CDFC: Collision Detection Based on Fuzzy Clustering for Deformable Objects on GPU's

David Mainzer
Clausthal University of Technology
Julius-Albert-Straße 4
38678 Clausthal-Zellerfeld, Germany
dm@tu-clausthal.de

Gabriel Zachmann
University of Bremen
Bibliothekstraße 1
28359 Bremen, Germany
zach@informatik.uni-bremen.de

ABSTRACT

We present a novel *Collision Detection Based on Fuzzy Clustering for Deformable Objects on GPU's* (CDFC) technique to perform collision queries between rigid and/or deformable models. Our method can handle arbitrary deformations and even discontinuous ones. With our approach, we subdivide the scene into connected but totally independent parts by fuzzy clustering, and therefore, the algorithm is especially well-suited to GPU's. Our collision detection algorithm processes all computations without the need of a bounding volume hierarchy or any other acceleration data structure. One great advantage of this is that our method can handle the broad phase as well as the narrow phase within one single framework. We can compute inter-object and intra-object collisions of rigid and deformable objects consisting of many tens of thousands of triangles in a few milliseconds on a modern computer. We have evaluated its performance by common benchmarks. In practice, our approach is faster than earlier CPU- and/or GPU-based approaches and as fast as state-of-the-art techniques but even more scalable.

Keywords

collision detection, fuzzy clustering, physics based animation, computer animation, cloth simulation

1 INTRODUCTION

Collision detection between rigid, and/or soft bodies is important for many fields of computer science . The underlying collision detection needs to check if collisions occur between a pair of objects as well as self-collisions among deformable objects. In many applications, an additional requirement is that the collision detection has to be calculated within milliseconds.

There exist various approaches that propose spatial subdivision for collision detection or approximate the surface of rigid and soft bodies. These algorithms employ axis-aligned bounding boxes (AABB), oriented bounding boxes (OBB) or Inner Sphere Trees (IST) [8] to reduce the computation time.

Most of the earlier efficient collision detection algorithms were sequential ones, which are perfect for devices that can execute only one instruction at a time. The current trend in computer architecture focuses on multi-core CPUs and many-core GPU's, and so many parallel collision detection algorithms have been proposed in the last years. The collision detection algorithm we present in this paper is a fast, fully GPU-based algorithm that can exploit data and thread-level parallelism.

1.1 Our Contributions

Our novel ***C**ollision **D**etection Based on **F**uzzy **C**lustering for Deformable Objects on GPU's* (CDFC) algorithm is designed for interactive and exact collision detection in complex environments and can handle objects movement and deformation at the same time. To achieve these features, our algorithm subdivides the whole scene into independent, overlapping parts in the first step. For the segmentation process, we use a GPU-based clustering algorithm called *fuzzy C-means*. For all clusters, we can execute the collision detection steps independently, and this offers the possibility to distribute the collision detection computation for the clusters to different GPU's.

Our novel approach is as fast as state-of-the-art collision detection algorithms but with the additional advantage that our collision detection can be distributed easily to more than only one GPU; thus it scales very well with the number of GPU's. Also, our collision detection algorithm works directly on all primitives of the whole scene, which results in a simpler implementation and can be implemented much more easily by other applications.

2 PREVIOUS WORK

In this section, we focus on those approaches that can handle collisions between deformable objects.

2.1 Approaches Using Bounding Volume Hierarchies

Using Bounding Volume Hierarchies (BVH) is the most common approach to speed up collision detection of rigid and deformable objects. A GPU-based linear BVH (LBVH) approach was presented by Lauterbach et al. [4]. Updating these LBVH over more than one GPU is difficult and leads to a huge communication overhead.

2.2 GPU-based Collision Detection

Most modern collision detection algorithms using BVH are GPU based. However, there are some approaches which use e.g. distance fields or space subdivision to improve their performance. A hybrid CPU/GPU collision detection technique based on spatial subdivision was presented by Pabst et al. [5]. They prune away non-colliding parts of the scene by using an adapted highly parallel spatial subdivision method.

3 SWEEP-PLANE TECHNIQUE USING PCA

Due to the fact that, our collision detection approach treats all objects in a scene at the same time, we make no differences between individual objects in the rest of this paper.

During the collision detection process we use an adapted version of the standard Sweep and Prune approach, a 1D version, hereafter referred to as *Sweep-Plane* technique. We compute the bounding box for every triangle. Each bounding box spans an interval $[S_i, E_i]$ for each triangle T_i on the x-axis. Sorting all intervals along the x-axis provides information about possible colliding bounding boxes.

There exists a downside of using bounding volumes, like AABB's or OBB's. If, for example, primitives are moving then in a significant amount of cases a huge number of false positives may occur, when we choose any of the *fixed* world coordinate axes as sweep direction. In our case, the best sweep direction is the one, that allows projection to separate the primitives as much as possible. In order to achieve the best sweep direction, even if the primitives move through 3D spaces, we compute the *principal component analysis* (PCA) [2] in every frame, because the direction of the first principal component maximizes the variance of primitives, after projection.

Therefore, we move the direction of the first principal component on the x-axis. Now we compute the bounding box intervals $[S_i, E_i]$ and use the x-axis, more specifically the direction of the first component of the principal component analysis, respectively, as sweep direction. As a consequence, combining Sweep-Plane and PCA reduces the number of primitive pairs tested for intersection.

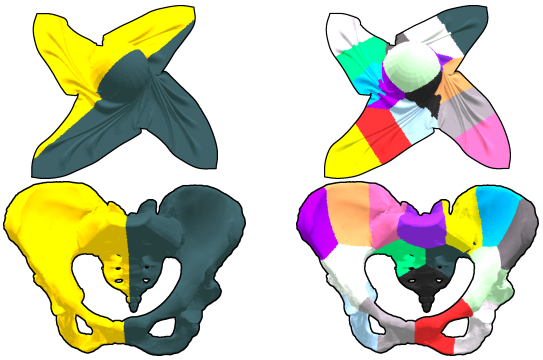

Figure 1: Examples of some high-detail objects, partitioned by fuzzy C-means into two (left column) and 16 clusters, respectively. From top to bottom: Cloth on Ball (92k) and Model of the Female Pelvis (200k).

4 OBJECT SUBDIVISION USING FUZZY C-MEANS

Using the first principal component as sweep direction only, will nevertheless produce false positives, because of the dimensional reduction in the Sweep-Plane step. The Sweep-Plane technique, used to separate the primitives, projects all 3D bounding volumes to 1D points.

To eliminate this kind of false positives we subdivide the scene (see Figure 1 for some examples) into connected components using *fuzzy C-means* (FCM) algorithm [1]. We use a fuzzy clustering algorithm because the triangles, which are located on the border between two clusters, have to be in both clusters. If adjoining clusters are not connected, then in some cases collisions across the border of the clusters would not be taken into account.

The FCM algorithm is a soft, or fuzzy, version of the well-know k-means clustering algorithm. The algorithm tries to minimize the total error, which is the sum of the squared distances of each data point to each cluster center, if we use the euclidean distance, weighted by the membership of the data point to each cluster, for all data points.

5 GPU-BASED COLLISION DETECTION

In this section we show how our method combines all previously introduced techniques. Algorithm 1 provides a short overview of the pipeline of our collision detection approach with the main procedures.

Algorithm 1 GPU-based Collision Detection

A line represents a massively parallel computation kernel

Input: triangles of all objects
Output: intersecting triangle pairs

```
1:  subdivide scene into c clusters using fuzzy C-means
2:  for all clusters do in parallel
3:      compute PCA and apply PCA
4:      compute AABBs and sort AABBs along x-axis
5:      collect all overlapping intervals
6:      for all overlapping intervals do in parallel
7:          if AABB intersect along y-axis then
8:              do triangle-triangle intersection test
9:          end if
10:     end for
11: end for
```

First of all, we subdivide the whole scene into independent, overlapping parts by fuzzy clustering. Thus, we use the centroid of all triangles to decide to what cluster a triangle belongs to.

Now we can do the following steps for every cluster independently. As described in Section 3, we do a principal component analysis using the centroid of the triangles of the cluster. The result of the PCA is applied to the triangles of the cluster, which means that the direction of the first component of the principal component analysis points along the x-axis (step "Clustering and PCA" in Figure 3 and 4).

We are now using the x-axis as sweep plane direction because this direction maximizes the variance of primitives after projection. Therefore, we compute the bounding box of all triangles of this cluster (step "Compute AABBs" in Figure 3 and 4).

After computing the bounding boxes for all triangles of this cluster, we sort them along the x-axis using a highly-tuned Radix Sort algorithm from the Thrust[1] library.

The next challenge is to collect all bounding box intervals which intersects in the x-dimension. In order to avoid counting overlapping bounding boxes twice, we only consider the start point (S_i) of a bounding box interval. In order to receive the required memory and the position where to put all possible colliding pairs, we use the prefix sum algorithm from the Thrust library. This step, see "Collect overlapping intervals" in Figure 3 and 4, takes up the most computation time in our collision detection algorithm. The problem is that it is not possible to access the memory completely coalesced, which slows down the computation process.

After collecting all possible colliding pairs, we verify whether the bounding boxes of both triangles overlaps in the y-dimension or not. We omit an bounding box overlap test for the z-dimension, because it takes more time to read the bounding box information from memory and to compare the values, than using the triangles vertices, which may potentially needed further in the case both triangles intersect, to test if the triangles overlap in the z-dimension. If that is the case, and both triangles overlap in all three dimensions, the algorithm performs a triangle-triangle intersection test.

[1] http://thrust.github.com/

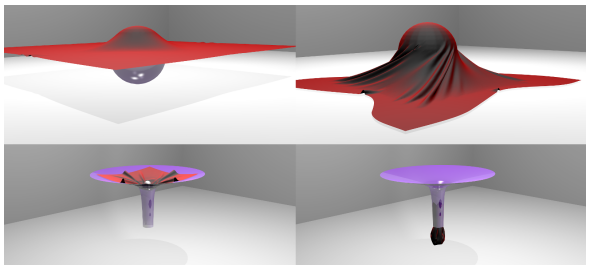

Figure 2: The upper row shows the frames 10 and 60 of the Cloth on Ball benchmark. The lower row shows the frames 125 and 375 of the Funnel benchmark.

Our collision detection algorithm compute all colliding triangle pairs and, if needed, the intersection point or line, respectively.

5.1 Accuracy & Limitations

Our collision detection algorithm will recognize every intersection between all triangles. Therefore, our approach perform bounding box intersection tests with all triangles of a cluster, to detect all colliding triangle pairs. However, in the case of significant differences in the size of the triangles, it could happen that a triangle is completely assigned to one cluster, but collides with a triangle which is completely assigned to an adjoining cluster. The reason for this is that, our approach use the centroid, which represents a triangle, for the clustering process. To prevent this, we have to decrease the membership value in the clustering step. This results in a higher degree of overlap between adjoining clusters. The size of the overlap has to be at least as large as the overall maximum distance from triangle's centroid to one of its vertices:

$$\max_{i=1,2,\ldots,n} \left(\max_{k=0,1,2} (\|C_i - vertex_{i,k}\|_2) \right) \tag{1}$$

6 RESULTS

We have implemented our collision detection algorithm on a NVIDIA GeForce GTX 480. Therefore, we used the CUDA toolkit 5.0 as development environment. For sorting and prefix computation steps we used Thrust, a parallel algorithms library.

6.1 Benchmarking

To evaluate the performance of our collision detection algorithm in different situations, we choose some often

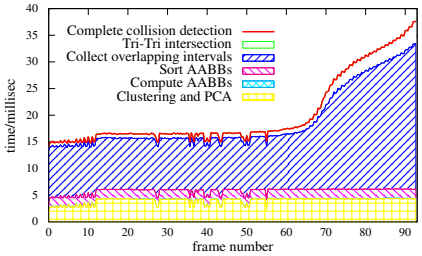

Figure 3: Collision detection time needed for Cloth on Ball (92k triangles) Benchmark.

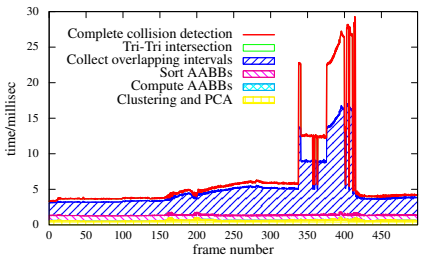

Figure 4: Collision detection time needed for Funnel (18.5k triangles) Benchmark.

used collision detection benchmarks to compare our results against other approaches.

In Table 1 we show the average collision detection time needed for all benchmarks compared with state-of-the-art collision detection algorithms.

Bench.	Our	CSt.	Pab.	HP	MC
Cl. on Ball	20.24	18.6	36.6	23.2	32.5
Funnel	6.53	4.4	6.7	–	–

Table 1: Timings (in ms) include both external and self-collision detection; CSt.[6], Pab.[5], HP[3], MC[7]

6.1.1 Cloth on Ball

In this benchmark a cloth (92k triangles) drops down on a rotating ball (760 triangles) (see Figure 2 upper row). Thereby the cloth has a huge number of self-collisions.

Figure 3 shows that the collision detection time needed to compute all collisions from frame 60 onwards increase because the number of self-collisions increase heavily like you can see on the Figure 2 (upper row). Our collision detection algorithm needs more time to collect all possible colliding triangles and has to do more intersection tests between them.

6.1.2 Funnel

A cloth (14.4k triangles) falls into a funnel (2k triangles) and passes through it, due to the force applied by a ball (1.7k triangles), who slowly increased in volume over the time (see Figure 2 lower row).

Figure 4 depicts that the collision detection time needed to compute all collisions increase slightly between frame 150 and frame 345. In these frames the cloth hit the funnel and slides a little bit into the funnel. From frame 345 onwards the ball push the cloth trough the funnel, and produces a huge number of self-collisions which results in an higher computation time needed for collision detection.

7 CONCLUSIONS

We presented a novel, accurate and fast collision detection algorithm which is completely GPU-based and needs no additional communication between host (CPU) and device (GPU) is necessary. Our *Collision Detection Based on Fuzzy Clustering for Deformable Objects on GPU's* technique can perform collision queries between rigid and/or deformable models consisting of many tens of thousands of triangles in a few milliseconds. Our results show that our collision detection algorithm is as fast as state-of-the-art approaches.

8 REFERENCES

[1] J.C. Bezdek. *Pattern recognition with fuzzy objective function algorithms*. Kluwer Academic Publishers, 1981.

[2] I. Jolliffe. *Principal component analysis*. Wiley Online Library, 2005.

[3] D. Kim, J.P. Heo, J. Huh, J. Kim, and S. Yoon. Hpccd: Hybrid parallel continuous collision detection using cpus and gpus. In *Computer Graphics Forum*, volume 28, pages 1791–1800. Wiley Online Library, 2009.

[4] C. Lauterbach, Q. Mo, and D. Manocha. gproximity: Hierarchical gpu-based operations for collision and distance queries. In *Computer Graphics Forum*, volume 29, pages 419–428. Wiley Online Library, 2010.

[5] S. Pabst, A. Koch, and W. Straßer. Fast and scalable cpu/gpu collision detection for rigid and deformable surfaces. In *Computer Graphics Forum*, volume 29, pages 1605–1612. Wiley Online Library, 2010.

[6] M. Tang, D. Manocha, J. Lin, and R. Tong. Collision-streams: fast gpu-based collision detection for deformable models. In *Symposium on Interactive 3D Graphics and Games*, pages 63–70. ACM, 2011.

[7] Min Tang, Dinesh Manocha, and Ruofeng Tong. Mccd: Multi-core collision detection between deformable models using front-based decomposition. *Graphical Models*, 72(2):7–23, 2010.

[8] Rene Weller and Gabriel Zachmann. Inner sphere trees for proximity and penetration queries. In *Robotics: Science and Systems Conference (RSS)*, Seattle, WA, USA, June/July 2009.