Proceedings of the 5th International Conference on Signal Processing and Machine Learning
DOI: 10.54254/2755-2721/108/2025.21193

Collision Detection Algorithms for Deformable Models: A Literature Review

Xiaoyue Wang[1,a,*], Zhongzheng Liu[2,b]

[1]International Department, Yuhuatai High School, Nanjing, 210012, China
[2]Tsinglan School International Department, Dongguan, Guangdong, 523808, China
a. 2625232145@qq.com, b. 20260239@tsinglan.cn
**corresponding author*

Abstract: Collision detection is one of the most important features of video games that involve models that might run into each other. Video game players want to see the two models moving against each other to crash but not go through each other; this is why collision detection has come about. For ordinary models, like a box or a ball, collision detection is quite straightforward: check if the bounding volume of the object has gotten into the bounding volume of another object. For deformable models, like a piece of cloth, there would be a great many of calculations for the collision, since it has a lot of bounding volumes in one model. As a result, a range of methods have been introduced to game developers to optimize players' experience and computers' performance. All of them have something to do with essential features like bounding volumes, but they use these basic things in different ways.

Keywords: Collision detection, bounding volume, deformable models

1. Introduction

Collision detection is one of the most performance-consuming factors in video games. With models in a particular scene hitting each other, a number of computations would occur and decide which pairs of models are colliding and which pairs are not. For rigid models, like *a ball* or *a box*, collision detection could be quite straightforward because nothing beyond this ball and box is taken into consideration. However, the story would be more complicated for *deformable models* that consist of large numbers of things like triangles, which have their own bounding volumes, since some of them might not supposed to be colliding even though the model is running into something. As a result, collision detection for deformable models has a great demand in the performance, with the huge amount of calculations for the computer to make. In order to optimize the detection and make it easier for the computers, this literature review analyzes three different methods to deal with collision detection for deformable models that can optimize the process in their own ways.

2. Background

2.1. Collision Detection

Collision detection for models basically detects the overlapping of bounding volumes(BV) that belong to different models. If any of the bounding volumes overlap another one, that means they are colliding, and elementary tests would occur. It is the same for deformable models that can undergo both inter-object collision and self-collision since being deformable means that the model consists of a great number of triangles so that they have their bounding volumes and can overlap each other.

Collision detection itself is divided into two kinds. One of them is called *Discrete Collision Detection*(CD), which tests the contact between models at specific time instances. The other sort of collision detection is called *Continuous Collision Detection*(CCD), which checks if the collision has been kept in a period of time.

2.2. Deformable Models

Deformable models are models that can both collide with other models and collide into themselves. Take a T-shirt, for example. A T-shirt can contact with another T-shirt; in addition to that, its sleeves can contact with each other as well. These models are different from rigid models that cannot perform intra-object collision.

2.3. Bounding Volumes

The bounding volume of a model is basically the volume of the box it is in. This is one of the fundamental requirements of collision detection, as mentioned in previous sections. The algorithms mentioned in this literature review all have to deal with this element of deformable models, and one of the ways to make it less of a burden is to use *Bounding Volume Hierarchies*(BVH).

3. Fast Collision Detection for Deformable Models Using Reference-Triangles

This section is talking about [1]. This paper in the section contributes to providing an algorithm called Reference Triangles that assigns the features(i.e., the edges, the vertices, and the face of triangles) of the colliding, or incident, triangles to an augmented triangle, which increases the culling efficiency of each collision and reduces the elementary tests for detecting collisions. This method particularly works for the deformable models that are triangulated.

3.1. R-Triangles

In every triangle, there are always three edges, three angles, three vertices, and one face. These are known as the features of the triangle. If the triangle is included in a deformable model, the number of features for each triangle never changes. As a result, if multiple triangles share sharing same features, like a cone colliding with a triangle in Fig.1 [1], then there would be plenty of unnecessary elementary tests that have to be undergone since all of the triangles are involved in a collision, despite that there's only one vertex incident to another object. As a result, the basic method to detect the collision of deformable models is extremely inefficient, and this can be improved by using Representative-Triangles(R-Triangles).

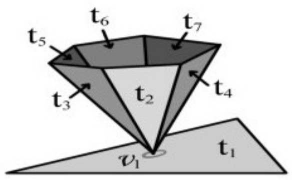

Figure 1: A cone that consists of multiple triangles colliding into a triangle with the same vertex [1]

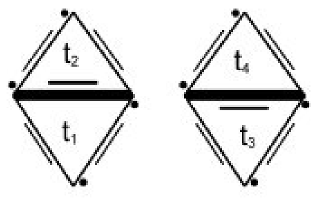

Figure 2: If these two models collide with the bold edge, there would only be one elementary test(edge test).

Representative-Triangles are the augmented triangles in deformable models. Instead of being ordinary triangles containing a number of vertices etc., R-Triangles carry the features of colliding triangles in the model. Unlike ordinary triangles in the model, the R-triangles would perform collision detection with the BVs of each feature instead of the BV of the whole triangle. All those characteristics of R-Triangles allow this method to detect collision to eliminate duplicate queries with improved culling efficiency. R-Triangles all have the following properties:

I Every feature must be represented by a triangle.
II Every feature can only be assigned to one triangle.
III If a feature is assigned to a triangle, then this triangle must be incident to the feature.

With all these requirements, the possibility of performing the same elementary test more than once can be eliminated by using R-Triangles. Fig.2 shows an example mentioned in [1], which has these two models colliding each other with the bold edges. Since the bold edges are assigned to triangles t1 and t4, respectively, the only elementary test that would occur would be the edge-edge test between these bold edges. The authors introduced two ways to assign the R-Triangles. One way is to maximize the number of unassigned triangles(about half of the total number), this is known as the *Maximal Schema*, and another way is to assign R-Triangles as much as possible, known as the *Uniform Schema*. Interestingly, these two ways could perform a similar result, and this might be completely a coincidence according to the authors. However, if the number of unassigned triangles exceeds half of the total amount of triangles in the model, there would be a drop in performance since it simply has created one more pair of incident R-triangles, forcing it to do more elementary tests. In the next two subsections, we're talking about the two goals achieved by the R-Triangle algorithm.

Algorithm 1: Using R-Triangles and feature bounding volumes to process candidate triangle pairs

```
Algorithm: processLeafPair(Node nodel, Node node2)
/* Determines if the two triangles represent compatible feature pairs */
if HasCompatible(nodel.tri, node2.tri) then
    if Overlaps(nodel, node2) then
        /* Elementary test construction */
        foreach Vert v represented by nodel.tri do
            if Overlaps(getBound(v), node2) then
                testVF(v, node2.tri)
            end
        end
        foreach vert v represented by node2.tri do
            if Overlaps(getBound(v), nodel) then
                testVF(v,nodel.tri)
            end
        end
        foreach Edge e1 represented by rode1 do
```

```
            foreach Edge $e2$ represented by node$2$ do
                if Overlaps(getBound($e1$) getBound(e2))then
                    testEE($e1.e2$)
                end
            end
        end
    end
end
```

3.2. Improving Culling Efficiency

The authors use the bounding volumes of features(or feature bounding volumes) to improve the overall culling efficiency. The use of bounding volumes of features in R-Triangles can minimize the amount of elementary tests since the features can only overlap features, but not a whole model. With that, the culling efficiency is improved by limiting the set of feature BV overlap tests to the R-Triangles that contain the features involved in collisions, ignoring all the triangles that are not incident to the other triangles.

3.3. Eliminating Duplicate Queries

The authors achieved this goal by using a simple idea: elementary tests are only dispatched for the compatible feature pairs represented by the triangle pairs. The details are given in Algorithm 1, and this algorithm describes how the method works.

Basically, the duplicate queries are eliminated because of this theorem from [1]: *For a pair of colliding features in contact, Algorithm 1 guarantees that exactly one elementary test on the features will be dispatched*, and this is proven in that literature.

3.4. Results

The authors used several benchmarks and implemented different algorithms for collision detection, and they found that R-Triangle was the fastest way to detect the collision of deformable models, comparing the amount of elementary tests and query time with several different algorithms. The result is given in Fig.3. The graphs compare the results of collision detection of the benchmarks with different methods. The methods are basic collision detection, ADJ, R-Triangle, and NODUPL [2,3,4].

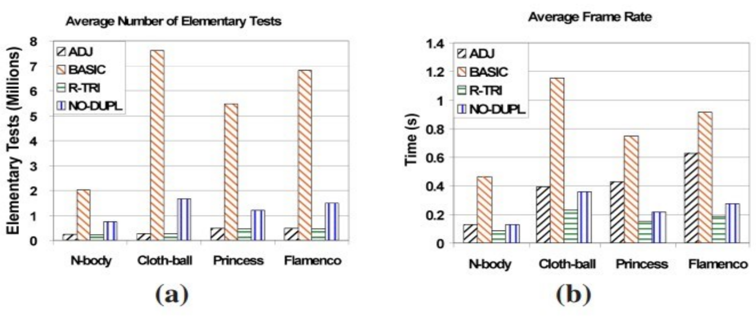

Figure 3: (a) the average number of elementary tests per frame for the benchmarks with all sorts of algorithms. (b) the average time per frame for the benchmarks with all sorts of algorithms. The algorithm for R-Triangle is represented by ”R-TRI”.

3.5. Advantages and Disadvantages

From the results, we can see that the collision detection query time has been clearly improved. The duplicate queries are eliminated without expensive memory accesses or unwieldy run-time data structures, leading to an improvement in performance. The culling efficiency is also improved by using the features instead of the whole triangles.

However, R-Triangles could still be improved. The first reason for that is that the percentage of false positives is huge (over 90 percent). In addition, any scene made up of a triangle soup gains no benefit from R-Triangles, since every triangle in the model represents all its own features. Finally, although the algorithm itself doesn't have a heavy demand on memory space, storing the BVs, which are critical for the R-Triangles, would cost a lot of memory space.

3.6. Their Future Work

The author is going to improve or implement their algorithm with the following three future works.

The first is Integration into Simulation, which means that they would like to fully implement the algorithm into a simulation system.

The next one is Element Bounding Volumes. There are several kinds of ways to assign the BV to models, what they're using is AABB [5], which is not the best BVH to implement but the one with the best simplicity and lowest cost. The algorithm might be even more efficient if they try better BVs. The last one is Dynamic Representative Re-assignment.

Although they found Maximal Schema and Uniform Schema, they're struggling to find one universal schema that is optimistic for all of the cases. Instead, they have found that Maximal Schema and Uniform Schema could both perform better than the other one in special occasions, meaning that there might be an optimal way to assign the R-Triangles for each case. Those cases are what the authors are trying to find in their future works as well.

4. Interactive Collision Detection Between Deformable Models Using Chromatic Decomposition

4.1. The Abstract

[6] made a new way to find when things touch each other, even when they're deformable objects. Researchers use colors to split a deformable object into different parts, and this helps us quickly see if any parts might touch without actually being next to each other. This makes it faster and better at knowing if things are really touching. They tried it on things like clothes and body parts, and it worked well and fast. It's a lot faster than older ways that were used before.

4.2. Results of the Methods Used

The researchers get two main results using the algorithm:

4.2.1. Chromatic Mesh Decomposition

The researchers put different colors into groups that will not be adjacent to each other.

4.2.2. Set-based Self-Collision Detection

The researchers with a way of organizing things in groups to figure out which things might bump into each other. Their method gets rid of things in each group that are far away using a special structure. They explain a quick method for deciding if things overlap, using simpler tests on one part of the computer and slightly more complex tests on another part. They make use of how things are positioned and how they move from one moment to the next to avoid checking if they overlap too many times.

4.3. Collision Detection

The researchers effectively enhance collision detection speed by categorizing objects into separate groups. This division optimizes the process, enabling efficient computation of potential collisions. By grouping objects, the algorithm focuses on relevant interactions, accelerating the overall detection process and improving performance in various applications.

5. Fast GPU-Based Collision Detection for Deformable Models

5.1. Abstract

[7] presents a streaming algorithm based on fast-GPU to execute collision queries between variable models. This paper presents a novel stream registration method to compact the streams and efficiently compute the potentially colliding pairs of primitives. They also use a deferred front-tracking method to lower the memory overhead. In practice, their algorithm can perform inter-object and intra-object computations on models composed of hundreds of thousands of triangles in tens of milliseconds.

5.2. Their contributions

They mainly present a kind of novel GPU-based detection algorithm that can extract a GPU to play as a kind of character that processes the stream, and this kind of stream processor will do well in processing the stream data and kernels at the same time. They also came up with a new fluid enrollment approach in order to assist in changing length figures configuration in an efficient way, which can be updated with lots of threads together. This provides them the flexibility to combine a lot of Orphan sets-based culling methods. [8], Representative triangles [1] and non-penetration filters [9], which is able to make the general implementation of GPU-based algorithm better; at the same time, in order to decrease the memory cost, they use a deferred-front-tracking plan, unlike the multi-core CPU-based method [10].

5.3. Background and related work

For continuous Collision Detection, many techniques are proposed to accelerate the performance of collision detection algorithms. These structures need to be improved for deformable models based on refitting approach or selective restructuring. In order to improve the performance of Continuous Collision Detection, including feature-based bounding volume. The other tech is able to obviously reduce the primitive measurements' quantity between the initial base on typical triangles. In place, this kind of approach could integrated with the limit capacity hierarchy; also, for GPU-based algorithms, they could also make the general execution of continuous collision detection algorithm better through more than one sequence of amplitude.

5.4. Streaming Continuous Collision Detection

In this part, they present CCD Continuous Collision Detection) algorithm, which is based on collision-streams. They suppose that the scene consists of several deformed objects. Their algorithm creates BVH for the whole scene, also their algorithms enforce the traversal from top to bottom in order to inspect for collisions between and *within objects.*

5.5. Three different commodity GPUs are used for testing their CCD algorithm

In practice, they select nodes for front-end latency based on the number of GPUs that can be stored and the exact size of the front end. If the front is huge. Select the grade connections of BVTT in order to decrease the usage of memory. In their benchmark, they observed a 21 percent acceleration in the Cloth benchmark, which could reduce memory consumption by 8.2X. For example, in complex scenarios where GPU memory is limited, delayed font tracking must be used to run CCD. To the Lion basic test that it was constructed by 1.6M triangles, more than 3GB of memory may be needed to keep the accurate BVTT front when their delayed front tracking only applies 340M.

5.6. How they use this algorithm

The general collision detection algorithm does not make hypotheses about the target or the motion; it is able to inspect all collisions between and within objects under object spatial accuracy and minimizes the number of basic tests between adjacent triangles. Their algorithm's performance as a feature of the quantity of cores or flowing processors is analyzed by them, too. In effect, collision flow can be calculated for 10 milliseconds between complicated deformation models comprised of hundreds of thousands of triangles using CCD (Continuous Collision Detection). Compared to previous algorithms based on CPU and GPU, they emphasize that we are faster and consume less memory. In addition, the combination of other elimination techniques is relatively simple. In place, this kind of approach could be incorporated with volume limitations; also, for CPU-based algorithms, the overall performance of CCD algorithms can be improved by more than one order of magnitude. In this article, they seek to develop appropriate GPU-based algorithms that can be used to decrease the amount of basic tests.

5.7. Comparison and Analysis

Compared to the majority of the earlier approaches for GPU-based collision checking, we can clearly see that the earlier methods executed the computations at image-space resolution, and their methods provided object-space accuracy and were much faster than those earlier methods. Their algorithm is a purely GPU-based approach. Some benefits of their methods include that by mapping the geometric and acceleration data structures and functional modules to streaming data and computation kernels, respectively, the CCD algorithm between deformable objects can exploit the parallelism on current GPUs. Also, they use front tracking to perform fine-grained tasks division. Generating a huge amount of little tasks and also allocating them between streaming processors in order to execute them in parallel.

5.8. Some Limitations

In my view, their methods probably have a little bit of limits. Firstly, even with deferred front tracking, the GPU still needs lots of memory space. Secondly, When their stream registration algorithm is able to assist changeable length data constructions and can be accessed frequently global memory, it slows down computation speed. Finally, for some kinds of small models, other GPU-based algorithms that only use shared memory in order to collect tasks could be faster than the collision streams.

5.9. Some of my opinions of their works

Their works find some new ways of algorithms like CCD(Continuous Collision Detection). They can correctly determine all the collisions between each object. In addition, compared to other kinds of culling techniques. In practice, these methods can improve the general performance of GPU-based a lot.

5.10. The Future Work

This article proposes a CCD stream algorithm for deformable objects to perform hierarchical updates and periodic and segmented calculations on GPU. In addition, their methods are flexible. In practice, this kind of algorithm will improve CCD performance on GPU-based architectures. By improving mapping to GPU architectures, they can further improve the performance of the algorithm. Eventually, they decided to expand a method to execute other inquiries, including range and interval inquiries.

6. Conclusion

In conclusion, as a researcher also author, I have been talking about three different approaches to deal with collision detection for deformable models. The first one, R-Triangle, relies on the use of triangles in the deformable model. By assigning the features of triangles, this approach optimizes the detection by eliminating repetitive processes, with better culling efficiency and less duplicate query time. The second one basically assigns a label to each part of the model and detects the collision of the bounding volumes that have the same label. They give different triangles different colors in order to make the detection easier and more efficient; also, this method can minimize the memory cost and maximize the efficiency of collision detection. The third one is the algorithms based on using GPUs, and by applying this method, as the researcher, I tried to minimize the internal storage cost. Also, streaming continuous collision detection has been presented that can allocate many small tasks between streaming processors in order to execute them in parallel. But it also has some limitations: we still need a lot of memory space and it also slows down computation speed. In the future, I'd like to do more research like this and learn more about this area. The main purpose of running this research is to develop a method that can avoid useless consumption in order to reduce memory costs and ensure quality. But there are still some disadvantages to be solved in the future. For example, a lot of errors still appear when these methods are used. Although the algorithm itself doesn't have a heavy demand on memory space, storing the BVs, which are critical for the R-Triangles, would cost a lot of memory space. As a researcher, I will improve the new methods by solving the problems mentioned above.

References

[1] Sean Curtis, Rasmus Tamstorf, and Dinesh Manocha. Fast collision detection for deformable models using representative-triangles. ACM Symposium on Interactive 3D Graphics and Games, 2008.

[2] Min Tang, Sean Curtis, Sung-Eui Yoon, and Dinesh Manocha. Interactive continuous collision detection between deformable models using connectivity-based culling. In Proceedings of the 2008 ACM symposium on Solid and physical modeling, pages 25–36, 2008.

[3] Marco Hutter and Arnulph Fuhrmann. Optimized continuous collision detection for deformable triangle meshes. 2007.

[4] Wingo Sai-Keung Wong and George Baciu. 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, pages 181–188, 2006.

[5] Gino van den Bergen. Efficient collision detection of complex deformable models using AABB trees. Journal of Graphics Tools, 2(4):1–13, 1997.

[6] Naga K Govindaraju, David Knott, Nitin Jain, Ilknur Kabul, Rasmus Tamstorf, Russell Gayle, Ming C Lin, and Dinesh Manocha. Interactive collision detection between deformable models using chromatic decomposition. ACM Transactions on Graphics (TOG), 24(3):991–999, 2005.

[7] Min Tang, Dinesh Manocha, Jiang Lin, and Ruofeng Tong. Collisionstreams: Fast GPU-based collision detection for deformable models. In Symposium on interactive 3D graphics and games, pages 63–70, 2011.

[8] Min Tang, Sean Curtis, Sung-Eui Yoon, and Dinesh Manocha. Interactive continuous collision detection between deformable models using connectivity-based culling. In Proceedings of the 2008 ACM symposium on Solid and physical modeling, pages 25–36, 2009.

[9] Min Tang, Dinesh Manocha, and Ruofeng Tong. Fast continuous collision detection using deforming non-penetration filters. In Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games, pages 7–13, 2010.

[10] 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.