a reef knot, resulting in a very uneven triangle distribution (Fig. 10).

The triangle spatial distribution density in these benchmarks is uneven in the most time steps. For example, in the cloth on ball and reef knot benchmarks, triangle distributions of areas under the sphere and near the knot are very dense, while triangle distributions of other areas are relatively sparse. Similarly in the flamenco test, the triangle density of the skirt of multiple layers of cloth is higher than that of the shirt, because the skirt consists of multiple layers of cloth. In addition, some benchmarks consist of meshes whose triangle sizes vary considerably.

5.2 Results and analysis

Active levels of the OTG are adjusted depending on the distribution of triangles in the hierarchy. In all benchmarks, the ratio between the number of frames in which each level is active and the number of total frames, and the ratio between

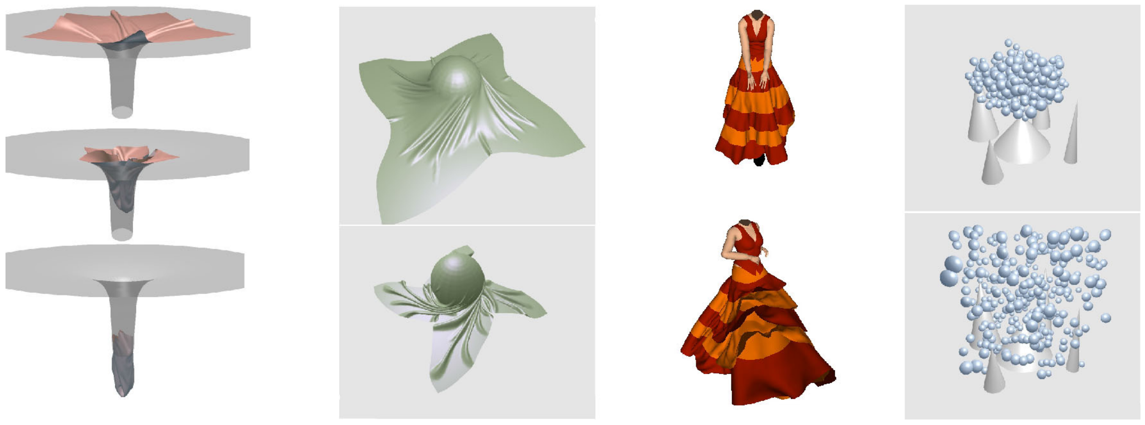


Fig. 9 Benchmarks used for performance measuring

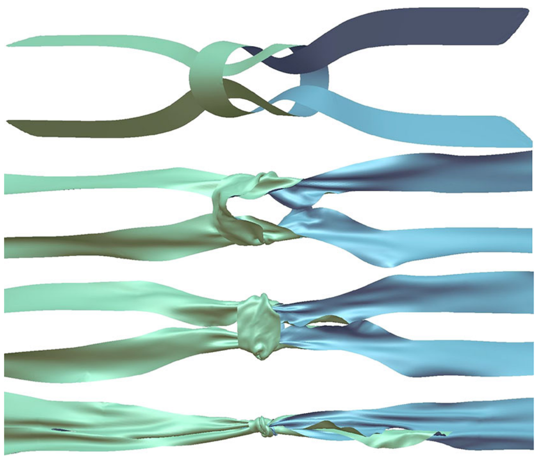


Fig. 10 Two pieces of ribbon are tied into a reef knot

Table 1 The ratio (in percentage) between the number of time steps in which each level is active (A) and the number of total time steps, and the ratio (in percentage) between the number of triangle-cell registrations (R) in each level and the number of the total registrations are given

	Level					
	3	4	5	6	7	8
Funnel						
A	–	100	100	100	23.6	–
R	–	4.7e−3	2.1e−1	70.8	29.1	–
Cloth ball						
A	1.6	100	100	100	87.5	3.4
R	1.6e−3	5.4e−2	3.5e−1	2.2	92.7	4.9
Flamenco						
A	4.9	100	100	100	87.2	–
R	1.5e−3	6.1e−2	5.4e−1	43.6	56.3	–
N-body						
A	12.5	100	100	100	96.0	10.1
R	2.9e−3	8.3e−2	4.7e−1	6.5	80.2	13.3
Reef knot						
A	–	100	100	100	75.3	–
R	–	4.5e−2	2.0e−1	10.7	88.3	–

the number of triangle-cell registrations in each level and the number of the total triangle-cell registrations are given in Table 1. Levels 1, 2 and 9 are inactive during the whole simulation for all benchmarks. In contrast levels, 4, 5 and 6 are always active. Most triangle-cell pairs are registered in levels 6 and 7.

The key idea of the OTG method is to reduce the number of unnecessary broad-phase tests resulting from an uneven triangle spatial distribution. We compare the average number of broad-phase tests per frame between the OTG$_t$ method and the VSS method which drastically reduces the broad-phase tests for a uniform subdivision. As Fig. 11 shows, the OTG$_t$ method yields a significant reduction in tests, by at least 39 %, across all benchmarks. The largest reduction, of 64 %, is observed in the reef knot test. This is because the density of the knot area is very high and the density of remaining areas is low, resulting in an inefficient subdivision for a uniform grid as expected. In contrast, the OTG$_t$ adaptively partitions the space according to the distribution density.

To demonstrate the further improvement of OTG$_t$, the number of broad-phase tests produced by the OTG$_s$ method is also given in this figure. In all benchmarks, the OTG$_t$ method partitions space for a better adaptation to triangle distributions, yielding less tests compared to OTG$_s$ by up to 36 %, and an average of 20 %.

A common feature of CD methods employing space subdivision techniques is the large memory requirement, because of the large number of broad-phase tests. This feature becomes a major limitation for GPU-based methods. However, memory usage for tests is not our main concern, because the two-stage scheme is designed to generate as few

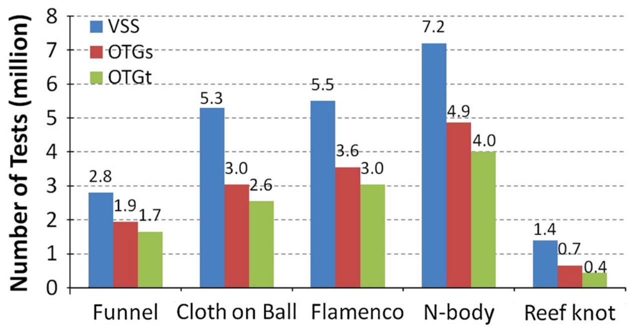


Fig. 11 Comparison of the average number of broad-phase tests in CCD among the VSS method, the OTG$_s$ method and the OTG$_t$ method

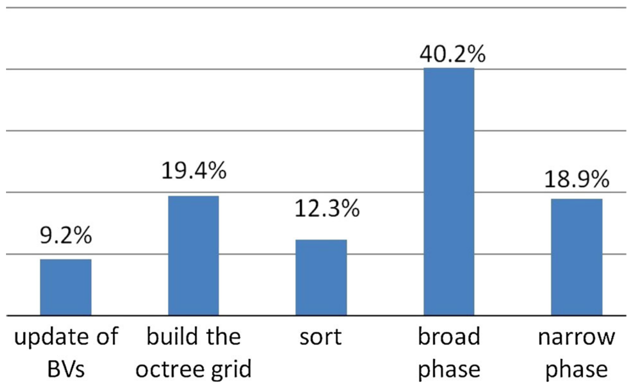


Fig. 12 Time proportions of each step for cloth on ball benchmark

Table 2 Timings of the two-stage scheme method for DCD and CCD, measured on a GTX 470 GPU and a GTX 780 GPU

		GTX 470		GTX 780	
	Triangles ×1,000	DCD (ms)	CCD (ms)	DCD (ms)	CCD (ms)
Funnel	18	4.7	5.1	3.7	3.4
Cloth ball	92	10.5	13.3	8.3	9.2
Flamenco	49	16.1	17.4	11.7	10.8
N-body	146	30.2	24.8	20.8	19.1
Reef knot	20	4.7	4.9	3.2	3.0

Table 3 The DCD and CCD timings of the VSS method run on our experimental environment with a GTX 780 GPU, and speedups of the two-stage scheme method

	DCD (ms)	Speedup (times)	CCD (ms)	Speedup (times)
Funnel	4.4	1.2	4.0	1.2
Cloth ball	10.2	1.2	11.1	1.2
Flamenco	11.2	0.9	11.0	1.0
N-body	18.7	0.9	24.5	1.3
Reef knot	4.3	1.3	4.4	1.5

as possible broad-phase tests. However, memory is required for two octrees. Maintaining a tree grid as a table requires a relatively large amount of memory. Since each node only stores an integer value, the memory size of the counter octree of nine levels is 512 MB, and the size of the type octree of nine levels is 64MB because the base level is unnecessary (the type of base level cells can be obtained from their parent cells). Fortunately, for the test scenes, this is not a limitation in an environment with a GTX 780 GPU whose memory is 3 GB. The number of OTG levels for all these standard benchmarks never reaches nine, which means a memory of 72 MB satisfies two octrees. For extreme situations, spatial hashing can be adopted to easily avoid using more than nine levels.

5.3 Performance

The performance of the OTG$_t$ method for DCD and CCD on benchmarks is given in Table 2. The results demonstrate that this method works well with scenes with uneven triangle spatial distributions. Execution times of CCD on steps for cloth on ball are presented in Fig. 12. The most expensive part is the broad phase which takes 40 %, while computation time of building the OTG takes 19 % of the total running time.

5.4 Comparison

We first compare the OTG$_t$ method against the VSS method, not only because the VSS method provides the fastest results for most benchmarks among previous methods we are aware of, but also the OTG$_t$ method is designed to address the uneven triangle spatial distribution issue, the main limitation of the VSS method. To remove uncertain factors arising from different hardware and environments, the VSS method is run on our experiment environment (scan with shuffle operations is used). Their performance measured on a GTX 780 GPU is given in Table 3 to provide a straightforward comparison.

Performance of the OTG$_t$ method for CCD on all benchmarks is faster with speedups of 1.5× and 1.3× for the reef knot and N-body benchmarks. For DCD, the VSS method gives slightly better performance for the flamenco and N-body benchmarks, while the OTG$_t$ method is around 1.3×, 1.2× and 1.2× faster for the reef knot, cloth on ball and funnel tests respectively. Although these benchmarks consist of triangles whose size also varies significantly and the VSS method is designed to handle this problem, the OTG$_t$ method is superior for the VSS method for most tests, which demonstrates that the OTG is able to address the uneven triangle size issue as well. However, the improvement in performance is not as remarkable as the significant reduction in the number of broad-phase tests, and even not observed for two of the DCD tests. This is mainly because the control bits scheme employed by the VSS method to remove the redundancy arising from the fact that a triangle pair can overlap multiple cells is more efficient than the token position technique used in the OTG approach, which cancels out part of the performance gain. It is also noticeable that the increase in CCD performance is greater than that in DCD. The reason is that by following [3], the BV is enlarged by a cloth thickness in the DCD case, while in the CCD case the BV is enlarged by a error tolerance which is much smaller than the cloth thickness. Therefore, BVs of triangle pairs are more likely to overlap in the case of DCD, thus there is less potential for the OTG to reduce the number of tests. This is also why the OTG_t method is faster for CCD of the N-body benchmark, but slower for DCD.

Fan et al. [6] address the issue of uneven spatial distribution in part by presenting a two level hierarchical grid. Their reported performance results are measured on a GTX 480 GPU, whilst we compare with results measured on a slightly slower GTX 470 GPU. The OTG_t method is faster for CCD in all except for the flamenco benchmark. For the cloth on ball, funnel and N-body tests, speedups of from 1.3× to 1.9× are observed. We do not provide a comparison of DCD performance, because for DCD in the narrow phase they only check collisions between triangle pairs, while we perform collision tests on VT and EE pairs to compute exact collision and contact information.

To the best of our knowledge, Tang et. al. [31] report the fastest results among GPU-based methods employing BVHs. They also run experiments on a GTX 480 GPU. CCD performance of the OTG_t method measured on a GTX 470 GPU is better in cloth on ball, flamenco and N-body benchmarks: 13.3 vs. 18.6 ms, 17.4 vs. 32.7 ms and 24.8 vs. 79.0 ms, respectively, although is slightly slower for the funnel test.

Finally, by comparing our CCD timings on a GTX 470 GPU to the fastest multicore CPU-based method [33] where experiments are run on a PC with four quad-core CPUs (16 cores) at 2.93 GHz, speedups of 1.6× and 1.3× are observed for the flamenco and cloth on ball tests, on which the multicore CPU-based method takes 27 and 32.5 ms. Performance of the OTG_t method on a GTX 780 GPU for these two tests is 2.5× and 1.7× faster. DCD timings of these methods are not provided.

6 Conclusion and future work

We present a GPU-based method using an OTG to accelerate CD of deformable objects by adaptively subdividing space according to triangle distributions. The OTG is dynamically built and adjusted. A two-stage scheme is introduced to optimise octree spatial subdivision and to reduce the number of broad-phase tests. This method addresses the issue of uneven triangle spatial distributions, a common limitation not addressed well in previous spatial subdivision-based CD approaches on GPUs. Further, the uneven triangle size problem can be addressed by this method as well, since it can be seen as a subcase of uneven spatial distributions. We have investigated the method by conducting a set of standard benchmarks, and observed increases in performance compared to previous CPU-based and GPU-based approaches.

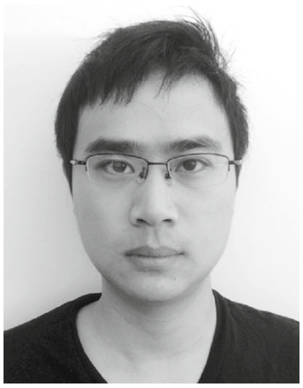
There are a number of avenues for our future work. We would like to develop CD methods that exploit new features of GK110 based GPUs including dynamic parallelism, hyper Q and grid management unit. Further utilising shuffle instructions to accelerate scan and other parallel primitives is also an area for further work.

Acknowledgments The cloth on ball and N-body benchmarks is courtesy of the UNC Dynamic Scene Benchmarks collection. The flamenco benchmark is courtesy of Walt Disney Animation Studios and was provided by Rasmus Tamstorf. The reef knot model was provided by David Harmon.

References

1. Alcantara, D.A., Sharf, A., Abbasinejad, F., Sengupta, S., Mitzenmacher, M., Owens, J.D., Amenta, N.: Real-time parallel hashing on the GPU. ACM Trans. Graph. **28**(5), 154:1–154:9 (2009)
2. van den Bergen, G., Van, G., Bergen, D.: Efficient collision detection of complex deformable models using aabb trees. J. Graph. Tools **2**, 1–13 (1998)
3. Bridson, R., Fedkiw, R., Anderson, J.: Robust treatment of collisions, contact and friction for cloth animation. In: ACM SIGGRAPH 2005 Courses, SIGGRAPH ’05. ACM, New York (2005)
4. Curtis, S., Tamstorf, R., Manocha, D.: Fast collision detection for deformable models using representative-triangles. In: Proceedings of the 2008 symposium on Interactive 3D graphics and games. I3D ’08, pp. 61–69. ACM, New York (2008)
5. Eitz, M., Lixu, G.: Hierarchical spatial hashing for real-time collision detection. In: Proceedings of the IEEE International Conference on Shape Modeling and Applications 2007, pp. 61–70. IEEE Computer Society, Washington, DC (2007)
6. Fan, W., Wang, B., Paul, J.C., Sun, J.: A hierarchical grid based framework for fast collision detection. Comput. Graph. Forum **30**(5), 1451–1459 (2011)
7. Gottschalk, S., Lin, M.C., Manocha, D.: OBBTree: A hierarchical structure for rapid interference detection. In: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH ’96, pp 171–180. ACM, New York (1996)
8. Govindaraju, N.K., Redon, S., Lin, M.C., Manocha, D.: Cullide: interactive collision detection between complex models in large environments using graphics hardware. In: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware, HWWS ’03, pp. 25–32. Eurographics Association (2003)
9. Grand, S.L.: Broad-phase collision detection with cuda. In: GPU Gems 3 (2007)
10. Greß, A., Guthe, M., Klein, R.: GPU-based collision detection for deformable parameterized surfaces. Comput. Graph. Forum **25**(3), 497–506 (2006)
11. Harris, M.: Parallel Prefix Sum (Scan) with CUDA. In: GPU Gems 3 (2007)
12. Heidelberger, B., Teschner, M., Gross, M.: Detection of collisions and self-collisions using image-space techniques. J. WSCG **12**(3), 145–152 (2004)

13. Heidelberger, B., Teschner, M., Gross, M.H.: Real-time volumetric intersections of deforming objects. In: VMV, pp. 461–468 (2003)
14. Hillis, W.D., Steele G.L., Jr.: Data parallel algorithms. Commun. ACM **29**(12), 1170–1183 (1986). doi:10.1145/7902.7903
15. Kim, D., Heo, J.P., Huh, J., Kim, J., Yoon, S.E.: Hpccd: hybrid parallel continuous collision detection using CPUs and GPUs. Comput. Graph. Forum **28**(7), 1791–1800 (2009)
16. Klosowski, J.T., Held, M., Mitchell, J.S.B., Sowizral, H., Zikan, K.: Efficient collision detection using bounding volume hierarchies of k-dops. IEEE Trans. Vis. Comput. Graph. **4**, 21–36 (1998)
17. Kroiss, R.R.: Collision detection using hierarchical grid spatial partitioning on the GPU. ProQuest Dissertations and Theses, University of Colorado at Boulder, p 45 (2013)
18. Larsson, T., Akenine-Mller, T.: A dynamic bounding volume hierarchy for generalized collision detection. Comput. Graph. **30**(3), 450–459 (2006). doi:10.1016/j.cag.2006.02.011
19. Lauterbach, C., Garland, M., Sengupta, S., Luebke, D., Manocha, D.: Fast BVH construction on GPUs. Comput. Graph. Forum **28**(2), 375–384 (2009)
20. Lauterbach, C., Mo, Q., Manocha, D.: gProximity: hierarchical GPU-based operations for collision and distance queries. Comput. Graph. Forum **29**(2), 419–428 (2010)
21. Mezger, J., Kimmerle, S., Etzmu, O.: Hierarchical techniques in collision detection for cloth animation. J. WSCG **11**(2), 322–329 (2003)
22. NVIDIA.: Cudashuffle: Cuda parallel prefix sum with shuffle intrinsics. In: Cuda samples (2013)
23. NVIDIA.: Cudpp: Cuda data parallel primitives library (2013)
24. Otaduy, M.A., Chassot, O., Steinemann, D., Gross, M.: Balanced hierarchies for collision detection between fracturing objects. Virtual Reality Conference, IEEE 83–90 (2007)
25. Pabst, S., Koch, A., Straer, W.: Fast and scalable CPU/GPU collision detection for rigid and deformable surfaces. Comput. Graph. Forum **29**(5), 1605–1612 (2010)
26. Palmer, I.J., Grimsdale, R.L.: Collision detection for animation using sphere-trees. Comput. Graph. Forum **14**(2), 105–116 (1995)
27. Provot, X.: Collision and self-collision handling in cloth model dedicated to design garments, vol. 97, pp. 177–189. Citeseer (1997)
28. Rodrïguez-Navarro, J., Sainz, M., Susïn, A.: GPU based cloth simulation with moving humanoids. In: Actas XV Congreso Espaol de Informtica Grfica, pp. 147–155 (2005)
29. Tang, M., Curtis, S., Yoon, S.E., Manocha, D.: Interactive continuous collision detection between deformable models using connectivity-based culling. In: Proceedings of the 2008 ACM symposium on Solid and physical modeling. SPM ’08, pp. 25–36. ACM, New York (2008)
30. Tang, M., Curtis, S., Yoon, S.E., Manocha, D.: Interactive continuous collision detection between deformable models using connectivity-based culling. In: SPM ’08: Proceedings of the 2008 ACM symposium on Solid and physical modeling, pp. 25–36. ACM, New York (2008)
31. Tang, M., Manocha, D., Lin, J., Tong, R.: Collision-streams: fast gpu-based collision detection for deformable models. In: Symposium on Interactive 3D Graphics and Games, I3D ’11, pp. 63–70. ACM, New York (2011)
32. Tang, M., Manocha, D., Tong, R.: Fast continuous collision detection using deforming non-penetration filters. In: Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games. I3D ’10, pp. 7–13. ACM, New York (2010)
33. Tang, M., Manocha, D., Tong, R.: Mccd: multi-core collision detection between deformable models using front-based decomposition. Graph. Models **72**(2), 7–23 (2010)
34. Teschner, M., Heidelberger, B., Müller, M., Pomerante, D., Gross, M.H.: Optimized spatial hashing for collision detection of deformable objects. VMV **3**, 47–54 (2003)
35. Teschner, M., Kimmerle, S., Zachmann, G., Heidelberger, B., Raghupathi, L., Fuhrmann, A., Cani, M.-P., Faure, F., Magnenat-Thalmann, N., Strasser, W., Volino, P.: Collision detection for deformable objects. Comput. Graph. Forum **24**(1), 61–81 (2005). doi:10.1111/j.1467-8659.2005.00829.x
36. Wong, T., Leach, G., Zambetta, F.: Virtual subdivision for GPU based collision detection of deformable objects using a uniform grid. Vis. Comput. **28**, 829–838 (2012)
37. Wong, W.S.K., Baciu, G.: A randomized marking scheme for continuous collision detection in simulation of deformable surfaces. In: Proceedings of the 2006 ACM international conference on Virtual reality continuum and its applications. VRCIA ’06, pp. 181–188. ACM, New York (2006)
38. Zhang, D., Yuen, M.F.: Collision detection for clothed human animation. In: Computer Graphics and Applications, 2000. Proceedings. The Eighth Pacific Conference on, pp. 328–337 (2000)



Tsz Ho Wong is a PhD student in Computer Science at RMIT University. He received the master’s degree in RMIT University, Australia, in 2010. His major research interests include physics simulation and GPU computing.

Geoff Leach is a lecturer in the School of Computer Science and Information Technology at RMIT University. His major research interests include computer graphics and distribution system.

Fabio Zambetta is a senior lecturer in the School of Computer Science and Information Technology at RMIT University. In 2004, Dr Zambetta earned his PhD degree in Universitë degli Studi di Bari. His major research interests include computer graphics, artificial intelligence and applied mathematics.