3^{rd} Workshop in Virtual Reality Interactions and Physical Simulation "VRIPHYS" (2006)
C. Mendoza, I. Navazo (Editors)

Kinetic Separation Lists for Continuous Collision Detection of Deformable Objects

Rene Weller[†] and Gabriel Zachmann[‡]

TU Clausthal, Germany

Abstract

We present a new acceleration scheme for continuous collision detection of objects under arbitrary deformations. Both pairwise and self collision detection are presented. This scheme is facilitated by a new acceleration data structure, the kinetic separation list. *The event-based approach of our kinetic separation list enables us to transform the continuous problem into a discrete one. Thus, the number of updates of the bounding volume hierarchies as well as the number of bounding volume checks can be reduced significantly.*
We performed a comparison of our kinetic approaches with the classical swept volume algorithm. The results show that our algorithm performs up to fifty times faster in practically relevant scenarios.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Geometric algorithms, Object hierarchies I.3.7 [Computer Graphics]: Animation, Virtual reality

1. Introduction

Environments with dynamically deforming objects play an important role in many applications, including medical simulation, entertainment, and surgery simulation. Virtually all of these applications require collision detection to be performed in order to avoid the simulated objects to penetrate themselves or each other. For example, in cloth simulations, we have to avoid penetrations between the cloth and the body, but also between the wrinkles of the cloth itself.

Most current techniques use bounding volume hierarchies (BVHs) to quickly cull parts of the objects that cannot intersect. Usually, a BVH is constructed in a pre-processing step, but if the object deforms, the hierarchy becomes invalid. In order to still use this well-known method for deforming objects, it is necessary to update the hierarchies after the deformation happened. Another problem that arises when using BVHs for self-collision detection is the problem of adjacency, because two adjacent subareas are always colliding by contact along their borders. Using swept volumes and lazy updating methods for continuous collision detection aggravates this problem. Moreover, most of the earlier methods in collision detection do not make use of the temporal and spatial coherence of simulations.

In order to avoid all those problems mentioned above, we propose an event-based approach for continuous collision detection. The rationale is as follows: We all know that motion in the physical world is normally continuous. So, if an animation is discretized by very fine time intervals, a brute-force approach to the problem of updating BVHs and checking for collisions would need to do this at each of these points in time, possibly utilizing swept BVs between successive times. However, changes in the *combinatorial structure* of a BVH and, analogously, collisions only occur at *discrete* points in time. Therefore, we propose to utilize an event-based approach to remedy this unnecessary frequency of BVH updates and collision checks.

Exploiting this observation, we present the novel *kinetic separation list*, which enables continuous inter- and intra-object collision detection for arbitrary deformations such that checks between bounding volumes (BVs) and polygons are done only when necessary, i.e., when changes in the moving front really happen.

This way, the continuous problem of continuous collision detection is reduced to the discrete problem of determining exactly those points in time, where the combinatorial structure of our kinetic separation list changes.

We use the framework of kinetic data structures (KDS) for the design and the analysis of our algorithms. To use this framework, it is required that a *flightplan* is given for every vertex. This flightplan may change during the motion,

† e-mail:weller@in.tu-clausthal.de
‡ e-mail: zach@in.tu-clausthal.de

maybe by user interaction or by physical events (like collisions). Many deformations caused by simulations satisfy these constraints, like keyframe animations and many other animation schemes.

The kinetic separation list is based on the kinetic AABB tree. In contrast to conventional AABB trees, only the combinatorial structure of the hierarchy is stored instead of real vertex positions of the BVs. An update of the hierarchy is only necessary, if this combinatorial structure changes, which happens much less frequent than changes of vertex positions. However, the kinetic AABB tree utilize the temporal and spatial coherence only for the update of an individual hierarchy.

Our kinetic separation list extends the same principle to collision detection between pairs of objects. We maintain the combinatorial structure of a separation list of a conventional recursion tree.

As a natural consequence of this event-based approach, collisions are detected automatically in the right order, so there is no further ordering required like in many other approaches. Therefore, our kinetic separation list is well suited for collision response.

In the following, we will restrict our discussion to polygonal meshes, but it should become obvious that our data structures can, in principle, handle all objects for which we can build a bounding volume hierarchy, including polygon soups, point clouds, and NURBS models. Our algorithms are even flexible enough for handling insertions or deletions of vertices or edges in the mesh during run-time.

2. Related Work

Many methods using bounding volume hierarchies have been developed for collision detection of rigid bodies and have also been adopted for deformable objects, including axis-aligned bounding volumes (AABBs) [vdB97, Pro97], *k*-Dops [KHM*98], OBBs [GLM96] and spheres [PG95]. Since the objects deform, the hierarchies must be updated regularly and the cost of these updates can be high. Van den Bergen [vdB97] showed that updating is about ten times faster compared to a complete rebuild of an AABB hierarchy, and as long as the topology of the object is conserved, there is no significant performance loss in the collision check compared to rebuilding.

Several techniques to speed up the updates during each time step were proposed, including top-down, bottom-up updates and hybrid strategies [Ber98].
Mezger et al [MKE03] accelerated the update by omitting the update process for several time steps. Therefore, the BVs are inflated by a certain distance, and as long as the enclosed polygon does not move farther than this distance, the BV need not to be updated.

There also exist some stochastic methods [KZ03, Lin93] for deformable collision detection, but they can not guarantee to find exact collisions and even a single missed collision can result in an invalid simulation.

BVHs are also used to accelerate continuous collision detection. Therefore, the BVs enclose the volume described by a linear [RKC02, BFA02] or screw motion [KR03] within two successive time steps, but most approaches are restricted to rigid objects.

Knott and Pai [KP03] used hardware frame buffer operations to implement a ray-casting algorithm to detect static interferences between polyhedral objects. Therefore, the precision is constrained by the dimension of the viewport. Another hardware-based approach is given by Heidelberger et al [HTG04]. They use layered depth images with additional information on face orientation for the collision detection. Govindaraju et al [GKJ*05] use chromatic decompositions and the GPU to speed up the triangle tests using 2.5D overlap tests. However, for the broad phase, they use bottom-up updates of an AABB hierarchy and a simple swept volume algorithm. Furthermore, the algorithm is restricted to polygonal meshes with fixed connectivity but it can handle also self collisions of the objects.

Another approach for the special case of morphing objects [LAM03], where the objects are constructed by interpolating between some morphing targets, is to construct one BVH and fit this to the other morph targets, such that the corresponding nodes contain exactly the same vertices. During runtime, the current BVH can be constructed by interpolating the BVs. Fisher and Lin [FL01] use deformed distance fields for the collision detection between deformable objects.

James and Pai [JP04] introduced the BD tree which uses spheres as BVs and leads to a sub-linear-time algorithm for models which represent the deformation as linear superposition of precomputed displacement fields. However, the deformation is restricted to reduced deformable objects.

Most algorithms for continuous collision detection for deformable objects do not test for self collision [CMT02], just use simple heuristics [FGL03] or are to expensive [VT00].

There also exist first approaches of collision detection using the event-based kinetic data structures (KDS): Erickson et al [EGSZ99] describes a KDS for collision detection between two convex polygons by using a so-called boomerang hierarchy. Other approaches [ABG*02, Spe01] use pseudo triangles for a decomposition of the common exterior of a set of simple polygons for collision detection. However, all these approaches could not be extended to 3D space or are much too expensive in practice. Another kinetic approach is given by [CS06]. They use an event-based version of the sweep-and-prune algorithm for multi-body collision detection. But the data structure is restricted to rigid objects.

3. Kinetic Data Structures

In this section we give a quick recap of the kinetic data structure framework and its terminology.

The kinetic data structure framework (KDS) is a framework for designing and analyzing algorithms for objects (e.g. points, lines, polygons) in motion, which was invented

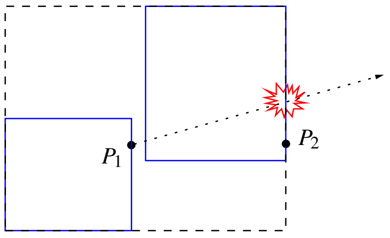

Figure 1: Kinetic AABB event: *When P_1, the maximum of the left child-box becomes larger along the x-axis than the overall maximum vertex P_2, an event will happen.*

by [BGH97]. The KDS framework leads to event-based algorithms that sample the state of different parts of the system only as often as necessary for a special task. This task can be, for example, the convex hull of a set of moving points and it is called the *attribute* of the KDS.

A KDS consists of a set of elementary conditions, called *certificates*, which prove altogether the correctness of the attribute. Those certificates can fail as a result of the motion of the objects. These certificate failures, the so-called *events*, are placed in an *event-queue*, ordered according to their earliest failure time. If the attribute changes at the time of an event, the event is called *external*, otherwise the event is called *internal*. Thus, sampling of time is not fixed, but determined by the failures of some conditions.

The quality of a KDS is measured by four criteria: A good KDS is *compact*, if it requires only little space, it is *responsive*, if we can update it quickly in case of a certificate failure. It is called *local*, if one object is involved in not too many events. This guarantees that we can adjust changes in the flighplan of the objects quickly. And finally, a KDS is *efficient*, if the overhead of internal events with respect to external events is reasonable.

In case of the kinetic AABB tree, the objects are a set of m polygons with n vertices; in the case of the kinetic separation list, they are a pair of BVs or a pair of polygons. Every vertex p_i has a flightplan $f_{p_i}(t)$. This might be a chain of line segments in the case of a keyframe animation or algebraic motions in the case of physically-based simulations. The flightplan is assumed to use $O(1)$ space and the intersection between two flightplans can be computed in $O(1)$ time. The flightplan may change during simulation by user interaction or physical phenomena, including collisions. In this case, we have to update all events the vertex is involved in.

The attribute is, in the case of the kinetic AABB tree, a valid BVH for a set of moving polygons. An event will happen, when a vertex moves out of its BV. In the case of the kinetic separation list, the attribute is a valid separation list, i.e., a list of non overlapping BVs in the traversal tree. An event will happen, if two BVs in the traversal tree will overlap, or if their fathers does not overlap anymore.

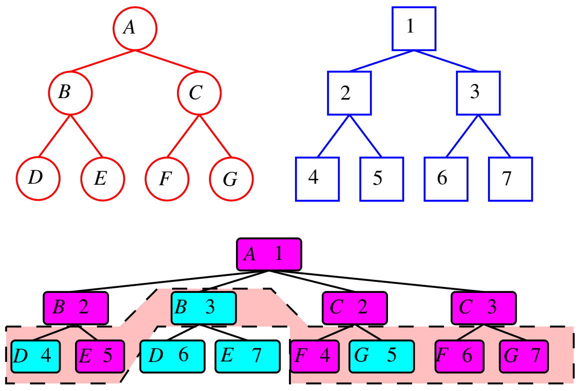

Figure 2: *This figure shows the complete traversal tree of two given BVHs. The overlapping nodes are colored red, the non overlapping nodes are colored blue. When we perform a collision check, we get a BVTT. Those BV pairs, where the traversal stops, build a list in this tree. We call it the* separation list. *This list consists of inner nodes, whose BVs do not overlap (B, 3), leaf nodes, where the BVs are leaves in the BVH that do not overlap (G, 5) and finally non-overlapping leaf nodes which contain leaves of the BVHs which do overlap (F, 6).*

4. The Kinetic AABB-Tree

In this section, we give a short recap of the kinetization of the well-known AABB tree [ZW06b].

We build the tree by any algorithm that can build a static BVH. However, instead of storing the actual extends of the AABBs with the nodes, we store references to those points, that determine the bounding box. For the theoretical runtime analysis, we assume that the height of the BVH is logarithmic.

After building the hierarchy, we compute the initial set of events. Basically, an event will happen, if a vertex becomes greater or smaller than the current respective extent of its box, which is only stored in the form of a reference to another vertex (see Fig. 1).

During runtime, we just have to process all those events in the event queue with a timestamp smaller than the current query time. When an event happens, we simply have to replace the old maximum or minimum along the axis, with the new one, and compute a new event for this BV. In addition, we have to propagate this change possibly to the upper BVs in the BVH.

In [ZW06b] we showed, that our kinetic AABB tree is compact, local, responsive, and efficient. Furthermore, the BVH is valid at every point of time, not only at the query times as is the case with most other update algorithms,

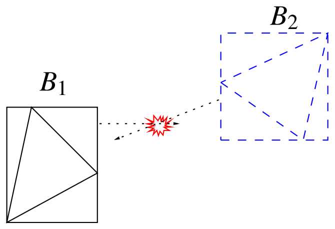

Figure 3: *If the BVs move so that they begin to overlap, we get an BV-overlap event.*

like bottom-up or top-down approaches. Moreover, the total number of events is bounded by nearly $O(n \log n)$ [†].

For a detailed description of the kinetic AABB, we would like to refer the interested reader to [ZW06b].

5. The Kinetic Separation List

So far, the kinetic AABB tree utilizes the temporal and, thus, combinatorial coherence only for the updates of individual hierarchies. In this section, we will describe a novel KDS specifically for detecting collisions between pairs of objects.

5.1. Kinetization

Our so-called *kinetic separation list* builds on the kinetic AABB tree and utilizes an idea described in [CL99, PCLM95] for rigid bodies. Given two kinetic AABB trees of two objects O_1 and O_2, we traverse them once for the initialization of the kinetic incremental collision detection. Thereby, we get a list, the so-called *separation list*, of overlapping BVs in the BV test tree (BVTT) (see Fig. 2). We call the pairs of BVs in the separation list *nodes*. This list contains three different kinds of nodes: Those which contain BVs that do not overlap (we will call them the *inner nodes*), leaves in the BVTT, where the BV pairs do not overlap (the *non-overlapping leaves*), and finally, leaf nodes in the BVTT that contain pairs of overlapping BVs, the so called *overlapping leaves*.

During run-time, this list configuration changes at discrete points in time, when one of the following events happens:

BV-overlap event: This event happens, when the pair of BVs of a node in the separation list which did not overlap so far, do overlap now. Thus, this event can happen only at inner nodes and non-overlapping leaves (see Fig. 3).

Fathers-do-not-overlap event: This event happens, if the BVs of a father of an inner node or a non-overlapping leaf in the BVTT do not overlap anymore (see Fig. 4). This could be inner nodes or non-overlapping leaves.

[†] The exact bound for the number of events is $O(n \log n \log^* n)$. But $\log^* n$ could be regarded as constant for all reasonable cases.

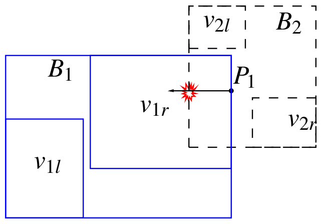

Figure 4: *When the fathers B_1 and B_2 of the BVs v_{1r}, v_{1l}, v_{2r} and v_{2l} do not overlap anymore, we get a fathers-do-not-overlap event.*

Leaves-do-not-overlap event: The fathers-do-not-overlap event cannot occur to overlapping leaves, because if their fathers do not overlap, then the leaves cannot overlap in the first place. Therefore, we introduce the leaves-do-not-overlap event.

Polygons-collide event: A collision between two triangles can only happen in overlapping leaves. If a non-overlapping leaf turns into an overlapping leaf, we have to compute the collision time and insert an adequate event into the event queue.

BV-change event: Finally, we need an event that remarks changes of the BV hierarchies. This event is somewhat comparable to flightplan updates of the kinetic AABB tree, but it is not exactly the same: This is, because an object in the separation list is composed of two BVs of different objects O_1 and O_2 and the flightplans are attributes of the vertices of only one single object. Therefore, not every flightplan update of an object affects the separation list (see Fig. 6).

In addition, a BV-change event happens, if the combinatorial structure of a BV in the separation list changes. Since we use kinetic AABB trees as BVH for the objects, this can happen only if a tree event or a leaf event in the BVH of an object happens. Surely, not all events cause changes at the separation list.

Assuming that the BVs of the object do not overlap at the beginning of the simulation, the separation list only consists of one node, which contains the root BVs of the two hierarchies.

During run-time, we have to update the separation list every time one of the above event happens according to the following cases:

BV-overlap event: Let K be the inner node with BVs V_1 of object O_1 and V_2 of object O_2. Here, we need to distinguish two cases:

- Node K is inner node: In order to keep the separation list valid after the event happened, we have to delete K from it and insert the child nodes from the BVTT instead. This means, if V_1 has the children V_{1L} and V_{1R}, and V_2 has the children V_{2L} and V_{2R} we have to put 4 new nodes, namely (V_{1L}, V_{2L}), (V_{1L}, V_{2R}), (V_{1R}, V_{2L}) and (V_{1R}, V_{2R}) into the list. Then we have to compute the next time point t, when (V_1, V_2) do not overlap. Furthermore, we have to compute

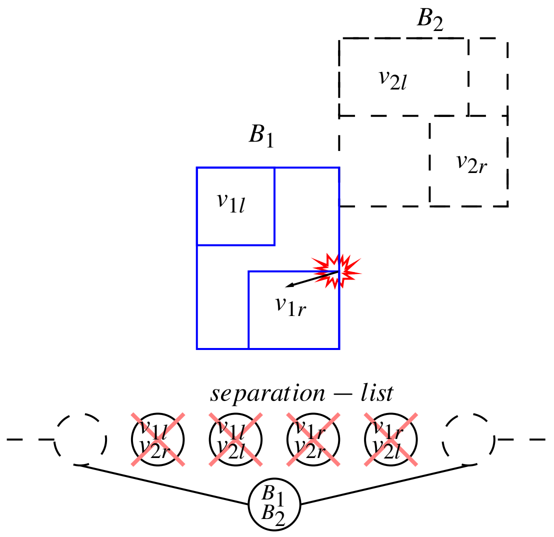

Figure 5: *If a fathers-do-not-overlap event happens, that means B_1 and B_2 do not overlap anymore. Thus, we have to remove their child BVs from the separation list and insert the new node (B_1, B_2) into it.*

the times t_i for the new nodes, when they will overlap. If $t_i < t$ we put a BV-overlap event in the queue, otherwise a father-do-not-overlap event.

- Node K is a not overlapping leaf: In this case we just have to turn the node into an overlapping leaf and compute the next leaves-do-not-overlap event (Fig. 7).

Fathers-do-not-overlap event: In this case, we have to delete the corresponding node from the separation list, and insert its father from the BVTT instead. Furthermore, we have to compute the new fathers-do-not-overlap event and BV-overlap event for the new node and insert the one which will happen first into the event queue (see Fig. 5).

Leaves-do-not-overlap event: If such an event happens, we have to turn the overlapping leaf into a non-overlapping leaf, and compute either a new fathers-do-not-overlap event or a BV-overlap event and put it into the event queue.

Polygons-collide event: A polygons-collide event does not change the structure of the separation list. Such an event must be handled by the collision response. But after the collision response, we have to compute the next polygons-collide event.

Note, that the polygons-collide events are reported in the correct order to the collision response module, this means, that that pair of polygons which collides first is also reported first. There is no other sorting required as it is by normal bottom-up strategies if we want to handle the first collision between two frames foremost.

BV-change event: If something in a BV in the separation list changes, e.g., the fligthplan of a vertex or the maximum or minimum vertex of a BV, then we have to recompute all events the BV is involved in.

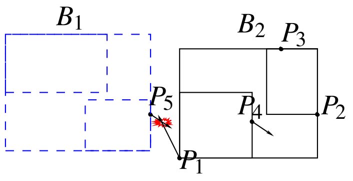

Figure 6: *If the flightplan of P_4 changes, this has no effect on the separation list, and thus, no BV-change event will happen due to this change.*

5.2. Analysis

For measuring the theoretical performance of our algorithm we use the four criteria of quality given for every KDS. First, we have to define the "validity" of a separation list:

Definition 1 We call a separation list "valid", if it contains exactly the non-overlapping nodes that are direct children of overlapping nodes in the BVTT plus the overlapping leaves.

Theorem 1 Our kinetic separation list is compact, local, responsive and efficient. Furthermore, we maintain a valid separation list at every point in time, if we update it as described above.

In order to prove the first part of the theorem, we assume, w.l.o.g, that both objects O_1 and O_2 have the same number of vertices n and the same number of polygons m.

In the worst case, it is possible that every polygon of object O_1 collides with every polygon of O_2. However, this will not happen in real world application. Thus, it is a reasonable assumption, that one polygon can collide with only $O(1)$ polygons of the other object. We will show the proof for both, the worst and the practical case:

Compactness: In order to evaluate the compactness, we have to define the attribute we are interested in. In the case of the kinetic incremental collision detection, this is the separation list. Thus, the size of a proof of correctness of the attribute may have size $O(n^2)$ in the worst case and $O(n)$ in the practical case.

For every node in the separation list, we store one event in the event queue, which will be at most $O(n^2)$ in the worst, respectively $O(n)$ in the practical case in total.

Furthermore, for every BV we have to store the nodes in the separation list in which it is participating, which could be at most $O(n^2)$ in the worst case or rather $O(n)$ in the practical case, too. Summarizing, the storage does not exceed the asymptotic size of the proof of correctness and thus, the data structure is compact.

Responsiveness: We will show the responsiveness for the four kinds of events separately:

- Leaves-do-not-overlap event: The structure of the separation list does not change if such an event happens. We just have to declare the node as not overlapping leaf and compute a new event which costs time $O(1)$, and the insertion into the event queue of the new event could be done in $O(\log n)$.

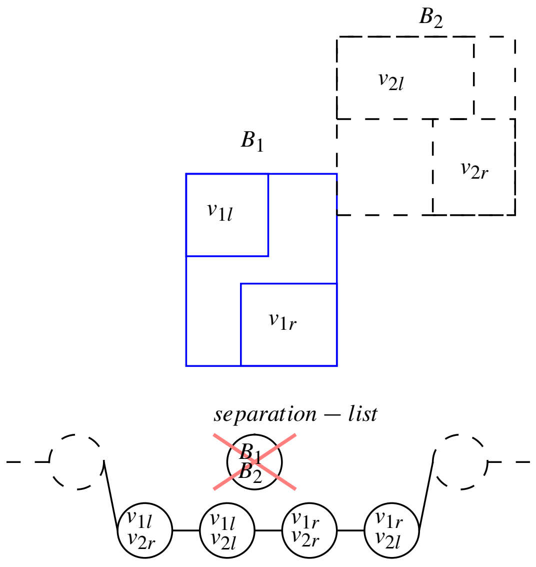

Figure 7: *If the BVs B_1 and B_2 overlap due to an BV-overlap event, we have to remove the corresponding node from the separation list and insert the pairs of their child-BVs v_{1r}, v_{1l}, v_{2r} and v_{2l}*

- BV-overlap event: The insertion of a new node into the separation list and deletion of the old node needs time $O(\log n)$. In addition we have to delete the links from the old BV to the old node in the separation list and insert the new ones. If we organise this lists of links as AVL-tree, we get costs of $O(\log n)$.
- Fathers-do-not-overlap event: The deletion of nodes and events takes time of $O(\log n)$ again.
- BV-change event: When this kind of event happens, the structure of our separation list does not change. We just have to recompute the event of the affected node. The insertion and deletion of an event costs $O(\log n)$.

  Overall, our data structure is responsive in all cases.

Efficiency: To determine the efficiency is bit more complicated, because it is not immediately obvious which events we should treat as inner and which as outer events. Clearly, leaves-do-not-overlap events, BV-overlap events and fathers-do-not-overlap event cause a real change of the attribute, the separation list, so these events are outer events. But classifying the BV-change events is more difficult. Those which happen due to flightplan updates clearly do not count, because they happen due to user interactions and could not be counted in advance. But there are also BV-change events which happen due to changes of the BV hierarchies, and they could be regarded as inner events.

Since we use the kinetic AABB tree, there are at most $O(n \log n)$ events in one single BVH. One BV could be involved in n nodes in the separation list. So there are $O(n^2 \log n)$ inner events in the worst case.

On the other hand, there may be $\Omega(n^2)$ outer events and thus the KDS is still responsive, even if we treat the BV-change events as inner events.

In the reasonable case we have at most $O(n \log n)$ inner events from the kinetic AABB tree and $O(n)$ outer events in the separation list and therefore our KDS is also responsive in this case.

Locality: We also have to be careful when showing the locality of our data structure. The objects of our kinetic data structure, are the nodes in the separation list, not the single BVs in the kinetic AABB hierarchies. Each node is involved in only $O(1)$ events and thus, our kinetic separation list is trivially *local*.

Otherwise, if the flightplan of one single BV changes, this could cause $O(n)$ BV-change events in the kinetic separation list, because one BV could participate $O(n)$ nodes in the worst case. However, this is compared to $O(n^2)$ total nodes in our kinetic separation list small and moreover, in the reasonable case there are at most $O(1)$ nodes affected by a flightplan update. Summarized, our kinetic separation list can be updated efficiently in all cases if a flightplan update happens.

Due to its length, the proof of the second part of the theorem is omitted here, but it can be found at [ZW06a].

Overall, our data structure fulfills all quality criteria of a kinetic data structure both in the practical and in the worst case. Our experiments in the last section show, that it performs very well in practical cases and that the running time is up to 50 times faster compared to other approaches.

Algorithm 1: Simulation Loop

```
while simulation runs do
    determine time $t$ of next rendering;
    $e \longleftarrow$ min event in event queue;
    while $e.timestamp < t$ do
        processEvent($e$);
        if $e$ = Polygons-Collide event then
            collision response;
        $e \longleftarrow$ min event in event queue;
    render scene;
```

5.3. Self-Collision Detection

BVHs are also used for self-collision detection. In general, collisions and self-collisions are detected in the same way. If two different objects are tested for collisions, their BVHs are checked against each other. Analogously, self-collisions of an object are performed by testing one BVH against itself. The main problem which arises when using this method in combination with discrete time sampling algorithms, is the problem of adjacency. E.g., the BVs of adjacent polygons allways overlap. Therefore, approaches which are not using temporal and spatial coherence, has to descent from the root of the BVTT down to all neighbours of a polygon at every query time. This are $O(n \log n)$ BV overlap tests, even if there are no colliding polygons.

Our kinetic separation list avoid the problem of adjacency. For self collision tests, we also test the BVH against itself, but we do this only one time for the initialisation. During run-time, pairs of adjacent BVs stay all the time in the separation list and their parents will never be checked for collision as it is with most other approaches.

6. Implementation Details

In this section, we describe the implementation details of our kinetic separation list, which differs in several points from the basic algorithms described above. Algorithm 1 shows the basic simulation loop.

6.1. Kinetic Separation List

First of all, it is not necessary to store the separation list explicitly. Instead, it is sufficient to link only the two colliding BVs in the kinetic AABB tree. Therefore, we use a simple list for every BV in the kinetic AABB hierarchy and store pointers to the colliding BVs in the other hierarchy. It is sufficient, to use a list, even if we have to delete or insert some pointers when an event appears, because in real world scenarios the degree of vertices is bounded and thus, a single BV does not collide with too many other BVs in the BVTT.

Moreover, if a fathers-do-not-overlap event happens, we do not simply add the father of the affected BVs into our separation list, because in most cases, the fathers of the fathers do not overlap either. Instead, we ascend in the hierarchy to the highest pair of BVs which does not overlap and then delete all its children that are in the separation list. Note, that the data structure is not responsive anymore if we proceed like this, because in the worst case, we have cost of $O(n^2)$ for one single event. However, if we simply proceed as described in the section before, we would have to process $O(n^2)$ events. Thus, the overall complexity is still the same. Equivalently, we do not insert just the children if a BV-overlap event happens. Instead, we descent directly to the deepest non overlapping-nodes in the BVHs.

As event queue, we use a Fibonacci heap. With this data structure, we can efficiently insert, delete and update events.

6.2. Event Calculation

The calculation of the events depends on the motion of the objects. At first we assume a linear motion of the vertices.

In the kinetic AABB tree, we get an event if a vertex P become larger than another vertex Q along some axis. Therefore, the computation of an event corresponds to line intersection tests in 2D.

More precisely, assume two vertices P and Q with velocity vectors p and q respectively, and at time t, we have $P_x(t) < Q_x(t)$. In order to get the next point of time $\bar{t}$ when P becomes larger than Q along the x-axis, we get $\bar{t} = \frac{Q_x(t)-P_x(t)}{p_x-q_x}$.

If $\bar{t} < 0$, there will be no event.

In the kinetic separation list, we get events if two BVs begin to overlap or do not overlap anymore.

Assume two BVs A and B with extreme points P^A_{imax} and P^B_{imax}, respectively, and minimum points P^A_{imin} and P^B_{imax}, respectively, with $i \in \{x,y,z\}$ at time t.

Algorithm 2: Event Calculation

```
Compute $f$ with $l \cdot f \le t \le l \cdot (f+1)$;
$\bar{t} = l \cdot (f+1)$;
**while** $t > l \cdot f$ **do**
    $\overline{p} = P_{l\cdot(f+1)} - P_{l\cdot f}$;
    $\overline{q} = Q_{l\cdot(f+1)} - Q_{l\cdot f}$;
    $p_f = \frac{\overline{p}}{l}$;
    $q_f = \frac{\overline{q}}{l}$;
    Compute $\bar{t}$ when $P$ gets larger than $Q$;
    $f = f+1$;
```

There are two different cases for events:

- Assume A and B overlap at time t and we want to get the point of time $\bar{t}$ when they do not overlap anymore. Surely, A and B do not overlap $\Leftrightarrow$ there exists an axis $i \in \{x,y,z\}$ with $P^A_{imax}(\bar{t}) < P^B_{imin}(\bar{t})$ or $P^B_{imax}(\bar{t}) < P^A_{imin}(\bar{t})$.
  Thus, we have to compute the points of time $\overline{t_i}$ for every axis $i \in \{x,y,z\}$ when P^A_{imax} becomes smaller than P^B_{imin} and P^B_{imax} becomes smaller than P^A_{imin}. We generate an event for the minimum of these $\overline{t_i}$.
- If A and B do not overlap at time t, we have to look for the time $\bar{t}$, when they overlap. A and B overlap $\Leftrightarrow P^A_{imax}(\bar{t}) \ge P^B_{imin}(\bar{t})$ and $P^B_{imax}(\bar{t}) \ge P^A_{imin}(\bar{t})$ for all axes $i \in \{x,y,z\}$.
  Thus we have to compute the points of time $\overline{t_i}$ for all $i \in \{x,y,z\}$, when P^A_{imin} becomes smaller than P^B_{imax} and P^B_{imin} gets smaller than P^A_{imax} too. We generate an event for the maximum of the $\overline{t_i}$.

We tested our algorithms with keyframe animations. Between two keyframes, we interpolated linearly. Therefore, we get paths of line segments as motion of the vertices.

Assume k keyframes $K_0, \ldots, K_k$. Let l be the number of interpolated frames between two keyframes. We want to compute, for the vertices P and Q with positions $P(t)$ and $Q(t)$, respectively, when the next event between these points will happen, i.e., when P will become larger along the x-axis than Q.

Therefore, we first have to determine the actual keyframe K_f with $l \cdot f \le t \le l \cdot (f+1)$. We get the actual velocity p_f and q_f for the two vertices by $p_f = P(l \cdot (f+1)) - P(l \cdot f)$ and $q_f = Q(l \cdot (f+1)) - Q(l \cdot f)$.

Now, we can compute time $\bar{t}$ when P gets larger than Q, as described in the previous section. If $\bar{t} \le m \cdot (f+1)$ we get the event for P and Q. But if $\bar{t} > l \cdot (f+1)$ we have to look at the next keyframe wether the paths of P and Q intersects, and so on (see Algorithm 2). Thus, we have to compute k line intersections for one single event in the worst case.

7. Results

We implemented our algorithms in C++ and tested the performance on a PC with a 3 GHz Pentium IV. For the continuous triangle test we used the method proposed in [ES99].

As benchmark, we used three different scenes of keyframe animations.

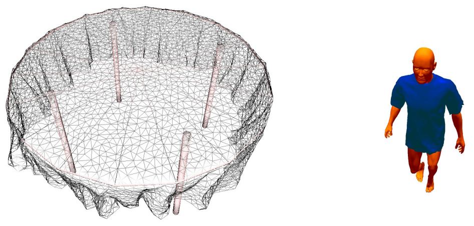
Figure 8: *A tablecloth falling down on a table and a virtual character animation including the simulation of a shirt.*

The first scene shows a tablecloth falling on a table. This scene contains both fairly static geometry on the top of the table and geometry with large deformations. We used this scene with several resolutions of the cloth, ranging from 2k to 16k faces (Fig. 8). The second scene shows a single swirling cloth in resolutions of 4K to 33K deforming polygons (Fig. 9). We used this scene in order to stress our algorithm: It contains very heavy deformation of the cloth and many self collisions. The last scenario show typical cloth animation: A male avatar with a shirt in resolutions from 35k to 90k deforming triangles walking around (Fig. 8).

We compared the performance of our algorithms with a classical swept volume algorithm for continuous collision detection: We updated the hierarchy with a bottom-up updating strategy. For the proper collision check, we constructed an AABB which enclosed the BVs at the beginning and the end of the frame.

First, we considered the number of events in our kinetic separation list compared to the number of checks the swept volume algorithm has to perform. In the high-resolution tablecloth scene, we have about 500 events per frame with our kinetic data structure compared to several tens of thousands collision checks with the swept volume. Since the computation costs for an event are relatively high, this results in an overall speed-up of about factor 50 for updating the kinetic separation list. The number of events rises nearly linearly with the number of polygons (see Fig. 11).

In the cloth animation scenes with the male avatar and the tablecloth, the gain of our kinetic data structures is highest, because the objects undergo less deformation than the swirling cloth, and thus we have to compute and handle less events. In these scenarios, we see a performance gain of a factor up to 50 compared to the swept volume algorithm (Fig. 10). This factor would increase even further, if the number of interpolated frames between two keyframes were increased. This is because the performance of the event-based kinetic data structures only depends on the number of keyframes and not on the total length of the scene or the number of collision checks.

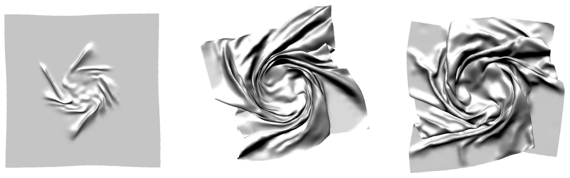
Figure 9: *The swirling cloth animation scene.*

Overall, the kinetic separation list performs best, and the running time of the updating operations is independent from the sampling frequency. This means, for example, if we want to render a scene in slow motion, maybe ten times slower, the overall costs for updating the hierarchies and the collision detection are still the same, while they increase for the swept volume algorithm by a factor of ten.

Moreover, the collisions are reported in the right order with our kinetic separation list. This is important for a correct collision response scheme. The collisions in the swept volume algorithms are reported in random order. If we would sort them, the gain by our algorithms would even increase.

8. Conclusions and Future Work

We introduced a novel data structure, the kinetic separation list, for continuous inter- and intra-collision detection between deformable objects, i.e., pairwise and self collision detection. The algorithm gains its efficiency from the event-based approach.

It contains a discrete event-based part, which updates only the combinatorial changes in the BVH, and a continuous part, which needs to compute only the time of future events after such a combinatorial change. Our algorithm is particularly well-suited for animations where the deformation cannot be restricted in some way (such as bounded deformations).

We presented a theoretical and experimental analysis showing that our new algorithm is fast and efficient both theoretically and in practice. We used the kinetic data structure framework to analyze our algorithm and showed that our data structure fulfills all quality criteria for good KDS. Moreover, our kinetic separation list is perfectly qualified for a stable collision response, because it naturally delivers the collisions ordered by time to the collision response module.

In practically relevant cloth animation scenes, our kinetic data structures can find collisions and self-collisions more than 50 times faster than a swept volumes approach. Even in scenarios with heavy deformations of the objects we observed a significant gain by our algorithm.

In the future, we plan to use our algorithms with other kinds of motion, including physically-based simulations and other animation schemes. In addition, it should be straightforward to extend our novel algorithms to other primitives such as NURBS or point clouds.

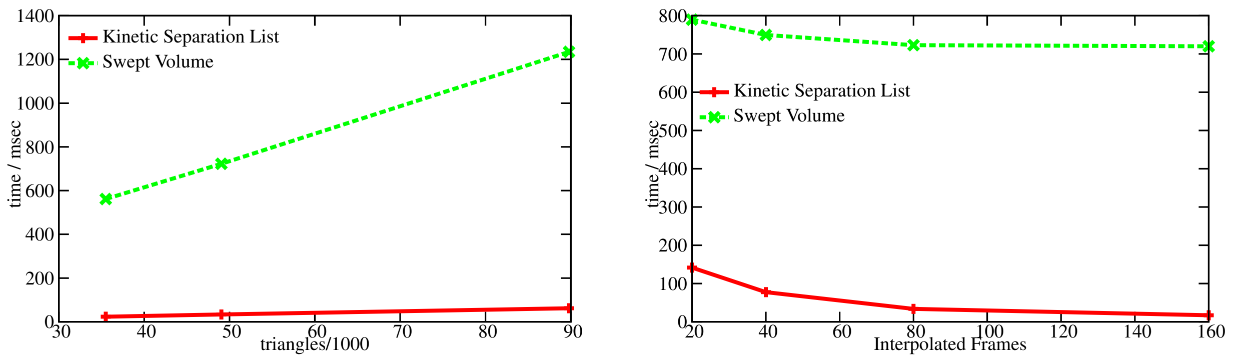

Figure 10: *The left diagram shows the average total time for updating the hierarchies and performing the inter- and intra-collision detection in the male avatar scene. We have an overall gain of about a factor of 20 with our kinetic separation list. The table on the right shows the update time for the same scene in the resolution of 49K triangles, depending on the number of interpolated frames in-between two key frames. Since the number of events only depends on the number of key frames and not on the number of interpolated frames, the average update time decreases if we increase the total number of frames.*

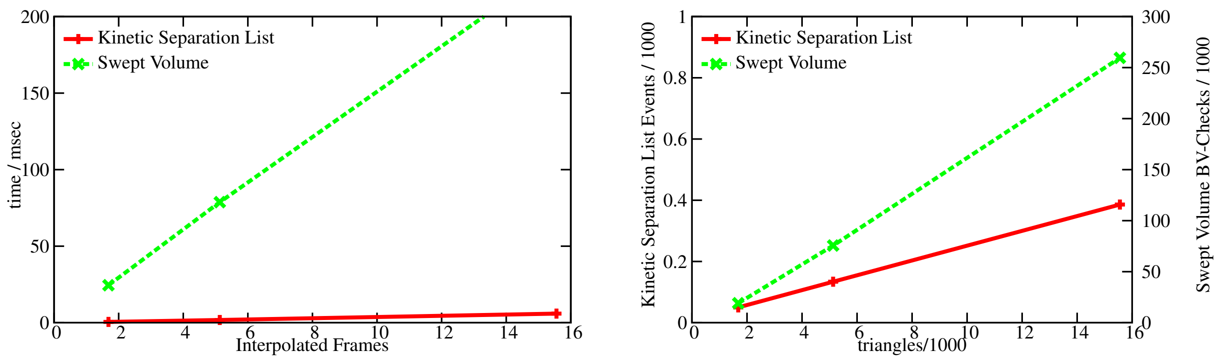

Figure 11: *The left diagram shows the total time, this means updating the hierarchies and the time for the collision check including self collision for the tablecloth scene. The gain of our kinetic data structures is about a factor of 50. The right diagram shows the number of events in our kinetic data structure compared to the number of collision checks we have to perform with the swept volume algorithm. The number of events is significantly smaller. Note the different scales.*

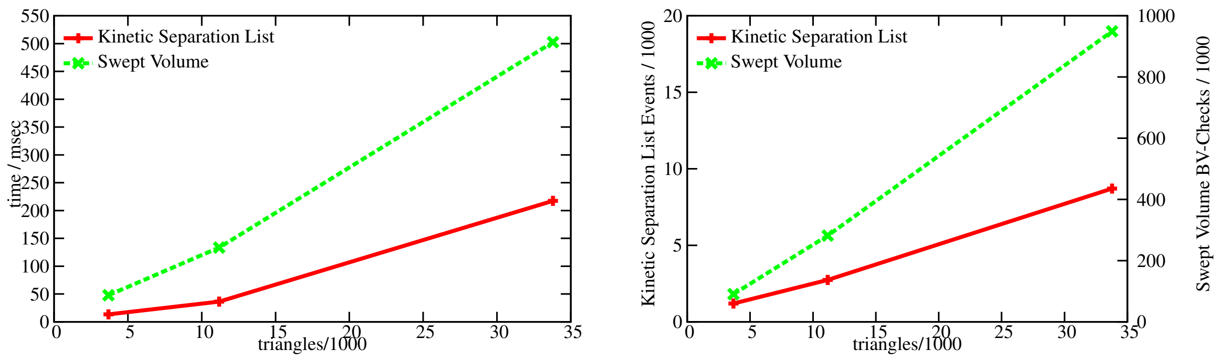

Figure 12: *The left diagram shows the time for updating and self collision check in the swirling cloth scene. Even in this worst case scenario for our algorithm, we have a gain of a factor about two for our kinetic data structure. This depends on the higher number of events in this scenario, which is shown in the right diagram. Again, note the different scales.*

References

[ABG*02] AGARWAL P. K., BASCH J., GUIBAS L. J., HERSHBERGER J., ZHANG L.: Deformable Free-Space Tilings for Kinetic Collision Detection. *I. J. Robotic Res. 21*, 3 (2002), 179 – 198.

[Ber98] BERGEN G. V. D.: Efficient Collision Detection of Complex Deformable Models using AABB Trees, Dec. 07 1998.

[BFA02] BRIDSON R., FEDKIW R., ANDERSON J.: Robust treatment of collisions, contact and friction for cloth animation, 2002.

[BGH97] BASCH, GUIBAS, HERSHBERGER: Data Structures for Mobile Data. In *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)* (1997).

[CL99] CHEN J.-S., LI T.-Y.: Incremental 3D Collision Detection with Hierarchical Data Structures, Nov. 22 1999.

[CMT02] CORDIER F., MAGNENAT-THALMANN N.: Real-time animation of dressed virtual humans. *Comput. Graph. Forum 21*, 3 (2002).

[CS06] COMING D., STAADT O. G.: Kinetic sweep and prune for multi-body continuous motion. *Computers Graphics 30*, 3 (May 2006).

[EGSZ99] ERICKSON J., GUIBAS L. J., STOLFI J., ZHANG L.: Separation-sensitive collision detection for convex objects. In *SODA '99: Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms* (Philadelphia, PA, USA, 1999), Society for Industrial and Applied Mathematics, pp. 327 – 336.

[ES99] ECKSTEIN J., SCHÖMER E.: Dynamic collision detection in virtual reality applications. In *WSCG'99 Conference Proceedings* (1999), Skala V., (Ed.).

[FGL03] FUHRMANN A., GROSS C., LUCKAS V.: Interactive animation of cloth including self collision detection. In *WSCG* (2003).

[FL01] FISHER S., LIN M. C.: Deformed distance fields for simulation of non-penetrating flexible bodies. In *Proceedings of the Eurographic workshop on Computer animation and simulation* (New York, NY, USA, 2001), Springer-Verlag New York, Inc., pp. 99–111.

[GKJ*05] GOVINDARAJU N. K., KNOTT D., JAIN N., KABUL I., TAMSTORF R., GAYLE R., LIN M. C., MANOCHA D.: Interactive collision detection between deformable models using chromatic decomposition. *ACM Trans. Graph. 24*, 3 (2005), 991–999.

[GLM96] GOTTSCHALK S., LIN M. C., MANOCHA D.: OBBTree: A hierarchical structure for rapid interference detection. *Computer Graphics 30*, Annual Conference Series (1996), 171–180.

[HTG04] HEIDELBERGER B., TESCHNER M., GROSS M. H.: Detection of collisions and self-collisions using image-space techniques. In *WSCG* (2004), pp. 145–152.

[JP04] JAMES D., PAI D.: Bd-tree: Output-sensitive collision detection for reduced deformable models. *ACM Transactions on Graphics (SIGGRAPH 2004) 23*, 3 (August 2004).

[KHM*98] KLOSOWSKI J. T., HELD M., MITCHELL J. S. B., SOWIZRAL H., ZIKAN K.: Efficient collision detection using bounding volume hierarchies of *k*-DOPs. *IEEE Transactions on Visualization and Computer Graphics 4*, 1 (/1998), 21–36.

[KP03] KNOTT D., PAI D. K.: Cinder: Collision and interference detection in real-time using graphics hardware. In *Graphics Interface* (2003), pp. 73–80.

[KR03] KIM B., ROSSIGNAC J.: Collision prediction for polyhedra under screw motions. In *SM '03: Proceedings of the eighth ACM symposium on Solid modeling and applications* (New York, NY, USA, 2003), ACM Press, pp. 4–10.

[KZ03] KLEIN J., ZACHMANN G.: Adb-trees: Controlling the error of time-critical collision detection. In *Vision, Modeling and Visualisation 2003* (November 2003), Ertl T., Girod B., Greiner G., Niemann H., Seidel H.-P., Steinbach E.,, Westermann R., (Eds.), Akademische Verlagsgesellschaft Aka GmbH, Berlin, pp. 37–46.

[LAM03] LARSSON T., AKENINE-MOELLER T.: Efficient collision detection for models deformed by morphing. *The Visual Computer 19*, 2-3 (June 2003), 164–174.

[Lin93] LIN M. C.: *Efficient collision detection for animation and robotics*. PhD thesis, 1993. Chair-John F. Canny.

[MKE03] MEZGER J., KIMMERLE S., ETZMUSS O.: Hierarchical Techniques in Collision Detection for Cloth Animation. *Journal of WSCG 11*, 2 (2003), 322–329.

[PCLM95] PONAMGI M., COHEN J., LIN M., MANOCHA D.: Incremental algorithms for collision detection between general solid models, 1995.

[PG95] PALMER I. J., GRIMSDALE R. L.: Collision detection for animation using sphere-trees. *Comput Graphics Forum 14*, 2 (June 1995), 105–116.

[Pro97] PROVOT X.: Collision and self-collision handling in cloth model dedicated to design garments. In *Proc. Graphics Interface '97* (1997), pp. 177 – 189.

[RKC02] REDON S., KHEDDAR A., COQUILLART S.: Fast continuous collision detection between rigid bodies, 2002.

[Spe01] SPECKMANN B.: *Kinetic Data Structures for Collision Detection*. PhD thesis, 2001.

[vdB97] VAN DEN BERGEN G.: Efficient collision detection of complex deformable models using aabb trees. *J. Graph. Tools 2*, 4 (1997), 1–13.

[VT00] VOLINO P., THALMANN N. M.: Accurate collision response on polygonal meshes. In *CA '00: Proceedings of the Computer Animation* (Washington, DC, USA, 2000), IEEE Computer Society, p. 154.

[ZW06a] ZACHMANN G., WELLER R.: *Kinetic Bounding Volume Hierarchies for Collision Detection of Deformable Objects*. Tech. Rep. IfI-06-02, TU-Clausthal, 2006.

[ZW06b] ZACHMANN G., WELLER R.: Kinetic bounding volume hierarchies for deformable objects. In *ACM International Conference on Virtual Reality Continuum and Its Applications (VRCIA)* (Hong Kong, China, 14–17 June 2006).