Kinetic Bounding Volume Hierarchies for Deformable Objects

Gabriel Zachmann* TU Clausthal

Rene Weller† TU Clausthal

Abstract

We present novel algorithms for updating bounding volume hierarchies of objects undergoing arbitrary deformations. Therefore, we introduce two new data structures, the kinetic AABB tree and the kinetic BoxTree.

The event-based approach of the kinetic data structures framework enables us to show that our algorithms are optimal in the number of updates. Moreover, we show a lower bound for the total number of BV updates, which is independent of the number of frames.

We used our kinetic bounding volume hierarchies for collision detection and performed a comparison with the classical bottom-up update method. The results show that our algorithms perform up to ten times faster in practically relevant scenarios.

CR Categories: I.3.5 [Computing Methodologies]: Computational Geometry and Object Modeling—Geometric algorithms, Object hierarchies; I.3.7 [Computing Methodologies]: Three-Dimensional Graphics and Realism—Animation, Virtual reality

1 Introduction

Bounding volume hierarchies (BVHs) for geometric objects are widely employed in many areas of computer science to accelerate geometric queries. Such acceleration data structures are used in computer graphics for ray-tracing, occlusion culling and collision detection, to name but a few; They are also used in other areas such as geographical databases, molecular simulation, or robotics. Usually, a bounding volume hierarchy is constructed in a pre-processing step which is suitable as long as the objects are rigid.

However, deformable objects play an important role, e.g. for creating virtual environments in medical applications, entertainment, and virtual prototyping [Teschner et al. 2005]. If the object deforms, the pre-processed hierarchy becomes invalid.

In order to still use this well-known method for deforming objects as well, it is necessary to update the hierarchies after the deformation happens.

Most current techniques do not make use of the temporal and spatial coherence of simulations and just update the hierarchy by brute-force at every time step or they simply restrict the kind of deformation in some way, in order to avoid the time consuming per-frame update of all bounding volumes (BVs).

On the one hand, we all know that motion in the physical world is normally continuous. So, if animation is discretized by very fine time intervals, a brute-force approach to the problem of updating BVHs would need to do this at each of these points in time. On the other hand, changes in the *combinatorial* structure of a BVH only occur at discrete points in time. Therefore, we propose to utilize an event-based approach to remedy this unnecessary frequency of BVH updates.

*e-mail: zach@in.tu-clausthal.de

†e-mail:weller@in.tu-clausthal.de

According to this observation, we present two algorithms to update hierarchies in a more sensitive way: we only make an update if it is necessary. In order to determine exactly when it is necessary, we use the framework of kinetic data structures (KDS). To use this kind of data structures, it is required that a *flightplan* is given for every vertex. This flightplan may change during the motion, maybe by user interaction or physical events (like collisions). Many deformations caused by simulations satisfy these constraints, like keyframe animations and many other animation schemes.

Beyond this, our algorithms can 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.

In the following, we first present a kinetization of a tree of axis aligned boundingboxes (AABBs) and show that the associated update algorithm is optimal in the number of BV updates (This means that every AABB hierarchy which performs less updates must be invalid at some point of time).

Moreover, we prove an asymptotic lower bound on the total number of update operations in the worst case which holds for every BVH updating strategy. This number is independent from the length of the animation sequence under certain conditions.

In order to reduce the number of update operations, we propose a kinetization of the BoxTree. A BoxTree is a special case of an AABB, where we store only two splitting axis per node. On account of this, we can reduce the overall number of events.

Finally, we present the results of a comparison to the running times of hierarchical collision detection based on our novel kinetic BVHs and conventional bottom-up updating, resp.

2 Related Work

Many methods using BVHs have been developed for collision detection of rigid bodies and have been also adopted for deformable objects, including axis-aligned bounding volumes (AABBs) [van den Bergen 1997; Provot 1997], k-Dops [Klosowski et al. 1998], OBBs [Gottschalk et al. 1996] and spheres [Palmer and Grimsdale 1995]. Since the objects deform, the hierarchies must be updated regularly and the cost of these updates can be high. [van den Bergen 1997] 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 [Bergen 1998]. [Mezger et al. 2003] 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 [Klein and Zachmann 2003; Lin 1993] 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.

[Knott and Pai 2003] 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. 2004]. They use layered depth images with additional information on face orientation for

the collision detection. Govindaraju et al [Govindaraju et al. 2005] 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. Furthermore, the algorithm is restricted to polygonal meshes with fixed connectivity. [Wong and Baciu 2005b] utilize the GPU for the intrinsic collision test for pairs of polygons. However, for the broad phase of the collision detection, there are still hierarchies or other techniques needed to reduce the number of candidate pairs.

[Lau et al. 2002] proposed a collision detection framework for deformable nurbs surfaces using AABB hierarchies. They reduce the number of updates by looking for special deformation regions. [Wong and Baciu 2005a] use a partitioning of a deformable surface into so called (π, β, I)-surfaces in order to prune noncolliding polygon pairs.

Another approach for the special case of morphing objects [Larsson and Akenine-Moeller 2003], 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 2001] use deformed distance fields for the collision detection between deformable objects.

[James and Pai 2004] 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.

There also exist first approaches of collision detection using the event-based kinetic data structures: [Erickson et al. 1999] describes a KDS for collision detection between two convex polygons by using a so-called boomerang hierarchy. [Agarwal et al. 2002] and [Speckmann 2001] developed a KDS using 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.

3 Overview of our Approach

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

The kinetic data structure framework is a framework for designing and analyzing algorithms for objects (e.g. points, lines, polygons) in motion which was invented by [Basch et al. 1997]. The KDS framework leads to event-based algorithms that samples the state of different parts of the system only as often as necessary for a special task. This task can be for example the computation of 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. This 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 failure of some certain conditions.

As an example we can assume the bounding box of a set of moving points in the plane. The bounding box is the attribute we are interested in. It is generated by four points $P^t_{\{max,min\},\{x,y\}}$ which have the maximum and minimum x- and y-values at a certain time point t. For every inner point P^t_i we have $P^t_i[x] < P^t_{max,x}[x]$, $P^t_i[y] < P^t_{max,x}[y]$, $P^t_i[x] > P^t_{min,x}[x]$ and $P^t_i[y] > P^t_{min,y}[y]$. These four simple inequations are the certificates in our example. If an inner point moves out if the bounding box due to its motion, e.g. $P^{t_2}_i[x] > P^{t_2}_{max,x}[x]$, this causes an external event at the point of time $t + \Delta t$ when $P^{t+\Delta t}_i[x] = P^{t+\Delta t}_{max,x}[x]$ (see Fig. 1). If $P^t_i[x] > P^t_j[x]$ and $P^{t_3}_i[x] < P^{t_3}_j[x]$ for points that are not in $P^t_{\{max,min\},\{x,y\}}$, this causes an internal event.

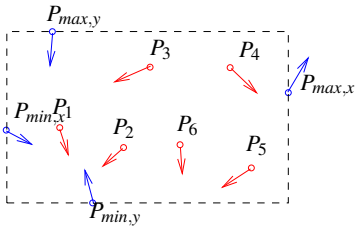

Figure 1: Assume a set of moving points in the plane. $P_{\{max,min\},\{x,y\}}$ for the current bounding volume of this points. At some time, P_5 will become smaller than $P_{min,y}$, this causes an event.

We measure the quality of a KDS 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.

Algorithm 1: Simulation Loop

```
while simulation runs do
    calc time t of next rendering
    e ← min events in event-queue
    while e.timestamp < t do
        processEvent(e)
        e ← min events in event-queue
    check for collisions
    render scene
```

In our case, the objects are a set of m polygons with n vertices. Every vertex p_i has a flightplan $f_{p_i}(t)$. This might be a chain of line segments in case of a keyframe animation or algebraic motions in 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 of a vertex 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 with.

The attribute is a valid BVH for a set of moving polygons. An event will happen, when a vertex moves out of its BV.

The kinetic data structures we will present have some properties in common, which will be described as follows.

Algorithm 2: Check{*BV a of object A, BV b object B*}

```
if overlap ( a, b ) then
    if a and b are leaves then
        test_primitives( a, b )
    else
        forall children a[i] of a do
            forall children b[j] of b do
                Check( a[i], b[j] )
else
    return
```

They all use an event-queue for which we use an AVL-Tree, because with this data structure we can insert and delete events as well as extract the minimum in time $O(\log k)$ where k is the total number of events.

Both algorithms run within the same framework for kinetic updates, which is explained in Algorithm 1.

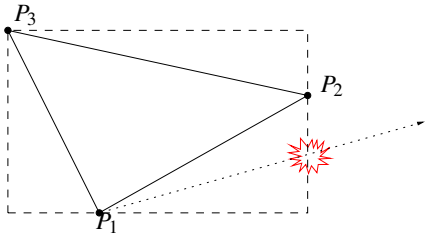

Figure 2: When P_1 becomes greater than the current maximum vertex P_2, a Leaf-Event will happen.

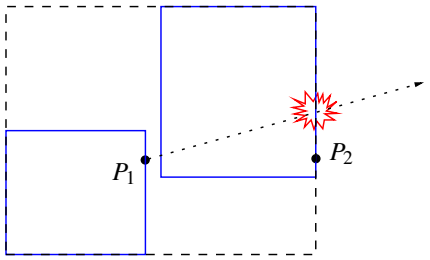

Figure 3: When P_1, the maximum of the left child-box becomes greater than the overall maximum vertex P_2, a Tree-Event will happen.

Furthermore, our algorithms use also the same procedure for the collision check (see Algorithm 2).

4 Kinetic AABB-Tree

In this section, we present a kinetization of the well known AABB tree. We build the tree by any algorithm which can be used for building static BVHs and store for every node of the tree the indices of these points that determine the bounding box. For our analysis of the algorithms it is only required that the height of the BVH is logarithmic in the number of polygons.

After building the hierarchy, we traverse the tree again to find the initial events.

There are three kinds of different events:

- *Leaf-Event*: Assume that P_1 realizes the BVs maximum along the x-axis. A leaf event happens, when the x-coord of one of the other points P_2 or P_3 becomes greater than $P_{1,x}$ (see Fig. 2).
- *Tree-Event*: Let K be an inner BV with its children K_l and K_r and $P_2 \in K_r$ is the current maximum of K on the x-axis. A Tree-Event happens when the maximum of K_l becomes greater than P_2 (see Fig. 3). Analogously Tree-Events are generated for the other axis and the minima. other axis and the minima.
- *Flightplan-Update-Event*: Every time a flightplan of a point changes, we get a Flightplan-Update-Event.

So after the initialization we have stored six events with each BV. In addition, we put the events in the event queue, sorted by timestamp.

During runtime, we perform an update according to Algorithm 1 before each collision check. In order to keep the BV hierarchy valid, we have to handle the events as follows:

- *Leaf-Event*: Assume in a leaf BV B, realized by the vertices P_1, P_2 and P_3, the maximum extend along the x-axis has been realized by P_2. With the current event, P_1 takes over, and becomes larger than $P_2[x]$. In order to maintain the validity of the BV hierarchy, in particular, we have to associate P_1 as the max x extent of B. In addition, we have to compute a new event. I.e, we have to compute all the intersections of the flightplans of all other vertices in B with P_1 in the xt-plane. An event for the pair with the earliest intersection time is inserted into the event queue (Fig. 4).

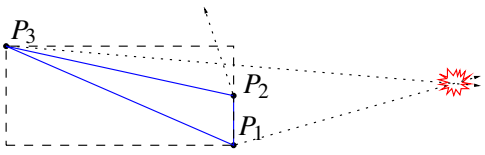

Figure 4: To keep the hierarchy valid when a Leaf-Event happens, we have to replace the old maximum P_2 by the new maximum P_1, and compute the time, when one of the other vertices of the polygon, P_2 or P_3 will become greater than P_1. In this example this will be P_3.

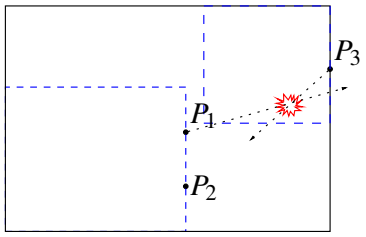

Figure 5: In Addition, we have to propagate the change to upper BVs in the hierarchy after a Tree-Event happend. After replacing the P_2 old maximum by the new maximum P_1 in the lower left box, we have to compute the event between P_1 and P_3, which is the maximum of the father.

But that is not necessarily sufficient for keeping the BVH valid. In addition, we have to propagate this change in the BVH to the upper nodes. Assume B be the right son of its father V, so we have check whether P_2 had been the maximum of V too. In this case, we have to replace P_2 by the new maximum P_1. In addition, the corresponding event of V is not valid any more because it was computed with P_2. So we have to delete this event from the event-queue and compute a new event between P_1 and the maximum of the left son of V.
Similarly we have to proceed up the BVH until we find the first predecessor $\overline{V}$ with $max_x\{\overline{V}\} \neq P_2$, or until we reach the root. In the first case we only have to compute another event between $max_x\{\overline{V}\}$ and P_1 and stop the recursion. (Fig. 5).

- *Tree-Event*: Let K be an inner node of the BVH and P_2 be the maximum along the x-axis. Assume further, P_2 is also the maximum of the left son. When a Tree-Event happens, P_2 will be replaced by P_1, which is the maximum of the right son of K (see Fig. 5). In addition, we have to compute a new event between P_1 and P_2 and propagate the change to the upper nodes in the BVH in the same way as described above for the Leaf-Event.
- *Flightplan-Update-Event*: When the flightplan of a vertex changes, we have to update all the timestamps of those events it is involved in.

For measuring the theoretical performance of our algorithm we use the four criteria of quality given for every KDS.

In addition, we want to show that our data structure is a valid BVH even if the object deforms. Therefore, we need the following definition.

Definition 1 *We call a kinetic AABB tree* valid*, if every node in the tree is a bounding volume for all polygons in its subtree.*

Theorem 1 *The kinetic AABB tree is compact, local, responsive and efficient. Furthermore, if we update the BVHs in the manner described above, then the tree is valid at every point of time.*

We start with the prove of the first part of the theorem.

- *Compactness*: For a BVH we need $O(m)$ BVs. With every BV we store at most six Tree- or Leaf-Events. Therefore, we need space of $O(m)$ overall. Thus, our KDS is compact.

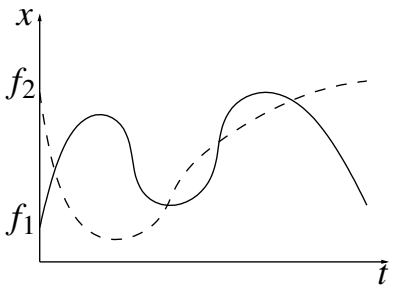

Figure 6: The flighplans are functions f_1 and f_2 in the xt-plane and similarly, in the yt- and zt-planes.

- *Responsiveness*: We have to show that we can handle certificate failures quickly.
  - Leaf-Events: In the case of a leaf event we have to compute new events for all points in the polygon, thus the responsiveness depends on the number of vertices per polygon. If this number is bounded we have costs of $O(1)$. When we propagate the change to upper nodes in the hierarchy, we have to delete an old event and compute a new one, which causes costs of $O(\log m)$ per inner node for the deletion and insertion in the event-queue, since the queue contains $O(m)$ events. In the worst case we have to propagate the changes until we reach the root. Thus the overall cost is $O(\log^2 m)$ for a leaf event.
  - Tree-Events: Analogously, for tree events we get costs of $O(\log^2 m)$.

  Thus the KDS is also responsive.
- *Efficiency*: The efficiency measures the ratio of the *inner* to the *outer* events. Since we are interested in the validity of the whole hierarchy, every event is an inner event because every event changes our attribute. So, the efficiency is automatically given.
- *Locality*: The locality measures the number of events one vertex is participating in. For sake of simplicity, we assume that the degree of every vertex is bounded. Thus, every vertex can participate in $O(\log m)$ events. Therefore, a flightplan update can cause costs of $O(\log^2 m)$. Thus, the KDS is local.

The proof of the second part of theorem is too long to be presented here, so we would like to refer the interested reader to [Zachmann and Weller 2006]. Note that by this theorem, the BVH is valid at every time point, not only at the moments we check for a collision as it is the case with most other update algorithms like bottom-up or top-down approaches.

5 Optimality of the kinetic AABB-Tree

In the previous section we have proven that our kinetic AABB can be updated efficiently. Since there are no internal events, we would also like to determine the overall number of events for a whole animation sequence, in order to estimate the running time of the algorithm more precisely.

Theorem 2 *Given n vertices P_i, we assume that each pair of flightplans, $f_{P_i}(t)$ and $f_{P_j}(t)$, intersect at most s times. Then, the total number of events is in nearly $O(n \log n)$.*

An exact and complete proof is too lengthy to be presented here but it can be summarized as follows. We consider all flightplans along each coordinate axis separately (see Fig. 6).

We reduce the estimation of the number of events on the computation of the upper envelope of a number of curves in the plane. This computation can be done by an algorithm using a combination of divide-and-conquer and sweep-line. During the merge step, a line is swept from one curve intersection to the next. Each intersection corresponds to an update in our kinetic BVH.

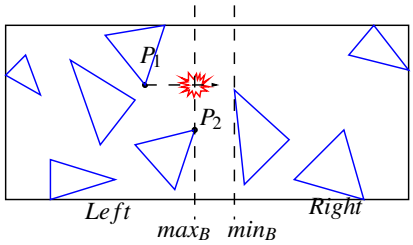

Figure 7: If a vertex in the left subtree becomes greater than the maximum P_2, a Tree-Event will happen.

It can be shown that the maximal number of curve intersections is in $O(\lambda_s(n) \log n)$, where $\lambda_s(n)$ is the length of a Davenport-Schintzel sequence. For given s, $\lambda_s(n)$ behaves nearly linear; more precisely $\lambda_s(n) \in O(n \log^* n)$ where $\log^* n$ is the smallest number m for which the m-th iteration of the logarithm is smaller than 1. For example, $\log^* n \leq 5$ for all $n \leq 10^{20000}$ [Agarwal and Sharir. 1995].

Furthermore, it can be shown that the problem of computing the upper envelope is in $\Theta(n \log n)$, which shows that our algorithm is optimal in the worst case.

For a detailed proof we refer the interested reader to [Zachmann and Weller 2006],

This demonstrates one of the strengths of the kinetic AABB tree: with classical update strategies like bottom-up, we need $O(kn)$ updates, where k is the number of frames. However, with our kinetic BVH, we can reduce this to nearly $O(n \log n)$ updates in the worst case. Furthermore, it is totally independent of the number of frames the animation sequence consists of (or, the frame rate), provided the number of intersections of the flightplans depends only on the length of the sequence in "wall clock time" and not on the number of frames.

Moreover, our kinetic AABB tree is updated only if the vertices that realize the BVs change; if all BVs in the BVH are still realized by the same vertices after a deformation step, nothing is done. As an extreme example, consider a translation or a scaling of all vertices. A brute-force update would need to update all BVs — in our kinetic algorithm, nothing needs to be done, since no events occur. Conversely, the kinetic algorithm never performs more updates than the brute-force update, even if only a small number of vertices has moved.

6 Kinetic Boxtree

The kinetic AABB tree needs up to six events for every BV. In order to reduce the total number of events, we kinetized another kind of BVH, the BoxTree [Zachmann 1995], which uses less memory than the kinetic AABB tree. The main idea of a BoxTree is to store only two splitting planes per node instead of six values for the extends of the box. To turn this into a KDS we proceed as follows:

In the pre-processing step, we build a BoxTree as proposed in [Zachmann 1995], but similarly to the kinetization of the AABB tree, we do not store real values for the splitting planes. Instead, we store that vertex for each plane that realizes it (see Fig. 7). We continue with the initialization of the events:

There are only two kinds of events:

- *Tree-Event*: Assume B is an inner node of the hierarchy with splitting plane $e \in \{x, y, z\}$ and assume further min_B is the minimum of the right subtree (or max_B the maximum of the left subtree). A Tree-Event happens, when a vertex of the right subtree becomes smaller than min_B with regard to the splitting axis e, or a vertex of the left subtree becomes greater than max_B (see Fig. 7).
- *Flightplan-Update-Event*: Every time if the flightplan of a vertex changes, a Flightplan-Update-Event happens.

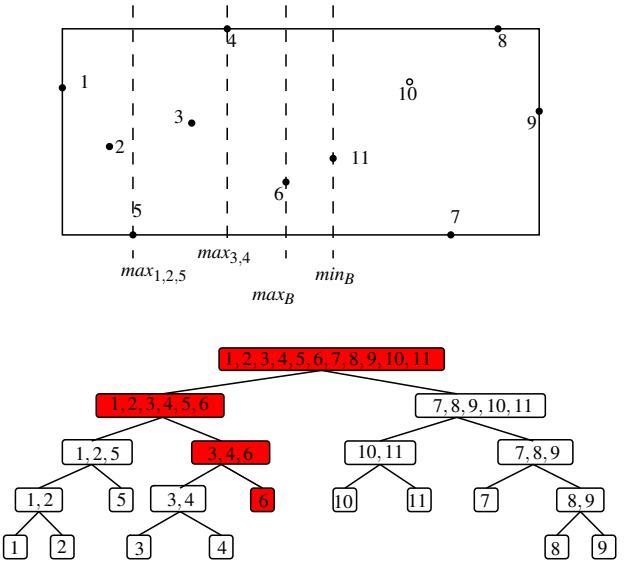

Figure 8: In order to compute a new event, we have to look which vertex can become greater than max_l. In the first level, this could be the maximum of the left subtree, the vertex 5, and all vertices in the right subtree of (1,2,3,4,5,6). On the next level it could be the maximum of the left subtree of (3,4,6), thus the vertex 4, and all vertices in the left subtree.

During runtime, we perform an update according to Algorithm 1 before every collision check. For keeping the BVH valid, we have to handle the events as described in the following:

Tree-Event: Let K be the node, where the Tree-Event happens and let P_{new} be the vertex in the left subtree of K that becomes greater than the current maximum K_{max}.

In this case we have to replace K_{max} by P_{new} and compute a new event for this node. The computation of a new event is more complicated than in the case of a kinetic AABB tree. This is because the number of possibilities of different splitting planes and because of the fact that the extends of the BVs are given implicitly.

For simplicity, we first assume that all BVs have the same splitting axis. In this case, we have to look for event candidates, vertices, which can become greater than the maximum, in a depth-first search manner (see Fig. 8). Note that we do not have to look in the left subtree of the left subtree, because those vertices would generate an earlier event stored with one of the nodes in the subtree.

If more than one splitting axis is allowed, we first have to search for the nodes, with the same splitting axis (see Fig. 9).

Then we have to propagate the change to the upper nodes: First we have to search a node above K in the hierarchy with the same splitting axis. If its maximum is also K_{max}, we have to replace it and compute a new event for this node. We have to continue recursively until we reach a node O with the same splitting axis but $O_{max} \neq K_{max}$, or until we reach the root.

Flightplan-Update-Event: If the flightplan of a point changes, we have to update all events it is involved in. Therefore, we once again start at the leaves and propagate it to the upper nodes.

In order to show the performance of the algorithm, we have to show the four quality criteria for KDS again.

Theorem 3 *The kinetic BoxTree is compact, local and efficient. The responsiveness holds only in the one-dimensional case. Furthermore, if we use the strategies described above to update the BVH, we get a valid BVH at every point of time.*

In the following we show, due to its length, only the first part of the proof, but the interested reader can find the second part in [Zachmann and Weller 2006].

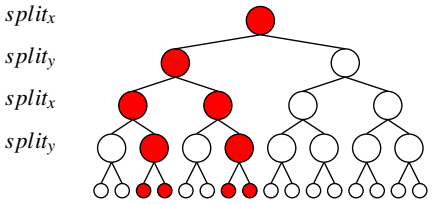

Figure 9: If more than one splitting axis are allowed, we have to search for the next level with the same splitting axis, when we want to look for the next candidates for an event. We have to visit the red marked nodes when we compute a new event for the root box.

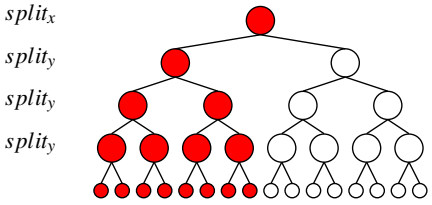

Figure 10: In the worst case, all levels have the same split axis, instead of the root. If we now want to compute a new event for the root, we have to traverse the whole tree.

- *Compactness*: We need space of $O(m)$ for storing the kinetic BoxTree. In addition, we get at most two events per node, so we have $O(m)$ events overall, so, the kinetic BoxTree is compact.
- *Efficiency*: Since we are interested in the validity of the whole hierarchy and every event leads to a real change of the combinatorial structure of the hierarchy, our KDS is also efficient.
- *Locality*: Assuming the tree is not degenerated, one polygon can be involved in at most $O(\log m)$ events, thus the KDS is local.
- *Responsiveness*: Not so clear is the responsiveness of our KDS, which is due to the costly computation of new events, where we have to go down the tree in dfs-manner.
  If all nodes have the same splitting axis, the computation of a new event costs at most $O(\log m)$, because of the length of a path from the root to a leaf in the worst case.
  Deletion and insertion of an event in the event-queue generate costs of $O(\log m)$ and in the worst case we have to propagate the change up to the root BV.
  Therefore, the overall cost for computing an event is $O(m \log^2 m)$ and thus the KDS is responsive in the one-dimensional case.
  But if the other nodes are allowed to use other split-axis too, it could be much more expensive. Assume that the root BV has the x-axis as split-axis and all other nodes have y as split-axis (Fig. 10). If an event appears at the root, we have to traverse the whole tree to compute the next event, so we have total costs of $O(m \log m)$ and thus, the KDS is not responsive.

The total number of events is nearly in $O(n \log n)$ which follows analogously to the kinetic AABB tree. Therefore, we have a second KDS for fast updating a BVH which uses less events than the kinetic AABB tree but the computation of one single event is more complicated.

7 Results

We implemented our algorithms in C++ and tested the performance on a PC running Linux with a 3.0 GHz Pentium IV with 1 GB of main memory. We used two different types of test scenarios, keyframe animations and simple velocity fields.

There are three scenes with keyframe animations, the first one shows a tablecloth falling on a table. We tested this scene with several resolutions of the cloth, ranging from 2k to 16k faces. This

scene shows the behaviour of our algorithms under heavy deformation. The two other keyframe scenarios show typical cloth animations. The first one shows a male avatar with a shirt in resolutions from 32k to 326k deforming triangles (Fig. 11), the other one a female avatar with a dress reaching from 65k to 580k deforming triangles (see Fig. 12).

For measuring the speed of updates when the flightplan changes, we used a benchmark with two spheres. Every point of a sphere is given a velocity vector which points away from the midpoint, so that the spheres expand regularly. When they collide, the velocity vectors of the colliding triangles are reversed. We tested this scene with resolutions from 2k to 40k triangles.

We compared the performance of our algorithms with a bottom-up updating strategy. Because we are primary interested in the speed of the updates, we do not use self-collision detection.

First, we consider the number of events. In the high-resolution tablecloth scene, we have about 400 events per frame and have to update only 1000 values for the kinetic AABB tree, and even less for the kinetic BoxTree (Fig. 13). In contrast, the bottom-up approach has to update 60 000 values. Since the computation costs for an event are relatively high, this results in an overall speed-up of about factor 5 for updating the kinetic AABB tree. The number of events rises nearly linearly with the number of polygons, which supports our lower bound for the total number of events of nearly $O(n \log n)$ (see Fig. 13).

The figure also shows that we need less events for the kinetic BoxTrees, but the the proper collision check takes more time since the kinetic BoxTree is susceptible for deformations.

A high amount of flightplan updates does not affect performance of the data structures, they are still up to 5 times faster than the bottom-up updates (see Fig. 15).

In the cloth animation scenes, the gain of the kinetic data structures is highest, because the objects undergo less deformation than the tablecloth, and thus we have to perform less events. In this scenarios we see a performance gain of a factor about 10 (Fig. 14 and 15). From Theorem 2, it is clear that this factor increases with the number of interpolated frames between two keyframes. 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.

Overall, the kinetic AABB 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 costs for updating are still the same, while they increase for the bottom-up-update by a factor of ten.

8 Conclusions and Future Work

We introduced two novel data structures and algorithms for updating a BVH over deformable objects fast and efficient. We presented a theoretical and experimental analysis showing that our new algorithms are fast and efficient both theoretically and in practice. We used the kinetic data structure framework to analyze our algorithms, and we showed an upper bound of nearly $O(n \log n)$ for the updates that are required at most to keep a BVH valid. We also showed that the kinetic AABB tree and kinetic BoxTree are optimal in the sense that they only need to make $O(n \log n)$ updates.

Our kinetic data structures update bounding volumes more than 10 times faster than a bottom-up approach in practically relevant cloth animation scenes. Even in scenarios with heavy deformations of the objects or many flightplan updates we have a significant gain by our algorithms.

We believe that the kinetic data structures are a fruitful starting point for future work on collision detection for deformable objects. We will try to improve the performance by using trees of higher order than binary trees.

The BoxTree uses less events, but is susceptible to deformations. So it could be a good strategy to rebuild parts of a BoxTree if the deformation is too strong. This could also speed up the algorithm.

Furthermore, we showed that our data structures are valid at every time point of simulation. So, they are optimally qualified for continuous collision detection algorithms. Actually, we are working on event-based approach for continuous collision detection of deformable objects, which makes use of the kinetic AABB tree.

Finally we plan to use our algorithms in other kinds of motion, including physically based simulations and other animation schemes and other applications like ray-tracing or occlusion culling.

References

AGARWAL, P. K., AND SHARIR., M. 1995. Davenport–schinzel sequences and their geometric applications. Tech. Rep. Technical report DUKE–TR–1995–21.

AGARWAL, P. K., BASCH, J., GUIBAS, L. J., HERSHBERGER, J., AND ZHANG, L. 2002. Deformable Free-Space Tilings for Kinetic Collision Detection. *I. J. Robotic Res. 21*, 3, 179 – 198.

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

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

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

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

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

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

HEIDELBERGER, B., TESCHNER, M., AND GROSS, M. H. 2004. Detection of collisions and self-collisions using image-space techniques. In *WSCG*, 145–152.

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

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

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

KNOTT, D., AND PAI, D., 2003. Cinder: Collision and interference detection in real–time using graphics hardware.

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

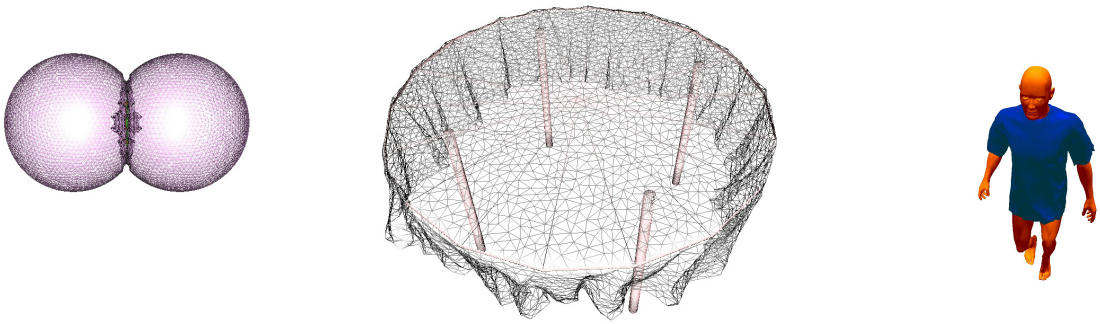

Figure 11: The scenes, with which we tested our algorithm: Two expanding spheres, a tablecloth falling down, and a cloth animation scene.

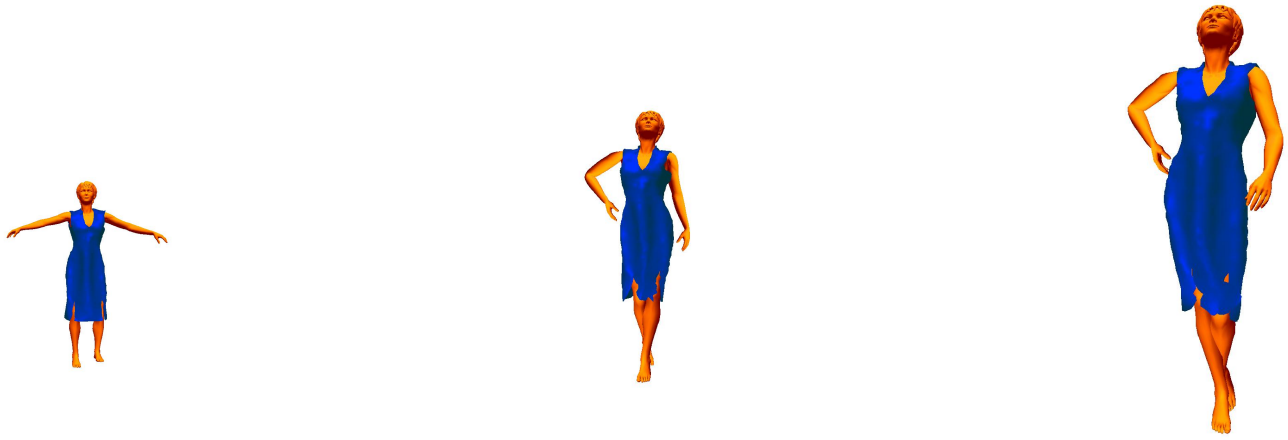

Figure 12: The second cloth animation scene shows a walking female avatar with a dress

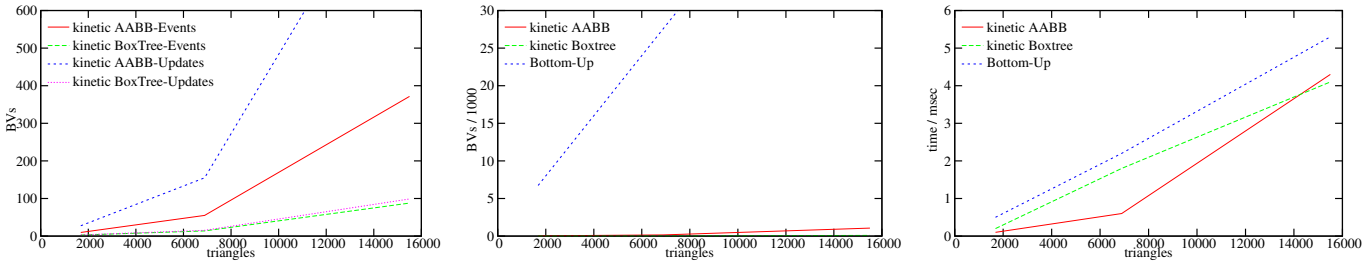

Figure 13: The left diagram shows the average number of events and updates per frame. The kinetic BoxTree has, as expected, the smallest total number of events and the smallest number of total updates per event. The diagram in the middle shows that the total number of updates is significantly lower than the updates needed by the bottom-up-strategy. Unfortunately, due to the relatively high deformation of the tablecloth and the high costs for the event-computation, the gain is lesser than expected, but there is still a significant gain for the kinetic AABB tree and the BoxTree (right diagram).

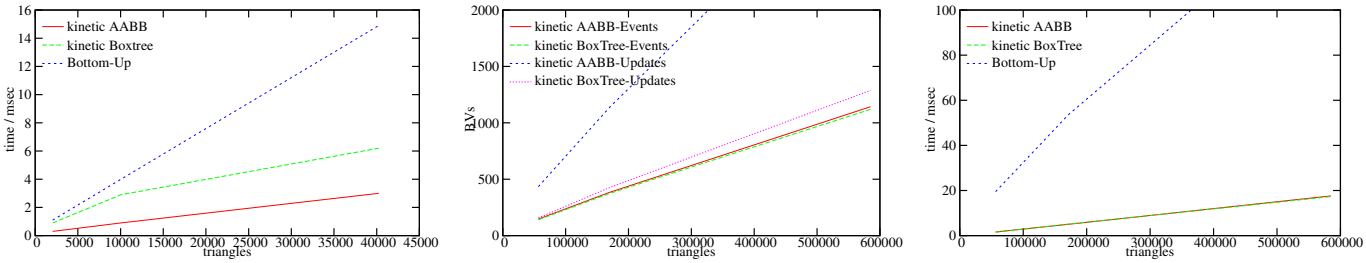

Figure 14: The left diagram shows the average total time for the updates and the collision checks, for the sphere scenes. This scene seems to be more appropriate for the KDSs than the tablecloth scene, despite the high amount of flightplan updates. The gain of the kinetic data structures compared to the bottom-up approach is more than a factor of five. The diagram in the middle shows the average number of events and updates for the cloth animation with the women. The ratio seems to be nearly the same as in the tablecloth scene. The diagram on the right shows the update times in the same animation scene. In this scene we have an overall gain of a factor about 10 for the kinetic AABB compared to the bottom-up-update.

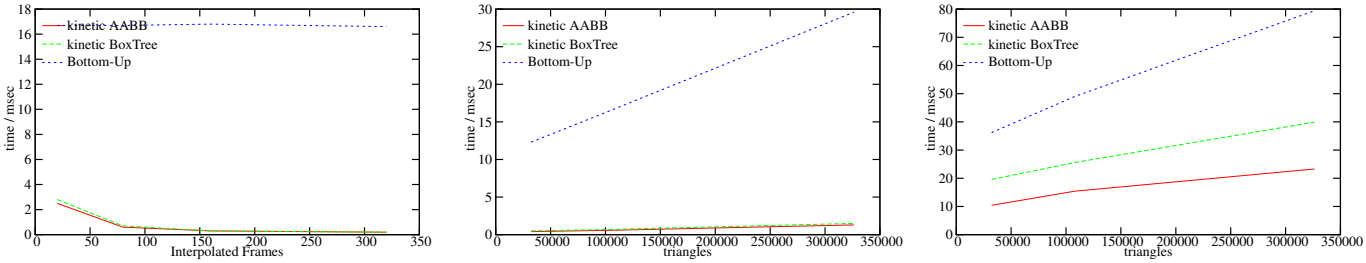

Figure 15: The left diagram shows the average update time for the cloth animation scene with the man, depending on the number of interpolated frames between two key frames. Since the number of events only depends on the number of key frames and not on the number of interpolated frames, so, the average update time decreases if we increase the total number of frames. The diagram in the middle shows the average update time for the cloth animation scene with the man and shows total time for this scene. In this scene we have an overall gain of a factor about 10 for the kinetic AABB compared to the bottom-up-update. The right diagram shows the total time, this means the time for updates and the proper check time.

LAU, R. W., CHAN, O., LUK, M., AND LI, F. W. 2002. Large a collision detection framework for deformable objects. In *VRST '02: Proceedings of the ACM symposium on Virtual reality software and technology*, ACM Press, New York, NY, USA, 113–120.

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

MEZGER, J., KIMMERLE, S., AND ETZMUSS, O. 2003. Hierarchical Techniques in Collision Detection for Cloth Animation. *Journal of WSCG 11*, 2, 322–329.

PALMER, I. J., AND GRIMSDALE, R. L. 1995. Collision detection for animation using sphere-trees. *j-CGF 14*, 2 (June), 105–116.

PROVOT, X. 1997. Collision and self-collision handling in cloth model dedicated to design garments. 177 – 189.

SPECKMANN, B. 2001. *Kinetic Data Structures for Collision Detection*. PhD thesis.

TESCHNER, M., KIMMERLE, S., HEIDELBERGER, B., ZACHMANN, G., RAGHUPATHI, L., FUHRMANN, A., CANI, M.-P., FAURE, F., MAGNENAT-THALMANN, N., STRASSER, W., AND VOLINO, P. 2005. Collision detection for deformable objects. *Computer Graphics forum 24*, 1 (Mar.), 61–81.

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

WONG, W. S.-K., AND BACIU, G. 2005. Dynamic interaction between deformable surfaces and nonsmooth objects. *j-IEEE-TRANS-VIS-COMPUT-GRAPH 11*, 3 (may-jun), 329–340.

WONG, W. S.-K., AND BACIU, G. 2005. Gpu-based intrinsic collision detection for deformable surfaces. *Computer Animation and Virtual Worlds 16*, 3-4, 153–161.

ZACHMANN, G., AND WELLER, R. 2006. Kinetic bounding volume hierarchies for collision detection of deformable objects. Tech. Rep. IfI-06-02, TU-Clausthal.

ZACHMANN, G. 1995. The boxtree: Exact and fast collision detection of arbitrary polyhedra. In *SIVE Workshop*, 104–112.