View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by eScholarship - University of California

UC Davis
IDAV Publications

Title
Kinetic Sweep and Prune for Multi-Body Continuous Motion

Permalink
https://escholarship.org/uc/item/18q0f3kk

Journal
Computers & Graphics, 30

Authors
Coming, Dan
Staadt, Oliver G.

Publication Date
2006

Peer reviewed

eScholarship.org Powered by the California Digital Library
University of California

Kinetic Sweep and Prune for Multi-body Continuous Motion

Daniel S. Coming, Oliver G. Staadt

Institute for Data Analysis and Visualization
Department of Computer Science
University of California, Davis

Abstract

We propose an acceleration scheme for real-time many-body dynamic collision detection. We kinetize the sweep and prune method for many-body collision pruning, extending its application to dynamic collision detection via kinetic data structures. In doing so, we modify the method from sample-rate driven to event-driven, with no more events than the original method processed, also removing the per-frame overhead, allowing our method to scale well in terms of frame-rates. Unlike many schemes for many-body collision pruning, ours performs well in both sparse and dense environments, with few or many collisions.

Key words: Methodology and Techniques–Graphics data structures and data types; Three-Dimensional Graphics and Realism-Virtual reality; Dynamic collision detection; Kinetic data structures

1 Introduction

Collision detection is a common and often critical requirement for virtual and augmented reality, haptic feedback devices, animation, and physical simulations. The work of collision detection is to determine if/when objects intersect, or interact, which is useful in providing realistic environments, accurate simulations, and both appropriate and robust user interactions. The level of accuracy in collision detection directly affects each of the aforementioned properties. While some situations involve relatively few objects, it is common to find scenarios with hundreds or more objects. Scenarios requiring *many-body collision detection* include: large or complex environments, collaborative environments, and complex or large-scale physical simulations (see Figure 1). Most of the scenarios we have mentioned involve objects whose motion is not fully known in advance and require interactive frame rates of 30-60 Hz, so they

require an on-line collision detection method. Haptics require even greater response rates at 1000 Hz or more.

Interactions commonly take the form of collisions, which are usually detected through the use of complicated intersection tests. Because of the expense of object intersection tests as well as the combinatorial number of possible tests among objects, many-body collision detection usually employs an acceleration method for pruning the majority of tests for object intersections. This is then paired with a method for reducing the number of feature intersection tests, along with more exact (and expensive) methods for finding interacting object features. The three primary classifications that apply to any of the above components of collision detection are static, pseudo-dynamic, and dynamic (also called continuous) [1]. Static refers to one-time methods that determine if objects interact in their current configuration. Pseudo-dynamic collision detection is the common extension of static methods to systems with moving objects by incrementally moving objects and performing static collision detection at a regular rate. Commonly, due to the irregular distribution of collisions in time, this rate is too frequent or too infrequent at any given point in simulation. It is well known that these methods suffer from *temporal aliasing,* missing collisions due to objects moving too much between collision detection samples. Increasing the sample rate yields greater levels of accuracy with diminishing returns and degraded performance. Dynamic collision detection considers time and motion information in order to give conservative results without missing collisions, also providing time and location information about collisions. Resolving collisions is straightforward given this information, since objects never interpenetrate. Dynamic collision detection does not suffer from temporal aliasing. The threshold where the accuracy of dynamic collision detection significantly diverges from pseudo-dynamic collision detection depends on how far objects move per sample interval. We have previously demonstrated that this threshold is when objects move by 1/3 of the smallest diameter of any feature, per sample interval, or *frame* [2].

Methods for pruning object intersections as well as feature intersections often use bounding volumes (BV) for simple preliminary tests, to reduce the number of expensive primitive tests that must be performed. World-centric BVs, described in world-coordinates and often used in many-body collision pruning, include axis-aligned bounding-boxes (AABB) [1; 3–5], discrete oriented polytopes (k-DOP) [5–7], and spheres [5; 8–13]. Spheres are also used as model-centric BVs. In contrast to most world-centric BVs, model-centric BVs like oriented bounding-boxes (OBB) [5; 14–18], are described in model-coordinates, and are often trivial to update for rotations. Model-centric BVs are useful for object feature collision pruning. Both pseudo-dynamic and dynamic BV intersection tests exist for BVs [8; 14].

Collision pruning methods utilize spatial and temporal coherence of objects in

consecutive frames. Spatial subdivision methods [8; 19] prune pairs of objects that cannot possibly collide because they are not in the same cell at the same time. Spatial subdivision suffers from the difficulty of choosing cell sizes as well as the performance ratio of work spent on the scheme to the number of collisions detected. Adaptive spatial subdivision [19] suffers from high constant factors in the cost of maintaining the spatial subdivision as well as pathological configuration cases.

Coherence also aids the sweep and prune method used by Cohen et al. [3], as well as bounding volume hierarchies (BVH), in efficiently reducing the number of bounding volumes that need to be compared. [4; 14–16; 20–23]

In this paper, we present a many-body dynamic collision detection method that operates at interactive rates. To this effect, our contributions are as follows:

- *Kinetize* the sweep and prune method for use with dynamic collision detection, removing temporal aliasing and enabling a significant increase in the possible number of objects for real-time performance.

- Provide an event-driven method with no per-frame overhead, minimal update costs, and excellent scalability in terms of frame-rates.

- Demonstrate a scheme for efficient dynamic collision detection, with only locally known motion, that is quickly responsive to changes in motion.

- Give an alternative to swept bounding volumes for reducing the number of dynamic object intersection tests.

- Define a metric for performance comparison between both event-driven and frame-based collision detection methods.

2 Related work

Most collision detection schemes work with a collision detection pipeline similar to the one proposed by Zachmann [24]. This involves updates to bounding volumes, pairwise bounding volume tests, and pairwise feature tests between possibly-intersecting objects. This is followed by collision response, performed for each interval of time, usually at a rate of at least 30 Hz, some beyond 1000 Hz. Such methods incur overhead for each time interval tested, spent updating bounding volumes and collision pruning data structures, regardless of the occurrence or frequency of collisions during the time interval.

2.1 Collision detection methods

Dynamic collision detection methods generate results that are valid so long as the motions of the involved objects are valid. Thus, dynamic methods are driven by the events that cause changes in objects' motions. Swept bounding volumes (SBV) [15; 16; 25–27], are pseudo-dynamic BVs that bound an object through its entire motion over a time interval, effectively reducing the number of dynamic intersection tests by casting the dynamic collision pruning problem into pseudo-dynamic acceleration schemes such as [3]. This still has the problem of being constrained by per-frame overhead, which is now more expensive due to the requirement to find tight bounds on not only the object, but its motion. Sampling too infrequently results in large SBVs, and thus more dynamic intersection tests. Yet sampling too frequently results in much overhead in computing the SBVs. Instead of an SBV that bounds motion, while itself not moving, we would like to use a BV that moves with the object it bounds [8; 14], to avoid the per-frame overhead of SBVs. We will call this moving BV a *kinetic BV*.

In I-COLLIDE, Cohen et al. [3] used *sweep and prune*, a pseudo-dynamic object collision pruning method which reduced 3D collision detection among AABBs into three separate 1D problems, taking advantage of spatial coherence for expected $O(n)$ performance. We note that sweep and prune is a pseudo-dynamic method with per-frame overhead. As a pseudo-dynamic method, it exhibits temporal aliasing, without proper precautions. We discuss sweep and prune in further detail in Section 3 and our extension of it for dynamic collision detection by *kinetizing* in Section 4.1.

Other methods using sweep and prune include: SOLID [21; 28; 29], Swift++ [30], and V-COLLIDE [31]. Each of these methods used sweep and prune for object collision pruning along with different pseudo-dynamic intersection tests: Enhanced GJK [32], Lin-Canny [33], and Separating Axis Theorem (SAT) with OBB-tree [17] respectively. Further, each method could benefit from using our extensions along with the dynamic version of their corresponding intersection tests. Larsson and Akenine-Möller [23] use sweep and prune with deformable objects by enlarging AABBs to enclose all possible deformations when possible, yet this reduces the quality of pruning. Alternatively, they regularly recompute AABBs to account for deformations, which increases per-frame overhead. Van den Bergen [22] has also presented a method for linearly translating objects that enhances sweep and prune by calculating swap times and performing them in time-sequential order by using a priority queue. This is shown to have better pruning than with SBVs, however AABBs are still updated each frame, incurring a per-frame overhead.

Eberly [14] discussed efficient dynamic collision tests based on SAT, applied

to polygons, spheres, and OBBs. He also provided a framework for hierarchical dynamic collision detection with OBB-trees, which we use. Cameron [34] extruded 3D objects into a hierarchical 4D structure and performed half-space intersection tests based on that. He noted, however that for many-body collision detection part of the method explicitly considered all $O(n^2)$ pairs of objects. Eckstein and Schömer [5] used hierarchical dynamic collision detection between complex objects, but did not resolve many-body dynamic collision detection, citing it as a "major bottleneck of multi-body simulation."

Hotz et al. [26; 27] used swept AABBs with space partitioning or coordinate sorting (similar to sweep and prune) for object collision pruning. Remaining object-pairs were tested for collisions using dynamic intersection tests. Our previous arguments against spatial subdivision and swept bounding volumes apply here.

Redon et al. [15] used swept OBBs and heuristically subdivided them when they grew too large relative to the object itself. The necessity of such a heuristic along with the involved work in subdividing SBVs seems evidence in favor of kinetic BVs over SBVs. They made no claims of scalability in the number of objects, however our tests with low frame-rates that have resulted in 30% more SBV overlaps, and thus worse collision pruning, than kinetic BVs.

2.2 Kinetic data structures

Kinetic data structures (KDS) [35] are methods for mobile data that "animate proofs through time," by maintaining a set of *certificates* and a schedule of events that predict certificate failures. For collision detection, a common use of KDS is to maintain proof of non-penetration of a pair of objects [19; 35; 36].

Basch et al. provided an efficient KDS [35] for maintaining the closest pair of points in a set, a method that naturally extended to maintaining closest features, which is useful for collision detection, however the closest pair of points or features can easily change $O(n^2)$ times without a collision even occurring. Maintaining the closest pair generates events too frequently for the case of collision detection, and with complex non-convex objects, evaluating the closest pair is expensive. Basch et al. [36] further provided methods for kinetic collision detection between two non-convex polygons by maintaining a KDS that kept track of the space between the polygons.

Kim et al. [8] utilized a KDS for space-partitioning, where the events were (bounding) spheres entering or leaving cells, as well as collisions. Their method required $O(\lg n)$ work each time an object entered or left a cell, often much more frequent events than collisions.

Often, collisions are irregularly distributed in time, so we propose an event-driven collision detection method that uses a KDS to avoid this per-frame overhead. This paper is an expanded version of our work on kinetic data structures for collision detection [37].

3 Overview

Our method allows the simulation to drive its action, processing events in time-sequential order and/or accepting motion updates as deemed necessary by the simulation. Due to the order of events, collision pruning, object intersection tests, and collision responses are handled in an interleaved manner. Figure 2 shows how we handle these actions. Note that the simulation can delegate work to the collision detection whenever it has extra time; the collision detection engine is robust and without penalty for invoking collision detection too frequently or infrequently.

In order to accomplish this, our methods draw on established techniques in both collision detection and computational geometry. Sweep and prune [3] provides object collision pruning while we use dynamic collision detection using OBB-trees [14] for exact feature-level collision detection. Kinetic data structures [35] are commonly used in computational geometry for efficiently handling data points with motion information.

Sweep and prune is a dimension reduction collision pruning technique that projects objects' geometries into 1D in x, y, and z, storing the extrema on lists. The method updates and sorts these extrema and prunes collisions by tracking when they swap positions. We go into more detail on sweep and prune in Section 4.1 and object intersection tests in Section 4.4. For now, we note that it is a good candidate for *kinetizing*, extending a static or pseudo-dynamic method with a kinetic data structure, because it depends on swaps that are straightforward to extend into swap events. For further discussion of swap events, see Section 4.2.

By *kinetizing* sweep and prune, we transform it from a frame-based method, dependent on updates at a regular interval, to an event-driven method. Many collision detection methods are frame-based due to convenient synchronization with rendering. However, collisions are often irregularly distributed in time, so detecting collisions is best done in an event-driven manner, where the events are: collision responses, motion changes, and BV intersections. All of these events signal the possibility of new collisions, so they can result in object intersection testing. In our case, as with sweep and prune, the status of BV intersections are signalled by swap events. We go into more detail on *kinetizing* sweep and prune in Section 4.1 and the events in Section 4.2.

4 Components of our method

In this section we present the components that make up our dynamic collision detection scheme *kinetic sweep and prune* (kinetic SP). Given descriptions of the motions of objects, we schedule events that signal when kinetic BVs start or stop intersecting. When kinetic BVs start intersecting, a pair objects could collide in the near future, so we perform dynamic complex object intersection tests to determine when and where the first point of intersection will occur. If a future intersection is found, we then schedule a collision response to occur at the first intersection time.

4.1 Kinetizing sweep and prune

We use a *kinetic sorted list* (KSL) [35], a KDS for maintaining a sorted list of 1D moving points (see Figure 3). This keeps the list continuously sorted, in spite of the values in the list changing, i. e. due to their motion. The following equation for motion accounts for velocity and acceleration (e. g., scenes with gravity):

$$m_{\mathbf{p}Ad}(t) = \mathbf{p}_{Ad} + \mathbf{v}_{Ad}t + \mathbf{a}_{Ad}t^2, d \in \{x, y, z\}, \tag{1}$$

where $m_{\mathbf{p}Ad}(t)$ represents the d component of the motion for point $\mathbf{p}_A$ associated with the AABB of object A, as a function of time t. The initial position, velocity, and acceleration are $\mathbf{p}_{Ad}$, $\mathbf{v}_{Ad}$, and $\mathbf{a}_{Ad}$ respectively. The representation of motion $m_{\mathbf{q}Ad}(t)$ for point $\mathbf{q}_A$ is similar. For each pair of adjacent list elements (i, j), we schedule an event for the first time $s_{i,j}$ when the elements cross (or could possibly cross, if motion descriptions are incomplete). The intersection of the motion functions is calculated by finding the roots of the difference of motions:

$$m_{\mathbf{p}Ad}(s_{i,j}) - m_{\mathbf{p}Bd}(s_{i,j}) = 0 \tag{2}$$

For brevity of notation, we define the following:

$$\begin{aligned} \Delta\mathbf{p}_d &= \mathbf{p}_{Ad} - \mathbf{p}_{Bd}, \\ \Delta\mathbf{v}_d &= \mathbf{v}_{Ad} - \mathbf{v}_{Bd}, \\ \Delta\mathbf{a}_d &= \mathbf{a}_{Ad} - \mathbf{a}_{Bd}. \end{aligned} \tag{3}$$

Substituting only the linear portion of motions (i.e., no acceleration) from Equation 1 into Equation 2, the following shows the prediction for intersection of linear motions:

$$\begin{aligned} \mathbf{p}_{Ad} + \mathbf{v}_{Ad}s_{i,j} - (\mathbf{p}_{Bd} + \mathbf{v}_{Bd}s_{i,j}) = 0, \\ s_{i,j} = -\frac{\Delta\mathbf{p}_d}{\Delta\mathbf{v}_d}. \end{aligned} \tag{4}$$

Including acceleration yields the following prediction for the intersection of quadratic motions:

$$\Delta \mathbf{p}_d + \Delta \mathbf{v}_d s_{i,j} + \Delta \mathbf{a}_d \left(s_{i,j}\right)^2 = 0,$$

$$s_{i,j} = \frac{-\Delta \mathbf{v}_d \pm \sqrt{\left(\Delta \mathbf{v}_d\right)^2 - 4\Delta \mathbf{a}_d \Delta \mathbf{p}_d}}{2\Delta \mathbf{a}_d}. \tag{5}$$

See [38] for discussion on avoiding numerical errors in this computation as well as a closed form solution for the roots of cubic polynomials.

Scheduled events are processed in time-sequential order and list elements swap positions, resulting in the destruction of two old adjacencies and the creation of two new adjacencies. The *kinetic sorted list* must respond in turn by descheduling up to two events and scheduling up to two new events. Processing each swap in this way is not optimal for certain circumstances [35]. However, we apply the *kinetic sorted list* to the sweep and prune method, where the swaps provide useful information relevant to collision detection. For this reason, theoretical bounds on kinetic sorting established by Abam and de Berg [39] do not apply to our method. In fact, by ensuring that each swap is processed in order, kinetic sorted lists eliminate temporal aliasing from sweep and prune.

For collision pruning, we *kinetize* sweep and prune [3]. We maintain three ascending-order sorted lists: one for each of x, y, and z axes. The elements on the lists are endpoints Min_{Ad} or Max_{Ad} of moving 1D intervals $I_{Ad}(t)$, obtained by taking the d components of the AABB of objects and their motions. We describe the AABB of object A by its minimal $\mathbf{p}_A$ and maximal $\mathbf{q}_A$ corner points (see Figure 3).

$$I_{Ad}(t) = \left[m_{\mathbf{p}Ad}(t), m_{qAd}(t)\right]. \tag{6}$$

We keep these intervals updated with current projections and motion as necessary, and let the kinetic sorted lists keep track of their sorted order. When two maxima, or two minima, swap positions, there is no effect. However, whenever a maxima and a minima swap positions on the list, a pair of intervals either begins or ceases to intersect, depending on whether the left element was a maxima or minima, respectively, prior to the swap. A pair of objects A and B can intersect only if their projected intervals $I_{Ad}(t)$ and $I_{Bd}(t)$ intersect on all three axes d, i. e. their AABBs overlap:

$$A \cap B \Rightarrow \bigwedge_{d \in x,y,z} \left(I_{Ad}(t) \cap I_{Bd}(t)\right), \tag{7}$$

where $\cap$ indicates intersection of objects or intervals and $\wedge$ is a logical AND operator.

Thus, sweep and prune maintains a basic set of separating axes, axes upon which objects' projections do not intersect, which act as "witness" to the non-penetration of objects. Further, this witness to non-intersection is maintained through swap events. Worst-case performance of this method is due to the possible number of swaps, $O(n^2)$. In practice, Cohen et al. [3] argue, the performance of the insertion sort is expected $O(n)$. Such optimal scenes for the method are sparse, have few collisions, or exhibit a high degree of temporal coherence due to either small object velocities or high sample rates. In these cases, we have measured that the number of swaps is often one or more orders of magnitude smaller than n. This indicates that the bottleneck no longer rests on the frequency of swaps, but instead on updating the list elements and verifying the sorted order of the lists. We can remove these costs by using a KDS to maintain a proof that the lists are sorted.

Kinetizing the sweep and prune method involves using *kinetic sorted lists* rather than static lists with insertion sort. Each list element has its own motion information. In addition, a priority queue stores predictions for when each adjacent pair of list elements will swap, e.g. by Equations 4 and 5. Figure 3 shows how kinetic SP obtains and represents these kinetic sorted lists from the objects' BVs and motions. The need for updates is reduced to only notifying kinetic SP when the motion of an object changes, rather than every time the position changes. Otherwise, the sweep and prune method is unchanged: we handle swaps the same, use AABBs, use one list for each of the x, y, and z axes, and maintain the overlap status of each pair of BVs.

4.2 Events

Our system works with two types of events: collision responses and swaps. The KDS for swaps involves a priority queue of swap predictions and a list of extrema with motion information. Figure 4 demonstrates how swaps are handled. When motions change, effectively, the same method is applied. Swap predictions for the neighbors of $\mathrm{Min}_i, \mathrm{Max}_i$ are descheduled, recalculated, and rescheduled; the difference is that nodes are not swapped, unless the motion change was a discontinuous change in position. Swaps and collision responses are stored in separate priority queues. To avoid unnecessary computation on events that can be cancelled, only the event time and associated object references are calculated in advance. The collision detection system processes each swap or collision response in time-sequential order, when prompted by the simulation. Additionally, the collision detection interface allows the simulation to notify the collision detection engine of motion changes for objects.

Basch et al. [35] mention that for a "real time system, it is possible that there is not sufficient time to completely process an event before the next

event appears." Processing the swap event, in our case, is *efficient* in terms of KDS's, because the number of internal events is the same as the number of external events, swaps, with $O(\lg n)$ work for scheduling. The number of swaps w is $O(n^2)$ in the worst case for linear motion, in a given time interval. Events, especially collision responses, can occur in bursts with arbitrarily close times such that any system would fall behind. We attempt to minimize the latency in such a scenario by using OBB-trees for efficient intersection tests of large complex models. Additionally, we cache the results of dynamic intersection tests, which are by their nature, predictive, and valid until the motions of the objects change. Our system is scalable to very high frame rates, and often would leave the CPU idle in a forced real-time situation. In such a case, it would be possible to utilize the extra time for caching future predictions and use a Timewarp Collision Detection [40] system for concurrency control.

4.3 Object motion

At certain times it is necessary to update the motion of an object (see Figure 5). Causes include collision response, user interaction, or other arbitrary updates from the simulation. We draw a contrast here, between a discontinuous change in position, and continuous *motion*. Many collision detection methods pay a cost related to objects merely moving through space, whether they could possibly collide with other objects or not. Among these methods are any pseudo-dynamic or spatial subdivision methods, but most notably the original sweep and prune, which Cohen et al. [3] measured to have spent significant amounts of time updating bounding volumes, due to discontinuous changes in objects' positions. Indeed, it was as much or more than the time spent sorting their $O(n)$ size lists with insertion sort, which did exhibit the expected $O(n)$ behavior.

For the kinetic sorted list, we can assume any piece-wise pseudo-polynomial function for motion, which is what KDS's are designed to handle. We assume motions are known at least locally, and that motions can change at any time, so long as the collision detection system is notified of the change. Though kinetic SP can efficiently and robustly prune collisions with higher order motions, we use piece-wise linear motions in our testing, for simplicity. This is due to the complications of higher-order primitive intersection tests necessary when objects' BVs overlap.

It is possible to provide kinetic SP with simplified bounds on motion, rather than exact descriptions. The bounds approximation must be conservative to avoid reintroducing temporal aliasing. This is especially useful when actual motions change very frequently or are expensive to parameterize and/or solve for roots. Also, this can be used to deal with noise in the source of motions

(e. g., tracked input devices). Further, for handling motions which include rotations, kinetic SP could work just like the original sweep and prune; place spheres around rotating objects, so they have rotation-invariant BVs as in [25], and then enclose this with an AABB, which is not much larger than the sphere [3]. Alternatively, the minima and maxima of the AABBs could each use separate piecewise linear or quadratic motions that expand and contract the AABB with the same period as rotation. For the sake of avoiding pathological cases where swaps happen at a high rate due to such oscillation as shown in [3], we use an AABB around a sphere for rotating objects. Dynamic intersection tests that handle rotations [41; 42] are not necessarily fast enough for interactive rates with many objects. This is because there is no closed form solution to those tests and numerical solvers are required for root finding.

Motions change at varying frequencies within scenarios with collision detection, especially with user interaction. Therefore, to maintain fast and smooth interaction, the KDS should be *responsive,* requiring updates for few events when motion changes. For kinetic SP, a motion change for an object can result in modification of up to six events, two per axis, each of which could require $O(\lg n)$ work for scheduling. In practice, this work, while comparable to that of a swap, is invoked much less frequently.

4.4 Practical concerns

Promptly following swap events that generate *active pairs* of objects (i. e., those whose BVs overlap), we perform hierarchical dynamic intersection tests between the corresponding complex objects. The minimum of the search interval for these tests is bounded by the current simulation time, typically when the last swap, collision response, or motion change occurred. The maximum starts unbounded. As with any dynamic collision detection, the results of these intersection tests become invalid for objects which undergo collision response or other motion changes. At such a time, *active pairs* which include one or more of the affected objects require re-testing.

For complex object collision tests, we use dynamic intersection tests with OBB-trees [14], based on interval arithmetic and SAT, which gives time, location, and feature information for collisions. We tested dynamic intersection tests with sphere-trees like [13], however OBB-trees yielded significantly faster performance. Figure 6 contains examples of complex objects, with over 6000 triangle faces, that rely on OBB-trees for efficient intersection testing.

We refer the reader to the literature on OBBs [5; 14–18] for information on hierarchical OBB intersection tests. Here it is sufficient to know that an OBB-tree is a hierarchy of OBBs, built by bounding and subdividing pieces of a

model, with a constant number of primitives in each leaf node. OBB-trees have been shown to be tight by Gottschalk et al. [17]. Testing for intersections between objects bounded by OBB-trees involves recursing through the trees, performing OBB intersection tests among tree nodes and primitive intersection tests among primitives bounded in leaf nodes. This continues until intersecting primitives are found. For dynamic intersection testing, the first intersection time is usually desired, so the search continues after bounding the maximum of the search interval with the first known intersection time.

Since the majority of the work spent maintaining a KDS is scheduling and descheduling events, it is crucial to choose an appropriate data structure for the priority queue. We chose the auxiliary two-pass pairing heap [43], due to its efficiency for priority queue operations. It features amortized $O(1)$ complexity operations for insert, meld (join with another pairing heap), amortized $O(\lg n)$ operations for removeMin as well as arbitrary removals and decreaseKey, all with low run-time coefficients.

5 Results and analysis

We tested many-body collision detection methods with ring-models composed of 64 triangles each (see Figure 1). Except where noted in specific experiments, collision detection was sampled at 1000Hz with a scene consisting of 1000 objects at 1% scene density, with velocities in random directions at a magnitude in the range of 0.5–1.5 times the radius of the average bounding sphere. Although we have demonstrated interactive rates with more complex objects, as in Figure 6, rings are sufficient to demonstrate the performance of collision pruning methods intended for complex objects.

As a simple approach to the problem, consider a dynamic collision detection system which tests all $n(n-1)/2$ possible object-object intersections and caches the results and handles the collisions in order. Whenever a collision response occurs or an object's motion changes, the system would need to re-test each affected object against each of the other $n-1$ objects. Let us refer to this method as cached dynamic collision detection. As we will show, it is quite inefficient due to the lack of collision pruning. However it is event-driven, performing work only as necessary per collision.

We gathered results on a 2GHz AMD Opteron 246 with 8 GB RAM. At any time, however, our method used only a small fraction of system memory. Collision response is performed as simple rigid-body collisions between equal-mass objects. While unnecessary for our method, our testing scenario requires that collision detection methods synchronize with the simulation's rendering at a regular frequency f (i. e., frame-rate). This ensures fair testing against frame-

based methods. For statistical relevance, since collision detection occurs at very small time-scales, we generate results by averaging CPU time spent on each of no less than five samples per data point, in addition to allowing simulations to complete between 100 and 2000 frames: one full simulation second per test. This gives us the ratio of CPU time to simulation time, a measure of real-time plausibility and a metric to compare event-driven methods against frame-based methods.

Further, we use uniformly random initialization methods for each object's position, velocity, orientation, and size. Independent of configuration, the results from each method, collisions including times and locations, agreed and were consistent throughout the tests. This is because dynamic collision detection does not suffer from temporal aliasing. Hence, the comparisons in Figures 7 and 8 are focused on improving the performance of dynamic collision detection, as well as extending it to high frequency domains without frequency-related overhead.

As is shown in Figure 7, we verify the scalability of sweep and prune, whether with SBVs or KDS's. We certainly see, on this logarithmic scale, the order of magnitude reduction in total cost by collision pruning with sweep and prune, with even further improvement by kinetic SP. The number of collisions that occur given each configuration depends heavily on the configuration, not just the number of objects. It is shown to illustrate the output-sensitive costs of each method; ideally, fewer collisions would correspond to less cost for detection. Both kinetic SP and cached dynamic collision detection are event-driven by the output, evidenced by how closely their costs increase and decrease proportionally with the number of collisions. However, while sweep and prune (with or without SBVs) also has an associated per-event cost, shown by the jumps in cost for 200 and 400 objects, the additional per-frame overhead leads sweep and prune's costs to monotonically increase. This is in spite of the large drop in collisions at the 700 objects data-point, at which even cached dynamic collision had a decrease in total cost. Kinetic SP actually decreases its total costs in each configuration with a drop in the number of collisions, easily showing its output-sensitivity and lack of overhead. Kinetic SP allows real-time collision detection among nearly twice as many objects as sweep and prune with SBVs.

Next we analyze why kinetic SP delivered consistent improvement on the already good sweep and prune, by breaking the methods into their primary components. Figure 8 demonstrates the per-frame overhead of sweep and prune; the cost of updating and sorting increases linearly with simulation frame rates. By contrast, kinetic SP only exhibits slight increases due to the simulation frequently polling to check if a collision response was ready. Otherwise, kinetic SP demonstrates minimal update costs (including motion change) and freedom from per-frame overhead as an event-driven method. Ideally, the simulation

would have a thread waiting for collision response events, instead of polling, but that would make direct comparison difficult.

5.1 Performance

Cached dynamic collision detection uses no collision pruning and so provides a base-line for comparison: $O(n(c+u))$, for an arbitrary length of simulation time. The number of objects n and the number of motion updates u are input factors, whereas the number of collisions c is output. The test simulation polls collision detection at a regular frequency f, and the base method must at least reply to each request, an $O(1)$ operation. Thus, the CPU time cost per second of simulation time is $O(n(c+u)+f)$.

Sweep and prune improves this with expected per-frame performance of $O(n+o)$, where o are the number of intersection tests performed, due to overlapping BVs. Updating BVs is $O(n)$, a cost once per frame. Sorting performance (i. e., the number of swaps w) depends instead on object motions, but can be $O(n^2)$ for long time intervals. With sample frequency, sweep and prune's cost per second of simulation is: $O((n+o)f+w)$. Assume $O(1)$ update cost for a SBV. Just the pruning cost for dynamic collision detection using sweep and prune with SBVs is $O(nf+w+c\lg c)$. We have shown that the per-frame update cost of $O(n)$ becomes a limitation in consideration of high frame-rates, but it is better than $O(nc)$ in most cases.

Kinetic sweep and prune removes the $O(nf)$ update cost, by taking on scheduling work for the kinetic sorted list, with each swap, motion change, or collision response. By using two-pass auxiliary pairing heaps, we keep this down to an amortized cost of $O(\lg n)$ for removal operations and $O(1)$ for insertions. With kinetic SP, coherence is used maximally, in fact, preserved each time a predicted swap occurs, thus it is not necessary to talk about its expected performance, but instead in terms of events. The total asymptotic bound on CPU time per second of simulation is $O((w+u+c)\lg n+c\lg c+f)$. Sample-rate f is artificially added due to responding to the test simulation, otherwise unnecessary. Thus our method scales especially well with regard to frame sample rate and also the number of objects, since the inherent $O(n)$ factors were reduced to $O(\lg n)$. This result confirms that our method performs very little work on inputs (n, u) and more work on output c, as experimental results showed. Further, our method is asymptotically faster than sweep and prune with SBVs so long as w, u, and c are not $\Omega(nf/\lg n)$. In practice, we've measured the following: $\{w,u,c\} \ll nf$, but they can vary for certain configurations.

Kinetic SP performs no more intersection tests than sweep and prune with SBVs. The kinetic BV is tighter; it does not expand to account for motions,

so it generates the same number or fewer *active pairs* for intersection testing than SBVs, at least when referring to the same base BV-type in both cases. Our experiments have shown that for frame-rates as low as 1Hz, SBVs resulted in more than a 30% increase over kinetic BVs in the number of dynamic intersection tests performed between objects. The extra 30% increase was extraneous, as it yielded no more collisions. This increase in intersection tests was on the order of the number of collisions detected for the tested interval. This differential grows with higher object velocities or lower frame-rates. As with sweep and prune, k-DOPs can substitute for AABBs, yielding even better pruning, but at increased cost proportional to k.

In the terms outlined by Basch et al. [35], the kinetic sorted list is *local* because the maximum number of events in the KDS that depend on one moving object is $O(1)$. In total, it is twelve with AABBs: up to two events per extrema per axis. Further, it is *compact*, because it only requires $O(n)$ space, up to $2n$ extrema and $2n - 1$ predictions per axis. Also, it is *efficient* in our case, as opposed to its original proposal, because it performs the same number of internal as external events, swaps.

To keep track of whether objects' BVs overlap, both kinetic and regular sweep and prune have an optional memory requirement of $O(n^2)$, a trade-off to avoid repeated computations. In addition to this, we add two priority queues, one of which stores swaps in $O(n)$ space, and the other stores collision responses in $O(n^2)$ space, with active removal of invalid nodes.

5.2 Robustness

Kinetic SP only requires an axis-projected BV such as AABB or k-DOP, along with the motion of the BV and a guarantee that the kinetic BV bounds the object for some specified time interval. An event could be generated for ending the validity of a kinetic BV, whereupon it can be recalculated. Kinetic SP works independent of the underlying description of objects, which could be made up of triangles, point sets, or parameterized functions, for example. Further, kinetic SP is independent of the motion of underlying objects, dependent only on the possibly simplified motion (e. g., Equation 1) of the kinetic BV provided. Rotations and even deformations can be accounted for in the motion of the kinetic BV, by assigning different motions to the minimum p_A and maximum q_A points of an object A. It is also necessary that some methods for testing object interactions are defined, such as intersection tests.

Our method can use pseudo-dynamic intersection tests by scheduling polling events that perform intersection tests on all *active pairs* of objects at the polling time. This introduces a per-frame overhead, but not due to kinetic SP.

Such costs may be acceptable for objects whose dynamic intersection tests are undefined or too costly. When used with pseudo-dynamic intersection tests, our method will not introduce missed collisions; it reports exactly when kinetic BVs overlap. Missed collisions are due to pseudo-dynamic testing.

6 Conclusions and future work

We have presented our many-body dynamic collision detection system. We have *kinetized* the sweep and prune method, extending its use to dynamic collision detection, resolving its temporal aliasing, removing its per-frame overhead, and making it more output sensitive. This allows kinetic sweep and prune to perform not only at interactive rates, but to scale well for even higher sample-rates. This is accomplished even with only locally-known object motions and is quick to respond to changes in motion, without requiring swept bounding volumes. Our method scales well with frame-rates, objects, and collisions, and is event-driven, allowing greater flexibility than frame-based methods.

As future work, we would like to apply the methods we have presented to high sample frequency settings, such as haptics. Additionally, we intend to apply kinetic SP with pseudo-dynamic intersection tests, for comparison. Finally, there are cases where traditional sweep and prune is more efficient than kinetic SP, such as when swaps and updates become unusually frequent. However, such a condition usually occurs for short durations of time and so warrants further investigation into a hybrid method.

Acknowledgements

This work was supported in part by an HP-CITRIS fellowship and a Microsoft-CITRIS grant. Thanks to Charles Martel, Yong Kil, Benjamin Ahlborn, and the members of IDAV for helpful discussion. Thanks to David Eberly for making the Wild Magic source code available.

References

[1] M. Held, J. Klosowski, J. Mitchell, Evaluation of collision detection methods for virtual reality fly-throughs, in: Proc. Seventh Canadian Conference on Computational Geometry, 1995, pp. 205–210.

[2] D. Coming, O. Staadt, Velocity-aligned discrete oriented polytopes for dynamic collision detection, Tech. Rep. CSE-2004-25, Department of Computer Science, UC Davis (September 2004).
[3] J. D. Cohen, M. C. Lin, D. Manocha, M. K. Ponamgi, I-COLLIDE: An interactive and exact collision detection system for large-scale environments, in: Symposium on Interactive 3D Graphics, 1995, pp. 189–196, 218.
[4] G. Zachmann, Minimal hierarchical collision detection, in: Proc. of the ACM Symposium on Virtual Reality Software and Technology, ACM Press, 2002, pp. 121–128.
[5] J. Eckstein, E. Schömer, Dynamic collision detection in virtual reality applications, in: Proc. The 7-th International Conference in Central Europe on Computer Graphics, Visualization, and Interactive Digital Media '99 (WSCG'99), Plzen, Czech Republic, 1999, pp. 71–78.
[6] C. Fünfzig, D. W. Fellner, Easy realignment of k-DOP bounding volumes, in: Proc. of Graphics Interface '03, A. K. Peters, Ltd., 2003, pp. 257–264.
[7] G. Zachmann, Rapid collision detection by dynamically aligned DOP-trees, in: Proc. of the Virtual Reality Annual International Symposium, IEEE Computer Society, 1998, p. 90.
[8] D.-J. Kim, L. J. Guibas, S. Y. Shin, Fast collision detection among multiple moving spheres, IEEE Transactions on Visualization and Computer Graphics 4 (3) (1998) 230–242.
[9] P. M. Hubbard, Collision detection for interactive graphics applications, IEEE Transactions on Visualization and Computer Graphics 1 (3) (1995) 218–230.
[10] P. M. Hubbard, Approximating polyhedra with spheres for time-critical collision detection, ACM Transactions on Graphics 15 (3) (1996) 179–210.
[11] I. Palmer, R. Grimsdale, Collision detection for animation using sphere-trees, Computer Graphics Forum 14 (2) (1995) 105–116.
[12] C. O'Sullivan, J. Dingliana, Real-time collision detection and response using sphere trees, in: Proc. of the 15th Spring Conference on Computer Graphics '99, 1999, pp. 83–92.
[13] S. Redon, A. Kheddar, S. Coquillart, CONTACT: Arbitrary in-between motions for collision detection, in: Proc. of IEEE Workshop on Robot-Human Interaction, ROMAN, 2001, pp. 106–111.
[14] D. Eberly, Dynamic collision detection using oriented bounding boxes, Geometric Tools, Inc. (2002).
URL `citeseer.ist.psu.edu/486021.html`
[15] S. Redon, A. Kheddar, S. Coquillart, Fast continuous collision detection between rigid bodies, Computer Graphics Forum 21 (3) (2002) 279.
[16] S. Redon, Y. Kim, M. Lin, D. Manocha, Fast continuous collision detection for articulated models, Tech. Rep. TR03-038, University of North Carolina at Chapel Hill (2003).
[17] S. Gottschalk, M. C. Lin, D. Manocha, OBBTree: A hierarchical structure for rapid interference detection, in: Proc. of the 23rd Annual Conference

on Computer Graphics and Interactive Techniques, ACM Press, 1996, pp. 171–180.

[18] S. A. Gottschalk, D. Manocha, M. C. Lin, Collision queries using oriented bounding boxes, Ph.D. thesis, UNC Chapel Hill (2000).

[19] M. de Berg, J. Comba, L. J. Guibas, A segment-tree based kinetic bsp, in: SCG '01: Proc. of the seventeenth annual symposium on Computational geometry, ACM Press, 2001, pp. 134–140.

[20] J. T. Klosowski, M. Held, J. S. B. Mitchell, H. Sowizral, K. Zikan, Efficient collision detection using bounding volume hierarchies of k-DOPs, IEEE Transactions on Visualization and Computer Graphics 4 (1) (1998) 21–36.

[21] G. van den Bergen, Efficient collision detection of complex deformable models using AABB trees, J. Graph. Tools 2 (4) (1997) 1–13.

[22] G. van den Bergen, Continuous collision detection of general objects under translation, in: Lecture Notes, Game Developers Conference, 2005.

[23] T. Larsson, T. Akenine-Möller, Collision detection for continuously deforming bodies, in: Eurographics 2001, 2001, pp. 325–333, short presentation.

[24] G. Zachmann, Optimizing the collision detection pipeline, in: Proc. First International Game Technology Conference (GTEC'01), Hong Kong, China, 2001.

[25] B. Mirtich, J. Canny, Impulse-based simulation of rigid bodies, in: Proc. of the 1995 Symposium on Interactive 3D Graphics, ACM Press, 1995, pp. 181–ff.

[26] C. Lennerz, E. Schömer, T. Warken, A framework for collision detection and response, in: 11th European Simulation Symposium, ESS'99, 1999, pp. 309–314.

[27] G. Hotz, A. Kerzmann, C. Lennerz, R. Schmid, E. Schomer, T. Warken, SiLVIA – A simulation library for virtual reality applications, in: VR, 1999, p. 82.

[28] G. van den Bergen, A fast and robust GJK implementation for collision detection of convex objects, J. Graph. Tools 4 (2) (1999) 7–25.

[29] G. van den Bergen, Proximity queries and penetration depth computation on 3d game objects, in: Game Developers Conference, 2001.

[30] S. A. Ehmann, M. C. Lin, Accurate and fast proximity queries between polyhedra using surface decomposition, Computer Graphics Forum 20 (3) (2001) 500–510.

[31] T. C. Hudson, M. C. Lin, J. Cohen, S. Gottschalk, D. Manocha, V-COLLIDE: Accelerated collision detection for VRML, in: R. Carey, P. Strauss (Eds.), VRML 97: Second Symposium on the Virtual Reality Modeling Language, Vol. C24-26, ACM Press, 1997, pp. 119–125.

[32] S. Cameron, Enhancing GJK: Computing minimum and penetration distances between convex polyhedra, in: Int. Conf. Robotics & Automation, 1997, pp. 3112–3117.

[33] M. Lin, J. Canny, Efficient algorithms for incremental distance computation, in: Proc. IEEE Conf. on Robotics and Automation, 1991, pp.

1008–1014.

[34] S. Cameron, Collision detection by four-dimensional intersection testing, IEEE Trans. Robotics and Automation 6 (3) (1990) 291–302.

[35] J. Basch, L. J. Guibas, J. Hershberger, Data structures for mobile data, Journal of Algorithms 31 (1) (1999) 1–28.

[36] J. Basch, J. Erickson, L. J. Guibas, J. Hershberger, L. Zhang, Kinetic collision detection between two simple polygons, Comput. Geom. Theory Appl. 27 (3) (2004) 211–235.

[37] D. S. Coming, O. G. Staadt, Kinetic sweep and prune for collision detection, in: Second Workshop in Virtual Reality Interactions and Physical Simulations, 2005, pp. 81–90.

[38] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, Numerical Recipes in C: The Art of Scientific Computing, Cambridge University Press, New York, NY, USA, 1992.

[39] M. A. Abam, M. de Berg, Kinetic sorting and kinetic convex hulls, in: SCG '05: Proc. of the twenty-first annual symposium on Computational geometry, ACM Press, Pisa, Italy, 2005, pp. 190–197.

[40] B. Mirtich, Timewarp rigid body simulation, in: SIGGRAPH '00: Proc. of the 27th annual conference on Computer graphics and interactive techniques, ACM Press/Addison-Wesley Publishing Co., 2000, pp. 193–200.

[41] B. Kim, J. Rossignac, Collision prediction for polyhedra under screw motions, in: SM '03: Proc. of the eighth ACM symposium on Solid modeling and applications, ACM Press, 2003, pp. 4–10.

[42] D. Eberly, Intersection of objects with linear and angular velocities using oriented bounding boxes, Geometric Tools, Inc. (1999).
URL `http://geometrictools.com/Documentation`

[43] J. T. Stasko, J. S. Vitter, Pairing heaps: experiments and analysis, Commun. ACM 30 (3) (1987) 234–249.

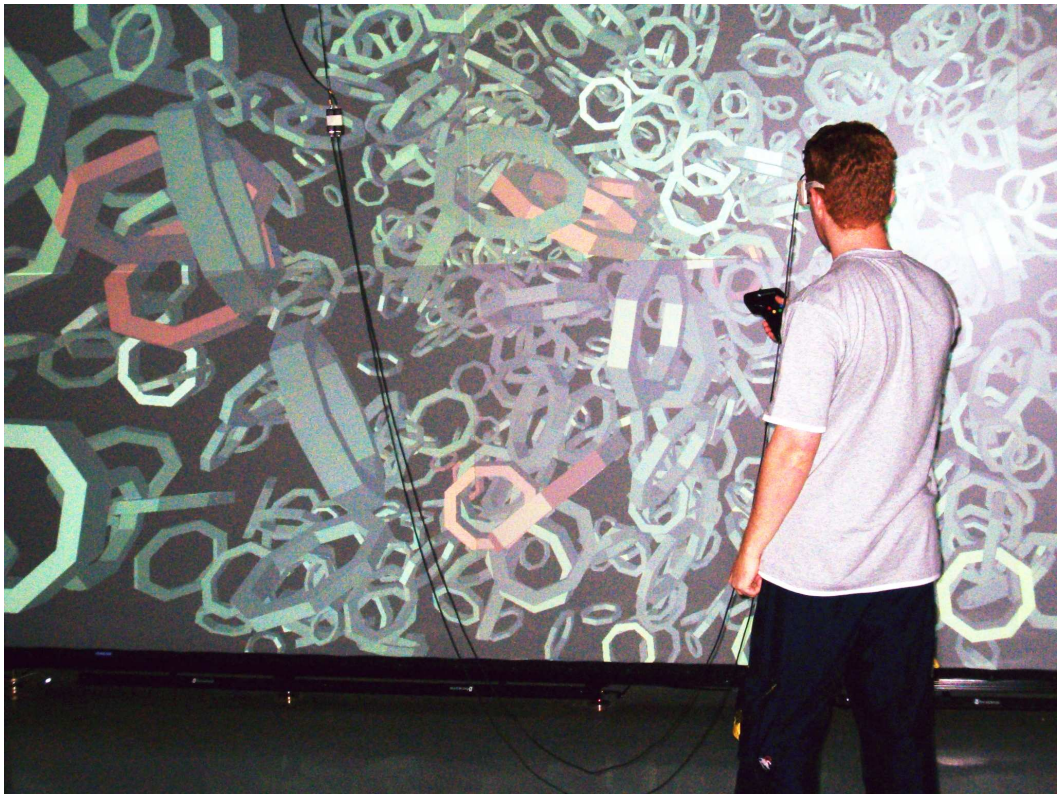

Fig. 1. Many-body (1000 rings) interactive collision fly-through scenario on a display wall, for visual exploration. Red models are colliding. Each has 64 triangles and its own motion.

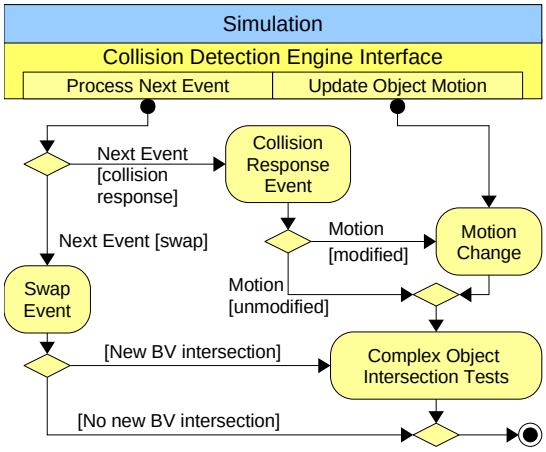

Fig. 2. The collision detection engine (CDE) offers an interface with two primary functions: event processing and updating object motions. Whenever motions are updated through the interface, the CDE maintains motion-dependent event predictions (swaps and collision responses) and may perform complex object intersection tests. Further, the CDE processes events, in time-sequential order as requested by the simulation. Collision response events callback to the simulation for collision response, with resultant motion changes being processed as before, as well as testing the next possible collision with the same objects. Swap events signal that a pair of AABBs may have started or ceased to intersect. In the former case, the CDE uses dynamic intersection tests to predict when and where a collision may occur.

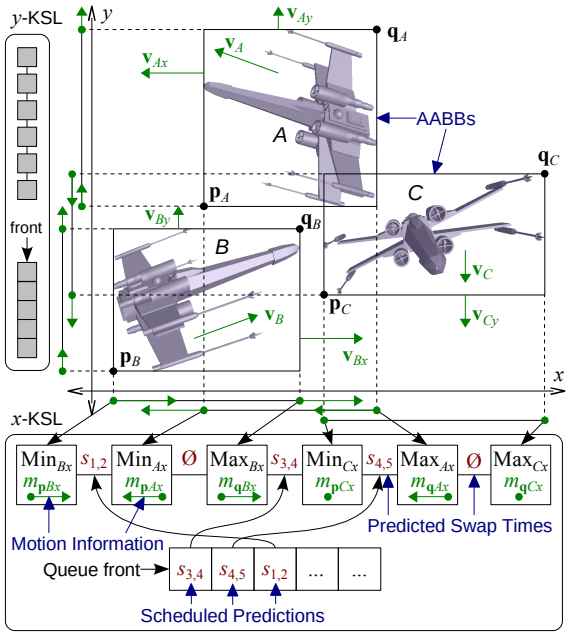

Fig. 3. Kinetic SP represents objects by their kinetic BVs. This is obtained for object A by projecting its geometry into an AABB $(\mathbf{p}_A, \mathbf{q}_A)$. Then for each axis d, the combination $m_{(\mathbf{p}|\mathbf{q})Ad}$ of the d component of the AABB and A's motion (velocity $\mathbf{v}_{Ad}$ in the linear case shown) are stored in kinetic sorted lists d-KSL. The process is similar for other objects, B and C. Each KSL has a priority queue of predictions $s_{i,j}$ for when adjacent list elements will swap. Predictions depend on object motions, e. g. by Equations 4 and 5. A $\emptyset$ prediction indicates that the corresponding pair of elements will not swap in the future, given their current motions. Details of the x-KSL have been shown; y-KSL is similar. Extension to 3D is straightforward.

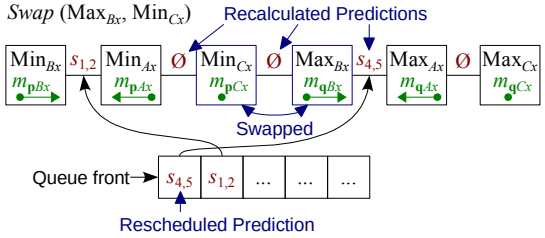

Fig. 4. Example of a swap: Given x-KSL from Figure 3, get the next prediction $s_{3,4}$ from the priority queue. Here, $s_{3,4}$ indicates the need to $Swap(\mathrm{Max}_{Bx}, \mathrm{Min}_{Cx})$. Deschedule non-null predictions among these swapping nodes and their neighbors, Min_{Ax} and Max_{Ax}. Next swap the nodes for Max_{Bx} and Min_{Cx}. Then, calculate swap time predictions (see Equations 4 and 5) for each swapped node with its new neighbor: $s_{2,3}$ for Min_{Cx} with Min_{Ax}; $s_{4,5}$ for Max_{Bx} with Max_{Ax}. Also recalculate $s_{3,4}$; nodes could swap again later with non-linear motion. Schedule any of these predictions that are non-null on the priority queue. Since a maxima swapped with a minima on its right, the BV's of the corresponding objects, B and C, overlap on the x-axis. They also happen to overlap on all other axes, so perform an intersection test between B and C.

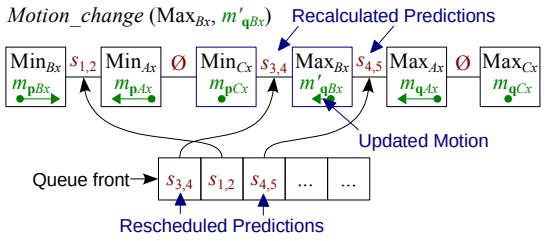

Fig. 5. Example of updating motion: Continuing the example from Figures 3 and 4, objects B and C will collide. Due to collision response, some or all of the extrema for both objects could require updates to their motions. Here we update the motion of Max_{Bx} to $m'_{\mathbf{q}Bx}$, first by copying the updated motion information to the node. Then recalculate predictions $s_{3,4}$ and $s_{4,5}$ for swaps with Min_{Cx} and Max_{Ax}, respectively, and if non-null, reschedule them.

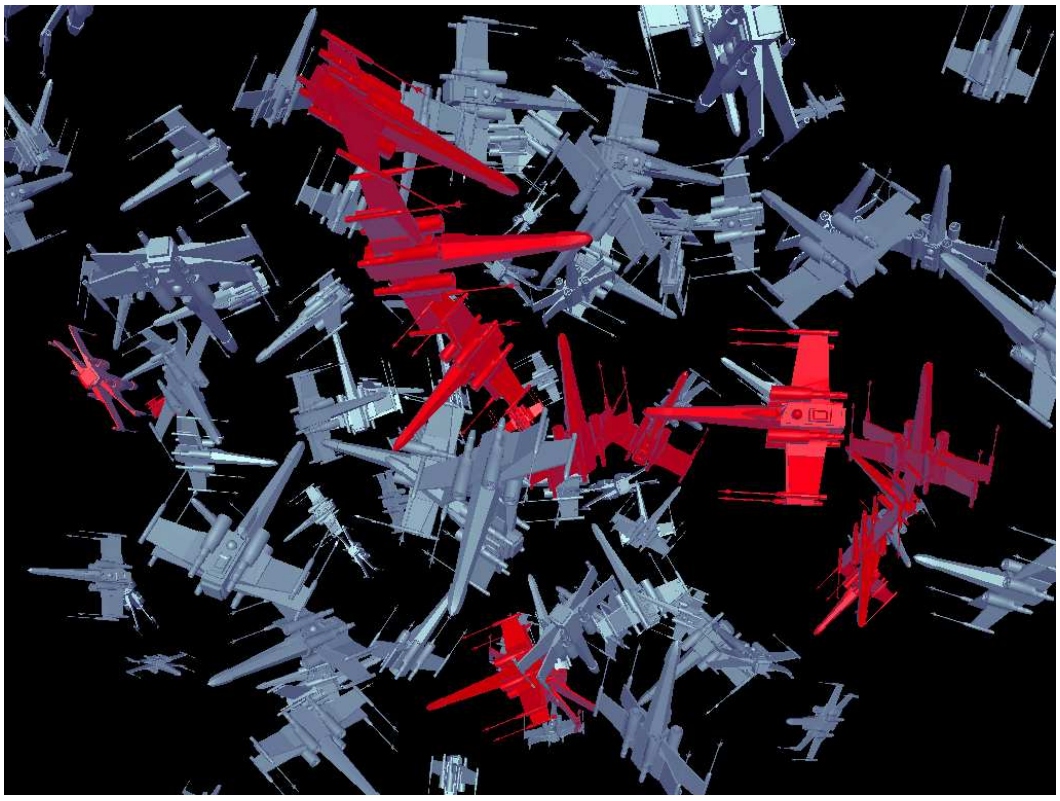

Fig. 6. Example of a complex model scenario requiring kinetic sweep and prune, along with OBB-trees for efficiency. Each of the 100 models is composed of over 6000 triangles, for a total of over 600,000 triangles; our collision detection engine sustains over 15 frames per second, including dynamic intersection tests. Red models are colliding.

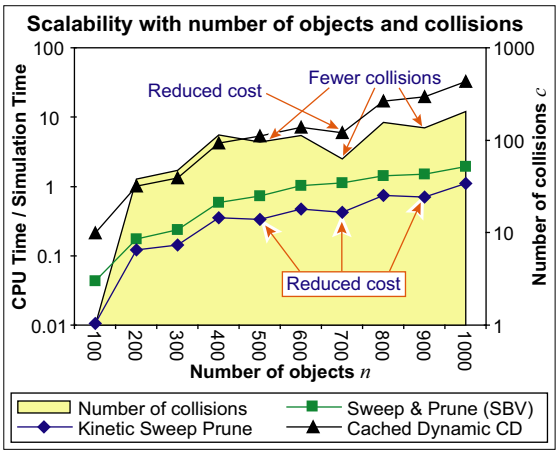

Fig. 7. Scalability of dynamic collision detection methods with respect to input (number of objects n) and output (number of collisions c). CPU time includes updates, pruning collisions, and intersection tests. Data reflects the ratio of CPU time compared to simulation time; real-time is less than 1. Reductions in the number of collisions per object in certain configurations result in proportionally much less work for output sensitive methods without per-frame overhead, favoring kinetic SP. Times are for 1000 frames.

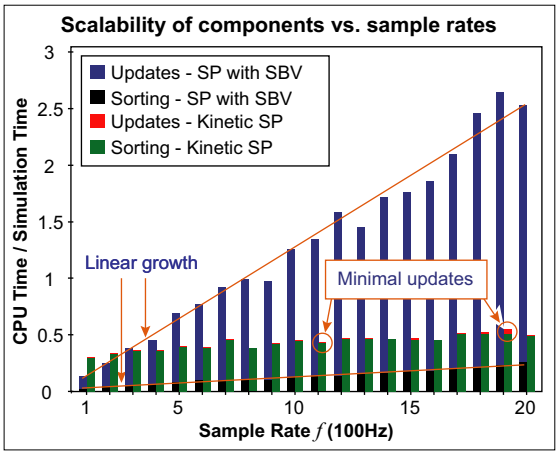

Fig. 8. Scalability in terms of collision detection sample frequency f, for components of sweep and prune with SBVs, contrasted with components of kinetic sweep and prune. Data reflects the ratio of CPU time spent compared to simulation time; real-time is less than 1. For sweep and prune, updating and sorting costs increase linearly with sample frequency. Kinetic SP only has minimal update costs. Sorting times show a slightly increasing cost, due to polling from the simulation.