Time Advancement and Bounds Intersection Checking for Faster Broad-Phase Collision Detection of Paired Object Trajectories

Proceso L. Fernandez Jr

Mary Aveline Germar

International Journal of Advances in Electronics and Computer Science, ISSN(p): 2394-2835 Volume-5, Issue-5, May.-2018
http://iraj.in

TIME ADVANCEMENT AND BOUNDS INTERSECTION CHECKING FOR FASTER BROAD-PHASE COLLISION DETECTION OF PAIRED OBJECT TRAJECTORIES

[1]MARY AVELINE GERMAR, [2]PROCESO FERNANDEZ

Department of Information Systems and Computer Science Ateneo de Manila University Philippines

Abstract- For self-driving mechanisms, the motion planning requires a reasonably fast algorithm for collision detection along the trajectories. We present three algorithms for the detection of collision among objects with predefined trajectories. The first algorithm uses the intersection of the path's bounding box. The second algorithm sequentially checks for intersection between each pair of corresponding axis-aligned bounding boxes (AABB) from the trajectories of the two paths. Lastly, the latter algorithm is modified using iterative time advancement to an estimated earliest possible collision time. Simulation experiments on a variety of pair trajectories demonstrate a significant speedup of the proposed algorithms over the existing baseline algorithm. They are, therefore, preferable alternatives for faster broad-phase collision detection in applications such as motion planning.

I. INTRODUCTION

Collision detection is commonly used in video games, simulations, and robotics. For motion planning, however, collision detection is still at its infancy. Common real-time collision detection algorithms, especially in video games, are used to determine collisions within a short span of time. For motion planning, however, the entire trajectories of the objects must be considered for early collision avoidance. Since trajectories may be curved, there is an additional complexity of handling possible collisions in between the initial and final orientations of the objects.

There are two stages of collision detection: broad-phase and narrow-phase. First, the broad-phase stage selects the candidates for the narrow-phase by removing objects that are certain to not intersect. In this stage, the object shapes may be simplified into bounding volumes. Then, the candidate objects undergo a narrow-phase collision detection that uses the exact shapes and the precise time of movements to determine if there is collision.

For the broad-phase collision detection, a popular technique is workspace-time bounding volume hierarchy (BVH) that contains a collection of bounding volumes for the object at specified time segments [3]. The bottleneck in this technique, however, lies in the creation of the BVH tree. To eliminate such bottleneck, an array or list data structure may be used to identify the configuration of the object in sequential order. With this, the objects can be directly checked for collision in each timestep.
In addition to the sequential data structure, the earliest possible collision time can also be used to advance the collision checking, and skip the time frames that are sure to have no collision. When a collision is detected on the advanced time segment, an early exit could be achieved.

II. RELATED STUDIES

Collision detection algorithms are often separated into two phases [1]. The initial phase, called the broad phase, uses basic shapes, such as spheres or boxes, which are large enough to encapsulate the entire object. These shapes are then checked for collision to remove the non-colliding objects and retain only the candidate pairs of objects for further checking. Afterwards, the narrow phase considers the actual shape and orientation of the candidate pairs. This phase is often more computationally expensive depending on the shape of the object.

A. Broad-phase Collision Algorithms

In the broad-phase collision detection, the two most common broad-phase techniques are spatial partitioning and bounding volume hierarchies.
In spatial partitioning, the whole space is divided into sections that are tested for collision. The simplest method is the sort-and-sweep techniques [1]. Axis-aligned bounding boxes (AABB) for the shapes are sorted according to either the x-axis or y-axis. Then, the bounds of the boxes are checked for overlaps in the selected axis. However, the clustering of the objects on an axis can make this method ineffective. Instead of focusing on a single axis, other spatial partitioning methods such as the k-d trees and binary spatial partitioning (BSP) trees use all the dimensions [10]. Since the space and not the object themselves are subdivided, the same pair of objects may be detected more than once on different partitions.

On the other hand, bounding volume hierarchy (BVH) removes the redundancy because the BVH considers the object itself and partitions its trajectory according to the bounding volume for the object in each partition. Consequently, the number of comparisons is reduced in a logarithmic scale. The commonly used shapes of bounding volumes are AABB, circle, and an oriented bounding box. There is no optimal shape for

all cases because the results depend on the shape and orientation of the objects being compared [1]. Compared to the BSP, the BVH may take longer to build depending on the tree depth. However, each object is only checked once for collision.
Due to the different advantages of both algorithms, the combination of spatial partitioning and BVH has been proposed as a Split BVH (SBVH) [4]. For each split, the SBVH decides whether to use a spatial split like BSP or an object split common in BVH trees.

B. Narrow-phase Collision Algorithms

The pitfall of the broad-phase algorithms is the possibility of false positives, since they often use the simplified version of an object's contours. Consequently, the narrow-phase collision algorithms use the actual shape of the object to provide more accuracy.

An example of the narrow-phase collision algorithm is the conservative time advancement (CA) [11, 12]. The CA can be used to calculate the earliest time of collision in between the initial time and final time by repeatedly advancing the objects by a given delta time (Δt) in between. The delta time is computed using the closest distance between the objects, since the time is a function of the distance. In the controlled conservative time advancement [8], an interpolation is used on candidate objects to determine the Δt. Unfortunately, the speed of the algorithm depends on several factors, such as the number of the time divisions, the threshold minimum distance, and the maximum iteration count. Moreover, the conservative time advancement is discrete and may fail to cover the intermediate positions between time steps if the object moves very fast. The main solution to this problem is to divide the Δt into smaller partitions, but this increases the computation time.

For narrow-phase collision detection, another issue is the actual shapes of the objects. AABBs can be easily checked for collision by determining the maximum and minimum values on each axis. For complex polygons, however, more complex algorithms are required. The Gilbert–Johnson–Keerthi (GJK) distance algorithm can be used for collision checking of convex polygons [9].

C. Data Structures

The polygonal objects are often represented as a set of points. For the broad-phase, however, the object silhouette is simplified using bounding volumes. For selecting a bounding volume, the things considered are: "inexpensive intersection tests, tight fitting, inexpensive to compute, easy to rotate and transform, and use little memory [1]."

In bounding volumes, the shape determines the necessary data. For example, a sphere is represented by the position of the center as a point with its corresponding coordinates, and its radius [1]. Another more common representation of a bounding box is the axis-aligned bounding boxes (AABB), which means that each edge of the boxes is parallel to an axis.

Besides the shape representation, a given algorithm defines the necessary data structure. For example, both BSP and BVH use trees to separate the elements of the spatial and object partitions, respectively. The contents of those trees, however, depend on the algorithm used. In BSP, the tree contains the whole objects; whereas, in BVH, the tree is for a single object that is partitioned into its smaller parts.

Another important consideration is the representation of the time element. To represent the area covered by an object over a span of time, swept volumes are used [13]. These swept volumes can be easily placed in the workspace, which is the two-dimensional space for the objects. However, the swept volumes are only accurate when it covers a short span of time. In order to represent a non-linear trajectory, a sequence of swept volumes is necessary. Thus, the workspace-time obstacle region (WT) is used by Schwesinger et al [3]. The workspace-time allows each object trajectory to be represented with the respective object configuration and time configuration.

D. Motion Planning

Motion planning requires an evaluation of the entire paths of the objects. This means that collision must be detected in multiple time steps throughout the time span of the object's entire path.

Ferguson et al. [2] proposed a technique that has three stages. First, the axis-aligned bounding volume of each paths are compared to one another for collision. The second stage uses individual time step advancement using a circular bounding volume. Starting from an initial time, the objects are moved by time step into the next position until either a collision between the circular bounding volumes is detected or the path is completed without collision. If there is collision, the last stage uses an oriented bounding box for each time step in a similar advancement technique in the previous stage.
The algorithm considers the time in the collision detection of paths. However, the proposed method uses discrete collision detection, which may fail to detect collision in between the time steps. Moreover, the multiple time step advancement may involve more computation time especially when the collision occurs at the end of the path.

An alternative approach has considered the time by using a workspace-time representation of the bounding volume hierarchy [3]. In the algorithm, the time becomes an additional dimension to the position of the object. The technique also considers continuous

collision detection using convex hulls that cover the area between the time steps. The main improvement of the technique comes from the usage of BVH trees for each path. The BVH trees reduce the comparisons required between each object pair.
The given technique, however, requires the creation of a bounding volume hierarchy for each path, and subsequently, a comparison of each of the BVH tree.In this study, we consider improving the broad-phase collision detection for motion planning.

III. TECHNIQUES

In this paper we propose three algorithms that use the following techniques: the bounding volume intersection (BVI), synchronized intersection (SI), and the speed-time advancement (STA). For each of these methods, the continuous collision detection uses a workspace-time (WT) axis aligned boxes (AABBs), which we describe here first.

A. Workspace-Time AABBs

Since the trajectory of an object may be curved, getting the positions at the initial and final times may fail to include a portion of the curve. Consequently, the path area is divided according to a time step. For each segment, the position of the object in the minimum and maximum times of the time step is computed. Then, the vertices are gathered to determine the maximum and minimum values for each axis. These values determine an AABB for the time step, as illustrated by a dashed box in Fig. 1.

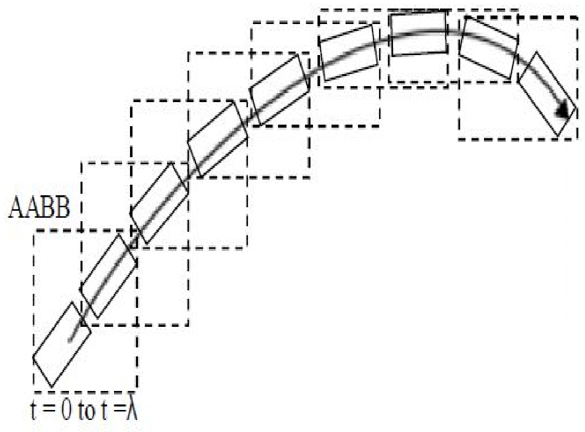


Figure 1: WT-AABB

The area covered by the entire path is then considered by gathering all the AABBs from the initial time to the multiple of the time step that is greater than or equal to the maximum time of movements for all paths. The collection of these boxes is then called the workspace-time (WT) AABBs (See Fig. 1).

B. Bounding Volume Intersection Algorithm

The bounding volume intersection (BVI) algorithm eliminates the WT-AABBs that are certain to not intersect. This is done through the following steps:

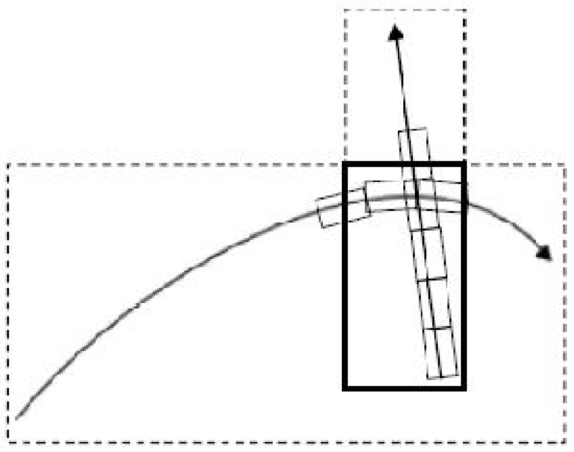


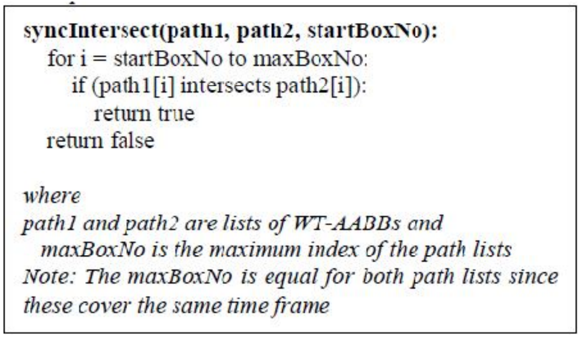
Figure 2: Intersection WT-AABBs

1. The bounding volumes of entire paths are obtained. For this stage, the minimum and maximum bounds of each path is obtained from all its WT-AABBs.
2. Then, the bounding volume of the intersection between the two object paths is obtained using the minimum and maximum values of the two paths' bounding volumes.
3. The WT-AABBs of the object trajectories within the intersection are then used for the BVH collision detection.

Depending on the bounding volume intersection, the AABBs for the BVH tree creation and collision detection would ideally reduce the computation time for collision detection.

C. Synchronized Intersection Checking

Since the WT-AABBs of each path is an array of AABBs, the synchronized intersection (SI) technique can take advantage of the WT-AABBs with an ordered time sequence. In each time step, the SI checks for the of WT-AABB pair of each trajectory for intersection. The computation complexity will only loop once through all WT-AABB pairs, in which each loop will have a basic AABB intersection test.



```
syncIntersect(path1, path2, startBoxNo):
    for i = startBoxNo to maxBoxNo:
        if (path1[i] intersects path2[i]):
            return true
    return false

where
path1 and path2 are lists of WT-AABBs and
    maxBoxNo is the maximum index of the path lists
Note: The maxBoxNo is equal for both path lists since
these cover the same time frame
```

Figure 3: Synchronized Intersection (SI) Algorithm

D. Speed-Time Advancement

Like the BVI, the speed-time advancement (STA) algorithm also attempts to reduce the WT-AABBs. In contrast to the SI, the main principle of the STA is to quickly advance through the earliest possible time of collision and check for actual movement from the calculated earliest possible collision time. If the earliest possible time of collision is within the current time step or the immediate next time step, the advancement is just minimal. Consequently, the SI is used for the succeeding WT-AABBs. The STA is a loop that stops when one of the following conditions is met: collision is detected or the end of the paths is reached.

STA is accomplished according to the following steps:

1. The minimum distance between the two AABBs is obtained for both the x-axis and y-axis. Since the formula for the actual distance has a computationally costly square root function, we simply estimate the distance using the maximum between the x and y distances.

$$Actual\ Distance = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

$$Estimated\ Distance = \max\left(|x_1 - x_2|, |y_1 - y_2|\right)$$

where (x_1, y_1) *and* (x_2, y_2) *are the closest points from their respective AABBs*

2. The estimated distance together with the maximum speeds of each object are used to compute the earliest possible time for a collision between the AABBs. This step disregards the actual direction of the WT-AABB, in order again to simplify the computations.
3. Both objects are then advanced from the current time to the earliest possible collision time.

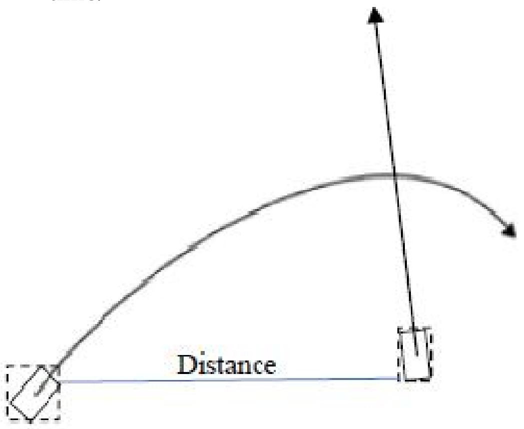


Figure 3: WT-AABBs at initial time

The advancement is done easily because the uniform time step indexing in the vectors containing the space-time AABBs enables quick computation for the correct target index.

4. Steps 1-3 are repeated until one of the following conditions is met:
   a. If the earliest time of collision is the current or immediate next time step, SI will be used henceforth.
   b. The WT-AABBs of the objects collide at the calculated earliest collision time.

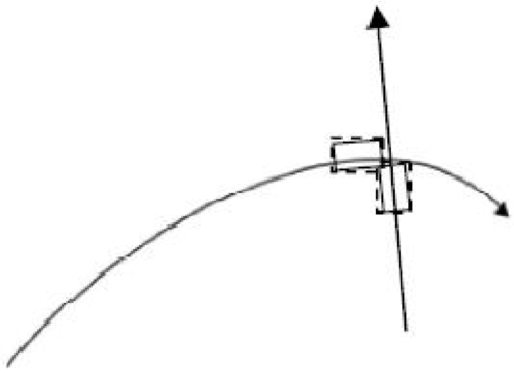


Figure 4: Collision detected

   c. The end time is reached without collision

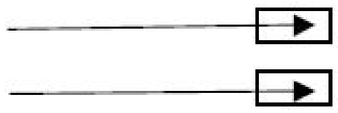
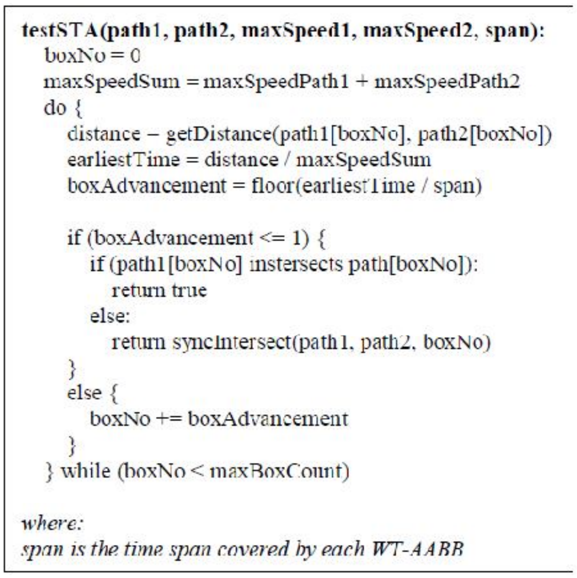


Figure 5: End of Path Reached



```
testSTA(path1, path2, maxSpeed1, maxSpeed2, span):
  boxNo = 0
  maxSpeedSum = maxSpeedPath1 + maxSpeedPath2
  do {
    distance – getDistance(path1[boxNo], path2[boxNo])
    earliestTime = distance / maxSpeedSum
    boxAdvancement = floor(earliestTime / span)

    if (boxAdvancement <= 1) {
      if (path1[boxNo] instersects path[boxNo]):
        return true
      else:
        return syncIntersect(path1, path2, boxNo)
    }
    else {
      boxNo += boxAdvancement
    }
  } while (boxNo < maxBoxCount)

where:
span is the time span covered by each WT-AABB
```

Figure 6: Speed-Time Advancement (STA) algorithm

IV. METHODOLOGY

For this study, the 2D shape representation of the top view of the object is considered. The third dimension is the time elapsed from the start of the object's path.

A. Algorithms

Different combinations of the mentioned techniques are used to form the algorithms used proposed in this study. The baseline algorithm is the BVH tree using the median as the splitting point. The following algorithms are proposed:

1. The first algorithm uses both BVI and BVH (BVI-BVH). The intersection of the volumes is obtained, and then used for the BVH.
2. The second algorithm uses SI only.
3. The third algorithm is STA, which could revert to SI when the earliest possible time of collision is within the current time step.

B. Data Set
The data set contains several test cases, with a pair of trajectories for each test. The simulations used two kinds of paths: straight and curved. For both kinds of paths, the experiment tested the following collision scenarios: (1) no intersection, (2) no collision but with WT-AABB intersection, and (3) collision cases. For straight path, the collision cases include collision at the start, middle, and end of the path. For the curved paths, the start and end points are randomly generated. Then, cubic spline interpolation is used to create the series of points along the curve.

C. Simulations
The simulations test the performance of the proposed algorithms compared to the reference BVH algorithm that includes tree creation and comparison. The tests were run 30 times for each scenario, and the lowest, average, and highest execution times for each scenario were recorded.

V. RESULTS AND ANALYSIS

For all paths, the BVI-BVH, SI, and STA algorithms performed better than the standard BVH. In general, the SI and STA are better than the BVI-BVH. Comparing the SI and STA, however, both average execution times are around the same range with the STA just slightly faster in most cases.

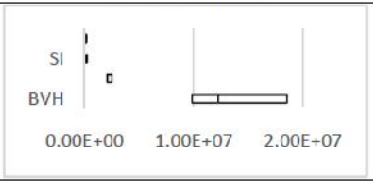
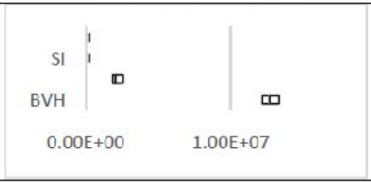
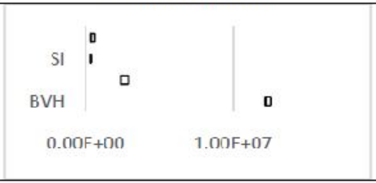
Straight Paths		
Collision near Start	Collision near Middle	Collision near End
		

Table 1: Results for Straight Path Collisions (Execution time in nanoseconds)

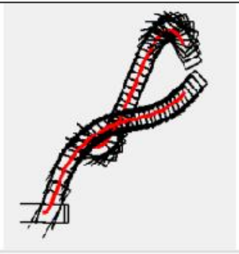
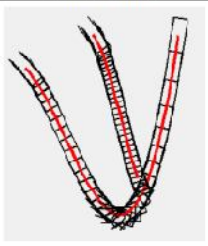
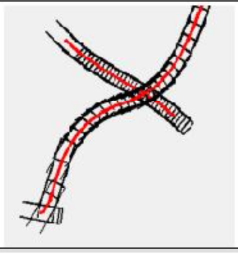
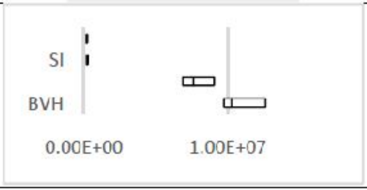
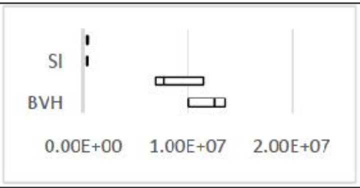
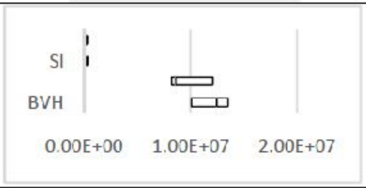
Curve Paths		
		
		

Table 2: Curve Paths Collision Results (Execution time in nanoseconds)

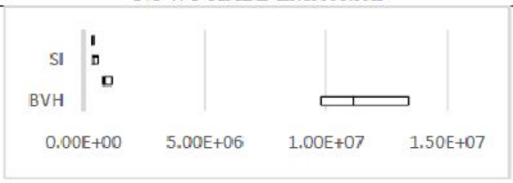
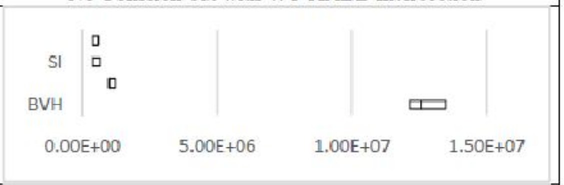
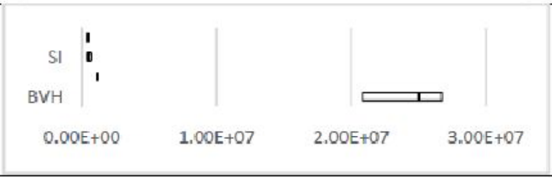
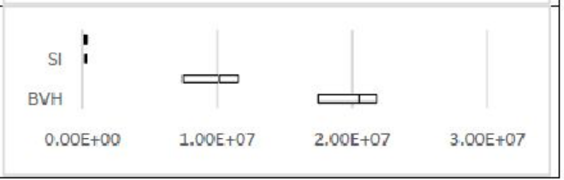
	No WT-AABB Intersection	No Collision but with WT-AABB Intersection
Straight Paths		
Curve Paths		

Table 3: Results for No Collision (Execution time in nanoseconds)

When there is no collision, there are two possible scenarios. First, when there is an intersecting area in the straight paths for the BVI, the performance is better than the collision cases. When there is no intersection on the parallel paths, the BVI significantly performed better than the intersection cases.

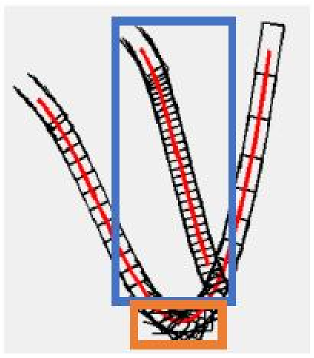


Figure 7: Separated Arcs (Curve 2)

On the curve paths, the SI and STA have more significant speedup than the BVI-BVH algorithm. Because of the curved movements, the BVI-BVH tends to include a large portion of some paths or in some cases, even includes the whole trajectory of one object. If the intersection cuts off an arc from a trajectory, the resulting BVH may be significantly unbalanced in terms of time coverage for those that include the bounding boxes on the both sides of the split arc. Consequently, the larger time area may detect more collision on the tree hierarchy rather than an early rejection on the early levels of the tree.

On the other hand, SI and STA performed better because these can detect the earliest collision without the need to create BVH trees. The STA also strategically advances to the next possible collision time without the false positives on the early levels of the BVI-BVH algorithm. The STA is slightly faster than SI in most cases, but the difference does not seem significant because the resulting ranges of time values are overlapping with one another. For the STA, the computation of the earliest possible time reduces the benefit of the time advancement. In SI, there is no such computation, but a direct comparison of corresponding AABBs in the array.

CONCLUSION

In this paper we propose three algorithms for broad-phase collision detection of paired object trajectories. These algorithms – BVI-BVH, SI, and STA – have been shown, through simulation, to perform better than the standard BVH. However, the performance of the BVI-BVH relies heavily on the trajectory shapes and may perform poorly if the resulting intersection between the trajectories covers a large number of the WT-AABBs. The SI and STA are faster than the BVI-BVH because they take advantage of the array data structure for the sequence of WT-AABBs. Unlike the BVI, the SI is unaffected by the shapes of the trajectories. Finally, the STA performs slightly better than the SI, based on average computation times. However, the range of computation times of the SI and STA are very similar because of the mentioned tradeoffs in the computations. Future research may explore further fine-tuning these algorithms.

REFERENCES

[1] Ericson, C. (2004). Real-time collision detection. CRC Press.

[2] Ferguson, D., Darms, M., Urmson, C., & Kolski, S. (2008, June). Detection, prediction, and avoidance of dynamic obstacles in urban environments. In Intelligent Vehicles Symposium, 2008 IEEE (pp. 1149-1154). IEEE.

[3] Schwesinger, U., Siegwart, R., & Furgale, P. (2015, May). Fast collision detection through bounding volume hierarchies in workspace-time space for sampling-based motion planners. In Robotics and Automation (ICRA), 2015 IEEE International Conference on (pp. 63-68). IEEE.

[4] Stich, M., Friedrich, H., & Dietrich, A. (2009, August). Spatial splits in bounding volume hierarchies. In Proceedings of the Conference on High Performance Graphics 2009 (pp. 7-13). ACM.

[5] Bonsai: Rapid Bounding Volume Hierarchy Generation using Mini Trees. (2015). Journal of Computer Graphics Techniques (JCGT), (3), 23.

[6] Ayellet, T., Ilan, S., & David P., D. (2013). Temporal Coherence in Bounding Volume Hierarchies for Collision Detection.

[7] A fast spatial partition method in bounding volume hierarchy. (2013). 2013 IEEE 4th International Conference on Software Engineering and Service Science, Software Engineering and Service Science (ICSESS), 2013 4th IEEE International Conference on, 15.

[8] Tang, M., Kim, Y. J., & Manocha, D. (2009, May). C 2 A: controlled conservative advancement for continuous collision detection of polygonal models. In Robotics and Automation, 2009. ICRA'09. IEEE International Conference on (pp. 849-854). IEEE.

[9] Gilbert, E., Johnson, D., & Keerthi, S. (1987, March). A fast procedure for computing the distance between complex objects in three space. In Robotics and Automation. Proceedings. 1987 IEEE International Conference on (Vol. 4, pp. 1883-1889). IEEE.

[10] Naylor, B., Amanatides, J., & Thibault, W. (1990). Merging BSP trees yields polyhedral set operations. ACM Siggraph Computer Graphics, 24(4), 115-124.

[11] Mirtich, B. V. (1996). Impulse-based dynamic simulation of rigid body systems. University of California, Berkeley.

[12] Mirtich, B. (2000, July). Timewarp rigid body simulation. In Proceedings of the 27th annual conference on Computer graphics and interactive techniques (pp. 193-200). ACM Press/Addison-Wesley Publishing Co..Chicago

[13] Xavier, P. G. (1997, April). Fast swept-volume distance for robust collision detection. In Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on (Vol. 2, pp. 1162-1169). IEEE

★★★