IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 31 March 2023, accepted 1 May 2023, date of publication 8 May 2023, date of current version 11 May 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3274202

RESEARCH ARTICLE

A Fast and Generalized Broad-Phase Collision Detection Method Based on KD-Tree Spatial Subdivision and Sweep-and-Prune

JIAQI CAO AND MONAN WANG
School of Mechanical and Power Engineering, Harbin University of Science and Technology, Harbin 230103, China

Corresponding author: Monan Wang (mnwang@hrbust.edu.cn)

This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grant 61972117, and in part by the Natural Science Foundation of Heilongjiang Province of China under Grant ZD2019E007.

ABSTRACT Various graphics applications use multibody collision detection, a critical technology in computer graphics, system simulations, and virtual reality. In these simulation environments, broad-phase collision detection, as part of collision detection, plays a critical role in ensuring that rejecting disjoint objects and collision detection is accelerated. Few existing methods implement collision detection of millions of objects in a general-purpose environment on the CPU. This paper proposes a broad-phase collision detection algorithm based on KD-Tree spatial subdivision and sweep-and-prune, which optimizes and accelerates broad-phase collision detection using a pre-sorting and temporal inference solution. Our method enables broad-phase collision detection for coherent and non-coherent settings for uniformly and non-uniformly sized objects respectively. Based on our proposed solution is tested in the context of complex scenarios and compared with other solutions available in the literature and in the industry. The experimental results show that our approach has a 1X to 2X performance improvement in virtual environments with up to 1024×10^3 objects, reaching the fastest collision detection speed of 119.45 milliseconds per frame in the test environment.

INDEX TERMS Collision detection, KD-tree, spatial subdivision, sweep and prune.

I. INTRODUCTION

Many computer graphics, animation, and visualization applications include different forms of multibody or n-body simulation. The core of the collision detection (CD) module is an essential part of the solid illusion of digital objects. On the other hand, it is one of the most time-consuming phases of the pipeline. The CD is usually the main performance bottleneck of physics-based computer simulations. Simulators need to check not only for potential primitive overlap between object pairs but also for many self-collisions within each model. This challenge is also encountered in other areas, such as haptics, robotics, and manufacturing. The performance of physics simulations is essential for computer animation's real-time and interactive nature. Therefore, many physics libraries use various techniques to accelerate the CD and response steps, such as bounding volume hierarchies (BVH), spatial subdivision, sweep and prune (SAP), and GPU acceleration [1]. A particularly challenging and critical feature in large-scale scenarios is the ability to detect collisions in real time.

One of the most common ways to implement a collision system is to divide the process into two or even three stages [2]. The first one, Broad Phase, aims to analyze all objects in the scene, select pairs likely to collide and discard other objects. This phase uses simplification techniques to improve the overall performance of the application process, which is essential to speed up the whole process, especially when there are many objects in the scene. The second phase, called the Narrow phase, aims to test the object pairs found in the previous phase by applying a more robust and accurate method. The third phase (optional) is the Exact Phase, where the computation is performed at the vertex level to obtain higher accuracy.

The associate editor coordinating the review of this manuscript and approving it for publication was Charalambos Poullis.

There are many algorithms dedicated to each stage. Many of these algorithms are only recommended for specific scenarios, hindering a fair performance comparison between them. Therefore, it is crucial to choose efficient and specific algorithms to perform for the scenarios. This problem is more prominent in multimedia application processes such as digital games and interactive simulations, where multiple execution scenarios may exist in addition to managing CPU threads and memory resources. In this paper, we focus only on broad-phase CD.

The broad-phase CD outputs a list of objects close enough to possible collisions by fast-speed approach detection; false global intersection detection is generally handled by shape simplification, spatial reasoning, and temporal reasoning. Shape simplification completely constrains each object with simple geometric elements, providing fast intersection checking. If the bounding volume of an object does not intersect, we can reject this object safely. Axis-aligned bounding boxes(AABB) [3], spheres [4], [5], oriented bounding boxes [6], and K-discrete oriented polygons [7], [8] are commonly used. Although simplified shapes act locally on each object, spatial reasoning works on the whole, reasoning about the distribution of objects and identifying relationships between them, such as clusters, separating axes, and order. Typically, this is achieved by using spatially partitioned data structures, spatial ordering, or hashing. Finally, temporal inference involves leveraging prior knowledge in the simulation and predicting its future behavior, such as identifying static objects, reusing computations and structures from previous frameworks, or even inferring trajectories to predict collisions. The more coherent the simulation, the more opportunities for temporal inference exist.

Modern broad-phase algorithms use all three techniques to achieve competitive performance. However, despite the fast broad-phase algorithms, most solutions tend to perform well in a few specific scenarios and fail to deliver the promised performance in several others. For example, game-oriented solutions typically assume that most objects are static and are therefore optimized for consistency. However, inconsistent behavior can come from player behavior and will result in severely degraded performance. Adaptive or generic solutions will provide consistent performance across multiple application domains and better handle these behavioral changes. Typically, the efficiency of CD algorithms is related to the number of objects involved in a 3D scene, and the algorithm's efficiency decreases significantly as the number of objects increases. Therefore, detecting possible intersection pairs between multiple objects in real-time frequency is often tricky.

On the other hand, most existing CD algorithms mainly deal with the case where the number of objects remains constant, while the actual number of objects involved in CD is often uncertain. In order to adapt to the more general situation, a robust algorithm needs to respond immediately to the above-mentioned special events. Therefore, making the algorithm meet the requirements and perform CD efficiently in unconventional situations becomes another challenge.

This paper proposes a scalable broad-phase CD method to simulate the generic building block for simulating millions of objects. The main contributions of our work are as follows: We adopt a spatial subdivision scheme based on KD-Tree that can handle CDs between objects of non-uniform sizes, coherent and non-coherent distribution; We combine it with an improved SAP algorithm to achieve good rejection; In addition, we use a storage scheme of a shared, linked list to pre-sort all objects before constructing the KD-Tree, which enables faster creation of the KD-Tree and also avoids the complex sorting steps of the SAP algorithm; Meanwhile, we also introduce temporal inference to divide dynamic objects and static objects to avoid meaningless CD between static objects. We test the complete solution with more than one million simulated objects in different scenarios. The results show that our solution is significantly competitive in the environment with a large number of simulated objects, and its average performance is 1 to 2 times higher than other excellent solutions.

II. RELATED WORK

Over the years, continuously improved CD techniques have gained several enhancements in algorithms and spatial data structures. According to the characteristics of the data they process, they can be divided into broad and narrow phases. The broad-phase stage is to avoid too many object queries entering narrow-phase CD, which leads to a long time and computational overload of CD. The methods that can be applied to the broad phase are BVH [9], spatial subdivision [10], [11], and SAP [12], [13]. Games, computer animation, and 3D interactive simulations require broad-phase, practical, and faster CD algorithms. The narrow phase determines the exact collision information after the broad-phase stage to identify potential collision objects, which can later calculate the penetration depth and collision deformation. The methods that can be applied to the delicate detection phase are distance field [14], [15], [16] and CD based on inter-triangle patches [17], [18]. Applications such as fabric simulation and virtual surgery often require stable and accurate narrow-phase CD algorithms.

These CD techniques are highly applicable and have good performance results in the areas where they are applied, and some researchers have reviewed them in reviews [1], [19]. For generic CD methods, such as Bullet Physics [20], PhysX [21], or the Computational Geometry Algorithms Library [22], which are open-source physics engine libraries, their performance is not very good. Their methods are based on stability and overall performance considerations, and their CD methods will not be very accurate, with problems such as false positives and misses. Their methods can achieve the performance requirements only when performing CD on a small number of objects; when the number of objects in the environment rises, their CD performance tends to decline exponentially.

Some researchers use a combination of spatial subdivision and SAP solutions for broad-phase CD. The spatial subdivision divides the space into different regions and tests whether objects intersect in the same region. This method will significantly reduce the time for combined testing, and spatial subdivision captures the concept of geometric coherence, where objects occupying the same space will form possible collision pairs. Other competing algorithms that can be used to determine the similar spatial ordering of objects typically rely on spatial subdivision using data structures such as BSP trees [23], [24], KD-Tree [25], octrees [26], and BVH [9]. Segmentation methods based on uniform grids [27], [28], hierarchical grids [29], and spatial hashing [11], [30] have also proven useful. These techniques are designed to locate collision searches for performance. Uniform segmentation is the simplest but severely reduces its effectiveness when using objects with significant variations in size [31].

Weller et al. [11] used spatial hashing for spatial subdivision, which does not require complex data structures (such as octrees or BSP trees), does not rely on the uniform size of primitives and uniform distribution of primitives, and applies to deformable or even topologically changing objects. Wei et al. [10] used a hybrid representation based on boundary volumes and spatial subdivision, aiming to generate tighter mutually exclusive boundary volumes in the preprocessing phase and to quickly reject irrelevant nearby objects in the broad phase to ensure narrow phase CD does not overload the tactile detection method.

We use KD-Tree in this paper, one of the hierarchical data structures commonly used in ray-tracing algorithms. It is a particular case of the BSP trees that recursively divides the space using a plane perpendicular to the axes of the coordinate system, and each internal node of the tree has a defined separation plane that creates two separate half-spaces. The left and right children of the original node of the tree contain these two half-spaces and redistribute object primitives (such as triangles). Those objects spanning the separation plane must be assigned to the two child objects. For a detailed description of the KD-Tree, refer to [32]. Serpa et al. [25], [33] used KD-Tree to accelerate broad-phase CD, which can effectively handle CD between non-uniformly sized primitives, and its performance is equally suitable for CD of uniform-sized primitives. Their solution optimizes a wide range of object distributions, motion coherence, and different object sizes of collision detection; however, it uses an array to store primitives, which will seriously affect its performance when adding or removing a large number of primitives.

The SAP algorithm, first proposed by [34], performs sorting using an insertion sort in $O(n)$ time under the assumption that the array is almost sorted. However, when temporal coherence is lost, sorting becomes a major bottleneck. In addition, in large-scale simulation, the number of false positives along the sweep axis will increase in a super-linear manner, resulting in unacceptable performance failures [35]. Tracy et al. [36] introduce a new segmented interval list structure that allows for inserting and removing primitives without the need for an entire axis. The algorithm suits large environments where many objects cannot move simultaneously. Liu et al. [37] alleviate the huge density along the sweep axes by using principal component analysis to select the optimal sweep direction, coupled with spatial segmentation to reduce false positive overlaps further. The parallel SAP algorithm used by Capannini et al. [12] for fast CD, propose a two-axis sweep-based method with a parallel SAP algorithm and a concise cache-friendly tree structure. It has good performance with multi-core performance and also tests the device's computational power.

Several researchers have used GPUs for CD acceleration, but most methods are based on uniform meshes [27], [28], [36], [37] and octrees [26], which only apply to modern desktop computers and not to portable mobile devices. Also, they rely on the relatively uniform distribution of collision primitives in space; their CD of non-uniformly sized primitives is also more challenging. Wang et al. [38] subdivided the large triangle into multiple sub-triangles to deal with the collision detection of non-uniformly sized primitives. Later, Wong et al. [26] adopted the octree mesh approach to handle CD with uneven distribution of primitives in the environment. Due to the processing requirements of computer hardware, it is not suitable to use GPUs to accelerate algorithms with hierarchical structures such as binary trees, and because of its parallelism requirements, the use of binary trees cannot achieve load balancing well [1].

III. KD-TREE-BASED SPATIAL SUBDIVISION METHOD

In virtual environments with a large number of objects, the SAP collision detection method is more efficient, but it is impractical to sweep all objects simultaneously, and its workload is tremendous; it requires high storage space and high computational power. In such large-scale virtual environments, we often use spatial subdivision schemes; KD-tree is a structure that allows positioning the segmentation in any direction. Since objects within different cells in the workspace subdivision cannot collide, these objects do not require any further cross-detection. It dramatically reduces the redundant CD computation. We use the KD-Tree-based spatial subdivision method and combine it with the SAP algorithm for implementing CD with a large number of objects environment, significantly reducing the SAP method's computational overhead.

A. CONSTRUCTION OF THE KD-TREE

A good construction of KD-Tree can significantly improve the performance of subsequent SAP. Hybrid methods use a "superstructure" based on a spatial subdivision method, usually a simple grid in which each cell is an instance of sweeping and pruning operating on a portion of the space. The "superstructure" using the KD-Tree-based spatial subdivision method is a binary tree structure similar to a BSP tree, whose division in 3D space consists of a plane, an axis-aligned bounding box (AABB), and a list of objects. In general, we use the basic structure of AABB for collision

detection. Although the sphere bounding box has a lower computational cost, the result is worse than AABB regarding rejection efficiency. The separation plane of KD-Tree is the axis-aligned plane that subdivides the space into two uneven halves, its AABB is the region separated by all the planes of the node's ancestors, and finally, the object list is a list of all objects within the node. Although a leaf node has no planes, the root node has an arbitrarily large AABB because it has no ancestors used to separate it. By subdividing the set of objects into groups, the tree provides enough information to assert that there are no conflicts between objects in different tree branches.

KD-Tree are usually constructed recursively in a top-down manner. There are two critical operations in constructing KD-Tree: one is to select the dividing dimensions, and the other is to select the exact location of the separation plane. In choosing the splitting dimension, the dimension with the most considerable variance or the most extensive dispersion is generally recommended because such a choice can divide the search space more evenly [39]. For selecting the separation plane location, the median value in the data structure is generally chosen as the location of the separation plane to achieve an equal amount of data in the left and right subtrees and construct a relatively balanced KD-Tree. The process of KD-Tree construction on the data requires frequent sorting algorithms to find the median point in the data; this increases the time to read the data during the construction process.

We use the pre-sorting algorithm in this paper [39] and [40], which uses a top-down width-first search algorithm to create the balanced KD-Tree. Before creation, the entire partition is quickly ordered in three-dimensional directions (e.g., in the direction of the x, y, and z axes in the Cartesian coordinate system) with a time complexity of $O(n \log n)$. At the same time, we create a shared multiple linked list that is shared throughout the creation of KD-Tree, and there are three pointer fields in the multiple linked lists. These three pointer fields store the ordered bounding box structure in the direction of the three Cartesian coordinate axes.

We choose the dimension with the most considerable dispersion in the three-dimensional directions in the whole partition, assuming that the largest dimension is the x-axis in the Cartesian coordinate system, and split in this dimension first. Then split in the direction of the second largest dispersion (assuming that this dimensional direction is the y-axis in the Cartesian coordinate system) and the direction of the most negligible dispersion (assuming that this dimensional direction is the z-axis in the Cartesian coordinate system) in turn. Assuming that the ordered linked list is arranged from the smallest to the largest, primitives smaller than the median belong to the left subtree, while primitives larger than the median belong to the right subtree (this paper uses this rule). The time complexity of selecting the median point is $O(1)$. In the splitting of the second largest dimension (the dimensional direction of the y-axis), the data elements belonging to the left and right subtrees are selected from the shared second largest dimension, respectively, with a time complexity of $O(1)$. The splitting of the partitions is carried out sequentially until reaching the specified number of subdivision spaces, assuming controlling the number of primitives in each subdivision space as T; for a collision detection environment with n primitives, subdivide a spatially complete binary tree whose number of spatial subtrees is $2\left(n/2T+1\right)$. A perfect KD-Tree structure makes the number of primitives in each subdivision space more uniform to achieve a more balanced load of the SAP algorithm in different spaces and to achieve the most optimal computational effect when implemented in parallel.

Taking the construction of a KD-Tree in two-dimensional space as an example, Fig. 1 shows the schematic diagram of its creation result. In the process of creating the KD-Tree, there will be cases where the center of the bounding box happens to be on the separation plane, and then always choose to place this bounding box in the left node; For the case where the bounding box intersects the separation plane, choose the region where the centroid of the bounding box belongs to as its note space. The above two cases may lead to false negatives, such as where a bounding box intersects the separation plane and is divided into either subinterval but crosses contact with a bounding box in the other subinterval. For such a case, the following subsection will detail the solutions to avoid false negatives.

B. UPDATE TO THE KD-TREE

The goal of the tree update algorithm is to update the shared ordered linked list and optimize the objects configuration of KD-Tree when the positions of the objects are updated. Changes in the KD-Tree due to object movement must be performed very efficiently and maintain a certain topology. Moreover, they can only be performed within a certain time-critical update and must not lead to a degradation of the structure to an irrecoverable state. The best way to update is to reconstruct, but it is computationally expensive and often not necessary.

Since we use a shared ordered linked list, when objects move, the shared linked list is first updated, and then the tree structure is adjusted and optimized. When updating, the sorted ordered linked list can easily delete and insert nodes while maintaining its ordered state. The update of KD-Tree is divided into two parts, updating the shared linked list and updating the tree structure; the update of the two parts only traverses once, the former updates the ordered linked list from the bottom to the top, updating the disorderly state generated by the movement of primitives to the ordered state, in which we will introduce the temporal inference method; the latter optimizes the tree structure from top to bottom, improving the position distribution of the bounding boxes in the nodes, and solves the situation of false negatives.

1) UPDATE SHARED LINKED LIST

When objects move, the spatial positions of objects change, and their positions in the ordered linked list move. In more detail, when the spatial positions of primitives change, the

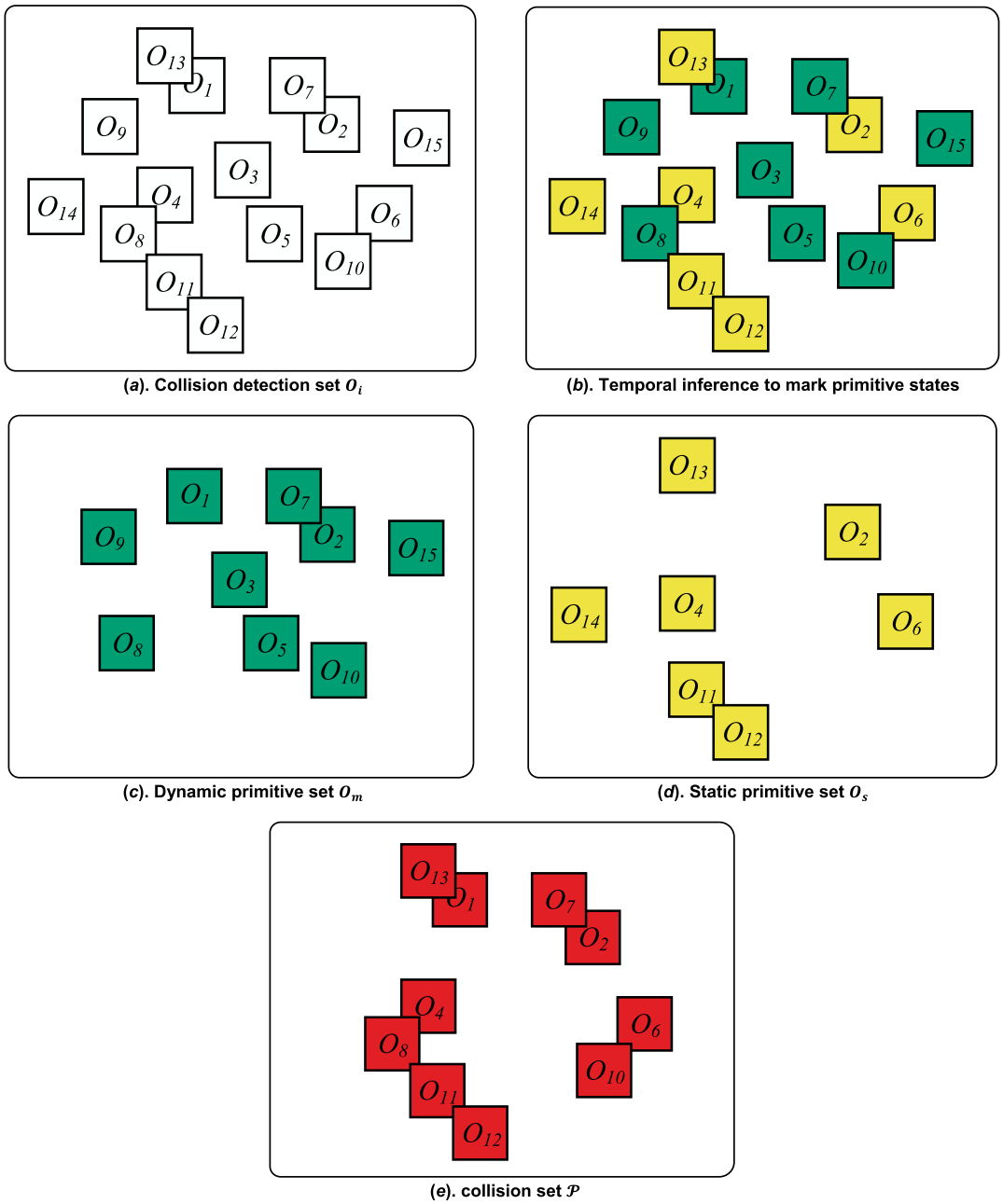


FIGURE 1. Collision detection solution for temporal inference. Only collision detections between dynamic primitives and collision detections between dynamic primitives and static primitives are performed each time update.

original ordered linked list becomes unordered. In this process, the regular operation of ordering the linked list is to swap the linked list nodes. We choose to swap the value in the node, and the position pointed by the pointer of the linked list does not change. When we update in this way, we directly perform quick sorting on the shared linked list. This operation avoids the node update of the linked list during the creation process, that is, it does not need to judge the partition of the child node again. In this way, we do not need to update the separation planes of the KD-Tree, and the separation plane is an abstract flat position space, whose specific position is determined by the position of the bounding box at the median. Moving an object or a group of objects from one node to another node does not change the space size of the ordered linked list, and its time complexity is $O(n \log n)$. In this process, we use a bottom-up traversal method to sort the shared linked list; We also introduce the method of temporal inference for determining whether the primitives are moved or not.

For our temporal inference solution, objects are classified as static primitives or dynamic primitives by expanding the AABB of the primitives with a small constant. The expanded AABB is used as a numerical margin only. For each primitive, at update time, the expanded AABB bounding box

tests the received non-expanded AABB bounding box for the next time node against the current time node of the object. If the received AABB lies entirely within the current expanded AABB, it can be considered static and maintain the non-expanded AABB bounding box for the next time node, marking the primitive as static. If not, the object is considered dynamic and is marked. The basic idea of our temporal inference solution is illustrated in Fig. 2, where the collision set is marked at each time update, as shown in Fig. 2(*b*), the dynamic primitives and static primitives are marked.

When performing the next SAP algorithm, only the CDs between dynamic primitives and the CDs between static primitives and dynamic primitives are performed. This approach is conservative, and the study in [18] uses a similar idea to label dynamic and static primitives, minus unnecessary CDs between static primitives.

2) UPDATE THE TREE STRUCTURE

The update of the tree structure adopts a top-down update strategy and is adjusted in two main parts, the internal nodes update and the leaf nodes update of the tree structure. The update of both behaviors in this process is traversed only once, and its time complexity is $O(n \log n)$.

For internal nodes: 1). When the bounding box intersects the separation plane, the bounding box can only be at the left or right node at the time of KD-Tree creation; if the bounding box of the primitive intersecting the separation plane is marked as a dynamic primitive, it is copied to another node space; the bounding boxes of static primitives intersecting the split plane are not copied; 2). Delete the bounding box of the dynamic primitive that was copied when the previous time node was updated and no longer intersects with the separation plane; and delete the bounding box that was copied at the previous time node and is marked as a static primitive at the current time node; 3). For the case that the bounding box of the dynamic primitive in the continuous time node always intersects with the segmentation plane, retain the copied bounding box in the node.

For leaf nodes: The above update of internal nodes will copy out a large number of bounding boxes of dynamic primitives into the leaf nodes. When the number of bounding boxes in the leaf nodes is greater than the set threshold, the nodes are further subdivided until the number of bounding boxes in the leaf nodes satisfies the threshold. The space subdivision in this process takes the same operation of updating the internal nodes as described above.

When updating internal nodes, the operation of copying the bounding boxes of dynamic primitives on the separation plane will increase the number of bounding boxes in child nodes, especially when there are a large number of bounding boxes intersecting with the separation plane, there will be a large number of copied bounding boxes in the child nodes, increasing the number of bounding boxes in the child nodes and thus increasing the computation time of SAP. Although some of the copied bounding boxes will be deleted during the update, the reduction is limited. This is the optimal solution to avoid the occurrence of false positives.

However, in our parent node, it does not increase the number of enclosing boxes in the whole environment, i.e., the copying operation does not increase the number of primitives in the environment. In our shared ordered linked list, it also does not replicate nodes. The replication operation only affects the internal nodes and leaf nodes. This method is conservative and does not miss any collisions. But in our root node, it does not increase the number of bounding boxes in the whole environment, i.e., a large number of copy operations will not increase the number of primitives in the environment. In our shared ordered linked list, the nodes of the linked list are not copied either. The copy operation only affects the internal nodes and leaf nodes. This method is conservative and does not miss any collisions.

IV. IMPROVED SWEEP AND PRUNE ALGORITHM

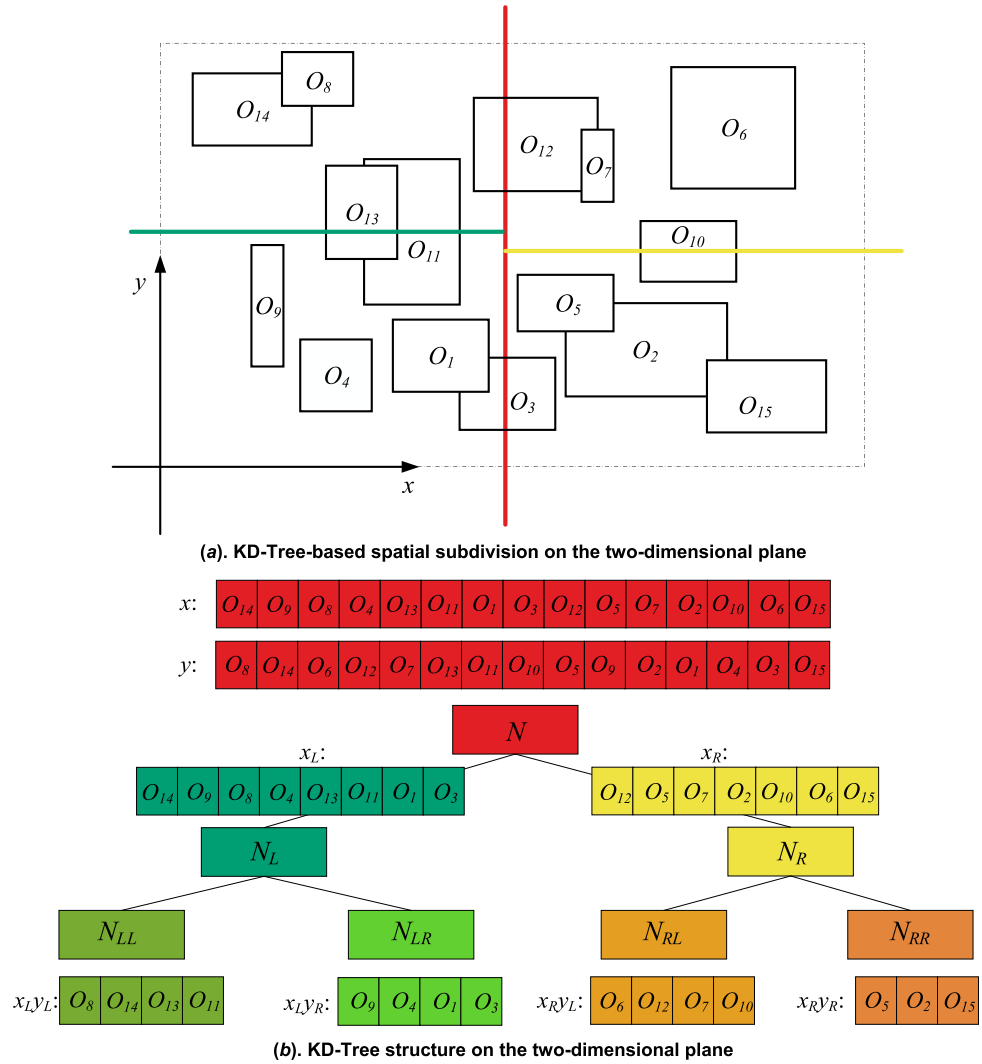
The solution of spatial subdivision based on KD-Tree can significantly eliminate objects that are not in the same space, providing a good prerequisite for the subsequent SAP algorithm. After the KD-Tree is created, each leaf node forms a small independent space, and only the bounding boxes of primitives in each leaf node are swept and pruned, which greatly reduces the amount of data to be processed by the SAP algorithm. In this section, we will detail the methods to optimize performance by improving the SAP algorithm.

A. SWEEP AND PRUNE

To determine whether two bounding boxes overlap, the algorithm reduces the three-dimensional problems to three simpler one-dimensional problems. The objects corresponding to these boundary boxes overlap if and only if the intervals of the two bounding boxes overlap in all three dimensions. To determine which intervals of the objects along an axis overlap, the list of the intervals is sorted.

The SAP algorithm on the one-dimensional axis [41] is described as: project on an optimal one-dimensional axis, which can be obtained by principal component analysis and covariance axes, to achieve the best rejection effect on a one-dimensional axis. The specific method is described as: Given n objects O_i in 3D, the goal of SAP is to find all overlapping pairs $\mathcal{P}$ of objects. Thus $\mathcal{P} = \{(O_i, O_j) \mid O_i \cap O_j \neq \emptyset, 1 \leq i \neq j \leq n\}$. Often, an object O_i is a simple bounding box such as an AABB or sphere that bounds more complicated geometry. We will assume that the objects are simple enough to allow us to determine whether $O_i \cap O_j \neq \emptyset$ in constant time. The original SAP algorithm can be described as Algorithm 1.

If the motion of the objects is highly coherent, then step 2 in this algorithm can be implemented efficiently as an insertion sort, and step 3 can be replaced by swapping operations between neighboring O_i. Note that step 4 becomes redundant for spheres since the exact overlap test between spheres can be implemented along the axial direction by comparing the distance between the centers of two spheres with the sum of



(*a*). KD-Tree-based spatial subdivision on the two-dimensional plane

(*b*). KD-Tree structure on the two-dimensional plane

FIGURE 2. Spatial subdivision and KD-Tree construction on the two-dimensional plane.

Algorithm 1 Basic SaP Algorithm

Input: a set of bounding volume $O = \{O_1, O_2, \ldots\ldots, O_n\}$;
three coordinate axes $\{\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}\}$;

Procedure:

Step1: Project O_i onto the $\boldsymbol{x}$-axis, obtain 1D intervals set $I_i = [m_i, M_i]$;

Step2: Sort m_i and M_i for all i, and obtain a sorted list L;

Step3: Sweep L and maintain an active list A as:

a) if m_i can be retrieved in L, add O_i to A;

b) for each $O_j \in A$, add pair (O_i, O_j) to $\mathcal{P}_x$;

c) if M_i can be retrieved in L, remove O_i from A;

Step4: Repeat Stepl-Step3 for $\boldsymbol{y}$-, $\boldsymbol{z}$-axis, obtain $\mathcal{P}_y$ and $\mathcal{P}_z$;

Step5: Report the final set of colliding pairs $\mathcal{P} = \mathcal{P}_x \cap \mathcal{P}_y \cap \mathcal{P}_z$;

Output: pairs of interacting simple volume $\mathcal{P}$

their radii. Moreover, the intervals I_i are the diameters of the spheres positioned on their projected centers.

This technique can be used as-is for small to medium scenes, or as an operator when dealing with large to large-scale scenes, but with less efficiency. To obtain the best performance, the choice of sweeping axes must be carefully considered. Some well-known strategies include the use of a maximum variance axis [25], approximate principal component analysis [37], and context-aware heuristics [12]. Regarding sorting algorithms, the usual choices are

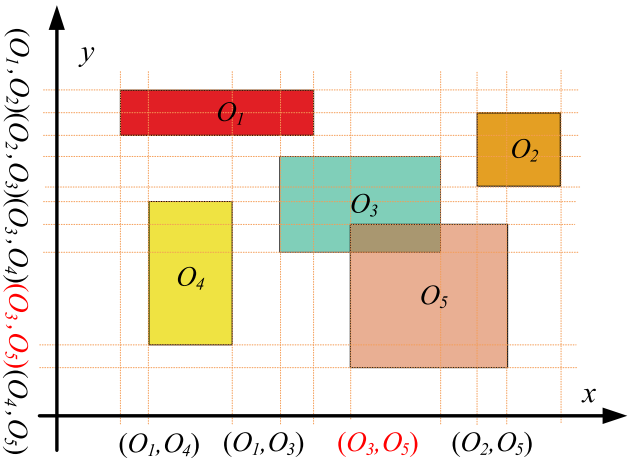


FIGURE 3. The SAP algorithm on a two-dimensional plane.

Quicksort and Radixsort, the latter being often used in parallel settings [12].

B. ALGORITHM OPTIMIZATION

We employ an optimized SAP algorithm that optimizes it during sweeping in all three axes. In the original SAP algorithm described above, the sweep is performed on the three Cartesian axes one by one, and the amount of data processed is relatively huge in a collision detection environment with a large number of primitives, which greatly wastes the time of SAP. Our improved SAP algorithm obtains potential collision pairs after sweeping and pruning on the first axis, and when sweeping and pruning on the other axis, only the candidate pairs on the first axis are swept and pruned to exclude the false positive collision pairs. Our improved SAP algorithm will get a significant reduction in the amount of data processed. The improved SAP algorithm on a two-dimensional plane is shown in Fig. 3. The solution we use obtains collision candidate pairs in the x-axis after obtaining (O_1, O_4), (O_1, O_3), (O_3, O_5) and (O_2, O_5), and then sweeping and pruning in the y-axis to obtain collision pairs (O_3, O_5) from the candidate pairs. It eliminates the complex computation of projecting on the y-axis to obtain more candidate pairs (O_1, O_2), (O_2, O_3), (O_3, O_4), (O_3, O_5), (O_4, O_5).

The KD-Tree-based spatial subdivision method we described above has sorted the bounding boxes before the creation of the KD-Tree and during the update of the KD-Tree. We combine it with the improved SAP algorithm, which can save further computation time for the SAP algorithm by eliminating the sorting operation of objects during the sweeping and pruning process.

Our improved scan pruning algorithm can be described as:

1) Obtain the objects O_i in one of the leaf nodes of the KD-Tree;
2) Project the intervals of each object O_i onto a certain coordinate axis, such as the x-axis in the Cartesian coordinate system, to obtain the one-dimensional interval extreme value of each object $\mathcal{I}_{xi} = [m_{xi}, M_{xi}]$;
3) Obtain the sorted list $\mathcal{L}_x$ in the leaf note;
4) Scan the sorted list $\mathcal{L}_x$ and compare whether the m_{xi} of several adjacent objects satisfy $m_{xi} \in \mathcal{I}_{xj}$; if so, it means that the projections of O_i and O_j on the x-axis coincide, and obtain the collision pair list $\mathcal{P}_x$ of the objects on the x-axis;
5) Project each object O_i in the obtained collision pair list $\mathcal{P}_x$ as the objects O_i to be detected on the y-axis, repeat the above operation, and obtain the collision pair list $\mathcal{P}_y$ that removes some false positive collision pairs;
6) In the same way, project each object O_i in the collision pair list $\mathcal{P}_y$ as the objects to be detected on the z-axis, eliminate all false positive collision pairs, and obtain the final collision pair list $\mathcal{P}$.

V. ALGORITHM CONTROL PIPELINE

Based on the spatial segmentation and SAP framework, we realized an efficient and robust algorithm for multibody CD in large-scale scenes by optimizing its control pipeline and event management and reorganizing its core data structure. The high-level description of our algorithm in the above two chapters facilitates the understanding of underlying concepts and theoretical analysis. The actual calculation also needs to be executed according to the pipeline of the specific CD algorithm. During the execution of the complete solution, many operations can execute in parallel. In this section, we describe the complete implementation of our solution.

We use a CD method based on the KD-Tree spatial subdivision and the SAP algorithm. During the implementation of the algorithm, construct the KD-Tree structure to divide the primitives in the space into different partitions; Store each small partition in the leaf node, and then sweep and prune the primitives in the leaf node. Due to the massive number of objects in space, it is difficult for the SAP algorithm to efficiently perform CD even if they are divided into small partitioned intervals. Therefore, in updating the KD-Tree structure, we introduce a temporal inference scheme to divide the objects in the space into dynamic primitives and static primitives. We only need to perform CD between dynamic primitives and between dynamic primitives and static primitives. For our proposed CD method, Fig. 4 shows the main control pipeline.

As we described in the pipeline, all primitives are presorted, and then the KD-Tree construction, update, and parallel sweep-and-prune operations are performed, with all events executed in chronological order. According to the event priority, the KD-Tree update and sweep-and-prune events are executed at each time iteration update. There are steps in the algorithmic control pipeline that are repetitive tasks that can be implemented using parallel multithreading, such as parallel sorting of k-dimensional (i.e., three-dimensional) data, constructing KD-Tree, and sweeping and pruning. In the pipeline of updating the KD-Tree, the process of temporal inference and optimization of the tree structure, which mainly updates the primitives within the leaf nodes of the tree structure, can be achieved by assigning a thread to each leaf node for parallel processing. The sweeping and pruning operation

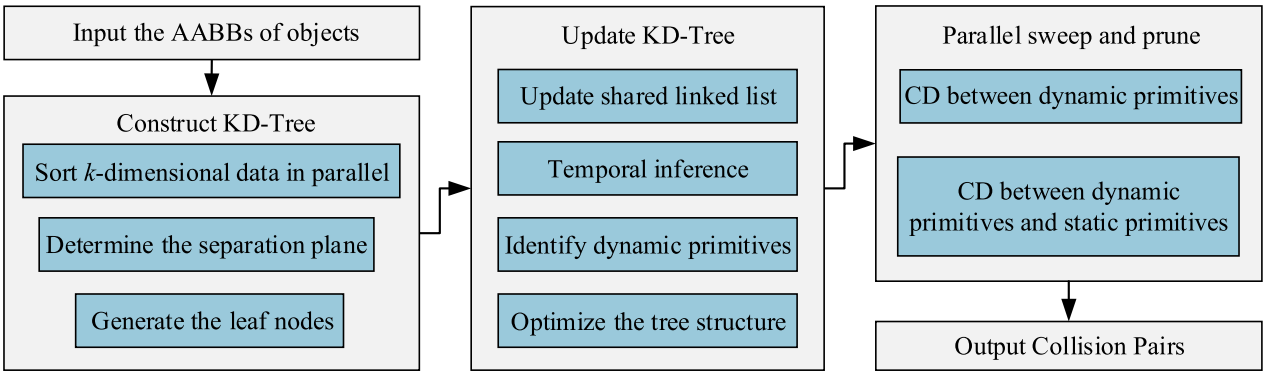


FIGURE 4. Control pipeline of our solution.

is performed only on the primitives within each leaf node, which can similarly be implemented by assigning one thread to each leaf node for parallel processing. We describe our CD method in detail in Algorithm 2.

In the step of obtaining the expanded interval extrema in our Algorithm 2, after temporal inference, the expanded extrema $\left[m'_{kj}, M'_{kj}\right]$ is calculated as:

$$M'_{kj} = \frac{M_{kj}(1+\varepsilon) + m_{kj}(1-\varepsilon)}{2} \tag{1}$$

$$m'_{kj} = \frac{M_{kj}(1-\varepsilon) + m_{kj}(1+\varepsilon)}{2} \tag{2}$$

where k denotes the projected axis, and the specific projection axis of k is determined by the direction of the maximum dispersion of the grandparent node of the leaf node, $k \in \{x, y, z\}$; ε denotes the expansion factor, which depends only on the object's size. If μ is the average object size, then we can use $\varepsilon = \mu/100$ as the default value; $\left[m_{kj}, M_{kj}\right]$ denotes the extreme value of the projection of objects on the k-axis before expansion.

VI. EXPERIMENTS AND RESULTS

We tested our approach on a Windows 10 device with an Intel i7-8700 CPU, 16G RAM, and an NVIDIA GeForce GTX 1660 GPU to evaluate our solution. In the tests, all time measurements were measured as accurately as possible and corresponded only to the time spent by the algorithm, with no other tasks in the measurements.

A. IMPLEMENTATION AND BENCHMARKING RESULTS

We adopt the benchmark scheme set by Serpa et al. [25] as the collision detection environment, design three distinct scenarios, namely *Free Fall*, *Brownian*, and *Gravity*. In a uniformly distributed environment, the Brownian scenario illustrates a typical case that can test the algorithm's overall performance. We used the *Free Fall* scenario to examine the algorithm's performance when some objects are stationary; Finally, we chose the *Gravity* scenario to test the algorithm intensively. Furthermore, we test the CD time for uniformly sized objects and non-uniformly sized objects, respectively; The CD for uniformly sized objects uses equal size cubes to simulate the objects to be detected, and the CD for non-uniformly sized objects uses randomly generated sized rectangles to simulate the objects to be detected.

Fig. 5 illustrates our test scenario, which shows the CD of 32×10^3 objects.

To verify the superiority of our solution, considering that some newer CD methods are challenging to implement or even we cannot repeat, we chose the following advanced CD methods to compare and evaluate with the method proposed in this paper:

CGAL: The CD algorithm [42] provided within the Computational Geometry Algorithms Library [22], which was initially developed by Zomorodian and Edelsbrunner [43], uses a hybrid method of interval/segmented trees and SAP technology. We label it as CGAL in this paper;

iSAP: A hybrid solution using mesh and SAP algorithm [36], designed to handle primarily static scenarios. This method uses a custom segmented list structure to simplify the migration of objects between grid cells when updating objects. We label it as iSAP in this paper;

GPU SAP: GPU-based mesh subdivision and scan pruning method [37], which we label GPU SAP in this paper;

KD-Tree: A SIMD-optimized hybrid algorithm of KD-Tree and SAP [25] with a shared array data storage approach. We label it KD-Tree in this paper.

B. PERFORMANCE

To test the performance of our approach, we tested CDs from 1000 to 1024×10^3 uniformly sized objects, and also from 1000 to 256×10^3 non-uniformly sized objects, separately. We limited our analysis to the 1000 milliseconds per frame mark to focus on the most competitive solution. Any algorithm performing that exceeds this point can safely be disregarded as competitive.

Fig. 6 shows the average collision ratio of *Free Fall*, *Brownian*, and *Gravity* scenarios. It represents the average ratio of collision pairs in each frame to all objects during detection. Based on the characteristics of the benchmark we use, there are relatively few collision pairs in the environment of *Brownian*; different from the environment of *Free Fall* and *Gravity*, many objects will remain in a collision state after falling. In calculating the average collision ratio,

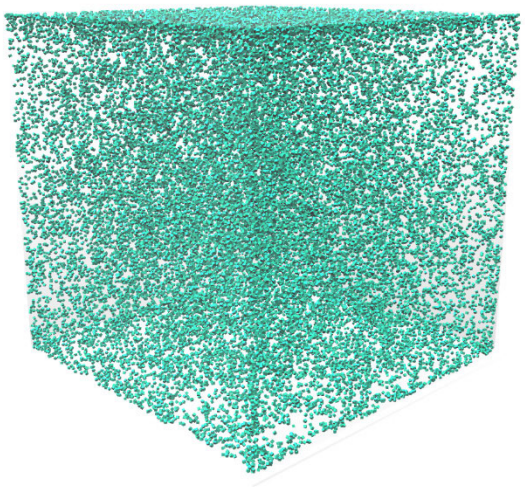
Algorithm 2 Collision Culling of Our Algorithm

Input: $O = \{O_1, O_2, \ldots\ldots, O_n\}$ // A set of AABBs
$\{x, y, z\}$
Output: $\mathcal{P}$ //A set of collision pairs

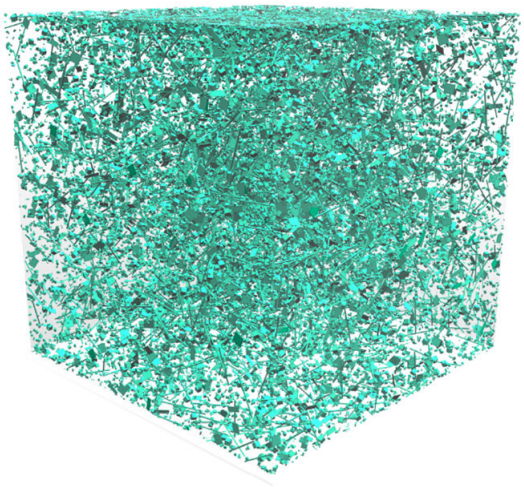
```
L_x ← null; L_y ← null; L_z ← null   //Establish a pre-sorting linked list
for each O_i ∈ O do
  L_x ← quick sort in the x-axis   //Complete the sorting of the linked list
  ...   //The same with the other two axis
  establish KD-Tree
  L_1, L_2, L_3, ..... ← leaf notes
end for
parallel execute   //Execute each L_1, L_2, L_3, ..... in parallel
 for each O_j in L do
  get the projection axis k   //The most extensive dispersion k ∈ {x, y, z}
   J_kj = [m_kj, M_kj]   // Obtain 1D interval extrema
  while time iteration do   //update at every time iteration
    update L_x, L_y, L_z
    get J'_kj = [m'_kj, M'_kj]   // Get expanded interval extrema
    update J_kj = [m_kj, M_kj]
    if m_kj <= m'_kj && M_kj <= M'_kj then
      mark as stationary O_s
    else mark as moving O_m
      optimize the leaf note   // Optimize leaf nodes and
                               //replicate moving objects on separate planes
    for each O_m in L do   //Sweep and prune
      if (O_m, O_j) is overlapping in the k-axis then   //m is not equal to j
       obtain collision pairs P_k ← (O_m, O_j)
       for each O_m in P_k do
         if (O_m, O_k) is overlapping in k̄-axis then   // m is not equal to j
                          //k̄-axis is any axis except the k-axis in {x, y, z}
          obtain collision pairs P_k̄ ← (O_m, O_k)
          for each O_m in P_k̄ do
            if (O_m, O_k̲) is overlapping in k̲-axis then
                    //k̲-axis is the last axis except for the k-axis and k̄-axis in {x, y, z}
             obtain collision pairs P_k̲ ← (O_m, O_k̄) // m is not equal to k̄
            end for
          end for
      end for
  end while
 end for
end parallel execute
P ← colliding pairs P_k̲ from all threads
Return: P
```

we also count collision pairs that have been stationary but maintain a contact state in the environment. The lower the average collision ratio, the better the rejection efficiency of the algorithm. In this comparison, Fig. 6 shows that the solution we adopted has a specific improvement in rejection efficiency and will significantly improve the efficiency of the subsequent narrow-phase CD.

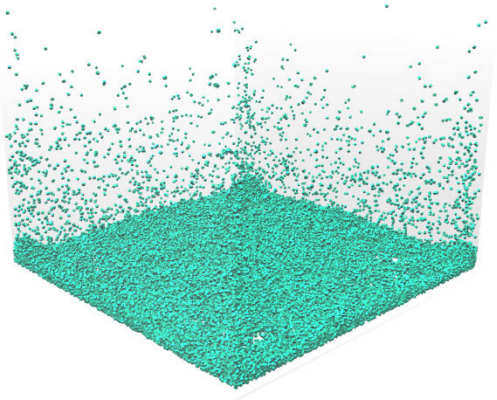
Fig. 7 shows the relationship between the number of detected objects and the detection time of each method. Fig.7(*a*), (*c*), and (*e*) show the CD performance for uniformly sized objects; Fig.7(*b*), (*d*), and (*f*) show the CD performance for non-uniformly sized objects, respectively. Our solution performs well in all test scenarios for different collision detection environments; when the number of objects in the environment is large, the advantages of our method are more manifested. Moreover, our method has at least 40% performance improvement compared to the best performance of the other four, with almost 1X to 2X performance improvement for detecting environments with more objects.



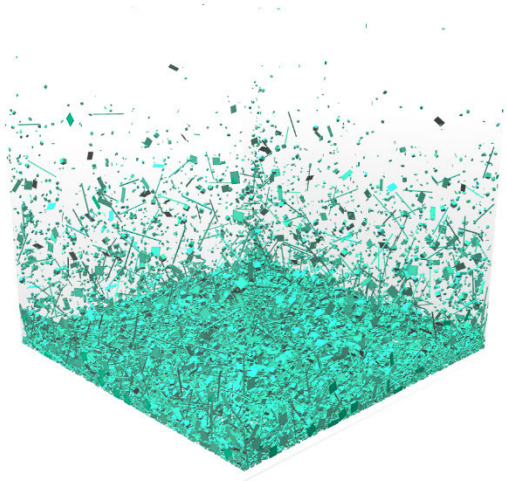
(*a*). *Brownian* - Uniform



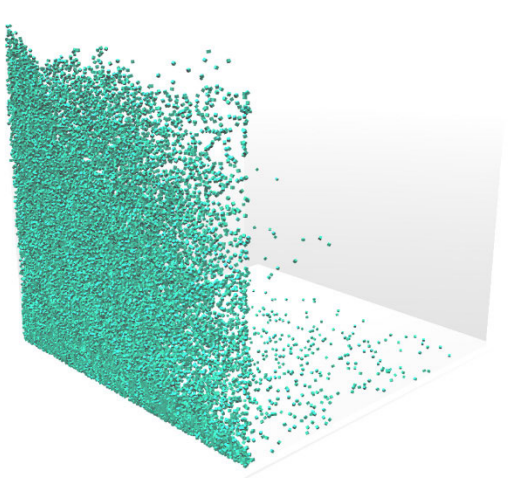
(*b*). *Brownian* - Non-uniform



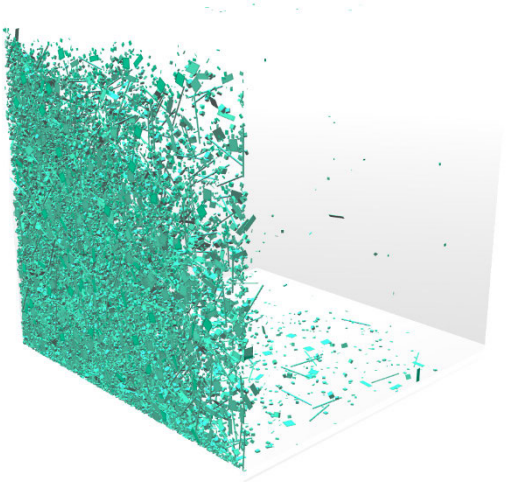
(*c*). *Free Fall* - Uniform



(*d*). *Free Fall* - Non-uniform



(*e*). *Gravity* - Uniform



(*f*). *Gravity* - Non-uniform

FIGURE 5. Collision detection with 32×10^3 objects in all scenes.

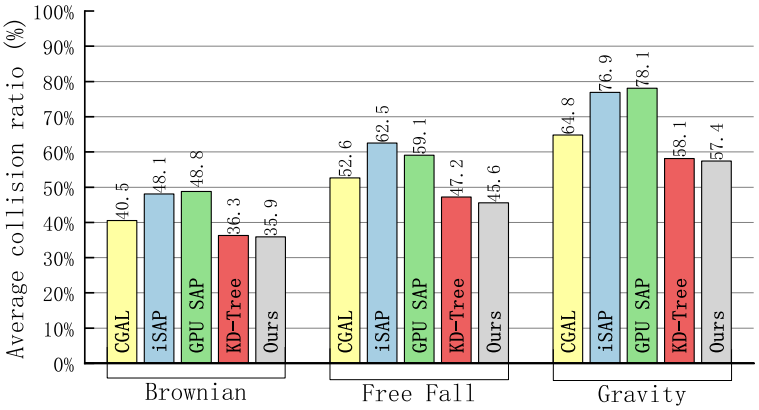


FIGURE 6. Average collision ratio in three scenarios.

TABLE 1. Performance of uniformly sized objects in the three test environments.

Number of Objects($\times 10^3$)		1	2	4	8	16	32	64	128	256	512	1024
Brownian	CGAL	0.19	0.39	1.22	3.42	9.52	23.30	55.79	131.08	344.28	964.69	--
	iSAP	0.13	0.31	0.77	1.91	4.69	12.59	36.64	100.13	301.79	--	--
	GPU SAP	0.32	0.41	0.67	1.21	2.62	5.66	11.46	24.41	60.91	193.97	843.17
	KD-Tree	0.31	0.61	1.03	1.92	4.02	8.46	17.43	36.43	76.48	158.93	370.12
	Ours	0.13	0.21	0.46	1.01	2.14	4.34	9.01	19.06	39.97	84.18	190.24
	Gain	0.0%	47.6%	45.7%	19.8%	22.4%	30.4%	27.2%	28.1%	52.4%	88.8%	94.6%
Free Fall	CGAL	0.49	1.36	4.68	14.91	31.18	66.75	153.48	364.14	922.93	--	--
	iSAP	0.09	0.22	0.56	1.31	3.11	7.56	18.06	47.75	117.37	269.42	754.56
	GPU SAP	0.24	0.41	0.81	1.57	2.93	5.92	13.76	30.16	72.07	218.78	918.72
	KD-Tree	0.08	0.17	0.39	0.79	1.57	3.17	7.24	14.95	29.22	65.52	168.35
	Ours	0.09	0.18	0.37	0.75	1.41	2.76	5.73	11.78	23.19	50.01	119.45
	Gain	-11.1%	-5.6%	5.4%	5.3%	11.3%	14.9%	26.4%	26.9%	26.0%	31.0%	40.9%
Gravity	CGAL	1.57	3.39	7.03	14.82	33.91	70.41	143.47	335.28	811.47	--	--
	iSAP	5.56	12.22	29.34	73.36	190.74	515.24	--	--	--	--	--
	GPU SAP	0.13	0.31	0.69	1.72	4.63	9.71	20.68	42.14	100.91	391.47	--
	KD-Tree	0.58	0.87	1.42	2.97	6.13	14.04	30.81	62.83	138.43	325.71	930.61
	Ours	0.14	0.29	0.59	1.24	2.56	5.39	8.31	17.11	39.97	107.14	371.59
	Gain	-7.1%	6.9%	16.9%	38.7%	80.9%	80.1%	148.9%	146.3%	152.5%	204.0%	150.4%

Table 1 shows the time performance for uniformly sized objects, from 1000 to 1024 $\times$ 10^3 objects; Table 2 shows the time performance for non-uniformly sized objects, from 1000 to 256 $\times$ 10^3 objects. We can see from the two tables that our solution does not reflect the advantages of our algorithm when the number of objects is small; when the number of objects is high, there is a significant acceleration of detection speed. As can be seen from the two tables, our solution does not show the advantage of our algorithm when the number of objects is small; when the number of objects is large, our solution has a significant speedup.

For the environment of *Brownian* with a relatively uniform objects distribution, we can see from Fig. 6 that all methods can show good rejection efficiency; due to a large number of objects in the environment being in motion, Fig. 7(*a*) and Fig. 7(*b*) shows that the performance of iSAP and CGAL are not so good. Table 1 and Table 2 show that our method has nearly 1X performance improvement in the environment of both uniform and non-uniform objects.

Based on the uneven distribution of objects for the *Free Fall* detection environment with a large number of stationary objects, we can see from Fig. 6 that the spatial subdivision based on KD-Tree shows a good rejection efficiency. Fig. 7(*c*) and Fig. 7(*d*) also show that the performance of KD-Tree and ours is significantly higher than other methods. In the environment of uniformly sized objects, our method has a performance improvement of about 40% compared with the best-performing method in the comparison; In the environment with non-uniformly sized objects, our method

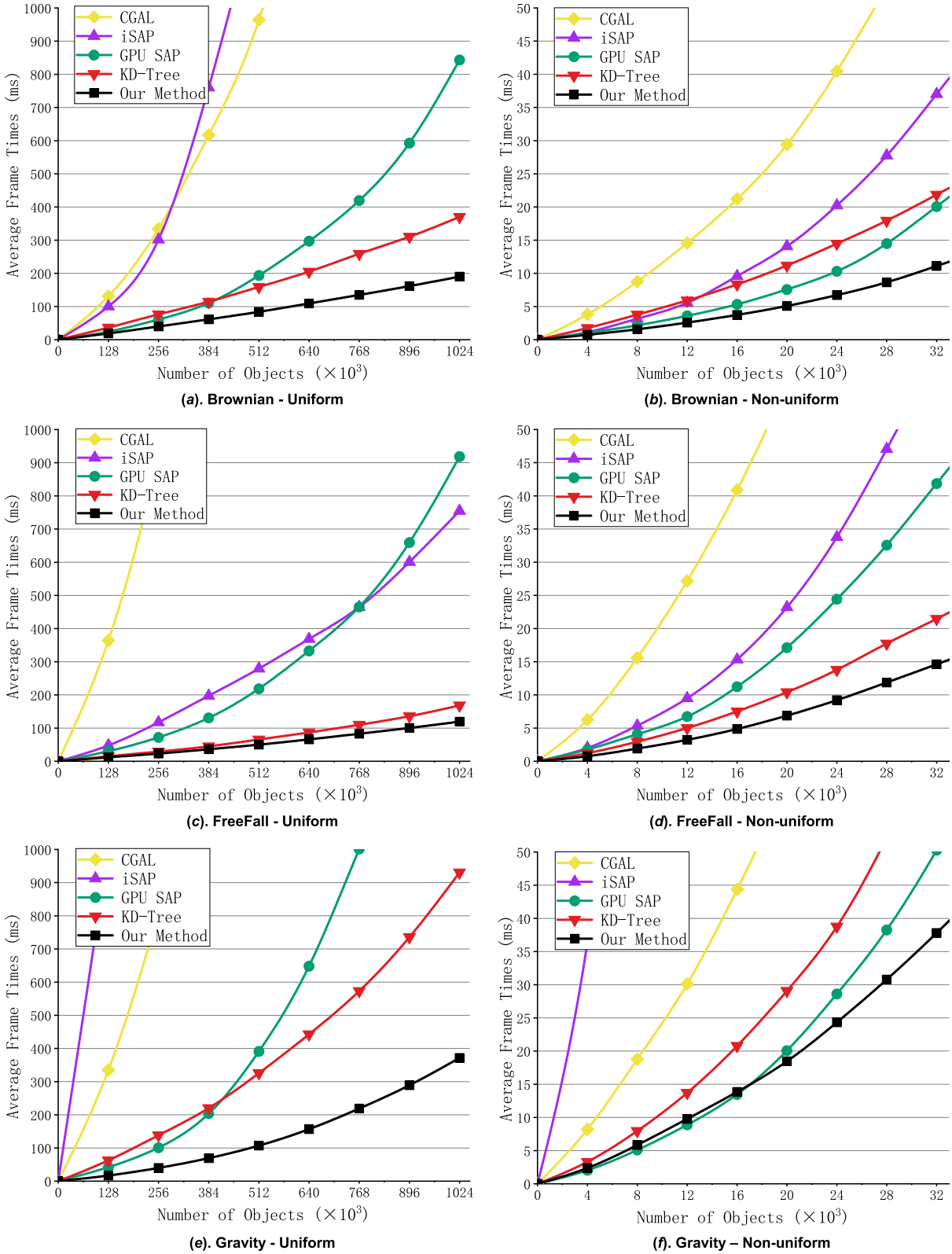


FIGURE 7. Performance comparison of the broad-phase CD algorithm for the environment of uniformly sized objects (left) and non-uniformly sized objects (right).

has nearly 1.2X speedup compared to the best-performing method.

For the more challenging *Gravity* environment, all methods have a high collision ratio due to the characteristics of its environment. From Fig. 7(*e*) and Fig. 7(*f*), we can find that the solution of iSAP and CGAL perform worst, and the detection time of all methods increases exponentially with the number of objects. Nevertheless, our solution can still have

TABLE 2. Performance of non-uniformly sized objects in the three test environments.

Number of Objects($\times 10^3$)		1	2	4	8	16	32	64	128	256
Brownian	CGAL	0.81	1.80	3.81	8.75	21.19	66.21	172.59	503.14	--
	iSAP	0.15	0.40	1.13	3.15	9.58	37.01	116.74	344.55	--
	GPU SAP	0.20	0.41	1.02	2.17	5.34	20.07	59.47	213.32	896.09
	KD-Tree	0.37	0.79	1.76	3.78	8.34	21.86	54.68	134.62	314.38
	Ours	0.15	0.32	0.73	1.57	3.74	11.10	29.67	68.38	156.67
	Gain	-2.8%	24.8%	39.7%	38.2%	42.8%	80.8%	84.3%	96.9%	100.7%
Free Fall	CGAL	1.43	3.07	6.24	15.59	40.87	111.74	291.98	891.09	--
	iSAP	0.24	0.68	2.01	5.37	15.33	60.96	160.36	465.17	--
	GPU SAP	0.41	0.88	1.77	4.07	11.24	41.87	120.91	456.92	--
	KD-Tree	0.20	0.48	1.21	2.94	7.51	21.46	65.37	188.68	489.16
	Ours	0.14	0.31	0.76	1.91	4.87	14.61	39.29	95.64	220.73
	Gain	42.9%	54.8%	59.2%	53.9%	54.2%	46.9%	66.4%	97.3%	121.6%
Gravity	CGAL	1.98	4.07	8.21	18.76	44.37	135.64	364.58	963.62	--
	iSAP	5.67	13.81	36.38	99.04	299.46	--	--	--	--
	GPU SAP	0.41	0.96	2.04	5.09	13.48	50.29	186.59	634.67	--
	KD-Tree	0.71	1.47	3.31	7.97	20.78	68.14	216.43	537.24	1321.57
	Ours	0.51	1.11	2.37	5.84	13.85	37.81	107.67	296.81	756.28
	Gain	-19.6%	-13.5%	-13.9%	-12.8%	-2.7%	33.0%	73.3%	81.0%	74.7%

better performance. In the collision detection environment of uniformly sized objects, our method has nearly 2X performance improvement compared with other best-performing methods; in the environment of non-uniform objects, our method achieves nearly 80% performance improvement.

We compare the GPU-based CD method with our solution, and as seen in Fig. 7, the GPU SAP method performs well when the number of objects is small; Its performance decays exponentially when the number of objects in the environment increases. The performance of the GPU-based solution also relies on the expensive data exchange between the GPU and the CPU, and it still suffers from the high memory latency in the GPU. In particular, reading the configuration of objects from global memory and writing the collision results back to global memory is still expensive due to the CUDA architecture [37].

The same CD methods using grid-based spatial subdivision and SAP, the GPU SAP method has a significant performance improvement compared to the iSAP method. Compared with uniformly sized objects, CDs with non-uniformly sized objects test the algorithm's performance and require the support of excellent computer hardware. However, under our hardware conditions, the performance of our solution is also superior.

VII. CONCLUSION AND FUTURE WORK

This paper proposes a general, scalable broad-phase CD algorithm with the basic technical framework of spatial subdivision and SAP. It adopts the spatial subdivision solution of KD-Tree and pre-sorts the objects before constructing KD-Tree to facilitate the subsequent construction of KD-Tree and faster execution of the SAP algorithm while improving the SAP algorithm; We also introduce a solution for temporal inference to avoid sweeping and pruning between static objects. Our solution can be applied to various collision detection environments, including complex environments with unevenly distributed objects and non-uniformly sized objects, and can achieve collision detection of more than one million objects. Experimental results show that our solution has similar performance to the compared methods in environments with a small number of objects; when the number of objects is enormous, it has up to 100.7% performance improvement with the best-performing method of the compared methods under the test conditions of typical cases, and up to 204.0% performance improvement with the best-performing method of the compared methods in harsher and more complex environments improvement.

In future research, we will extend our work in several directions. First, we are working to combine the solution of this thesis with narrow-phase CD to maximize the CD rate and propose a general and complete CD method. Second, we will introduce GPUs to accelerate our solution further to improve the CD efficiency and apply it to collision detection of a more significant number of objects.

REFERENCES

[1] M. Wang and J. Cao, "A review of collision detection for deformable objects," *Comput. Animation Virtual Worlds*, vol. 32, no. 5, pp. 1–10, Sep. 2021.

[2] S. Kockara, T. Halic, K. Iqbal, C. Bayrak, and R. Rowe, "Collision detection: A survey," in *Proc. IEEE Int. Conf. Syst., Man Cybern.*, Montreal, QC, Canada, Oct. 2007, pp. 4046–4051, doi: 10.1109/ICSMC.2007.4414258.
[3] G. V. D. Bergen, "Efficient collision detection of complex deformable models using AABB trees," *J. Graph. Tools*, vol. 2, no. 4, pp. 1–13, Jan. 1997.
[4] P. M. Hubbard, "Collision detection for interactive graphics applications," *IEEE Trans. Vis. Comput. Graphics*, vol. 1, no. 3, pp. 218–230, Sep. 1995.
[5] R. de Sousa Rocha and M. A. F. Rodrigues, "An evaluation of a collision handling system using sphere-trees for plausible rigid body animation," in *Proc. ACM Symp. Appl. Comput.*, Mar. 2008, pp. 1241–1245.
[6] S. Gottschalk, M. C. Lin, and D. Manocha, "OBBTree: A hierarchical structure for rapid interference detection," in *Proc. 23rd Annu. Conf. Comput. Graph. Interact. Techn.*, 1996, pp. 171–180.
[7] J. T. Klosowski, M. Held, J. S. B. Mitchell, H. Sowizral, and K. Zikan, "Efficient collision detection using bounding volume hierarchies of k-DOPs," *IEEE Trans. Vis. Comput. Graph.*, vol. 4, no. 1, pp. 21–36, 1998.
[8] G. Zachmann, "Rapid collision detection by dynamically aligned DOP-trees," in *Proc. IEEE Virtual Reality Annu. Int. Symp.*, Atlanta, GA, USA, Apr. 1998, pp. 90–97, doi: 10.1109/VRAIS.1998.658428.
[9] X. Wang, M. Tang, D. Manocha, and R. Tong, "Efficient BVH-based collision detection scheme with ordering and restructuring," *Comput. Graph. Forum*, vol. 37, no. 2, pp. 227–237, May 2018.
[10] L. Wei, H. Zhou, and S. Nahavandi, "Haptic collision detection on disjoint objects with overlapping and inclusive bounding volumes," *IEEE Trans. Haptics*, vol. 11, no. 1, pp. 73–84, Jan. 2018.
[11] R. Weller, N. Debowski, and G. Zachmann, "KDet: Parallel constant time collision detection for polygonal objects," *Comput. Graph. Forum*, vol. 36, no. 2, pp. 131–141, May 2017.
[12] G. Capannini and T. Larsson, "Adaptive collision culling for massive simulations by a parallel and context-aware sweep and prune algorithm," *IEEE Trans. Vis. Comput. Graphics*, vol. 24, no. 7, pp. 2064–2077, Jul. 2018.
[13] B. Qi and M. Pang, "An enhanced sweep and prune algorithm for multi-body continuous collision detection," *Vis. Comput.*, vol. 35, no. 11, pp. 1503–1515, Nov. 2019.
[14] F. Liu and Y. J. Kim, "Exact and adaptive signed distance fields computation for rigid and deformable models on GPUs," *IEEE Trans. Vis. Comput. Graph.*, vol. 20, no. 5, pp. 714–725, May 2014.
[15] D. Koschier, C. Deul, M. Brand, and J. Bender, "An hp-adaptive discretization algorithm for signed distance field generation," *IEEE Trans. Vis. Comput. Graph.*, vol. 23, no. 10, pp. 2208–2221, Oct. 2017.
[16] H. Xu and J. Barbic, "6-DoF haptic rendering using continuous collision detection between points and signed distance fields," *IEEE Trans. Haptics*, vol. 10, no. 2, pp. 151–161, Apr. 2017.
[17] H. Wang, "Defending continuous collision detection against errors," *ACM Trans. Graph.*, vol. 33, no. 4, pp. 1–10, Jul. 2014.
[18] M. Tang, T. Wang, Z. Liu, R. Tong, and D. Manocha, "I-cloth: Incremental collision handling for GPU-based interactive cloth simulation," *ACM Trans. Graph.*, vol. 37, no. 6, pp. 1–10, Dec. 2018.
[19] M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrmann, and M.-P. Cani, "Collision detection for deformable objects," *Comput. Graph. Forum*, vol. 24, no. 1, pp. 61–81, 2005.
[20] E. Coumans and Y. Bai. (2022). *Pybullet: A Python Module for Physics Simulation for Games, Robotics and Machine Learning*. [Online]. Available: http://pybullet.org
[21] NVIDIA. *NVIDIA PHYSX SDK 3.3.4 Documentation*. [Online]. Available: https://gameworksdocs.nvidia.com/PhysX/3.3/PhysXGuide/Index.html
[22] (2022). *CGAL User and Reference Manual*. CGAL Editorial Board. [Online]. Available: https://doc.cgal.org/5.4.3/Manual/packages.html
[23] R. G. Luque, J. L. D. Comba, and C. M. D. S. Freitas, "Broad-phase collision detection using semi-adjusting BSP-trees," in *Proc. Symp. Interact. 3D Graph. Games*, Apr. 2005, pp. 179–186.
[24] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, no. 9, pp. 509–517, Sep. 1975.
[25] Y. R. Serpa and M. A. F. Rodrigues, "Flexible use of temporal and spatial reasoning for fast and scalable CPU broad-phase collision detection using KD-trees," *Comput. Graph. Forum*, vol. 38, no. 1, pp. 260–273, Feb. 2019.
[26] T. H. Wong, G. Leach, and F. Zambetta, "An adaptive octree grid for GPU-based collision detection of deformable objects," *Vis. Comput.*, vol. 30, nos. 6–8, pp. 729–738, Jun. 2014.
[27] Q. Avril, V. Gouranton, and B. Arnaldi, "Fast collision culling in large-scale environments using GPU mapping function," in *Proc. Eurograph. Symp. Parallel Graph. Vis.*, May 2012, pp. 71–80, doi: 10.2312/EGPGV/EGPGV12/071-080.
[28] D. Barbieri, V. Cardellini, and S. Filippone, "Fast uniform grid construction on GPGPUs using atomic operations," in *Proc. Adv. Parallel Comput.*, Sep. 2014, pp. 295–304.
[29] W. Fan, B. Wang, J.-C. Paul, and J. Sun, "A hierarchical grid based framework for fast collision detection," *Comput. Graph. Forum*, vol. 30, no. 5, pp. 1451–1459, Aug. 2011.
[30] M. T. B. H. M. Müller, D. Pomeranets, and M. Gross, "Optimized spatial hashing for collision detection of deformable objects," in *Proc. VMV*, Nov. 2003, pp. 1–14.
[31] H. Mazhar, T. Heyn, and D. Negrut, "A scalable parallel method for large collision detection problems," *Multibody Syst. Dyn.*, vol. 26, no. 1, pp. 37–55, Jun. 2011.
[32] M. Hapala and V. Havran, "Review: Kd-tree traversal algorithms for ray tracing," *Comput. Graph. Forum*, vol. 30, no. 1, pp. 199–213, 2015.
[33] Y. R. Serpa and M. A. F. Rodrigues, "Broadmark: A testing framework for broad-phase collision detection algorithms," *Comput. Graph. Forum*, vol. 39, no. 1, pp. 436–449, Feb. 2020.
[34] J. D. Cohen, M. C. Lin, D. Manocha, and M. Ponamgi, "I-collide: An interactive and exact collision detection system for large-scale environments," in *Proc. Symp. Interact. 3D Graph.*, Monterey, CA, USA, 1995, p. 189.
[35] Q. Avril, V. Gouranton, and B. Arnaldi, "Collision detection: Broad phase adaptation from multi-core to multi-GPU architecture," *J. Virtual Reality Broadcast.*, vol. 10, pp. 1–13, Jan. 2014.
[36] D. J. Tracy, S. R. Buss, and B. M. Woods, "Efficient large-scale sweep and prune methods with AABB insertion and removal," in *Proc. IEEE Virtual Reality Conf.*, Mar. 2009, p. 191.
[37] F. Liu, T. Harada, Y. Lee, and Y. J. Kim, "Real-time collision culling of a million bodies on graphics processing units," *ACM Trans. Graph.*, vol. 29, no. 6, pp. 1–8, Dec. 2010.
[38] T. H. Wong, G. Leach, and F. Zambetta, "Virtual subdivision for GPU based collision detection of deformable objects using a uniform grid," *Vis. Comput.*, vol. 28, nos. 6–8, pp. 829–838, Jun. 2012.
[39] Y. Cao, H. Wang, W. Zhao, B. Duan, and X. Zhang, "A new method to construct the KD tree based on presorted results," *Complexity*, vol. 2020, pp. 1–7, Dec. 2020.
[40] R. A. Brown, "Building a balanced k-d tree in O(kn log n) time," *J. Comput. Graph. Techn.*, vol. 4, no. 1, pp. 50–68, 2015.
[41] D. Baraff, "Dynamic simulation of nonpenetrating rigid bodies," Cornell Univ., Ithaca, NY, USA, Tech. Rep., 1992. [Online]. Available: https://hdl.handle.net/1813/7115
[42] L. Kettner, A. Meyer, and A. Zomorodian. (2022). *CGAL-5.3.2 User Reference Manual*. [Online]. Available: https://doc.cgal.org/5.3.2/Box_intersection_d/index.html
[43] A. Zomorodian and H. Edelsbrunner, "Fast software for box intersections," in *Proc. 16th Annu. Symp. Comput. Geometry*, May 2000, pp. 129–138.

JIAQI CAO was born in Harbin, China, in 1995. He is currently pursuing the Ph.D. degree with the Mechatronic Engineering Department, Harbin University of Science and Technology. His research interests include animation simulation, image processing, and computer vision.

MONAN WANG received the B.S. and M.S. degrees in mechatronic engineering from the Harbin University of Science and Technology, in 1995 and 2000, respectively, and the Ph.D. degree in automation from Harbin Engineering University, in 2004. From 2004 to 2007, she was an Associate Professor. Since 2007, she has been a Professor with the Mechatronic Engineering Department, Harbin University of Science and Technology. Her research interests include the simulation of the dynamic process of bone fracture healing and computer-aided medical treatment.