State University of New York at Stony Brook
Computer Sciense Department

Collision Detection of Deformable Models

Research Proficiency Exam Report

by

Eli Packer

The research work is being carried out
under the supervision of Prof. J.S.B Mitchel

September 2005

abstract

Collision detection is a fundamental operation used in a variety of areas such as computer graphics, computer animation, robotics, computational geometry, visualization and more. Often the problem of detecting collisions turns out to be the bottleneck of the application. The reason is that in involves testing pairs of primitives, thus quadratic in nature. Hence, when large datasets are used in real-time applications, it is most important to have collision tests done efficiently.

The collision detection problem has been studied extensively in the literature and many algorithms have been developed in this area. However, the research of the variant of collision detection in which objects may deform is still new and was not explored widely. It is agreeable that there is still a lot of room for improvements. The known collision detection algorithms might perform very poorly when objects deform since further actions have to be taken as a result of a more complex model. In such cases, those algorithms might be found unsuitable.

We survey the work that have been done in collision detection in general and in collision detection of deformable models in particular.

We plan to research collision detection of deformable models in order to develop efficient algorithms that will perform well in practice. We describe our current status, the application we developed and our future plans.

Contents

Chapter 1

Introduction

Collision detection is an important operation in dynamic and deformable models. Most usually when collision occurs, a specific reaction should follow. For example changing the velocity and direction of movement, merging objects, modifying the topology, breaking objects into separate smaller pieces and more. Frequently, collision changes the flow of the software at hand. For instance, when a robot is detected to collide with the object it has to lift, its task will change from detecting the object into carrying it [7].

There are two main reasons for the need for efficient collision detection algorithms: real-time environments and large models. Most often collision detection is performed in real-time environments that require an interaction with users. Unless the detection is fast enough, it loses its relevance and results in awkward performance. Frequently, the models that are used consist of very large geometric models that consume a lot of time to process. Brute-force algorithms to detect collision take $\theta(n^2)$ detection tests for n primitives. Such performance is undoubtedly far from sufficient, thus much more efficient techniques must often be used.

Approaches to collision detection can be divided into two main classes of algorithms. The first one is feature-based which uses temporal and spatial coherence to maintain the close features. One example is to tile a grid of voxels and assign objects to the voxels that contain them. The second and most widely class uses hierarchical bounding volumes (or BV, for short). The idea is that levels of bounding volumes cover the objects. At the top level, the bounding volume bounds the entire model, and at the bottom one, it bounds primitives. In between, it bounds subsets of primitive (see Section 2.1 for more details). The use of both classes is to efficiently prune

subsets of primitives that are well separated such that two primitives in different subsets cannot undergo collision. There were few hybrid methods that combined the two classes.

Deformable models is an area that is widely researched in previous years. The idea is to define rules that generate deformations. Frequently the target is physics-based simulation in which real world models are desired. In this area real-rime performance are often required, thus the algorithms should be efficient.

When objects can deform, the situation is more complicated for detecting collisions. Unlike with rigid bodies, a deformable object may undergo self collision which has to be tested as well. More importantly, the efficiency of the data structures we described above may degrade significantly. The reason is that the static nature of rigid bodies cannot be exploited anymore. Consider, for example, a BV data structure. It is usually constructed in a preprocessing step and remain static during the simulation. This is most appropriate when the objects are rigid. However, when the objects deform, the BV hierarchy quality might degrade in the form of very big bounding volumes which result in many more collision tests to perform. In such cases, it might be needed to reconstruct or update the BV in order to maintain efficient performance. However, those tasks might take significant time. As a result, data structures that perform well with rigid bodies might not be suitable to deformable models. There is still no convincing technique that provides good solution to the above problem in the form of performance that can be compared to rigid body simulation. Some of the research dealing with deformable models made different assumptions on the dynamicity of the model, thus obtaining nice performance with the techniques discussed above. However, those assumption pose a limitation and suitable to certain models and situations.

We developed an application to simulate particles moving freely in 3D. Particle simulation can be applied in several applications such as molecular modeling and can be extended to simulate polygonal meshes where the particles represent the vertices. We detect the collisions among the particles using two algorithms: the first one is the *Deformable Spanners* [13] and the second uses regular grid of voxels that hold the particles. Those two techniques are opposite in nature. The first one uses sophisticated data structure that behaves well in theory, while the second one is very simple with no good worst case bound. We were interested to take those two as instances of techniques and to understand under which circumstances one outperforms the other. We plan to test more techniques and try to optimize the process by, for instance, mixing different techniques in different situations to achieve hybrid work. We tested our software and present experimental results.

The rest of this report is constructed as follows: We start by describing and surveying the area of collision detection in Chapter 2. In Chapter 3 we present the collision detection of deformable models problem and survey related work . We describe our work on collision detection and plans for the future in Chapter 4. We conclude in Chapter 5.

Chapter 2

Collision Detection

As we mentioned in Chapter 1, approaches to collision detection can be divided into two main classes of data structures. Both come to speed up collision detection performance by pruning collision tests between primitives that are too far to collide. Most collision detection algorithms use a data structure that belongs to one of these classes. Interestingly, within those classes, there is still a huge room to use different approaches in order to speed up performance in certain situations and models. The ideas behind both are simple and usually easy to implement. For more background, we refer the reader to surveys done in this area [30], [24]. Next we present both in depth.

2.1 Hierarchical Bounding Volumes

The basic idea of the BV approach is to maintain a hierarchy of bounding volumes that cover the entire model with geometric objects, most usually using a trees. A typical way of such is the following. On the bottom of the hierarchy, each atomic primitive (or, less frequently, a few ones) is bounded separately. Then we gradually bound groups of bounding volumes or primitive subsets as we climb up the tree until we converge to one BV that bounds the entire model at the root. Figure 2.1 illustrates the idea. [1]

Collision detection queries are performed top-down, comparing the different BV's

[1] This example illustrates eight separate primitives while usually sets of rigid bodies composed of many primitives are involved. We chose this example for the sake of simplicity.

for overlapping and prune ones that do not overlap. More precisely, it is performed as the following. We start from the root, gradually descend along the tree, testing nodes that possibly overlap. For each node, we check if its children overlap. If they do, we go one level down and check all its grandchildren for overlap (6 such tests) and so on further down. In any case, when we process a node we proceed independently with it for detecting collision between primitives that are placed below. When we reach the leaves, we explicitly perform collision detection of the corresponding primitives. See Figure 2.2(b) for an illustration for the work on the input at Figure 2.2(a) in which ellipsoid 3 translates. A modification of this algorithm which is usually adapted, is whenever two children overlap, we check the one with the smaller bounding volume with the children of the other. See Figure 2.2(c) for an illustration of the same input in Figure 2.2(a). It is immediate from the process above that it is very important that close primitives be placed close to each other in the tree. By doing so, we maintain smaller BV's and less overlapping tests to perform in general.

Sometimes it is necessary to update the BV hierarchy when objects transform [26]. This is performed by updating the BV along the hierarchy from the leaves all the way up to the root. See Figure 2.3 for an illustration where a particle translates, resulting in updating the ancestors' BV. In cases a rigid body transforms, all the BV's of the tree have to be recomputed in such a bottom-top manner.

2.1.1 Designing Bounding Volume Hierarchies

It is important to adapt the appropriate BV type to the model at hand in order to achieve efficient performance. In this section we describe many design attributes that one has to consider when coming to design collision detection algorithm with hierarchical bounding volumes. All these attributes were either applied or considered in the literature. In Section 2.1.2 we present many examples relevant for the discussion below.

Bounding volume type. This property determines the geometric type used to bind the objects of the model. Frequently, the following formula is used for measuring the efficiency of each type:

$$T = N_v * C_v + N_p * C_p + N_u * C_u \tag{2.1}$$

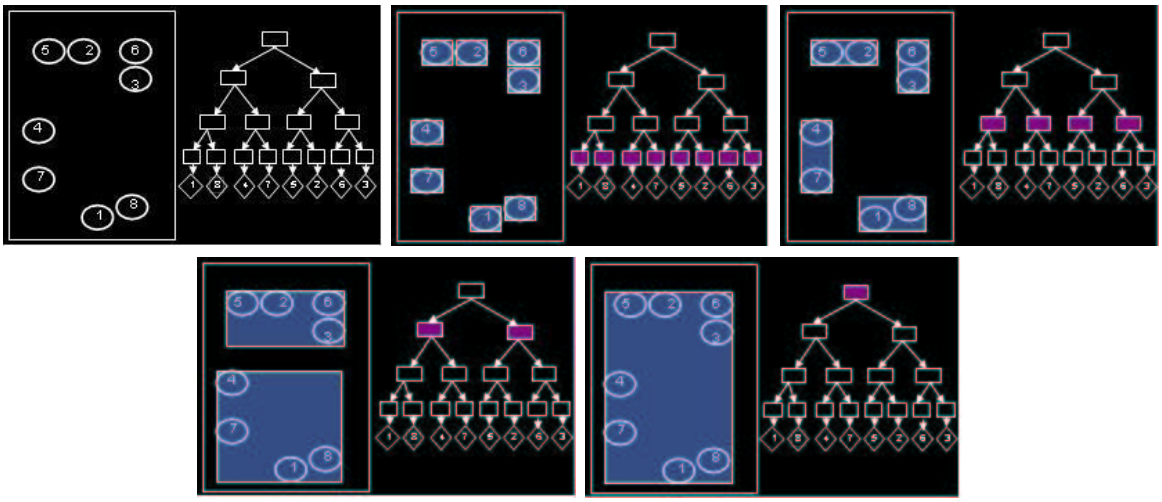


Figure 2.1: Different levels of hierarchical BV: the model is on the left of each sub figure and its respective data structure appears on the right. Different sub figures highlight different levels of the hierarchy.

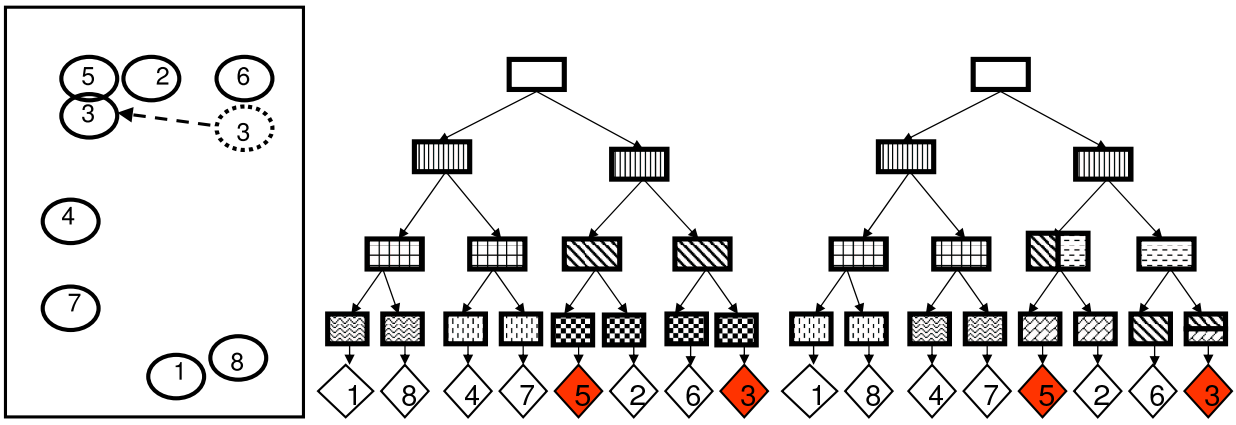


Figure 2.2: Testing for Collisions. Nodes with the same texture are tested for overlap.

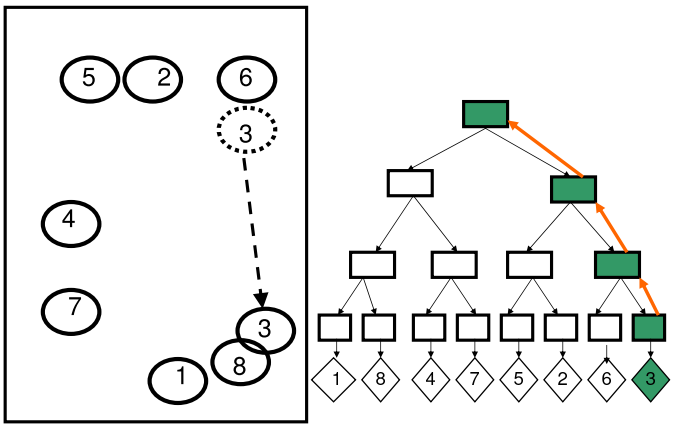


Figure 2.3: Updating the BV when an object transforms.

N_v is the number of bounding volume pairs tested for overlap, N_p is the number of primitive pairs tested for collision, N_u is the number of nodes in the hierarchy that require updates (in case updates are necessary) and C_v, C_p and C_u are the costs for these operations, respectively. It is important to emphasize that there is a trade off between the numbers of tests represented by N_v and N_p and the cost represented by C_v and C_p. The more simple the bounding volume is, the bigger are N_v and N_p, but the smaller are C_v and C_p. Consider for example an *axis aligned bounding box* (AABB for short). The test for overlapping is very cheep, but we might have a lot of redundant computations to perform as AABB might approximate the elements at hand very poorly. This is the case, for example, with long-thin oriented polygons. Their AABB volumes are much bigger, resulting in possibly many tests for overlap that might be prevented had we allowed the bounding volume to orient (obtaining *oriented bounding box* or OBB for short). The BV would have been much smaller and approximate the primitives much better in such case. The result would be possibly much less overlapping tests. However, the computations involving OBB, both finding the bounding volumes and the overlapping test between two OBB's, are much more complicated and time consuming than those of AABB.

Gottschalk et al. [14] suggest that simple BVs like spheres and AABB would perform very well when two models are far apart but when they are in close proximity

with multiple number of closest features, the number of tests increases, thus more complicated BV's might work better.

Among the BV used in the literature we can mention *spheres*, *AABB*, *OBB*, *discrete orientation polytopes* (k-dops), *Spherical Shells*, *convex hulls* and *C-trees*.

Object permutation within the data structure. We mentioned that locality in bounding elements is very important in order to reduce the number of overlapping tests. The question that is raised here, is how to achieve good locality. We could do it by finding the best matches of a complete graph for each level of the tree, but the costs would be exponential. Another possibility to obtain good approximation is to use a greedy algorithm in which for each group of primitives we search for the closest ones and bound them together. Sometimes, we can exploit model properties, such as an embedded subdivision within the model.

Data structure updates. Sometimes it is necessary to update the BV [26]. In this case, when an object transforms, we update its respective BV's, starting from the leaf, all the way to the root. This process could be costly and must be integrated into the BV decision.

Approximation. Since a collision detection query might be costly in certain situations, it could be possible to stop at certain level of the hierarchy and to decide whether a collision exists based on the current overlapping. The question that rises here is how good the BV approximates the primitive.

Degree of the data structure. This attribute determines the possible degrees of each node of the data structure. Frequently a binary tree is used, but other trees (for instance with degrees 4 and 8) have been used as well. Bigger degrees should be applied when there are more chances for collisions, thus reducing the height of the tree.

Data structure construction: top-down or down-top. Determines how to construct the BV tree. In the top-down approach, the entire model is gradually subdivided according to some criterion. In the down-top approach, primitives are paired at first, and then BV's are paired, converging into the root.

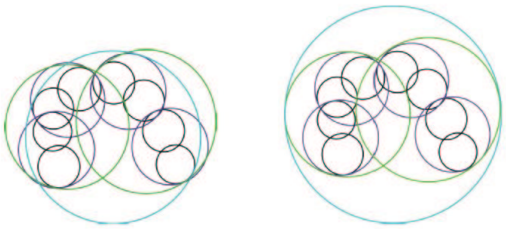


Figure 2.4: Wrapped (left) and layered (right) sphere hierarchies.

Detect or Report. If reporting the colliding primitives is not important, it would be desirable to apply an algorithm that only detects collision, which might be less costly. Most of the work done so far concentrated on reporting.

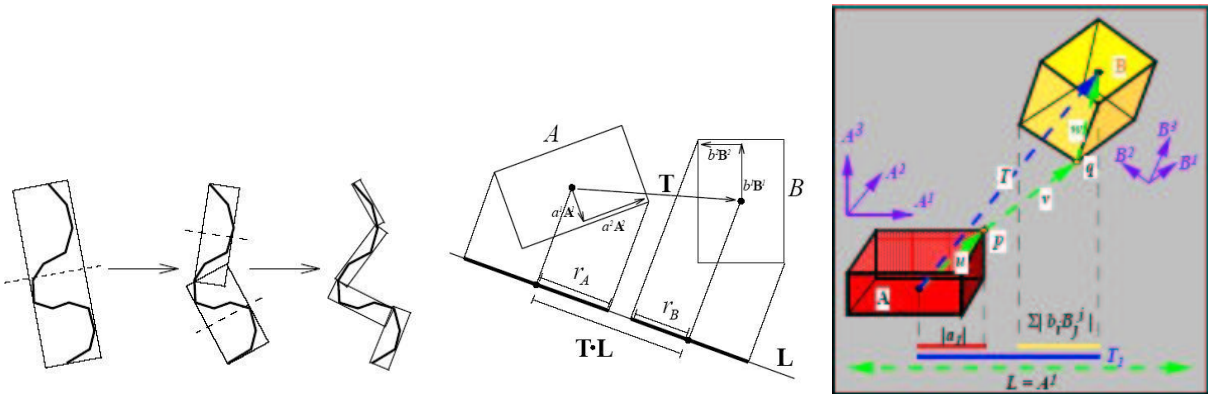
Data structure type. Most usually, trees were applied. However, there were cases in which DAG was preferred.

Wrapped Hierarchy vs. Layered Hierarchy. There are two possible ways to build a hierarchy of BV's [1]. In a layered hierarchy, each BV is a result of bounding its immediate children. In a wrapped hierarchy, the BV correspond to the bounding volume of all the primitives (located at the leaves) in the respective sub tree. While the first one is easier to build and maintain, the second one better approximates the model, allowing tighter BV fitting. It was shown that they produce the same hierarchies if the BV is K-dops (AABB is a special case of a K-dop). Figure 2.4 (from [1]) depicts the two kinds of hierarchies when spheres were used as the BV.

Hybrid data structure. It might be worthwhile to use different BV at different levels of the hierarchy. For example, using a tighter bounding volume at the top in order to decrease the number of branches used greatly, but using less tight and simple bounding volume at lower levels.

2.1.2 Related Work

In this section we present related research and highlight the contributions and interesting ideas of each. We emphasis that this list is not complete and new results are published very often.



(a) (b) (c)

Figure 2.5: The work with oriented bounding boxes.

OBBTree: A Hierarchical Structure for Rapid Interference Detection (Gottschalk et al.) [14]

This work used OBB (Oriented Bounding Box - a rectangular bounding box at an arbitrary orientation in 3-space). They were able to test two OBBs for overlap in about 100 operations on average which is about one order of magnitude faster than previous algorithms. They showed that although the computations involved are more complicated than the ones with AABB, the performance with OBB is asymptotically faster in many situations. In order to create the BV tree, they used a top-down approach in which they partition the polygons according to which side of the plane their center point lies on. Figure 2.5(a) depicts the hierarchy of OBB. Figures 2.5(b) and (c) depict the method used for testing overlap of two OBB's. It is done with projection onto different axes — fifteen projections are required in the general case.

Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPS (Klosowski et al.) [26]

This work applied *discrete orientation polytope (k-dop)* as the bounding volume. k-dop is a convex polytope whose faces are determined by the union of k half-spaces

Figure 2.6: A comparison of three BV for the same object. From left to right: AABB, OBB, k-dop.

formed from the set of planes whose normals are chosen in k different fixed directions and whose position along the normal is defined by the extreme point of the object in the normal's direction. A comparison with AABB and OBB is illustrated in Figure 2.6. This kind of bounding volume allow flexibility using different k's for balancing between the time to test overlap and the tightness of the approximation. When k is large, the respective k-dop approximate the convex hull. Figure 2.7 depicts different levels in the hierarchy with different sets of k-dops. This work was aimed for virtual reality in which some of the parts are stationary and the others are dynamic. Thus the primitives were partitioned into two respective sets. They show that it is important for the k-dop orientation normals to come as pairs of collinear and opposite oriented vectors and that two k-dops to have the same set of respective half-spaces in order to make the overlap test efficient. Under those assumptions, overlapping test is simply performed by $k/2$ interval overlap tests, which is as trivial as checking two AABB's for overlap and simpler than the checks between two OBB's or convex hulls. The idea is that two k-dops do not overlap if at least one results of projecting onto one orientation of the k-dop do not overlap. Since they allow objects to rotate, during each rotation the k-dop hierarchy must be updated to maintain the same k-dop orientations. Since this update can be costly, they developed two approximation approaches that work well in practice. Their hierarchical data structure was a binary tree which was built in top-down approach. The split was according to a plane orthogonal to one of the three coordinate axes, such that it is aligned to a representative point on the splitting triangle. The axis was chosen based on objective functions on the volumes that the children occupy (such as min-max). They compared among those functions.

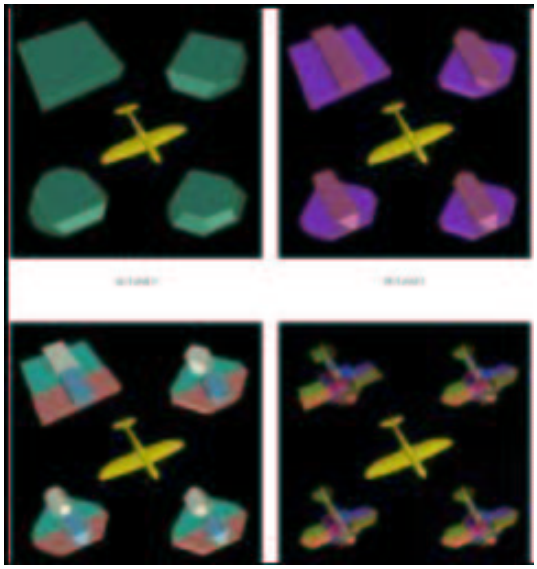


Figure 2.7: Different levels of the hierarchy with different k's and orientations. Each square corresponds to different level, while inside each, four different k's with different orientations are used.

Approximation Hierarchies and S-bounds (Stephan Cameron) [7]

S-bounds were developed to speed up the interference detection among robots. They provide a way of localizing the search for collisions, thus speed up the query. They are generated by forming aligned boxes of the primitives in the model. Consider Figure 2.8. It illustrates a robot and a stationary set. The task is to detect a collision between the two (Figure 2.8(a)). Then the bounding box of each primitives is computed (Figure 2.8(b)). Figure 2.9(a) illustrates the work on the bounding boxes. The primitives of each object are united, providing a bounding box of both. Then an intersection of the two is computed (denoted by I). Those are the two rectangles in Figure 2.8(c). The next step is to find the primitives that intersect I (Figure 2.8(d)). This process is reiterated until it converges into small elements that are explicitly tested for collision (Figure 2.8(f)). This scheme was also applied with convex hulls (Figure 2.9(b)). A 3D illustration is provided in Figure 2.9(d).

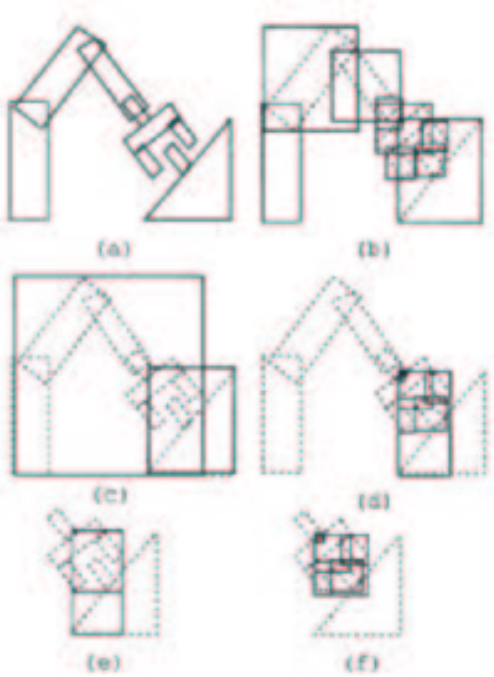
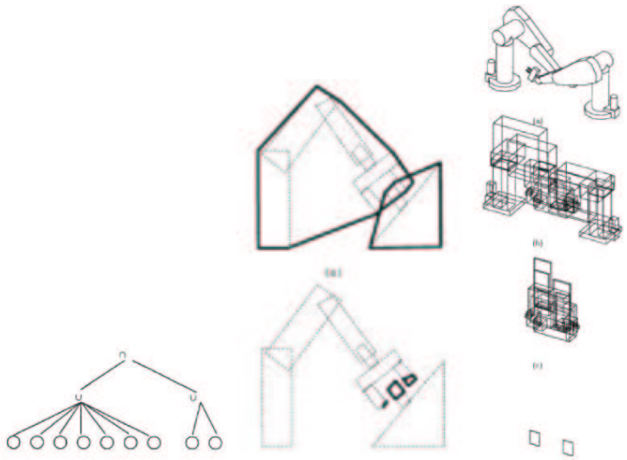


Figure 2.8: The steps in detecting a collision between a robot and a stationary set.



(a) (b) (c)

Figure 2.9: S-bounds

Box-Trees for Collision Checking in Industrial Installations (Haverkort et al.) [19]

A box-tree is a hierarchy which uses AABB. The starting point of this work was that a query time for box-tree is $O(n^{\frac{2}{3}} + k)$, where n is the total number of boxes and k is the number of boxes intersecting in the query range. This bound was referred as disappointing. This work describes an algorithm that generalizes a 2D kd-interval tree to 3D, while partitioning the input boxes into three subsets, according to the orientation of their longest edge, and construct separate box-trees for these subsets. The worst case query time achieved was poly-logarithmic, when working on CAD models of industrial installations. There are other settings in which such performance cannot be achieved.

Spherical Shell: A Higher Order Bounding Volume for Fast Proximity Queries (Krishnan et al.) [28]

This work demonstrates that BV could be pretty exotic. The BV chosen here is an intersection between the space between two concentric spheres and a cone whose apex coincides with the common center of the two spheres. See Figure 2.10 for an illustration. While the overlap test is only slower by a small factor compared to OBB, they provide local cubic convergence (the BV approximates the surface accurately up to the second order if the surface is expressed as a Taylor's series) to the underlying geometry. This results in improved overall performance for proximity queries between two objects in close proximity configurations or between two objects with high-curvature surfaces.

Approximating polyhedra with spheres for time-critical collision detection (Hubbard) [22]

This work used a hierarchy of bounding spheres. The idea is that spheres centers were positioned at the medial axis of the object in question which was approximated by voronoi diagram (Figure 2.11(a)). Then spheres were placed at the voronoi vertices and then the bounding hierarchy was generated accordingly by merging close spheres. An octree of spheres was also applied (Figure 2.11(b)). They listed a number of requirements the preprocess work of building the BV tree should meet: it must be

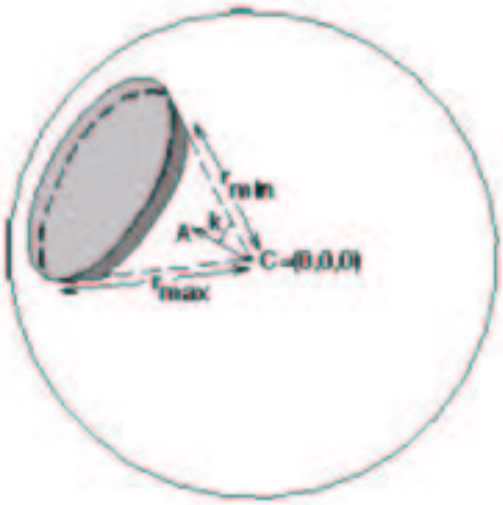
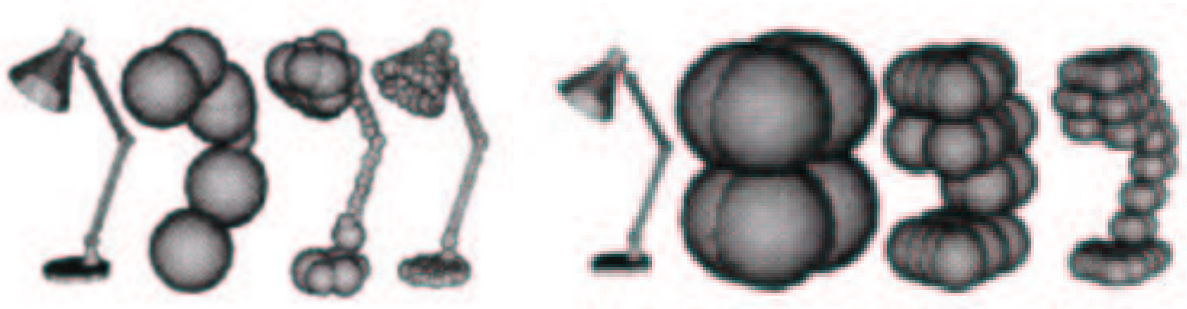


Figure 2.10: A spherical shell

automatic, building useful hierarchies without user intervention, must ensure that each hierarchy is an effective search structure (efficient pruning) and generates the best tight fit in each level of the hierarchy as possible.

2.2 Feature-based and Spatial Subdivision

The idea is to subdivide the space into separate regions such that each holds a list of primitives whose reference points intersect it. Then, in order to test for collisions, it is sufficient to test primitives in the same region or in two regions that share a common boundary. Clearly the regions should be designed such that no primitive intersects two non-neighbor regions, making the above heuristic useless. Among the subdivisions used in the literature, we can list uniform grid of unit voxels, oct-trees, bsp-trees, kd-trees and R/R*-trees. The first one is very simple spatial subdivision but it corresponds to a subdivision that is independent of the model shapes. Hence, it could be very useful when the shape is distributed uniformly, but may behave badly if many primitives are placed in few voxels, resulting in a quadratic behavior. Data structures like oct-trees and bsp-trees are examples of feature-based subdivisions that come to correspond to the shapes of the model. Thus, they better distribute the primitives along their structure than the simple grid, resulting in potentially less



(a) Spheres are placed according to the medial axis (b) Octrees of spheres

Figure 2.11: Spheres are used to bound the objects.

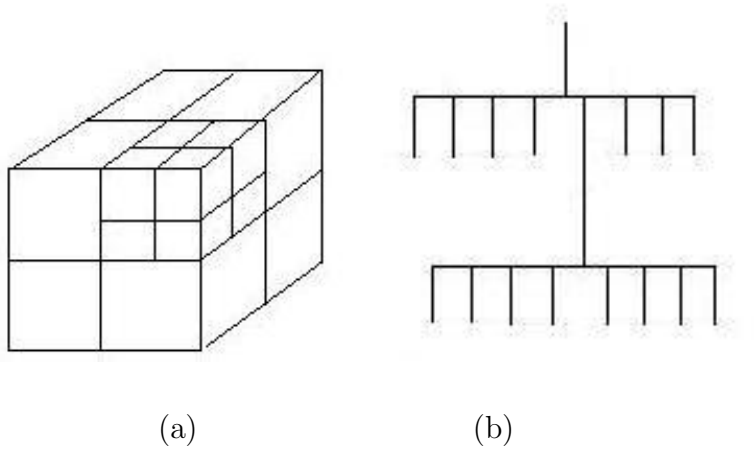
collision tests to perform for each query. However, they are more complicated and require more time for construction and updates.

2.2.1 Subdivision Data Structures

In this section we present popular subdivisions that have been used for collision detection tests.

Uniform Grid. The space is subdivided uniformly into a grid of cubes, or *voxels*. Only voxels that actually contain a reference point of primitives are explicitly allocated. They hold lists of the corresponding primitives. Then the collision detection query is performed by testing all pairs that belong to the same voxel or to two neighbor voxels (that share common boundary). The idea is that two primitives whose reference point belong to non-neighbor voxels are too far to collide. This data structure requires updates as objects translate from one voxel to another by updating the voxels' lists.

Oct-Trees The idea is to subdivide the space hierarchically to eight octant around a point by three axis aligned planes tangent to that point (this point should be determined as the center of the region being subdivided in some sense). Each box created in this process is subdivided only if the number of primitives it contains is

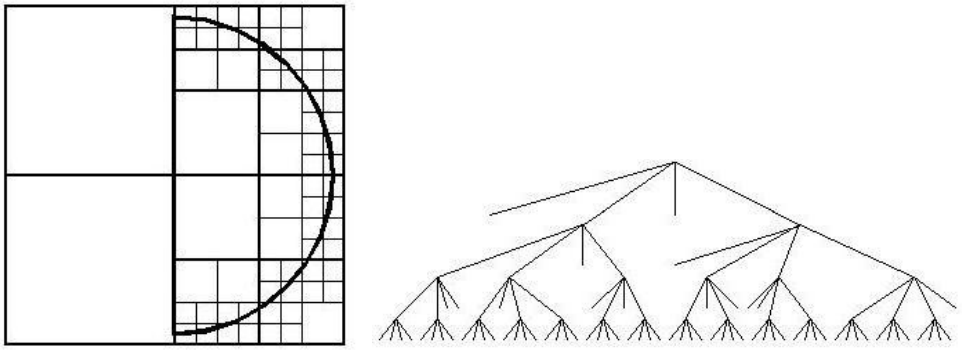


(a) (b)

Figure 2.12: An octree: (a) the space subdivision (b) the respective octtree

above some predefined threshold. The subdivision terminates also if the current box is smaller than another threshold. Only leaves of the corresponding tree hold lists of primitives. The result is a subdivision that correspond to the primitive distribution (see Figure 2.12 an illustration; A 2D illustration of an octree for a half circle is illustrated in Figure 2.13; the 2D version of the oct-tree is called quadtree), for a half circle is illustrated in Figure 2.13). As with the uniform grid subdivision, collision tests are performed within leaf nodes or among two leaf nodes with tangent boxes. The advantage of this data structure is that the maximum number of primitives in any list is usually constant, in which case potential bad cases as the uniform grid may have are prevented. The cost we pay for that attribute is time to construct the oct-tree and more difficult updates as primitives transform.

Two other data structures are *kd-trees* and *bsp-trees*. They are similar to oct-trees, but the subdivision operation is performed with respect to the distribution of the primitives. While the cells in the kd-tree are rectangular, the bsp-trees are polygonal, thus generalizing the oct-tree and kd-tree data structures. For more information on those data structure, refer to [8].



(a) (b)

Figure 2.13: A quadtree of a half circle: (a) the plane subdivision (b) the respective quadtree

2.2.2 Related Work

Below are two examples of work that use spatial subdivision. Another work that use such BV's and relate to deformable models are presented in the following chapter.

The R*-tree: An Efficient and Robust Access Method for Points and Rectangles (Backmann et al) [4]

In this paper, an R*-tree is represented. It is a variant of an R-tree which is a B-tree like structure that is defined as the following. A non-leaf node contains entries of children associated with their bounding volumes. A leaf node contains entries of the form object with its bounding volume. Parameters M and m are the maximum and minimum number of entries in a node respectively (for both non-leaves and leaves), such that $2 \leq m \leq M/2$. The data structure is dynamic and allows insertion and deletion. Hence, in order to maintain the number of entries of each node as required, operations such as *split* has to be performed occasionally. The criterion in an R-tree is the size of the BV (AABB in this case). The R*-tree is defined as an R-tree, but the criteria here contain margin and overlap of bounding boxes as well. By that it was possible to improve the performance as suggested in Figure 2.14. At its top, a set of 10 elements placed under one node have to be spitted as the upper bound M was acceded. Below, four possibilities are presented. The three left splits are achieved

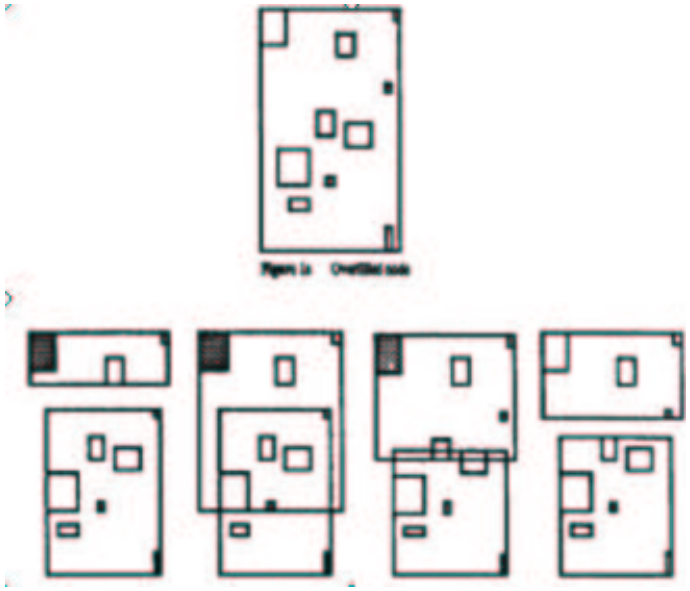


Figure 2.14: Several split possibilities with R-tree and R*-tree

with R-tree for different ratios of m and M. The right split, achieved by R*-tree is better than the other in the sense that overlap does not occur and the two new nodes are balanced. The R*-tree can be efficient with database systems organizing both multidimensional points and spatial data.

Broad-Phase Collision Detection Using Semi-Adjusting BSP-trees (Luque, Comba and Freitas) [32]

They proposed a new structure called Semi-Adjusting BSP-tree for representing scenes composed of thousands of moving objects. An scheduling algorithm evaluates locations where the BSP-tree becomes unbalanced, uses several strategies to alter cutting planes, and defer updates based on their re-structuring cost. The motivation of the work is to extend static nature constrain of spatial data structures to handle moving objects. The idea is to maintain data structure that updates itself as objects move without the need of complete reconstruction. They partitioned the space using BSP-trees (Figure 2.15(a)), and maintain his efficiency by scheduling and performing various operations (such as the ones in Figures 2.15(b)-(d)). Those updates are performed when the quality of the structure degrades.

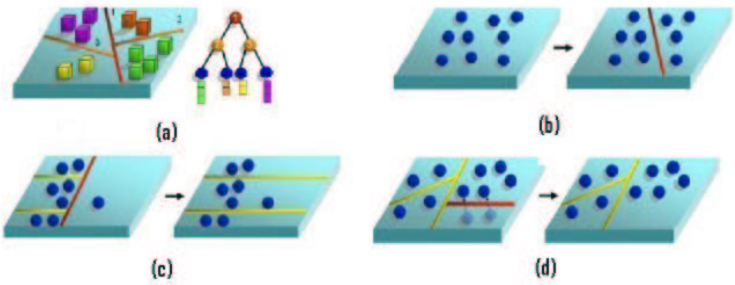


Figure 2.15: BSP-trees: (a) The partition and corresponding tree (b)-(d) The result of operators split, swap and merge respectively.

2.3 Discussion: Asymptotic Time Complexity

The approaches we described in this chapter are the most popular in this area and usually improve the performance of detecting collision greatly in comparison to a brute-force method that tests all primitive pairs. However, in terms of worst case asymptotic behavior, they do not make any improvement. Clearly, the running time of a brute-force algorithm is $\Theta(n^2)$ for n primitives. We next show that this worst case hold for the approaches we presented above even if the number of real collisions is much smaller.

- **Bounding Volumes.** Consider a set of n skinny parallel sticks and sphere BV. They are arranged such that a bounding sphere of each one intersects all other sticks. The result is testing all pairs of leaf nodes and quadratic time (note that the size of the tree is linear in n).
- **Spatial Subdivision.** Consider a grid of voxels partitioning the space. It best performs when the objects are well distributed. However, when this is not the case and the object reference points are located in few voxels, a quadratic performance is achieved. (if the number of voxels in which objects are equally distributed is C, then the number of collision tests is in the order of $\Theta(C(n/C)^2) = \Theta(n^2)$.)

Chapter 3

Collision Detection of Deformable Models

Deformable collision detection is an essential component in interactive physically-based simulation and animation. Among applications that apply deformable models and need to detect collisions we can mention surgery simulations, entertainment/games, robotics, computational biology and cloth animation.

Although a lot of research have been devoted for the detection of collisions, most of the work were aimed for rigid bodies and not proved to perform well when bodies undergo deformation. The prime reason for that is the following. When object remains rigid, its respective data structures (which does not depend on the body configuration) remains effective — when it transforms, it suffices to transform a reference associated point. However, if the body deforms, those data structure might turn out to be inappropriate. It results in either inefficient collision detection process or a need to update or reconstruct the data structure, which consumes time. Consider a BV hierarchy. Whether it is needed to recompute the BV when the body transforms or not, the hierarchy should not change as it reflects the shape of the body, regardless to its configuration. However, if the body deforms, it is mandatory to update respective BV's. Worse, the desirable locality property, where close primitives are placed close in the data structure, may not hold anymore. There are even models in which the deformation causes topology modification. The result is many and large BV's that overlap which require many more collision tests to perform. In order to improve the performance, it might be required to reconstruct the data structure occasionally. This work will consume time and still the performance degrades in between two such

reconstructions.

Several research projects were devoted in recent years to cope with this challenge. It was shown that simple BV's (mostly AABB and spheres) are much more useful for deformable models than more complicated ones. The reason is that the weight of BV's updates is large here, thus simple BV's for which this operation is cheaper are much more desirable. Another popular approach is to assume small or local deformation in which the BV hierarchy does not degrade much. Hence, data structures that were suitable for rigid bodies were found appropriate here too.

Recently, Graphics processing units (GPUs) have been increasingly used for collision and proximity computations. GPUs are good in optimizing 3D vectors and matrix operations, and complex computations on the frame-buffer pixel or image data. Different algorithms have exploited these capabilities to compute interference or overlapping regions or to cull away portions of the models that are not in close proximity. On the other hand, GPU-based collision detection algorithms often suffer from limited precision. The low precision and sampling errors can result in missed collisions between two objects. Examples of work using GPUs are [17], [3].

Three techniques that recently became popular in this area are *Stochastic methods*, *Distance Fields* and *Image-Space*. We survey each below.

However, there is still a consensus that for general models there is still a large room for improvements and there is still no method that solves this problem very efficiently without inducing restricting assumptions on the model at hand.

A survey on this topic can be found at [38]. In the following section we describe research in this area.

3.1 Related Work

We survey both research that were directly aimed to collision detection of deformable models and techniques that may be used for deformable models but were not directly assigned to collision detection but may assist with it.

Efficient Collision Detection of Complex Deformable Models using AABB Trees (Van Den Bergen) [39]

This work used AABB as the BV. As it has been shown that OBB are faster for rigid bodies [14], this work presents a way to speed up overlap tests between AABB's. They also show how to quickly update AABB trees when bodies deform, thus finding AABB tree as a method of choice for collision detection of complex models undergoing deformation. The way is to refit the BV's along the tree in a bottom-up order. In order to avoid the overhead of recursion, the data structure is implemented as an array. Although the refitting may degrade the overlap quality, the claim that unless radical deformation occurs, the data structure will maintain good approximation quality, thus does not degrade performance substantially. the situation is almost mainly to do with a quick updating of the tree by refitting boxes in tree in a bottom-top fashion, updating with no recursion, partitioning the plane orthogonal to the longest axis and splitting box into two halves (not necessarily balanced). An observation presented here is embedded in the formula that holds only for AABB and k-dops ([26])

$$B(S1US2) = B(S1)UB(S2)$$

where $S1$ and $S2$ are two volumes of two nodes and B is the function that generates BV. If this function holds, it allows us to construct and refit the bounding volume hierarchy in a bottom-top fashion.

BucketTree: Improving Collision Detection Between Deformable Objects (Ganovelli et al.) [12]

The hierarchy used here is constructed regardless of the distribution of the primitives. The space is subdivided recursively into eight octants and only the ones that contain primitives are retained. The bounding volume is AABB. They used three sorted arrays for each axis, each stores all the primitives, in order to maintain the position of the primitives in space.

Robust Treatment of Collisions, Contact and Friction for Cloth Animation (Bridson et al.) [5]

They presented an algorithm to efficiently and robustly process collisions, contact and friction in cloth simulation. Since cloth is very thin, even small interpenetrations can lead to cloth protruding from the wrong side. This is visually disturbing and can be difficult to correct. They developed collision handling algorithm that works with any method for simulating the internal dynamics (i.e. stretching, shearing, and bending) to efficiently yet robustly produce visually complex motion free from interference as in Figure 3.1. The key idea is to combine a fail-safe geometric collision method with a fast (non-stiff) repulsion force method that models cloth thickness as well as both static and kinetic friction. Their robust geometric collision algorithm is the first scheme that guarantees no dynamic self-interference of cloth. For the purposes of demonstrating their collision handling, they used a simple mass-spring model for the internal cloth dynamics, as opposed to a true constitutive model. In their basic model, particles are arranged in a rectangular array with structural springs connecting immediate neighbors. Diagonal springs provide shear support, and springs connected to every other node resist bending. They used AABB hierarchy in their implementation. It is built bottom-up once at the beginning of the simulation using the topology of the cloth mesh. In one sweep they greedily pair off adjacent triangles to get parent nodes, then in a sweep in the opposite direction pair off these to get the next level up, and so on alternating sweep directions until they are left with one root node. They also recalculate the hierarchy for each iteration of the collision algorithm

Interactive animation of cloth including self collision detection (Fuhrmann et al.) [11]

They presented a system for interactive animation of cloth, which can be used in e-commerce applications, games or even in virtual prototyping systems. In order not to restrict the shape of the cloth they used a triangulated mesh, which serves as basis of a mass-spring system. Their algorithm is able to animate a piece of cloth in real-time. During the animation, collisions with the environment and self collision, which is integrated easily in this system, are handled. They mentioned that self collision detection can be handled by either resolving them after they happened or never letting them occur. They solve the problem approximately by not testing particles against triangles and edges against each other, but they consider only pairs of particles. In order to keep the particles apart, they added constraints between nearby particles. In

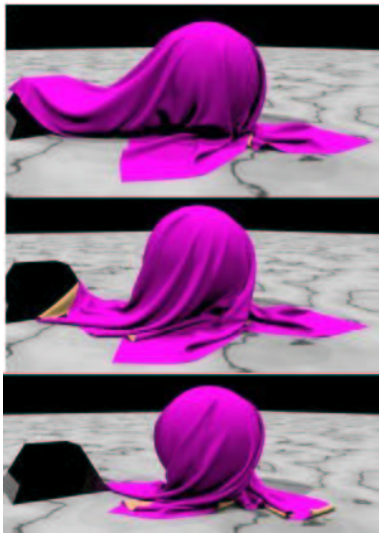


Figure 3.1: The friction between a rotating sphere and a piece of cloth draped over it creates a complex structure of wrinkles and folds

order to speed up the tests, they built a BV hierarchy of the triangles and tested the hierarchy against itself for collisions. Since they were testing only particles against each, other the hierarchy wasbased on them and not on triangles. The structure of the hierarchy does not change during animation and is computed from the flat cloth in advance. After each time step the hierarchy is updated from bottom to top. The bounding boxes are made larger to take into account the distance the particles should keep. Finally, the hierarchy is traversed in order to find nearby particles. They claim that this algorithm is useful as an approximate method for interactive virtual reality applications.

Collision and Self-collision Handling in Cloth Model Dedicated to Design Garments (Provot) [35]

They presented a method for collision handling applied to the semi-rigid mass-spring cloth model. This method supports multiple collision detection. The work is based on time intervals. Within an interval $[t_0, t_0+\Delta t]$, the position and velocities can be computed. They detect whether one or more collisions occurred during this time interval. The types of collisions involved are *point-triangle* and *edge-edge*. They used boudning boxes hierarchy to optimize the process and another optimization based on

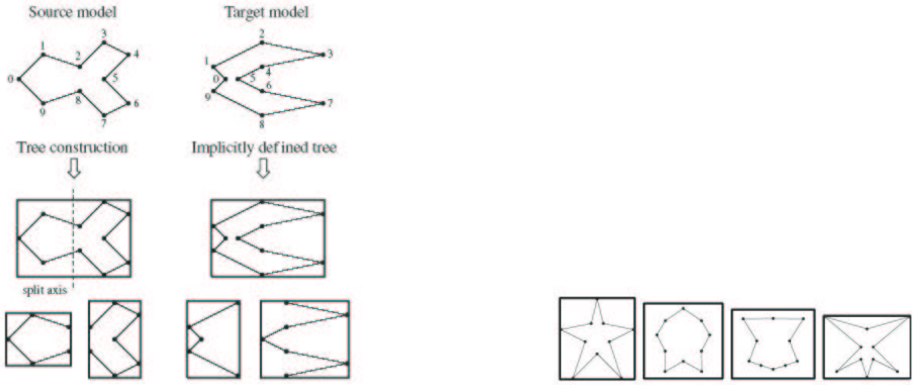
the property that when a given zone (provided it is a connex) has a sufficiently "low curvature", it cannot self-intersect.

Efficient collision detection for models deformed by morphing (Larrson and Moller) [29]

They presented an efficient collision detection algorithm specifically developed for models deformed by mesh morphing (also called mesh blending). They used several BVs (AABB, K-dops and spheres). They showed how the bounding volumes can be conservatively updated using the same morphing function as used for the vertices. Since only the bounding volumes, and not the entire geometry in them, are updated, this operation is extremely fast. Consider Figure 3.2. They constructed BV hierarchy for source and target shapes (Figure 3.2(a)) that correspond to the common topology of the shapes. Then during morphing (illustrated in Figure 3.2(b) for another set), the BV is morphed with the same function and not necessarily tightly bounds the shape. In addition, the bounding volumes are only updated lazily in a top-down manner, which further improves performance. This way of refitting the volumes is expected to work very well for all types of deformations where there is an efficient bounding-volume update function available that is not dependent on knowing the details of the model geometry inside the bounding volumes but still able to refit each volume reasonably tightly.

CULLIDE: Interactive Collision Detection Between Complex Models in Large Environments using Graphics Hardware (Govindaraju et al.) [18]

They addressed the problem of collision detection among moving objects, either rigid or deformable, using graphics processing units (GPUs). The claimed that at a broad level, such algorithms can be classified into two categories: use of depth and stencil buffer techniques for computing interference and fast computation of distance fields for proximity queries. However, such algorithm suffer from three limitations: the bandwidth to and from the graphics cards is not increasing as fast as computational power, many of these algorithms are mainly restricted to closed objects and most of the current algorithms are designed for a pair of objects and not intended for large environments composed of multiple moving objects. Given an environment composed of triangulated objects, their algorithm computes a potentially colliding set (PCS). The PCS consists of objects that are either overlapping or are in close proximity. They use visibility computations to prune the number of objects in the PCS. This is



(a) Hierarchy of source and target shapes (b) BV during morphing

Figure 3.2: data structure for [29]

based on a linear time two pass rendering algorithm that traverses the list of objects in forward and reverse order. They used a three steps pipeline: object level pruning, sub-object level pruning and exact collision detection (first two are done in GPU and the third in CPU). The advantages of their algorithm is the following. It is relatively simple and makes no assumption about the input model. It can even handle "polygon soups". It involves no precomputation or additional data structures (e.g. hierarchies). As a result, its memory overhead is low. It can easily handle deformable models and breakable objects with deforming geometry and changing topology. Their algorithm doesn't make any assumptions about object motion or temporal coherence between successive frames. It can efficiently compute all the contacts among multiple objects or a pair of highly tessellated models at interactive rates.

Based on this work, they developed another algorithm that performs visibility queries on the GPUs to eliminate a subset of geometric primitives that are not in close proximity [16]. They presented an extension to CULLIDE to perform intra-object or self collisions between complex models. The main extensions over COLLIDE are generalizing the formulation of PCS to check for both inter-object and intra-object collisions and improving the pruning and culling algorithm to compute collision-free subsets based on visibility computations.

Interactive Collision Detection between Deformable Models using Chromatic Decomposition (Govindaraju et al.) [15]

They devised an algorithm to accelerate the process of detecting collision in deformable models that consist of triangle mesh for cloth modeling and medical simulation. They detect collisions among prism that represent the swept volume of triangles between two time steps. They define a chromatic subdivision that constitutes independent sets which partition the mesh. They use graph coloring algorithm on the dual graph of the mesh to construct the subdivision. Using this subdivision, they perform two types of collision detection: intra-set (testing primitives of the same set) and inter-set (testing primitives from different sets). The algorithm is a combination of several techniques in four stages (two for the broad phase and two for the narrow phase) which is discussed below.

- AABB hierarchy is embedded on the model. The purpose is to cull primitives that do not overlap.
- Cull more non-overlapping primitives with the use of 2.5D and 1D overlap tests. The 2.5D tests correspond to the work with GPU introduced in [18] (see above) and 1D tests correspond to similar tests on the projections of the AABB and performed on the CPU.
- Perform primitives overlapping tests using vertex-edge and edge-edge tests.
- Perform similar tests on adjacent primitives.

Using those tests, they achieved a much better performance due to the avoidance of many false tests that were not performed thanks to their broad phase steps.

Dynamic collision detection in virtual reality applications (Eckstein and E. Schomer [10]

They presented data structures and algorithms for dynamics collision detection in virtual reality applications. The idea is to detect collision of the swept volumes of the objects induced between two time steps and report contact time. They use hierarchies of BV's, each can be either a sphere or an OBB (the smaller one) which is built in a top-down fashion. Each BV encloses the respective objects, both at the beginning and at the end of a motion step. The degree of each node is between 2 and

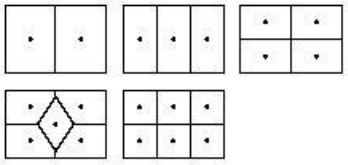


Figure 3.3: The partition regions for the splitting step

6 and depends on the worst quality of the different BV's and that directs them in the split step. They present two such methods. In the first one, they first compute the BV of the entire scene and compute its geometric center. Then in each step they look for a face which maximizes the distance from its BV center. The center of this face will be a new center. Now they redecompose the faces to the new set of centers, based on the minimal distances to centers. The second method splits the space to 2 to 6 regions (see Figure 3.3) and partitions the faces accordingly.

BD-Tree: Output-Sensitive Collision Detection for Reduced Deformable Models (James and Pai) [23]

They introduced the Bounded Deformation Tree, or BD-Tree, which can perform collision detection with reduced deformable models at costs comparable to collision detection with rigid objects. The idea is that the algorithm is efficient only when the deformation is very limited (such as models with vibrating objects). They used a wrapped hierarchy of spheres. Since the deformation is limited, the hierarchy topology will not have to be updated. They update the bounding spheres by recomputing the center as a weighted average of the points, and the radius is increased such that it will bound the effect of each component (see Figure 3.4 for an illustration).

Optimized Spatial Hashing for Collision Detection of Deformable Objects (Tescher et al.) [37]

They proposed an approach to collision and self collision detection of dynamically deforming objects that consist of tetrahedrons (but can be adapted to other object primitives). The proposed algorithm employs a hash function for compressing a potentially infinite regular spatial grid. It can be generated very efficiently and does not

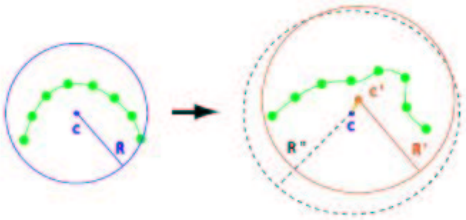


Figure 3.4: BD-tree: the effect of a deformation step on the center and the radius of the bounding sphere

require complex data structures. The algorithm classifies all object primitives with respect to small grid AABB. Therefore, a hash function maps the 3D boxes (cells) to a 1D hash table index. As a result, each hash table index contains a small number of object primitives that have to be checked against each other for intersection. In a first pass, all vertices of all objects are classified with respect to these small 3D cells. In a second pass, all tetrahedrons are classified with respect to the same 3D cells. If a tetrahedron intersects with a cell, all vertices, which have been associated with this cell in the first pass, are checked for interference with this tetrahedron (see Figure 3.5 for an illustration). In order to have an efficient performance, the optimize the parameters of the algorithm which are the hash table size and grid cell size. The complexity of the algorithm is $O(npq)$ where n is the number of primitives, p is the average number of cells intersected by a tetrahedron and q is the average number of vertices per cell. If p and q are constant (that is when the cell size is chosen to be proportional to the average tetrahedron size and there are no hash collisions), the running time is linear on the input. The algorithm does not detect whether an edge intersects with a tetrahedron.

Collision and Self-Collision Detection: Efficient and Robust Solutions for Highly Deformable Surfaces (Volino and Thalmann) [33]

They detected collisions using a hierarchical algorithm that takes advantage of curvature properties giving us full power of hierarchical algorithms for self-collision situations. As they intended to cope with highly deformed surfaces, like those obtained for surface wrinkling, they needed an algorithm that keeps being very efficient for self-collision detection. The idea of their work is to speed up self-collision detection by taking advantage of adjacency, using some geometrical properties in order to

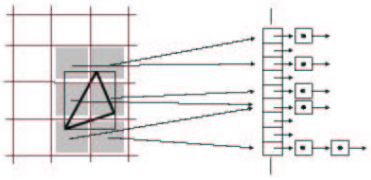


Figure 3.5: Hash values are computed for all grid cells covered by the AABB of a tetrahedron. The tetrahedron is checked for intersection with all vertices found at these hash indices

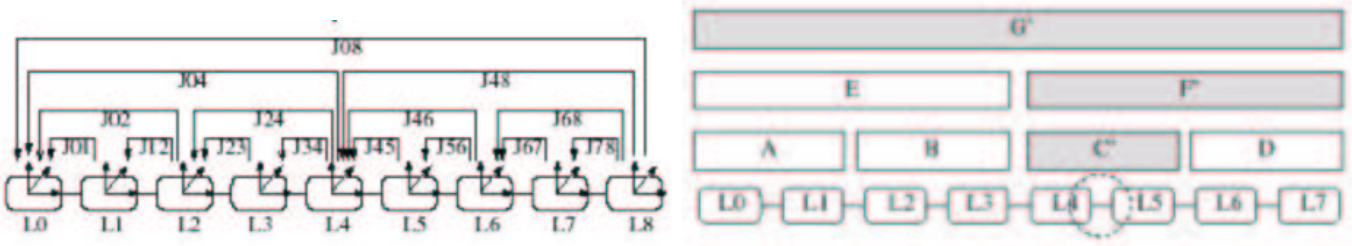
skip collision detection within some areas or between adjacent areas. By applying those properties, they got an average self-collision detection time proportional to the number of colliding elements.

Real-Time Simulation of Deformable Objects: Tools and Applications (Brown et al.) [6]

This paper presents algorithms for animating deformable objects in real-time. It focuses on computing the deformation of an object subject to external forces and detecting collisions among deformable and rigid objects. The targeted application domain is surgical training. exploit the fact that many deformations are local. By propagating forces in a carefully ordered fashion through an elastic mass-spring mesh, they effectively limit the computations to the portions of objects that undergo significant deformations. To accelerate collision detection, they pre-compute hierarchical representations for all objects in the scene; when objects are being deformed, they only update those parts of the hierarchies that need to be modified. They use a hierarchy of spheres. In order to be able to use BV algorithm efficiently with deforming bodies, propose a new sphere tree whose balanced structure is computed only once. When an object deforms, the structure of its tree remains fixed, i.e., no sphere is ever added or removed; only the radii and positions of some spheres are adjusted. The update is done bottom-up, only for deforming triangles; each sphere is updated once. Thus it is efficient when the deformation is local.

Efficient Maintenance and Self-Collision Testing for Kinematic Chains (Lotan et al.) [31]

This is an example for assigning data structure for a specific problem. The kine-



(a) the hierarchy (b) a deformation of a link

Figure 3.6: data structure for [31]

matic chain is studied in robotics and is a useful model for motion of biological macro-molecules. It is a serial linkage composed of N links and $N-1$ joints. In this model, local changes have global effects. Thus a change to a joint angle may bring together pieces of chain that were previously far a part. They used OBB as the bounding volume. Consider Figure 3.6. Figure 3.6(a) illustrates a hierarchy of 8 links with 9 joints. Then the joint between $L4$ and $L5$ transform (Figure 3.6(b)). Under this case, the links of the subtree, E, cannot collide with each other, but they can collide with links of the subtree, F'. Also note that collision may occur between links of subtree F'. These observations are taken care of during the processing of a query. The worst case time for a query is $O(n^{4/3})$, but they mentioned that in practice the performance is much better.

Collision Detection for Deformable Necklaces (Guibas et al.) [1]

Closely related to the work above. They studied deformable necklaces — flexible chains of balls, or *beads*, in which only adjacent beads may intersect. They use spheres as the BV. They achieved a time of $O(n^{2-2/d})$ in d-dimensional, $d \geq 3$, for collision checking. The underline assumption on the beads is that the ratio between the sizes of each pair is bounded by a constant factor. The main contribution is that their BV can be efficiently maintained under deformation and remain tightly fitting. They used wrapped sphere hierarchy to achieve better fitting. Its construction takes $O(n \log n)$ time and maintain the hierarchy in $\theta(n \log n)$ time, while in practice the actual time is close to linear.

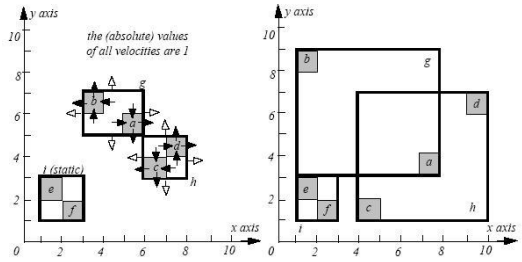


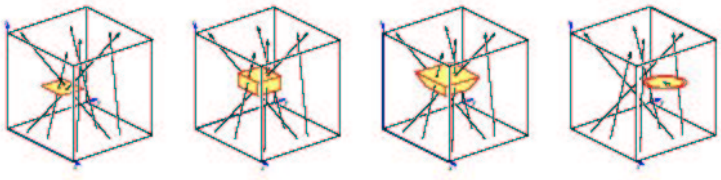
Figure 3.7: TPR*-tree

The TRP*-Tree: An Optimized Spatio-Temporal Access Method for Predictive Queries (Tao et al.) [36]

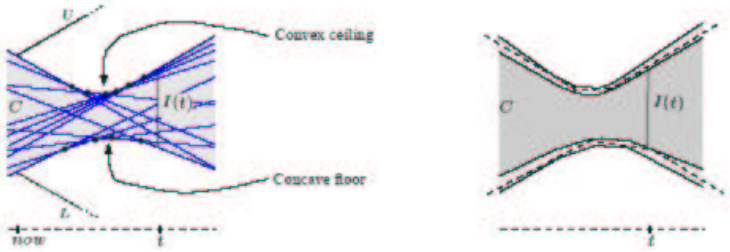
Based on R*-trees (TPR stands for Spatio Temporal R tree), they developed a dynamic data structure for predictive queries on moving objects. The idea is to use the insertion/deletion algorithms of the R*-tree to minimize penalty metrics to enhance performance. The velocity of each object (along each axis) has to be constant and known in advance for some period of time in the future. The bounding boxes has 2^d velocities (d is the model dimension), a positive and negative velocities to each axis which are determined as the maximum/minimum velocities of the bounded objects. Then, the bounding cubes extend according to the respective velocities and use for time queries. Since the bounding boxes get bigger, they should be updated. It is done either when new velocities are given or when performing insertion/deletion. Consider Figure 3.7. The model contains 6 objects with velocities as indicated. The initial configuration is on the left, and the configuration after 2 seconds are depicted on the right. Note how bounding boxes are extended when their corresponding objects move.

STAR-Tree: An Efficient Self-Adjusting Index for Moving Objects (Magdalena et al.) [34]

This is another work based on R*-trees to various range queries (STAR stands for Spatio-temporal self-adjusting R-tree) on space-time. The possible ranges for plane-time are depicted in Figure 3.8(a). The data structure maintains auxiliary information to update itself locally when performance deteriorates. The idea is to project each



(a) Possible space-time range queries



(b) Approximating the bounding segment

Figure 3.8: STAR tree

axis and find the bounding segment, as time passes. Consider Figure 3.8(b). The left part represents the places along a specific axis objects are located along time (the straight lines). Let C denote this structure. Two polygonal chains (upper and lower boundaries) bound those lines. Each vertex on those mark a topological change on the bounding segment. Thus those correspond to time points where the corresponding data structure has to be updated. In order to reduce the number of updates (thus enhance performance), they use a data structure that was presented in [2]. This data structure is an ε-approximation of C which contain less vertices (the right part of Figure 3.8(b)). This data structure requires $1/\sqrt(\varepsilon)$ vertices to exist. Thus ε is a trade off between approximation and complexity of C.

A Morphology-Based Topology Modification Algorithm for Deformable Models (Duan et al.) [9]

They proposed a topology modification algorithm based on mathematical morphology. The idea is to construct a signed distance function around colliding regions and then apply closing operation to create a distance function. In order to apply

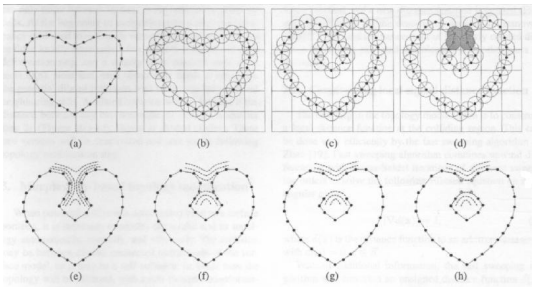


Figure 3.9: An example of topology modification when cllision occurs

topology modification, they used a particle-based collision detection. When collision occurs, the topology around the collision will be modified (see Figure 3.9). They were interested more in dilated collision in which dilated surfaces collide. They used hybrid approach for that which includes uniform occupancy grid for fast collision rejection and bounding spheres within the same or neighbor grid cell to detect dilated collision.

3.2 More Approaches for Deformable Models

In this section we describe few approaches for detecting collision of deformable models. More details can be found at [38].

3.2.1 Stochastic Methods

The idea is to provide inexact methods to speed up the detection step. This can be justified by several observations. First, polygonal models are just an approximation of the true geometry. Second, the perceived quality of most interactive 3D applications does not depend on exact simulation, but rather on real-time response to collisions. At the same time, humans cannot distinguish between physically-correct and physically-plausible behavior of objects.

There are two methods that use probabilistic principles in fairly different ways. The first one (An average case approach) does not check explicitly for collision of the primitives. Instead, is uses a probability for collision estimated from the distribution of the polygons inside a grid, and combinatorial reasoning about the probability that

a cell contains many polygons from both two objects. The second method (Stochastic Collision Detection Based on Randomly Selected Primitives) select random pairs of colliding features as an initial guess of the potential intersecting regions. This method can be further augmented by ensuring that the sampling covers features from the entire body and that the features are already close enough. When the object moves or deforms, the method consider temporal coherence in the sense that if a pair of features is close enough at a time step, it may still be interesting in the next one. This form *active pairs* that are tested. In Addition, random pairs are added to the list of active pairs at each time step to track more collisions.

3.2.1.1 Related Work

Hierarchy Accelerated Stochastic Collision Detection (Kimmerle et al.) [25]

They presented a new framework for collision and self-collision detection for highly deformable objects such as cloth in which they trade-off accuracy for computation time. They combined two approaches: the first manages a set of randomly chosen pairs of geometric primitives which iteratively converge to local distance minima; the second is hierarchy of bounding volumes (k-dops) which helps to pick appropriate random pairs. The way they choose active pairs to be tested is the following. At first a set of active pairs is chosen randomly. Each of these pairs then iteratively converges to a local distance minimum by repeatedly replacing each element by its topological neighbor with smaller distance compared to the other elements (Figure 3.10(a)). If the distance falls below a given proximity threshold, a collision is detected (Figure 3.10(b)). A local distance minimum where no active pair is present, such as in Figure 3.10(c), might eventually lead to an undetected collision (Figure 3.10(d)). If, during or at the end of a convergence process, the elements of an active pair are far from each other, they are discarded. At each step of the animation loop, they update each active pair by iterating it until convergence is reached at a new local minimum resulting from the motion of the bodies. To overcome the drawback of possible pure stochastic collisions detection they combined it with a bounding volume hierarchy of k-DOPs. First, a collision detection on the k-DOP -hierarchy is performed, returning all colliding leaves of the hierarchy tree. Then in the second step, a stochastic collision detection where the random sample pairs are created only on the colliding leaves.

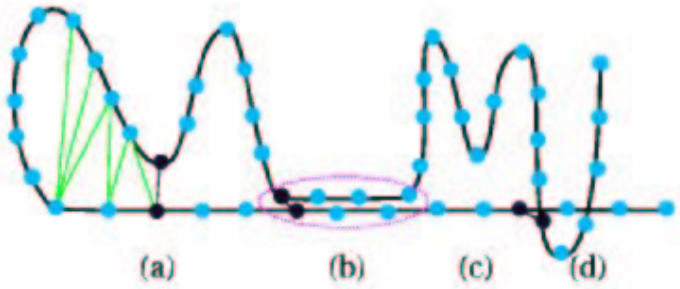


Figure 3.10: Stochastic Collision Detection. The active pairs of points are shown in black

3.2.2 Distance Fields

Distance fields specify the minimum distance to a closed surface for all points in the field. The distance may be signed in order to distinguish between inside and outside. The evaluation of distances and normals needed for collision detection and response is extremely fast and independent of the geometric complexity of the object. A distance field $D : R^3 \rightarrow R$ defines a surface as the zero level set, i. e. $S = \{p|D(p) = 0\}$. In contrast to other implicit representations, a simple function evaluation also yields the Euclidean distance to the surface. Collision detection between different objects is carried out point-wise when using distance fields. If both deformable objects are volumetric, vertices on the surface of one object are compared against the distance field of the other object and vice versa. A collision has occurred if $D(p) < 0$. Distance fields do not only report collisions, but also compute the penetration depth at the same. In order to take deformations of the objects into account, the distance fields are updated before each time step. The actual collision detection is carried out by a hierarchical method. Although efficient algorithms for computing distance fields have been proposed recently, this generation is still not fast enough for interactive applications, where distance fields have to be updated during run-time due to deforming geometry. However, distance fields provide a highly robust collision detection, since they divide space strictly into inside and outside. It is also worth mentioning that the method is well-suited for volumetric objects.

3.2.3 Image-Space

This approach commonly process projections of objects to accelerate collision queries. Since they do not require any pre-processing , they are especially appropriate for environments with dynamically deforming objects. Furthermore, image-space techniques can commonly be implemented using graphics hardware. They can be used to detect collisions and self-collisions. Image-space techniques usually work with triangulated surfaces. Topology changes of objects do not cause any problems. Since image-space techniques work with discretized representations of objects, they do not provide exact collision information. The accuracy of the collision detection depends on the discretization error. Thus, accuracy and performance can be balanced in a certain range by changing the resolution of the rendering process.

3.2.3.1 Related Work

Volumetric Collision Detection for Deformable Objects (Heidelberger and Teschner) [21]

The algorithm is based on a Layered Depth Image (LDI) decomposition of the intersection volume. LDI data structure stores multiple depth values per pixel, thus can be used to approximate volumes. While most methods consider surfaces, this work is volumetric. The LDI provides a discrete representation of the intersection volume and allows volume-based collision queries. The generation of LDI is accelerated by graphics hardware. First they use AABB (not hierarchical data structure; thus no preprocessing work is required) to find collision regions. Then, within these regions, they compute LDI representation. Finally, using Boolean operations on the LDI representation, they obtain the intersection. The three stages are illustrated in Figure 3.11. Another work with LDI is [20].

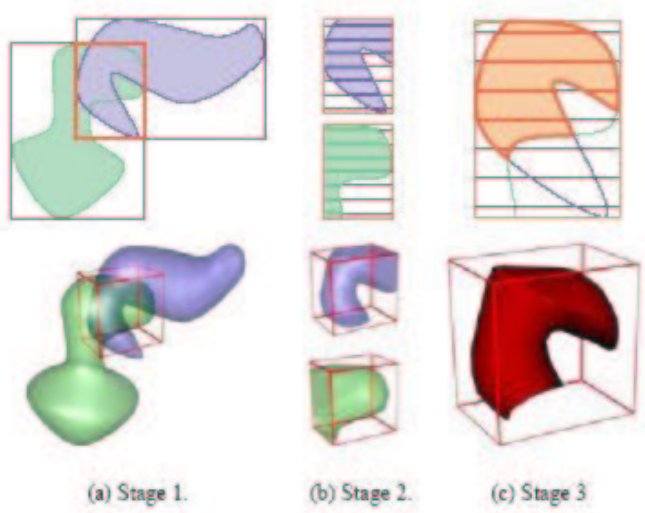


Figure 3.11: Collision detection steps of [21]

CInDeR: Collision and Interference Detection in Real-time using Graphics Hardware (Knott and Pai) [27]

This paper proposes an image-space method for detecting interference between solid polyhedral bodies. The algorithm makes use of virtual ray casting to determine which portions of the edges of the polyhedrons in question lie within volumes enclosed by other polyhedrons. Among the properties of this techniques are:

- No special data structures are required.
- No preprocessing of models is required.
- The algorithm's expected asymptotic running time is linear in both the number of objects being tested and the number of polygons comprising the objects.

The precision of interference computations is constrained by the resolution of the frame buffer region that they render into. Interference detection is performed by point-sampling the scene and looking for object edges that are interior to other polyhedrons. This is done using hardware raycasting.

Chapter 4

Current Work

We implemented a software tool to detect collision among particles that translate in space. Particles are defined as points and a collision among them is defined as a pair of particles being close one to each other. Such software could be useful for molecular modeling applications. Another way to illustrate the collision is refer to particles as balls with the same radius. Our work can relate to several models. First, it could simulate deforming bodies if we attach topology to the particles, such that they approximate the body boundaries (see [9]). Another potential model can be of molecular arrangements where particles correspond to molecules (thus have volume). Currently we only support balls of the same size, but we can partially relax this assumption if we bound the ratio between the biggest and smallest ball. We use two techniques in our implementation, each represents an extreme approach. The first one, *Deformable Spanners* is an implementation based on the paper by Gao et al. [13] [1]. They presented a sophisticated data structure on particles in 3D, whose one ability is to perform collision detection. The purpose of this work is to provide nice theoretic bounds on construction query times. However, the constants for detecting collision are huge. Thus, it is interesting to check how it behaves in practice. We describe this work in detail in Section 4.1. On the other hand, our second technique, *Uniform Grid*, is very easy to understand and implement. However, theoretically it can provide quadratic worst case time if the particles are not well distributed. We present results obtained with two techniques, where for the Uniform Grid approach we use several voxel's sizes.

[1] The respective code was provided by the authors of the paper. The author of this work would like to thank An Nguyen from Stanford University for helpful advises and for implementing specific code upon request

Another ability we implemented is for the sake of using hybrid methods. The idea is to be able to seamlessly switch between two techniques, whenever a potential to improve performance by that is detected. Such cases are encouraged whenever switching to a different technique improve the query time. Obviously this improvement should beat the time for this switch. We are currently able to do that with the two above techniques, but it is still unknown how to efficiently detect when this switch is worthwhile.

We use *Visual C++ .NET environment* for our implementation and extensively use the openGl library.

4.1 Deformable Spanner

In this section we describe in detail the work of Gao, Guibas and Nguyen [13].

For a set S of points in R_d, an *s-spanner* is a graph on S such that any pair of points is connected via some path in the spanner whose total length is at most s times the Euclidean distance between the points. The main idea of this work is to provide a dynamic deformable spanner that is going to be maintained when points(or particles) translate. They proposed a $(1+\varepsilon)$-spanner with $O(n/\varepsilon^d)$ edges for an ε parameter. The data structure they used is *hierarchy of discrete centers*. A set of discrete centers with radius r for point set S is defined as a maximal subset $S' \subseteq S$ such that the balls with radius r centered at the discrete centers contain all the points of S but any two centers are of a distance at least r away from each other. The hierarchy is constructed as the following. S_0 is the original point set S and S_i is a set of discrete centers of S_{i-1} with radius 2^i, for $i > 0$. The data structure supports dynamic insertion and deletion of particles. Belo is a result summery of their work:

- For any given ε, the spanner built using the above technique gives a stretch factor of $1+\varepsilon$.
- The total number of edges is $O(n/\varepsilon^d)$ and the maximum degree of G is $O(log\alpha/\varepsilon^d)$ where α is the ratio between the distance of the furthest particles and the closest one, or the aspect ratio.
- Dynamic insertion and deletion of points in the spanner takes $O(log\alpha/\varepsilon^d)$.
- The spanner can be constructed in time $O(log\alpha/\varepsilon^d)$.

- The kinetic spanner is efficient, responsive, local and compact. Specifically, the total number of events in maintaining G is $O(n^2 logn)$ under pseudo-algebraic motion. Each event can be updated in $O(logn/\varepsilon + d)$ time. Each point is involved in at most $O(logn/\varepsilon^d)$ certificates.

There are several applications of this work such as well separated decomposition, all near neighbors query, $1+\varepsilon$-nearest neighbor, k-center and our interest, collision detection. They describe a method to find the closest pair in the following way. They construct a $1+\varepsilon$-spanner where $\varepsilon < 1$. Then the closest pair must have an edge in G. Otherwise, ε would have to be greater than 1. They use a kinetic priority queue to maintain the closest pair. Using this method, they can check whether there exists a collision by testing the closest pair. However, this is still not a general collision detection. Omitted from their paper, the way to do it is to traverse the nodes at the lower level, both check lower level children of the same node, and ones whose parents share an edge. The work is linear, but the constants involved are huge.

Figure 4.1 depicts a 2D input set and the respective deformable spanner hierarchy (the red edges). Two particles (surrounded in top-left image) were found to be colliding.

4.2 Experimental results

We used the following structure for our experiments. It includes eight equally structured sets of particles that are initially placed at different octants in space. Then each moves towards the origin, while each particle has a random noisy factor. By doing that, the clusters simultaneously collide and then separate by continuing moving along their general trajectories. We tested several such clusters. In each time we use different density in each cluster. Figure 4.4 contain screen shots of the simulations. A zoom in is depicted in Figure 4.3. See Figure 4.2 for a 2D illustration.

We tested the grid method with different voxel sizes, the deformable spanner and a brute force approach which simply check all pairs.[2] Tables 4.2 and 4.3 and Figures 4.5,4.6 and 4.7 depict the results (see Table 4.1 for abbreviations).

[2]For the tests with 64000 particles, it was impossible with our machines to test the deformable spanner and the brute force approach. However, it was evident that the grid outperforms both.

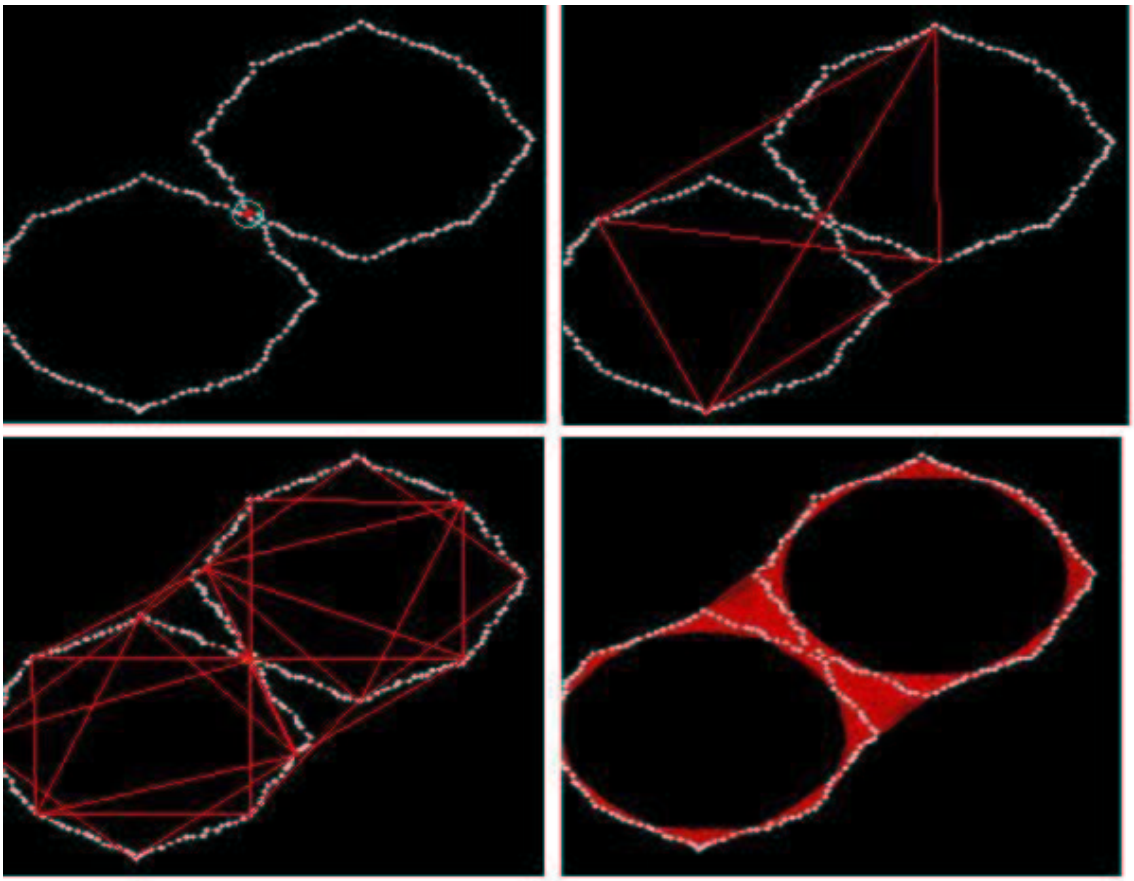


Figure 4.1: Deformable spanner example: different layers

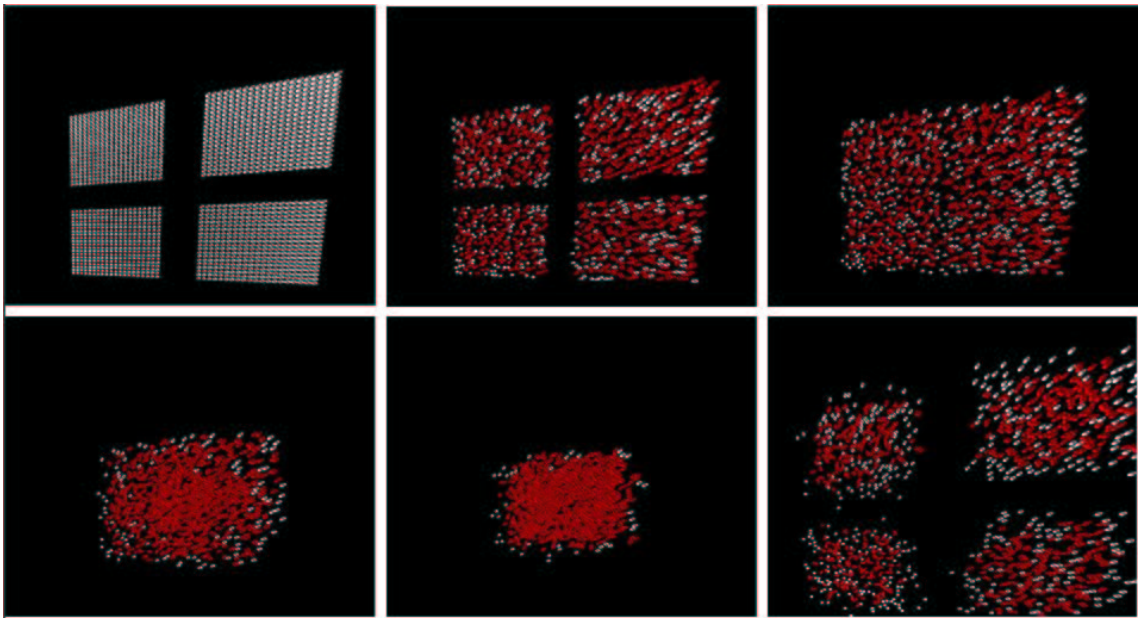


Figure 4.2: 2D particle set - 1600 particles (colliding particles are painted in red)

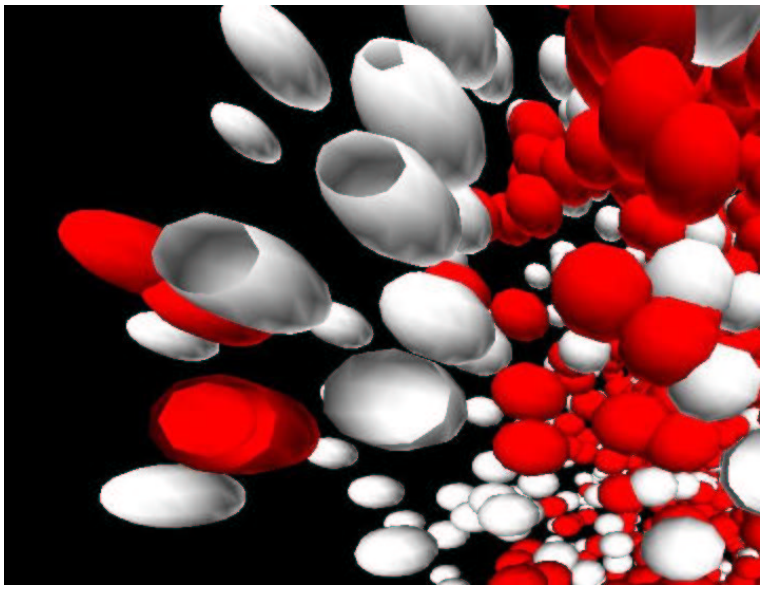


Figure 4.3: A zoom in on a 3D set (colliding particles are painted in red)

Abbreviation	**Explanation**
#par	input number of particles
#col	number of collisions
#tes	overall number of tests performed (in millions)
time	total time (sec.)
bf	brute force

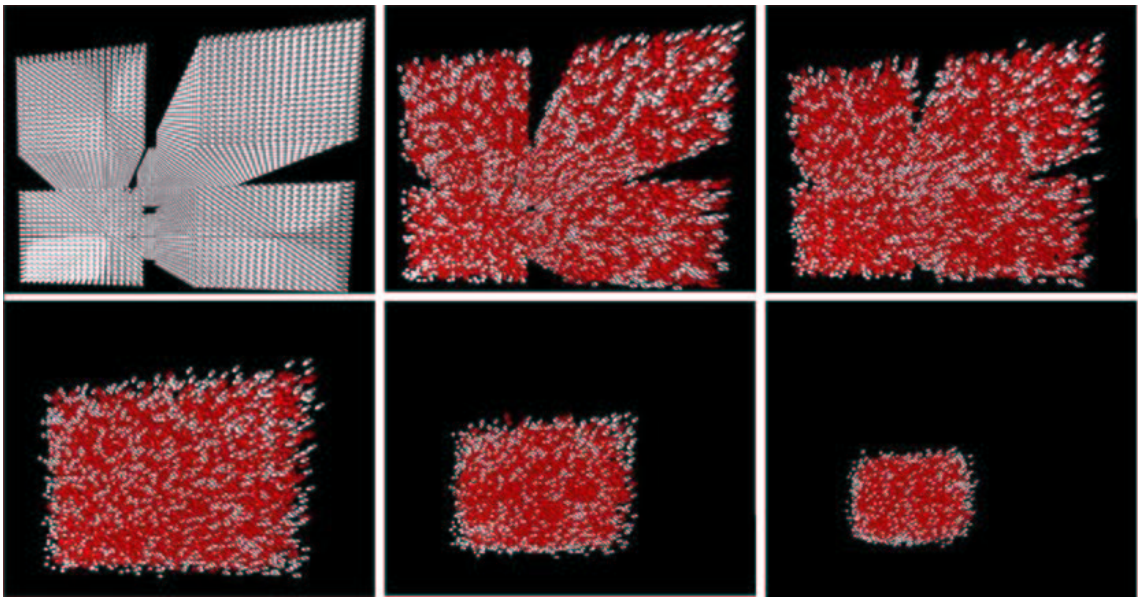
Table 4.1: Abbreviations

		grid size					
		0.125		0.25		0.5	
#par	#col	#tes	time	#tes	time	#tes	time
1000	3237	0.67	0.41	4.47	0.39	19.37	0.83
8000	233473	43.63	4.93	280.86	13.38	1224.77	54.29
27000	2730301	498.18	40.14	3202.03	160.87	13934.85	956.27
64000	15411495	2785.77	223.35	17920.65	942.09	78091.97	8067.43

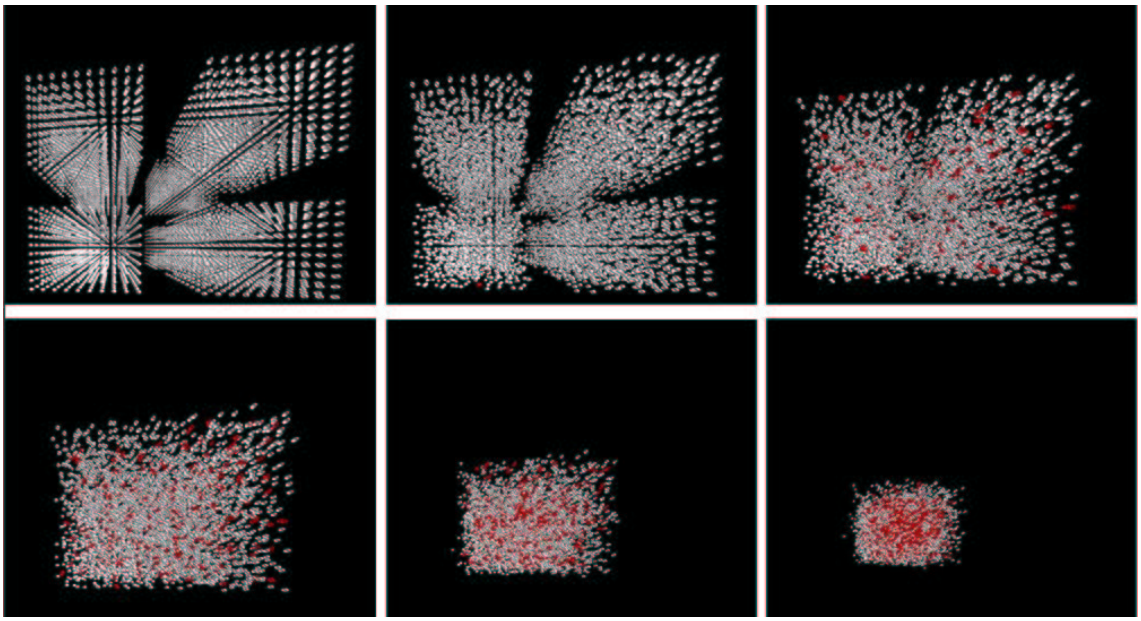
Table 4.2: Experiment results

The immediate conclusion is that the grid method beats the deformable spanner algorithm for those input sizes.[3] More then that, the deformable spanner is even slower than the brute force approach. This results hold even for big datasets (the 64000 particles). It is very likely that the same conclusions would have been achieved with other datasets. Since the deformable spanner time is linear (but with huge constants), it would be interesting to get results for much bigger input sets. However, such datasets would evidently be impractical for real-time simulations, at least for machines such as the one we used (note that even the bigger datasets we used are not suited for real-time applications). Not surprisingly, the running time is proportional to the number of tests being performed. This is true for all approaches. Among the various grid sizes we used, it is evident that a small size of voxel is greatly desired. Thus it is recommended to use the smallest voxel size so that two colliding particles are located in either the same voxel or in two neighbor ones.

[3]The time for 1000 particles was too fast for the time resolution to be precise. However, it indicates that grid outperforms the deformable spanner here too.



(a) 64000 particles (colliding particles are painted in red)



(b) 8000 particles (colliding particles are painted in red)

Figure 4.4: 3D particle sets

		def spanner		brute force	
#par	#col	#tes	time	#tes	time
1000	3237	3.24	2.16	49.95	1.67
8000	233473	2600.26	701.822	3199.6	306.47
27000	2730301	29306.16	8602.29	36448.65	3760.53

Table 4.3: Experiment results

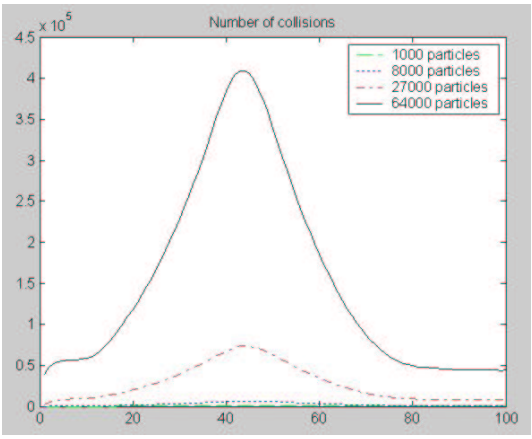


Figure 4.5: 3D particle set: number of collisions

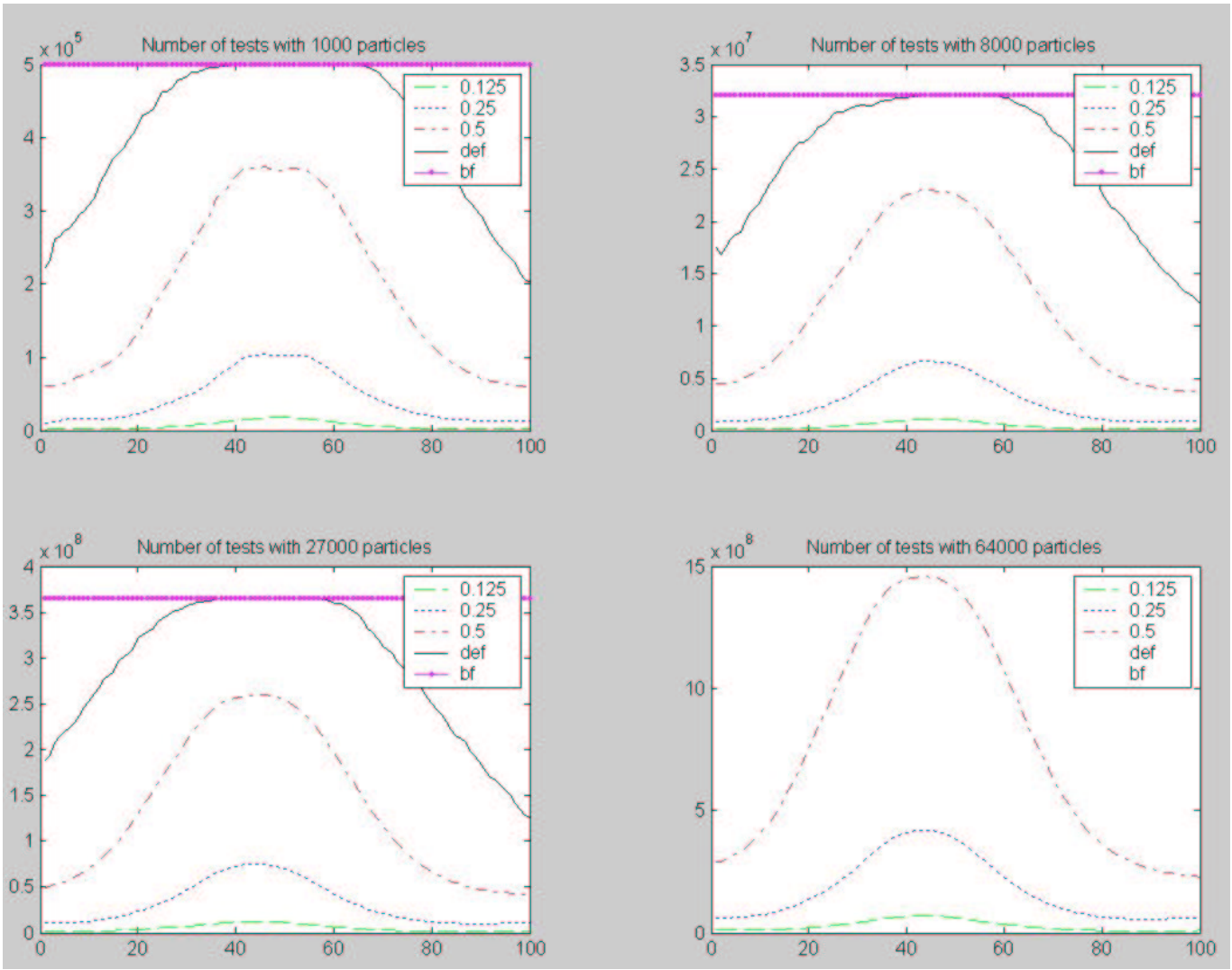


Figure 4.6: 3D particle set: number of tests

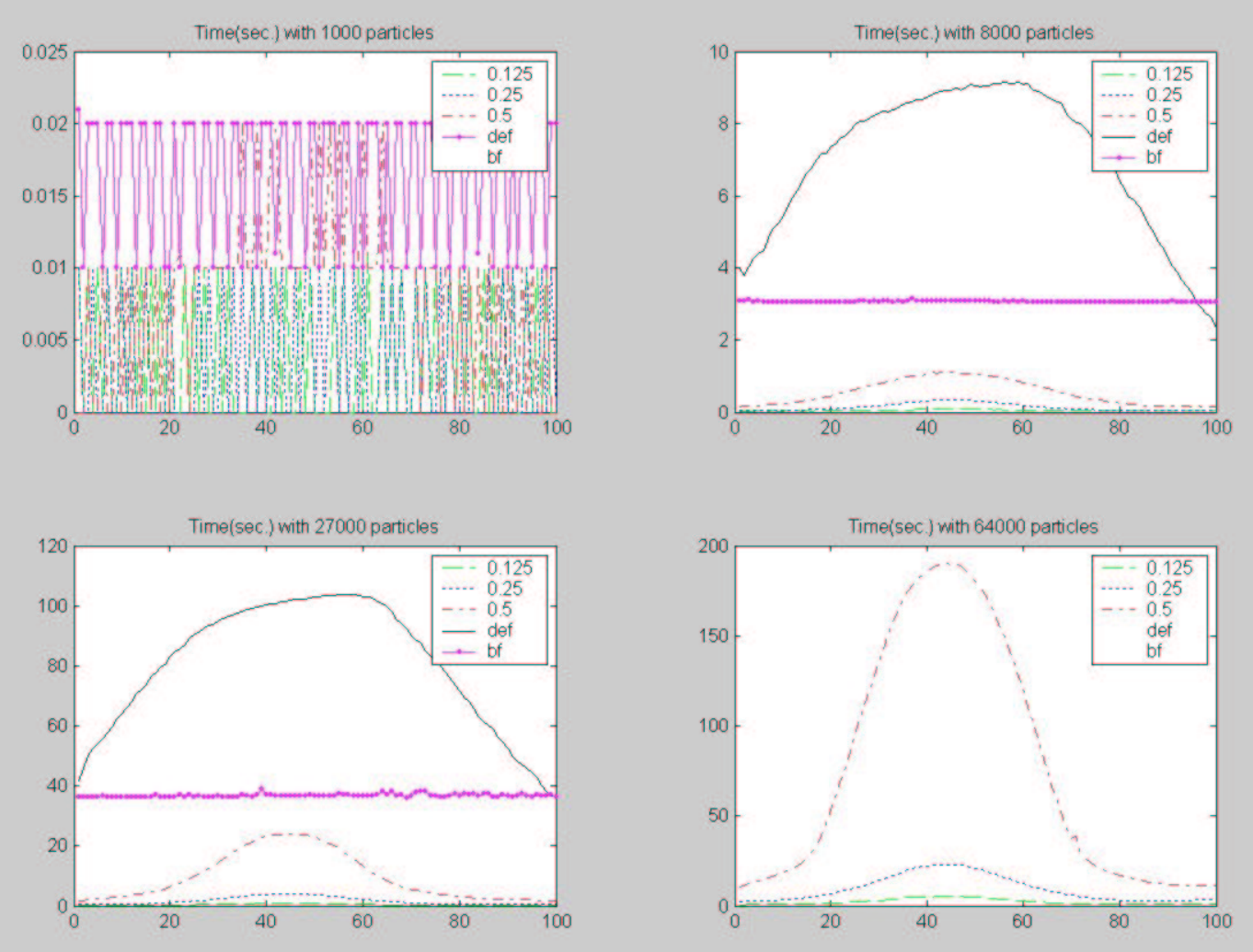


Figure 4.7: 3D particle set: time

4.3 Future Work

There are several directions to extend our work further. One direction is to add topological relationship among the particles in order to simulate surfaces. This will require the ability to modify the topology in cases of collisions (possibly using the technique in [9]). Another possibility is to support different sizes of balls (not supported by the current deformable spanner), or even more shapes to have a more general model. Another direction for future research is to add other types of techniques to be used in a hybrid approach and to establish criteria which one to select in which situations.

We presented several research work performed in the area of deforming models. Many assumed that the topology or the initial structure of the input remain static or were not proved to work well in such cases. An interesting direction would be to relax this assumption in order to support more dynamic models.

Chapter 5

Conclusions

Collisions detection is one of the most important and challenging topics in computer graphics. As performing collision detection is a costly task and since often the model contains very large input and requires real-time performance, efficient algorithms are crucial. The challenge is even greater when objects may deform, since it complicates the common data structure and induces big overhead. However, the research for deformable models is still young, although very active in recent years. Hence, it might have still room for nice enhancements.

In this work we presented the various popular approaches developed for collision detection. We concentrated on the two main approaches in this area, namely *hierarchy of bounding volumes* and *spatial subdivision*. We described them in detail and presented research that have been performed in many variants of each. We focused on the work that have been performed with deformable models. This work usually apply popular data structures in a way to speed up the performance when objects deform or restrict the models' degrees of freedom to make the work more efficient.

Both for rigid and deformable bodies there are no unique method that beats others in any situation. Different methods behave differently under different situations. There are always trade offs to consider when applying the two approaches above. Those depend on the model. We described those trade offs in detail.

We presented our current work that comes to detect collision among moving particles in space. We applied two different techniques and compared the results. This work still has no valuable contributions, but we regard it as a starting point for the future. Evidently, there might be many directions for extension.

Bibliography

[1] P. Agarwal, L. Guibas, A. Nguyen, D. Russel, and L. Zhang. Collision detection for deforming necklaces. *Comput. Geom. Theory Appl.*, 28(2-3):137–163, 2004.

[2] P. K. Agarwal and S. Hal-Peled. Maintaining approximate extent measures of moving points. In *SODA '01: Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 148–157, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics.

[3] G. Baciu and S. Wong. Hardware-assisted self-collision for deformable models. 2002.

[4] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The r*-tree: An efficient and robust access method for points and rectangles. *Proc. SIGMOD Conf. on Management of Data*, pages 322–331, 1990.

[5] R. Bridson, R. Fedkiw, and J. Anderson. Robust treatment of collisions, contact and friction for cloth animation. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 594–603, New York, NY, USA, 2002. ACM Press.

[6] J. Brown, S. Sorkin, C. Bruyns, J. Latombe, K. Montgomery, and M. Stephanides. Real-time simulation of deformable objects: Tools and application, 2001.

[7] S. Cameron. Approximation hierarchies and s-bounds. *Symposium on Solid Modeling Foundations and CAD/CAM Applications tection system for large-scale environments*, pages 189–196, 1991.

[8] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Heidelberg, Germany, 1997.

[9] Y. Duan, H. Zhao, and H. Qin. A morphology-based topology modification algorithm for deformable models.

[10] J. Eckstein and E. Schomer. Dynamic collision detection in virtual reality applications. 1999.

[11] A. Fuhrmann, C. Gross, and V. Luckas. Interactive animation of cloth including self collision detection.

[12] F. Ganovelli, J. Dingliana, and C. O'Sullivan. Buckettree: Improving collision detection between deformable objects. *SCCG2000 Spring Conf. on Comp. Graphics*, 2000.

[13] J. Gao, L. J. Guibas, and A. Nguyen. Deformable spanners and applications. In *SCG '04: Proceedings of the twentieth annual symposium on Computational geometry*, pages 190–199, New York, NY, USA, 2004. ACM Press.

[14] S. Gottschalk, M. C. Lin, and D. Manocha. Obbtree: A hierarchical structure for rapid interference detection. *Comp. Graphics, 30(Annual Conf. Series)*, pages 171–180, 1996.

[15] N. K. Govindaraju, D. Knott, N. Jain, I. Kabul, R. Tamstorf, R. Gayle, M. C. Lin, and D. Manocha. Interactive collision detection between deformable models using chromatic decomposition. 2005.

[16] N. K. Govindaraju, M. C. Lin, and D. Manocha. Quick-cullide: Efficient inter- and intra- object collision culling using gpus. *Proc. of IEEE VR*.

[17] N. K. Govindaraju, M. C. Lin, and D. Manocha. Fast and reliable collision detection using graphics hardware. 2004.

[18] N. K. Govindaraju, S. Redon, M. C. Lin, and D. Manocha. Cullide: interactive collision detection between complex models in large environments using graphics hardware. In *HWWS '03: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 25–32, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[19] H. J. Haverkort, M. de Berg, and J. Gudmundsson. Box-trees for collision checking in industrial installations. *Annual Symposium on Computational Geometry*, pages 53–62, 2002.

[20] B. Heidelberger, M. Teschner, and M. Gross. Detection of collisions and self-collisions using image-space techniques.

[21] B. Heidelberger, M. Teschner, and M. Gross. Volumetric collision detection for deformable objects.

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

[23] D. L. James and D. K. Pai. BD-Tree: Output-sensitive collision detection for reduced deformable models. *ACM Transactions on Graphics (SIGGRAPH 2004)*, 23(3), Aug. 2004.

[24] P. Jimnez, F. Thomas, and C. Torras. 3D Collision Detection: A Survey. *Computers and Graphics*, 25(2):269–285, Apr. 2001.

[25] S. Kimmerle, M. Nesme, and F. Faure. Hierarchy accelerated stochastic collision detection. In *Vision, Modeling, and Visualization 2004*, Stanford, California, 2004.

[26] J. T. Klosowski, J. S. B. Mitchell, H. Sowizral, and K. Zikan. Efficient collision detection using bounding volume hierarchies of k-dops. *IEEE Tr. on Visualization and Comp. Graphics*, 4(1):21–36, 1998.

[27] D. Knott and D. Pai. Cinder: Collision and interference detection in real-time using graphics hardware.

[28] Krishnan, Pattekar, Lin, and Manocha. Spherical shell: A higer order bounding volume for fast proximity queries. *Proceedings of the Third International Workshop on Algorithmic Foundations of Robotics*, pages 177–190, 1994.

[29] T. Larsson and T. Akenine-Mller. Efficient collision detection for models deformed by morphing. *The Visual Computer*, 19, 2003.

[30] M. C. Lin and S. Gottschalk. Collision detection between geometric models: a survey. pages 37–56.

[31] I. Lotan, F. Schwarzer, D. Halperin, and J.-C. Latombe. Efficient maintenance and self-collision testing for kinematic chains. *symposium on Computational geometry*, pages 43–52, 2002.

[32] R. G. Luque, J. L. D. Comba, and C. M. D. S. Freitas. Broad-phase collision detection using semi-adjusting bsp-trees. In *SI3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 179–186, New York, NY, USA, 2005. ACM Press.

[33] P. V. Nadia and Magnenat-Thalmann. Collision and self-collision detection: efficient and robust solution for highly deformable surfaces. *Sixth Eurographics Workshop on Animation and Simulation*, pages 55–65, 1995.

[34] C. M. Procopiuc, P. K. Agarwal, and S. Har-Peled. Star-tree: An efficient self-adjusting index for moving objects. In *ALENEX '02: Revised Papers from the 4th International Workshop on Algorithm Engineering and Experiments*, pages 178–193, London, UK, 2002. Springer-Verlag.

[35] X. Provot. Collision and self-collision handling in cloth model dedicated to design garments.

[36] Y. Tao, D. Papadias, and J. Sun. The tpr*-tree: An optimized spatio-temporal access method for predictive queries.

[37] M. Teschner, B. Heidelberger, M. Mueller, D. Pomeranets, and M. Gross. Optimized spatial hashing for collision detection of deformable objects. 2003.

[38] M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrmann, M.-P. Cani, F. Faure, N. Magnetat-Thalmann, W. Strasser, and P. Volino. Collision detection for deformable objects. *Computer Graphics Forum*, 24(1):61–81, March 2005.

[39] G. van den Bergen. Efficient collision detection of complex deformable models using aabb trees. *J. of Graphics Tools*, 2(4):1–13, 1997.