AN EFFICIENT COLLISION DETECTION ALGORITHM FOR POINT CLOUD MODELS

Mauro Figueiredo[1], João Oliveira[2], Bruno Araújo[2], João Pereira[2]
[1] Instituto Superior de Engenharia, Universidade do Algarve, Faro, Portugal
mfiguei@ualg.pt
[2] IST/INESC-ID, Rua Alves Redol, 9, 1000-029 Lisboa
{joao.oliveira, bruno.araujo, jap}@inesc-id.pt

Abstract

Point clouds models are a common shape representation for several reasons. Three-dimensional scanning devices are widely used nowadays and points are an attractive primitive for rendering complex geometry. Nevertheless, there is not much literature on collision detection for point cloud models.

This paper presents a novel collision detection algorithm for point cloud models. The scene graph is divided in voxels. The objects of each voxel are organized in R-trees hierarchies of Axis-Aligned Bounding Boxes to group neighboring points and filter out very quickly parts of objects that do not interact with other models. The proposed algorithm also uses Overlapping Axis-Aligned Bounding Boxes to improve the performance of the collision detection process. Points derived from laser scanned data typically are not segmented and can have arbitrary spatial resolution thus introducing computational and modeling issues. We address these issues and results show that the proposed collision detection algorithm effectively finds intersections between point cloud models since it is able to reduce the number of bounding volume checks and updates.

Keywords: *Collision detection, virtual environments, surface segmentation, point cloud processing.*

1. INTRODUCTION

Point cloud models are an increasingly attractive representation used as the basis to capture and measure reality rapidly in an increasing number of applications such as environmental surveying, structure analysis and archaeology [1]. Point cloud models also share a remarkable similarity with a very popular computer game representation of the 80s, numerous Sinclair Spectrum games used axonometric projection of point models to convey details of buildings, interiors and avatars. A crucial element to enable laser scanned point models to be used in a similar scenario is collision detection of point clouds. In general interactive virtual environments often need very fast collision detection queries to simulate physical behaviour and to allow the user to interact. However, there is practically no literature on determining collisions between two sets of points.

This paper describes a novel collision detection algorithm for point cloud models.

The scene graph is organized in voxels. To speed up the process of finding collisions, for each voxel, each object is represented by an R-tree data structure of Axis-Aligned Bounding Boxes (AABB) defined in its own local coordinate system. The R-tree organizes spatially its geometry, grouping neighbouring points.

The proposed algorithm is also based in the use of the Overlapping Axis-Aligned Bounding Box (OAABB) to improve the performance of the collision detection process. In addition a traversal algorithm for collision detection for point clouds that takes advantage of the OAABB is also presented, improving performance by reducing the number of bounding volume checks and updates.

Results show that the proposed approach uses effectively the R-tree structure and the OAABB concept to find intersections between point cloud models at interactive rates. In addition, unlike CAD objects which typically contain object hierarchies and the data is already segmented into surface groups, point data sets derived from laser scanned data do not have such information thus presenting computational issues. We address these issues and present a solution that adapts to point sets derived from different laser scanners and spatial settings.

The paper is organized as follows. Section 2 presents collision detection approaches for the determination of intersections between polygonal and point cloud models. Section 3 describes the VIZIR project that highlighted the need to develop an efficient collision detection algorithm for point cloud models. Section 4 describes the data structures for the representation of the scene graph that are used to improve the performance of the novel collision detection algorithm, which is presented in Section 5. Section 6 presents the evaluation results using CAD models and addresses laser scanned point sets. Conclusions and future work are presented in Section 7.

2. RELATED WORK

Currently, there are many implementations of collision detection schemes for interactive system, most of them only support polygonal models. Frequently, they use bounding volume hierarchies (BVH), spatial subdivision methods and more recently use graphics hardware to accelerate collision detection by hardware. There is a lack of collision detection systems for point cloud models.

Bounding volume hierarchies are frequently used to organize the triangles of an object to improve the performance of the collision detection process, by reducing the number of pairs of bounding volume tests. The classic scheme for hierarchical collision detection is a simultaneous recursive traversal of two bounding volumes trees A and B.

Several types of bounding volumes are available. Bounding spheres can be used [2]. SOLID [3] and OPCODE [4] use axis-aligned bounding boxes (AABB). RAPID [5], V-COLLIDE [6], PQP [7], H-COLLIDE [8], use oriented bounding boxes (OBB). QuickCD [9] and Dop-Tree [10] uses k-dops; and Swift++ [11] uses convex hulls (CH). There are also hybrid approaches like QuOSPOs [12] that use a combination of OBBs and k-dops.

The main advantage of SOLID, OPCODE and Box-Tree is that AABBs are faster to intersect.

RAPID approximates 3D objects with hierarchies of oriented bounding boxes (OBBs). The main advantage of RAPID is that OBBs are better approximations to triangles reducing effectively the number of intersecting operations.

V-COLLIDE solves the broad-phase of the collision detection using a sweep-and-prune operation to find pairs of objects potentially in contact. It uses RAPID to find in the narrow phase which pairs of objects intersect.

H-COLLIDE is a framework to find collisions for haptic interactions. It uses a hybrid hierarchy of uniform grids and trees of OBBs to exploit frame-to-frame coherence. It was specialized to find collisions between a point probe against 3D objects.

The QuickCD and Dop-Tree implementations build a hierarchy tree of discrete orientation polytopes (k-dops). The main advantage of using discrete orientation polytopes is that k-dops are better approximations to the underlying geometry than AABBs with the advantage of its low cost compared to OBBs.

Swift++ builds a hierarchy of convex hulls and intersection is tested using a modified Lin-Canny closest feature algorithm.

He [12] uses a hybrid approach that combines OBBs and k-dops called QuOSPOs. This approach provides a tight approximation of the original model at each level.

Another class of hierarchical data structures used for collision detection are spatial partitioning representations: regular grids [13, 14, 15, 16], octrees [17, 18], BSP-trees [19] and R-trees [20].

Spatial subdivisions are a recursive partitioning of the embedding space occupied by objects. In general, spatial partitioning structures are used as a secondary representation for the collision detection process.

The main idea behind all space partitioning methods is to exploit spatial coherency. For each object, we check for collision only objects of the neighborhood, eliminating comparisons with those objects that are far away and therefore cannot be colliding.

Zyda [13] uses grids to find overlapping objects in the broad phase. García-Alonso [14] also uses uniform grids to find exact collisions between 3D objects for the narrow phase. Teschner [21] use uniform grid subdivision combined with hashing to reduce storage requirements for collision and self-collision detection of deforming objects that consist of tetrahedrons. Eits [22] also uses a spatial grid inspired by the work of Teschener together with 1D hash table to find collisions between deformable tetrahedral models. A hybrid approach is presented by Gregory [8] using regular grids, where each occupied grid cell stores an OBBs tree of those triangles on that cell.

Hubbard [17] approach for finding collisions in real time is based on a *time-critical* computing algorithm and on octrees of spheres. Kitamura [18] algorithm for collision detection uses an octree for each object. Ganovelli [16] also associate an octree of axis aligned bounding boxes with each object, and keeps this hierarchy efficiently and dynamically updated for deformable objects.

Luque [19] uses semi-adjusting BSP-trees for representing scenes composed of thousands of moving objects.

Figueiredo [20] combines AABBs with R-trees to implement an efficient collision detection algorithm that determines intersecting surfaces.

Various approaches have been recently introduced using existing graphics accelerated boards (GPU) [23, 24, 25, 26] or dedicated hardware [27] to accelerate collision detection by hardware.

Algorithms using graphics hardware use depth and stencil buffer techniques to determine collisions between convex [23] and non-convex [24] objects. CULLIDE [25] is also a GPU based algorithm that uses image-space occlusion queries and OBBs in a hybrid approach to determine intersections between general models with thousands of polygons. MRC [26] deals with large models composed of dozens of millions of polygons by using the representation of a clustered hierarchy of progressive meshes (CHPM) as a LOD hierarchy for a conservative errorbound collision and as a BVH for a GPU-based collision culling algorithm.

These GPU-based algorithms are applicable to both rigid and deformable models since all the computations are made in the image-space. Collision detection methods using GPUs have the disadvantage that they compete with the rendering process, slowing down the overall frame rate. Furthermore, some of these approaches are pure image based reducing their accuracy due to the discrete geometry representation.

All these collision methods have been applied only to polygonal objects. Recently Klein [28] presented a novel approach for collision detection of point clouds. They construct a point hierarchy of bounding volumes to enclose the points at different levels of the hierarchy. Points are stored in the hierarchy leaves. Each node stores a sufficient sample of the points plus a sphere covering of a part of the surface. Given two point cloud hierarchies, two objects are tested for collision by simultaneous traversal. At the leaves, an intersection is determined by estimating the smallest distance.

Recently, Kim [31] et. al show the performance benefits of using compression of out-of-core AABBs for collision detection of polygon models that do not fit in main memory, namely they show that the resources of the CPU can be used to compensate the I/O lag of reading uncompressed data structures.

3. VIZIR

The VIZIR project sets out to develop new visualization and interaction algorithms of massive out-of-core data. The 3D model of study consists of approximately 700 laser scans of the Batalha monastery, ~2 billion points, exceeding 100 GBytes.

Collision detection is an important interaction cue to help user navigation in the virtual world. Unfortunately not much work exists with solutions for collision detection with point clouds.

Before the full complexity of the model can be addressed, an efficient and reliable collision detection solution is needed for point clouds.

For this purpose a simple scenario was designed to evaluate different user collisions that can occur whilst navigating and exploring a 3D point cloud model.

In this scenario a subset of the model was chosen that enabled the user's polygonal avatar, which is represented as a point cloud for collision detection purposes, to pass through open doors, walk alongside walls, but is stopped when colliding with the point cloud (Figure 1, 2).

In addition standard collision detection tests were carried out, and collisions with points obtained from CAD models were also tested.

In the next section we present our solution for efficiently detecting collisions with point clouds.

two R-tree bounding volumes trees *A* and *B*. The approach presented takes advantage of the OAABB and is implemented to avoid visiting the same node several times to improve performance. It visits the nodes of object *A* once. The OAABB is an approach introduced by [29] to improve collision detection performance. Consider two objects, A and B, whose corresponding axis-aligned bounding boxes are overlapping and therefore are candidates for collision. The OAABB is defined as the volume that is common to two axis-aligned bounding boxes (Figure 4).

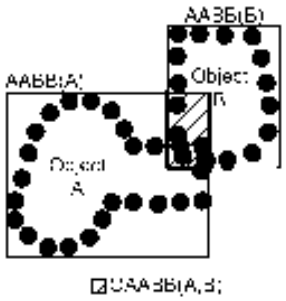

Figure 4: The OAABB is shown for two point cloud models intersecting.

Figure 5 presents the pseudo code of the novel approach.

```
Collide (A, B)
1: AABB_B(A)=M_B←A • AABB_A(A) //update BV
2: if (AABB_B(A) do not intersect AABB_B(B))
        return
3: Determine OAABB_B(A, B)
4: DescendRtree( B, OAABB_B(A,B))
5: for each point P(B) finside OAABB_B(A,B)
6:    Update point P(B) into coord. system of A P_A(B)
7:    DescendRtree( A, OAABB_A(A,B), P_A(B))
```

Figure 5: Pseudo-code for finding two intersecting objects.

The collision detection algorithm first checks if objects *A* and *B* are disjoint (line 1-2 in Figure 5). The bounding volumes of each object are originally computed in the object's local coordinate system, $AABB_A(A)$ and $AABB_B(B)$, respectively. The transformation matrix that converts the local representation of object *A* into the local coordinate system of object *B* is defined as $M_{B\leftarrow A}$. The bounding volume of object *A* is updated to the coordinate system of object *B*, by computing the cover axis-aligned bounding box, $AABB_B(A)$. Once the bounding volumes of each object are in the same coordinate system they can be checked for overlap. If this pair of AABBs does not overlap, then the corresponding two objects are not intersecting and the process ends. If they overlap, then the system determines the Overlapping Axis-Aligned Bounding Box, $OAABB_B(A,B)$ of the two objects (line 3 in Figure 5), which is defined in the local coordinate system of object *B*.

The next step of the collision detection process determines the points from object *B* inside the OAABB (line 4 of Figure 5). As mentioned before, the points of object *B* are organized in a bounding volume R-tree. The points of *B* are stored at the leaf nodes of the R-tree. By descending this R-tree, the points of object *B* outside the $OAABB_B(A,B)$ are filtered out. Only points at the leaf nodes inside the $OAABB_B(A,B)$ are candidate for collision.

The objects of object *B* inside the OAABB are transformed into the coordinate system of object *A*, $P_A(B)$ (line 6).

Then, the collision detection algorithm descends the bounding volume R-tree for object *A* (line 7 of Figure 5). In this step it finds points of object *A* inside both the OAABB and points in close proximity of object B.

6. EXPERIMENTAL RESULTS

This section presents the performance evaluation results of the novel collision detection algorithm for point cloud models described in this paper.

6.1. Using Points from CAD Models

This section shows that the proposed collision detection for point cloud models is effective in determining collisions in real time. It is also shown that it compares favorably with other approaches that determine collisions with a model´s polygons instead of with a model´s vertices.

To evaluate this, two case studies were designed. The first case study, evaluates the performance of the system with a real maintenance application, with interpenetrations between 3D models. The second case study, tests the performance of the collision detection algorithm for very close proximity when there are no intersections.

The first case study represents user operations to assembly the components to build a digger mechanism (Figure 6, left). For this application, it is necessary to allow the user to interactively carry out assembly and disassembly operations on the virtual prototypes in a realistic way. The three-dimensional virtual prototypes need to simulate physical properties realistically and interactively.

The functional modules used by this application are collision detection, constraint recognition, constraint satisfaction, constraint management and constraint motion. The automatic constraint recognition process uses collision detection services for various purposes such as (a) to provide collision response to stop object penetration, (b) to identify colliding parts to support the recognition of assembly relationships between the assembly parts, (c) to simulate constrained motion, (e) to simulate kinematics motion and sliding, thus assisting users to carry out precise object manipulations.

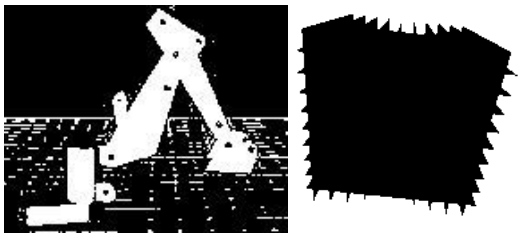

Figure 6: Test case scenario of: left) a Digger model; right) the grid scene.

The second example is a scene with two grids from a collision detection benchmark suite [31] (Figure 6, right). This benchmarking system is used to to compare pairwise static collision detection algorithms for rigid objects. This benchmark generates a number of positions and orientations for a predefined distance in close proximity and no interpenetration. It does not test

performance of collision detection approaches when intersections occur.

Table 1 presents the complexity for the digger and grid case studies.

The digger scenario has five parts that are assembled in a sequence of five hundred and seven intersecting steps. At each step it is found an intersection between parts of the scenario that are recognized and assembled appropriately. This experiment was conducted by the user executing the required assembly operations. The executed path was recorded and it was repeated only the intersecting steps to obtain the data values.

The second scenario has two equal grid objects defined each one of them with forty thousand and eighty points. The benchmark generated six thousand and thirty eight steps that positioned the two objects very close to each other, but not touching each other.

With these two case studies, the performance of the novel collision detection approach for cloud point models is evaluated.

Table 1: Complexity of the case studies.

	Digger	Grid
Number of Objects	5	2
Number Points	7356	46080
Number of steps	507	6038

All the experiments run in an Intel Core 2 Duo T7300, 2GByte of RAM memory at 2GHz.

The execution times, presented in this section, include only the time to determine collisions and do not include time for rendering or motion simulation.

Table 2 presents these times. The proposed collision detection algorithm for point cloud models achieves interactive rates in real industrial applications as desired.

Table 2: Collision detection time to find intersecting surfaces.

Time in milliseconds per step to determine intersections	3D models of Point Clouds
Digger	0.03
Grid	0.21

The time to determine the collisions between two objects depends on: (1) the cost of intersecting and updating bounding volumes; and (2) on the number of such operations. Table 3 shows the number of operations executed to determine intersections. This table shows that the number of bounding volume updates is significantly lower than the number of bounding volume intersections. The update of a bounding volume is a more expensive operation than a bounding volume intersection.

Table 3: Operations per step to determine intersections.

Number of operations	*Digger*	*Grid*
AABBs tests	149	634
AABBs updates	26	131

Table 4 shows the time and the number of operations executed to run the two test cases with the same collision detection algorithm, but it does not use the OAABB concept. This table presents results for the same traversal scheme to find collisions using only R-Trees.

Table 4: Traversal scheme for collision detection using R-Trees and not using the OAABB.

	Digger	*Grid*
Time to find intersections (ms)	0.15	3.79
Number AABBs tests	230	4810
Number AABBs updates	174	4625

The performance of the collision detection approach proposed is better when it uses the OAABBs.

From comparison of Tables 3 and 4, it is possible to see that the number of bounding volume checks and updates is reduced significantly by the use of the OAABB.

It is also important to compare the performance of this algorithm with other collision detection systems, although public collision detection toolkits are supported by polygonal models. Table 5 presents the times obtained for the two case studies with the S-CD, PQP, RAPID, OPCODE and Dop Tree collision detection toolkits. The times presented were obtained in the determination of the first intersecting triangle.

Table 5: Time to find first triangle intersection.

Time to find first triangle intersection (milliseconds)	*Digger*	*Grid*
PQP	0.94	8.99
RAPID	0.36	6.02
OPCODE	0.08	0.61
Dop Tree	1.26	7.09
S-CD	0.25	2.29

Tables 2 and 5 shows there is an improvement in performance for the collision detection approach supported by point cloud models. This improvement can be explained by the fact that the novel approach presented in this paper is being supported by point cloud models and, in this way, it does not make triangle checks to find intersections, which is an expensive operation. For this reason, there is a difference on the number of intersections determined with a collision detection using polygonal models and the approach described in this paper. However, for the digger case study there was only 1,1% different answers reported, which is a low error probability for the new collision detection algorithm for point cloud models.

6.2. Using Points from Scanned DataSets

Collision detection algorithms designed for polygonal and CAD models can rely on the concept of collision between subset surfaces. This has the advantage that searches for instance can be faster as we are dealing with only subsets rather than the entire model. In addition since we are typically only interested in detecting the collision between surfaces, a small standard tolerance constant is used in the literature. However point clouds derived from laser scans present two main differences: they are not segmented, and points are only samples of the surface, making an actual collision between points a less likely event. Point based rendering algorithms such as QSplat [33] change the thickness and shape of a point splat to better convey visually the underlying surface while viewing in close range. Similarly we use the average closest point distance of a point divided by two to create bounding boxes at point level that ensure collision detection of the surface they represent. In addition for each voxel we use an octree to segment points into smaller working sets.