Collision Queries: Models and Algorithms

Marta Franquesa–Niubo* and Pere Brunet
Departament de Llenguatges i Sistemes Informatics
Universitat Politecnica de Catalunya

October 8, 2003

Abstract

In this paper, a review of models and algorithms that are conceived to solve the collision queries problem is presented. Here, the type of query in proximity and collision computations is stated. The phases to solve the collision detection, prediction and proximity queries are exposed and classified. A collection of methods that are found in the literature are explained.

1 Introduction

Several classifications refered to collision and proximity queries are presented in this paper. Many classifications can be done from different application contexts. Hundreds of papers have been published on several aspects of these queries, bringing many data structures and reporting distinct information depending on the interested query.

2 Problem Classification

The collision detection problem can be formulated in several forms depending on the type of output sought and on the constraints imposed on the inputs. Constraining the inputs is a usual way of simplifying the complexity of problems. Thereby, in many approaches objects are assumed to be polyhedra, with planar faces, and convex, as we will see in the following sections.

*contact name: marta@lsi.upc.es

2.1 Collision Handling

The two main issues involved in the so called collision handling are those detecting that a collision has ocurred (*collision detection*) and then describing a response as an effect of collision (*collision response*). This document is focused on collision detection properly and not on collision response that is a dynamic problem which involves phisical laws to predict the behaviour of the objects that would penetrate. For a discussion of response algorithms, see [Kam93, Bar90, MW88, Can86, FHA90]. Collision detection is a geometric interference problem that depends on the space relationship of objects.

2.2 Mesure

There are at least [LM03] three forms of the distance queries, exact, approximate and boolean. The exact form asks for the exact distance between objects. The approximate form yields an answer which is within some given error tolerance of the mesure. The boolean form reports whether the exact mesure is greater or less than a given value. Although the Euclidean norm is the most common norm by which distance is computed, distance can be defined in terms of other norms. For example, the l_∞ or *max–norm distance* [VKK+03]. Under this last norm, the distance between two points x and y (in d dimensions) is represented as D_∞ and is defined as:

$$D_\infty(x, y) = \underset{i}{max} \mid x_i - y_i \mid \qquad i = 1, 2, .., d$$

2.3 Environments and Hybrid Collision Detection

When more than two objects are moving on the space the most obvious problem which arises is the $O(n^2)$ collision detection between all the n objects. This is the so called *allpairs problem.* It is obvious that this is an undesirable property of any collision detection algorithm and several techniques have been proposed to deal with it. Hybrid collision detection, a term introduced in [KTAK94], refers to any collision detection method which first performs one or more iterations of approximate tests to interfering objects in the entire workspace and then performs more accurate test to identify the objects parts causing the interference. [Hub95] and [CLMP95] also propose hybrid algorithms for collision queries.

Hubbard [Hub95] refers to two collision tests as two phases, the *broad phase*, where approximate interferences are detected, and the *narrow phase* where exact collision is performed. In [OD99, O'S99] these two phases are defined and classified as: The *narrow phase* itself may also consist of several levels of intersection testing between two objects at increasing levels of accuracy, the last of it may be fully accurate. The most accurate level is refered as *narrow phase: exact level* and the phase of testing for collision using increasing levels of accurancy is refered as *narrow phase: progressive refinement levels.*

In large environments (or, also called, *complex systems*), where the environment is composed by thousands of objects, the *broad phase* itself may consist of two phases too. At the first one, collision detection tests are performed to find subspaces (or subsets of objects) of the entire workspace where collisions can occur, rejecting, at the same time, all the space regions where interference clearly can not be possible. At the second phase, the collision test refers to determine the candidate interfering objects (objects identifiers that can cause collision). We shall refer to these phases as *broad phase: progressive delimitation levels* and *broad phase: accurate broad level*.

As we will see on the remainding of this paper the phases of the global collision detection problem can be summarized as:

- *Broad phase: progressive delimitation levels*: This phase restricts the subspaces where a collision can be found by using, commonly, hierarchies of space subdivisions (subsection 3.3).
- *Broad phase: accurate broad level*: In this phase the approximate test is performed to identify interfering objects in the entire workspace (in small environments with low number of objects) or in the subspaces found in the previous phase (in large environments) using a coarse representation of object shapes (simplified forms of objects such as bounding volumes, subsection 3.2).
- *Narrow phase: progressive refinement levels*: objects are usually modeled by hierarchies of simple shapes (subsections: 3.3 and 3.4). Hierarchical approximations allow to determine which object–parts are susceptible of interfering.
- *Narrow phase: exact level*: The interference tests need only to be applied within the relevant object–parts selected in the previous phase. In this phase a tightly representation of object shapes is used to accurately identify any object parts that cause interference (subsection 3.1).

The entire broad phase and the narrow phase at progressive refinement levels help to reduce the botleneck of $O(N^2)$. Figure 1 summarizes the main features of each phase. In the figure, N indicates a very high number of objects, while n indicates a subset of the N original objects.

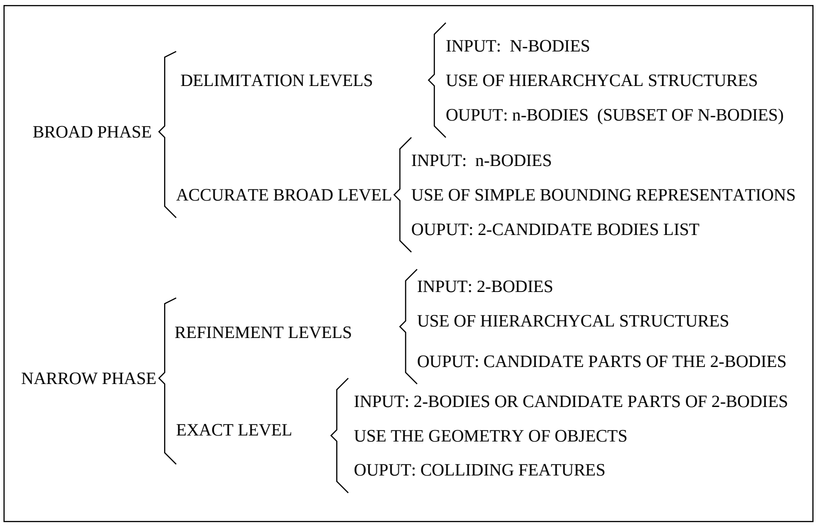

BROAD PHASE
DELIMITATION LEVELS
INPUT: N-BODIES
USE OF HIERARCHYCAL STRUCTURES
OUPUT: n-BODIES (SUBSET OF N-BODIES)
ACCURATE BROAD LEVEL
INPUT: n-BODIES
USE OF SIMPLE BOUNDING REPRESENTATIONS
OUPUT: 2-CANDIDATE BODIES LIST
NARROW PHASE
REFINEMENT LEVELS
INPUT: 2-BODIES
USE OF HIERARCHYCAL STRUCTURES
OUPUT: CANDIDATE PARTS OF THE 2-BODIES
EXACT LEVEL
INPUT: 2-BODIES OR CANDIDATE PARTS OF 2-BODIES
USE THE GEOMETRY OF OBJECTS
OUPUT: COLLIDING FEATURES

Figure 1: Phases of the global collision detection problem

2.4 Temporality

The collision detection problem can be solved from two different point of views, *static collision* and *dynamic collision* (where objects are moving). One way to treat the dynamic collision detection is sampling time and reducing the problem to multiple calls to static inteference test. This approach is called *multiple interference detection*. Dynamic collision can be achieved in other ways: swept volume interference, extrusion in 4D space [Cam90] and trajectory parameterization [KR03]. For more information about other dynamic collision approaches see [JTT98, JTT01].

2.5 Type of Query

As pointed out in [LM03], there are two forms of collision detection or proximity query, boolean or enumerative. The boolean distance query computes whether the two point sets, belonging to diferent objects, have at least one point in common. The enumerative form yields some representation of the intersection set.

Algorithms can be classified in different classes depending on the type of query. Four kind of algorithms can be distinguished:

- *ECA*: Exact Collision Algorithms: Algorithms that are focused on finding the exact interference regions computing the object parts that are involved in the collision if it exist.
- *ST*: Separability Test: Algorithms that the main goal is centered on reject collision making use of a minimum–tolerance distance ε.
- *NCIT*: Non Clear Interpenetration Test: Algorithms that the main goal is pointed on finding a non clear interpenetration between objects, making use of a minimum–tolerance of interpenetration.
- *GBA*: General Broad Algorithms: Algorithms that are based on bounding volumes and either reject collisions or detect possible collisions without using a pre–determined tolerances.

The behaviour of each algorithm class is showed in figure 2. As it can be seen, the ECA algorithms solve the collision detection problem in the *narrow phase: exact level*, while others (ST, NCIT and GBA) solve the problem in the previous phases. In figure 2 we have to consider two kind of LISTS. In the ECA algorithms the returning LIST is a list that contains the object features (object parts) involved on the detected interference, while in the ST and NCIT cases the LIST contains the object identifiers involved on the collision.

2.6 Type of Objects

To understand how objects can collide we have to consider which kind of objects may be present in the workspace. Below we consider the collision detection problem depending on types of objects are

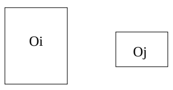
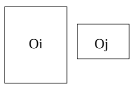
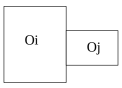
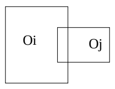
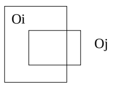
ε	ECA	ST	NCIT	GBA
	NO COLLISION	TRUE	TRUE	TRUE or FALSE
	NO COLLISION	FALSE + LIST	TRUE	TRUE or FALSE
	COLLISION + LIST	FALSE + LIST	TRUE	TRUE
	COLLISION + LIST	FALSE + LIST	TRUE	TRUE
	COLLISION + LIST	FALSE + LIST	FALSE + LIST	TRUE

Figure 2: Kind of the algorithm results. ECA: Exact Collision Algorithms, ST: Separability Test, NCIT: Non Clear Interpenetration Test. GBA: General Broad Algorithms. Where LIST means the list of objects involved in the collision

involved in. General and detailed information about interference between object types can be found in [Mou97, Kam93, JTT01, LM03].

In what follows there is a possible classification of collision detection problem according to the type of elements involved in:

- Plane collide with a point mass

  It is only needed to evaluate the sign of the expression $(p - x) * N$. Where p is the point location, x is any point on the plane, and N is the normal to the plane. See figure 3.

- Point to point collision

  Two points are considered to collide if the distance between the points is less than a pre–determined tolerance.

- Polygons collision in 3D

  The collision detection with two polygons is performed by testing for penetration of each vertex point of one polygon through the plane of the other, and checking that no two edges intersect. This technique can be applied to plane surfaces defined as two polygons of three points each.

- Convex polyhedra collision

  A common technique used for collision detection of convex polyhedra is performed by checking the face of each polyhedron against the faces of the other one and doing the same process in the other way round. This method can be accomplished in $O(N^2)$ time, where N is the number of estimated faces of each polyhedron [PS85].

- Non convex polyhedra collision

  When objects are not convex it is possible to process the interference detection in a two different approaches:

  – Objects can be subdivided in a collection of convex objects and apply the same process explained above as well, or
  – Applying an algorithm developped ad-hoc

- Surfaces collision

  In most cases surfaces are aproximated as a collection of triangulated polygons. In this case, the 3D polygons collision detection method presented above can be applied.

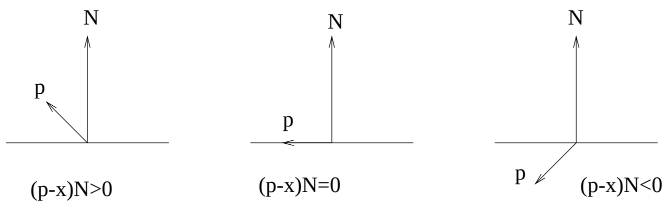


Figure 3: Collision detection with plane and point mass

2.7 First Algorithms Classification

As mentioned above, the collision detection problem can be formulated in several forms depending on the type of output sought and on the constraints imposed on the inputs. Taking into account the output sought we can classify the methods in five different classes:

- Algorithms that are interested to find the first contact point between objects.
- Algorithms that are focused on finding the exact interference regions computing the object parts that are involved in the collision if it exist.
- Algorithms that are centered on reject collision making use of a minimum–tolerance distance.
- Algorithms that are focused on finding a non clear interpenetration between objects making use of a minimum–tolerance of interpenetration.
- Algorithms that are based on bounding volumes and either reject collisions or detect possible collisions without using a pre–determined tolerance.

3 Model Representations

In this section several data structures found in the literature that are used to solve collision queries are described. From a general point of view, we can classify the data structures in different classes depending on the criterion followed to model the workspace and objects. As it has been exposed in subsection 2.3, before a pair of objects are tested for interference a previous test by using a coarse representation of objects is usually done. This is a conservative process of no–interference of both sets of polygons (one set for each object). Furthermore, applying recursively the conservative test to more approximative representations of objects point out to represent objects as representation hierarchies. The criterion followed to build hierarchical representations, as we will see, can be based on space partitioning or bounding volume hierarchies.

- Geometric Models: Collision, proximity and interference queries are computed by using the geometry of the two possible candidate objects. In the narrow phase at the exact level the queries are formulated by using the geometry of objects.
- Bounding Volumes: The scene objects are bounding by volumes of simply geometry. Those volumes permit to ask for collision more easly than the original object representations. Bounding volumes are used in the accurate broad level phase.
- Space Partitioning Trees: In this class, space decomposition techniques in a hierarchical way are considered. Examples of space partitioning hierarchies include BSPtrees [TN87, NAT90], Octrees, Bintrees, Kdtrees and their extensions [Sam90, SW98, PG93]. Space partitioning tress are often used in the progressive delimitation levels of the broad phase.
- Bounding Volume Trees: In this class hierarchical volume representations that bound set of geometric object primitives are included. Examples of bounding volumes hierarchies are: Sphere–trees [Hub93, Hub96, PG95, OD99, BO03], axis–aligned–bounding–box–trees [Ber97, AdBG$^+$01, LAM01, KZ03], oriented–bounding–boxe–trees [GLM96, BCG$^+$96, Got00] , kDOPtrees [KHM$^+$98, MKE02] and ConvexHullTree [EL01, OL03]. Bounding volumes trees are used in the narrow phase: progressive refinement levels.

3.1 Geometric Models

The methods to solve collision detection at the exact level of the narrow phase will be more and less conditioned by the geometric model selected. Then, in this subsection a brief description of geometric models to represent rigid objects is presented.

Constructive models represent a point–set as a combination of primitive point–sets. Each primitive is represented as an *intance* of a primitive type and combination operations are set boolean operations (union, intersection, difference and complement).

Boundary representations (B–rep) describe an object in terms of its surface boundaries: vertices, edges and faces. Some B–reps are restricted to planar, polygonal boundaries, and may even require faces to be convex polygons or triangles. Determining what constitutes a face can be particularly difficult if curved surfaces are allowed. Curved faces are often approximated with polygons. Alternatively, curved surfaces can also be represented as surface patches if the algorithms that process the representation can treat the resulting intersection curves, which will, in general, be of higher order than the original surfaces. Many B–rep systems support only solids whose boundaries are *2–manifolds* [FDF+93].

Space decomposition representations represent an object as a union of disjoint regions of the space called *cells*. Cells can be cubes or spheres for instance. Cells that contain part of the object are usually labeled as *black* and the rest are labeled as *white*. The most used representations for space decomposition are *voxels* and *octrees*. In the voxels case the space is partitioned in a uniform way while in the second case, octrees, the subdivision is hierarchical.

There exists several algorithms to compute collision detection in the narrow phase at the exact level (making use of the geometry of objects). In this subsection we include the most commonly known algorithms to solve the narrow phase at the exact level.

Dobkin and Kirkpatrick [DK83, DK90] describe a former algorithm that detects the collision of two polyhedra in $O(log^2 v)$, where v is the total number of vertices in the polyhedra. Despite the method reports only one collision point even if multiple parts of the objects collide and requires convexity, it is an early algorithm of theoretical importance that we have to consider. It precomputes the Dobkin–Kirkpatrick hierarchy for each polyhedron and uses it to perform the runtime query. The method may be useful when only detection is required.

• Linear Programing Algorithms: Detecting interference between two convex polytope can be formulated as: two convex polytopes do not overlap, if and only if there exists a separation plane between them. The linear constraints are formulated by imposing that all the vertices of the first polytope lie on one half-space of this plane and those of the other polytope lie on the other half-space [Sei90, CW96].

• Feature based Algorithms: In this class are included the Lin–Canny *LC* [LC91] algorithm and its improvements. These algorithms are focused on the relationships between vertices, edges and faces of the two polytope, that constitute the set of *features*. The emphasis is pointed out in detect whether two polytopes are touching or not. The algorithm constructs the *Voronoi Region* of the features of each polytope. The voronoi regions constitute the set of points closer to a feature than any other. Figure 4 shows the voronoi regions associated to a vertex, edge and face. LC algorithm computes the distance between the closest feactures of two polytopes to achieve if they collide. The method takes two features of the two polytopes, then by using the voronoi regions, it calculates if one feature is inside or outside of the voronoi region of the feature belonging to the other object. If it is not, the algorithm takes the *adjacent* feature and continue the exploration. The algorithm takes $O(f)$ time computing, being f the number of features. The method exploits coherence because closest features will not change significatively between frame to frame. The algorithm have been implemented into the I–COLLIDE [CLMP95, PML97] collision detection software which is available on the *world wide web*. The LC algorithm can not compute penetration between polytopes. Mirtich et al. [Mir98] present the V–CLIP algorithm to overcome the limitations of the I–COLLIDE.

• Simplex based Algorithms: In this case the polytopes are treated as the convex–hull of a point set. Gilbert et al. [GJK88] presented the former algorithm belonging to this class, called the *GJK* algorithm. The GJK detects collision and gives a mesure of interpenetration. After Cameron [Cam97] developed an

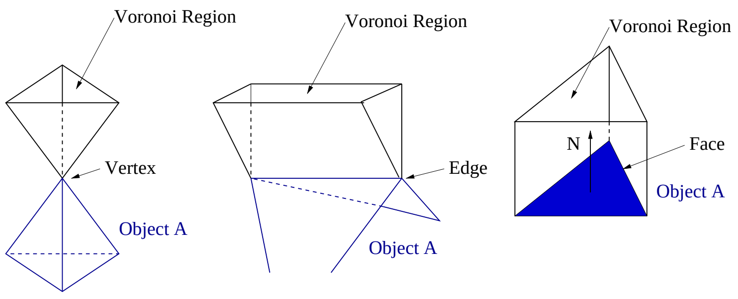


Figure 4: Voronoi regions of a vertex, an edge and a face

improved version of the GJK to produce the so called *Enhanced GJK* algorithm.

3.2 Bounding Volumes

Many bounding volumes and hierarchies of them have been studied. Thus, in this section a collection of the most used bounding volume to solve collision detection for polyhedral rigid bodies is presented. Other types of shapes for bounding volume for other class of geometric objects have been proposed and published but we will point on the most used types. The bounding volumes are used in the broad phase at the accurate broad level.

3.2.1 Sphere

To define an sphere it is only necessary to store the center coordinate (x, y, z) and the radii (r). Thus, two spheres intersect if the distance of their midpoints is equal or less than the sum of their radius (fig. 5). Then, the interference test between two objects bounded by spheres is fast and efficient. The most disadvantage when using this representation is that an sphere does not limit accurately the shape of the objects. When object shapes are more spherichal the representation is better than when object shapes are less spherical. As Ritter [Rit90] pointed out, the cost of building one sphere is of order $O(n)$, where n is the number of object vertices. The advantage of this bounding volume is its invariance with respect the axes, thus the rotation transformation does not affect the bounding volume.

3.2.2 AABB

Axis–Aligned–Bounding–Box (AABB) is a bounding box where its facets are aligned with the global scene axis. This bounding volume is easy to build, it is only needed to compute the minimum and maximum coordinates of the object that bounds. The building cost is of order $O(n)$, where n is the number of object vertices. It is only necessary six floats to store it, $x_{min}, y_{min}, z_{min}$ and $x_{max}, y_{max}, z_{max}$. The collision test is performed by projecting each edge of the box to the coordinate axes and then compute the onedimensional overlap test with the onedimensional resulting interval projected, the boxes can overlap

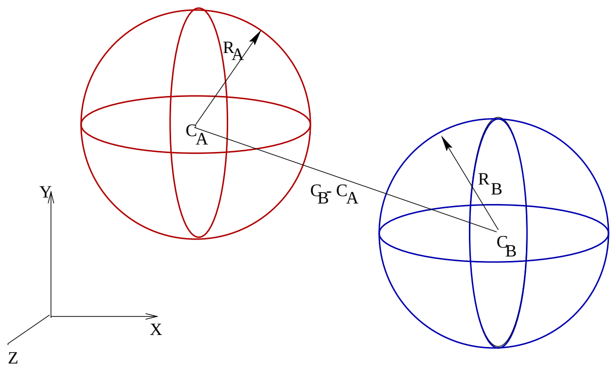


Figure 5: Sphere A and sphere B collide iff $d \leq R_A + R_B$, where $d \Leftrightarrow \mid C_B - C_A \mid$, in this example do not collide

if and only if their intervals overlap in all three dimensions. Thus, the test is very fast, it is reduced to simple arithmetic comparisons (overlap test between axis coordinates) (see fig. 6). AABB is inefficient for long thin objects at arbitrary orientations and needs to keep recomputing bounding boxes for rotating objects, $O(n)$ again.

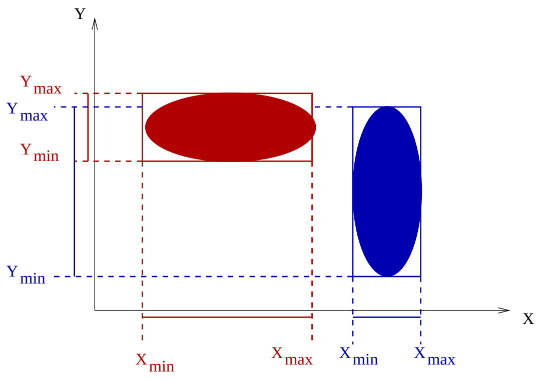


Figure 6: AABB collision: Boxes intersect if and only if projections to coordinate axes intersect.

3.2.3 OBB

Oriented–Bounding–Box is an object bounding box that is not necessarily aligned to global coordinate axes, thus an OBB is a rectangular bounding box at an arbitrary orientation. In an ideal case, the OBB would be oriented such that it encloses an object as tightly as possible. Thus, the OBB is the smallest possible bounding rectangular box of arbitrary orientation that can enclose objects geometry. To store an OBB it is needed 8 coordinates (one for each vertex) or a center point and 3 vectors. When rotating the object enclosed it is only needed to apply the transformation matrix to the 8 coordinates or to the vectors, depending on the stored information selected. The cost of building an OBB of an object with n vertices is of order $O(n \lg n)$. Overlaps between two OBB are computed by 15 simple axis projections test, about 200 arithmetic operations, by making use of the *separating axis theorem* [GLM96]. This theorem tells us that we have only 15 potential separating axes (see fig. 7). If overlap occurs on every single separating axis, the boxes intersect. Thus, to determine whether two boxes intersect or not is very simple and fast.

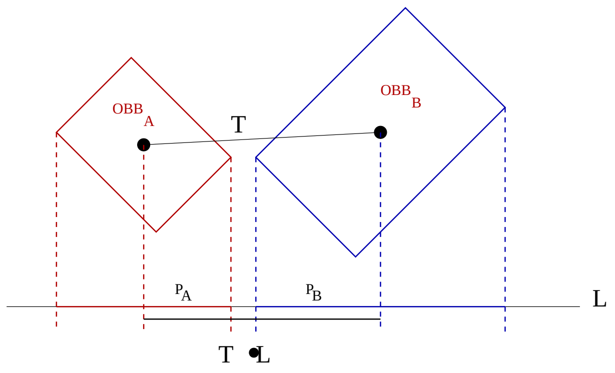


Figure 7: OBB interference test using the separating axis theorem: objects A and B do not intersect because their OBBs do not intersect: $\mid T \cdot L \mid > P_A + P_B$

3.2.4 k–DOP

A *k–Discrete–Oriented–Polytope (k–DOP)* is a convex polytope whose facets are determined by halfspaces whose outward normals come from a fixed set of k orientations [KHM+98]. There exists several possibilities for determining the $k/2$ fixed direction to use. There are four possible choices [KHM+98] 6–DOPs (which are AABBs), 14–DOP, 18–DOP and 26–DOP. To build a 14–DOP we have to find the minimum and maximum coordinate values of the vertices of the primitive along seven directions, defined by vectors: (1,0,0), (0,1,0), (0,0,1), (1,1,1), (1,-1,1), (1,1,-1) and (1,-1,-1). Thus, this particular k–DOP uses the 6 halfspaces that define the facets of an AABB, together with eight additional diagonal halfspaces that serve to *cut off* as much of the eight corners of an AABB as possible. 18–DOP also derives 6 of its halfspaces from those of an AABB, but augments them with 12 additional edges of an AABB, these 12 halfspaces are determined by six axes, defined by the direction vectors (1,1,0), (1,0,1), (0,1,1), (1,-1,0), (1,0,-1) and (0,1,-1). Finally 26–DOP is simply determined by the union of the defining halfspaces for the 14–DOPs and 18–DOPs, utilizing the 6 halfspaces of an AABB, plus the eight diagonal halfspaces that cut off corners, plus 12 halfspace that cut off edges of an AABB. To build one k–DOP we have to find the output object positions for every of the k selected directions, then its generation is linear $O(n)$. The k–DOPs are good fitting in many cases because they do not leave *empty corners* as it can be in the AABB case. The collision test between two k–DOPs is rapidly performed doing $k/2$ 1D overlap test (analogous to AABB/AABB test, but taking into account each outward normal). Needs to keep recomputing k–DOP when moving objects, usually this is recomputed from the transformed original k–DOP and not from the object $O(k)$, the result increases the bounding volume, but the solution is much better that reconstruct the k–DOP from the object. Figure 8 shows an example of the bounding volumes introduced up to now.

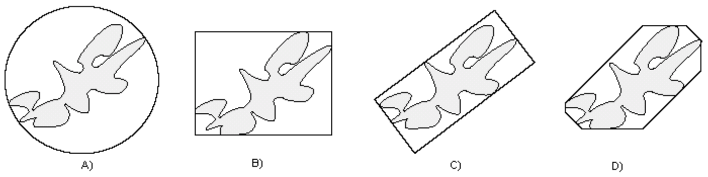


Figure 8: Approximation of an object by four bounding volumes: A) Sphere, B) AABB, C) OBB and D) k–DOP (where $k = 8$) [KHM$^+$98]

3.2.5 Convex–Hull

The convex–hull of a set of points can be defined as the smallest convex set containing the points or, more precisely, as the intersection of all convex sets containing the points. One can show that the convex–hull of a set of points is exactly the set of all possible convex combinations of the points. With convex–hulls intersection test are more complicated than for other BVs (as spheres, for instance) but they can approximate most objects a lot better. The convex–hull of an object can be built in $O(n \lg n)$ expected time [BKOS00]. Ehmann and Lin [EL01] use a minimum distance computation algorithm based on *voronoi marching* as a subroutine to test proximity of a pair of convex polyhedra. Otaduy and Lin [OL03] create a bounding volume hierarchy of convex–hulls. In the last paper, a new representation of the objects to be used in multiresolution collision detection is defined.

Ponamgi, Manocha and Lin [PML95, PML97] present an algorithm for collision detection between general B–rep solid models in dynamic environments. For each non–convex polyhedral object, they compute its convex–hull and the *AABB* (axis–aligned–bounding–box) enclosing the convex hull. Then, for each pair of objects, they determine if the bounding boxes are overlapping using a *Sweep–and–Prune* technique presented in [CLMP95]. For each intersecting bounding box pair, check if the convex–hulls of the objects are colliding using a penetration detection algorithm. For each intersecting convex–hull pair, they compute the areas of intersection on the convex–hulls. The faces comprising these areas are actual features on the original object or are faces (introduced by the convex–hull) covering features of the object. The features on the areas of intersection are represented as a pre–computed hierarchies. These areas are traversed using a hierarchical version of the *Sweep and Prune* technique to find exact collision.

3.2.6 Summary of Bounding Volumes

This section summarizes the features of the most common bounding volumes which have been exposed.

When we are looking for a selection of the bounding volumes we have to take into account the advantages and disadvantages of each one.

Advantages of BV:

- *sphere*: Fast and easy to compute, bounding volume is invariant with respect to rotation, collision detection test is very fast (requires 3adds, 3subs, 3squares, 1 root), reject test result is absolutelly reliable.

- *AABB*: Fast and easy to build, simple arithmetic test for interference testing and very fast. Easy to compute the overlap *region*
- *OBB*: Tight fit with underlying object. Relatively easy to test for intersection.
- *k–DOP*: Good fit in many cases. Overlap test between two bounding volumes analogous to AABB/AABB test.
- *Convex–Hull*: Good fit.

Disadvantages of BV:

- *sphere*: Approximation to underlying object may be poor, collision test has low specificity
- *AABB*: The bounding volume is not invariant with regard to rotation. Bad objects fit in many cases
- *OBB*: Overlap test based on the separating axes theorem still rather expensive (200 operations). Not invariant to rotation transformation.
- *k–DOP*: Not so easy to compute. Is not invariant with respect rotation.
- *Convex–Hull*: Not so easy to compute. Is not invariant with respect rotation.

Table 1 shows a comparison between the BVs. The criteria selected are: the *building cost* of the bounding volume of a polyhedra with n vertices, the *tightness* that mesures in a orientative way the degree of approximation between the underlying object and the BV, the *memory* occupancy to store the BV, the *Num. ops Inter* that indicates the number of operations needed to perform the interference test between two BV and finally, *rotation dependency* and *transform* the operations needed to perform tranformations when the object is moving.

3.3 Space Partitioning Methods

The space subdivisions or space partitioning methods are currently used to achieve collision detection between N–bodies in the space (broad phase: progressive delimitation levels, see sect. 2.3). Althought, they have been used, also, to perform collision between two objects (narrow phase:progressive refinement levels, see sect. 2.3). Different methods have been proposed to overcome the bottleneck of $O(N^2)$ pairwise tests in an environment composed of N objects. In this subsection the most commonly used space partitioning methods are exposed.

Criteria	Bounding Volumes				
	Sphere	AABB	OBB	k–DOP	Convex–Hull
Building cost	$O(n)$	$O(n)$	$O(n \lg n)$	$O(n)$	$O(n \lg n)$
Tightness	poor	maybe poor	midle	good	the best
Memory (bytes)	14	24	96	$12 * k$	$O(n)$
Num. ops Inter	8	6	200	k	$O(n^2)$
Rotation	invariant	recompute	tranform BV	transform BV	recompute
Transformation	1	$O(n)$	8	$2 * k$	$O(n \lg n)$

Table 1: Comparison between bounding volume features

3.3.1 Regular Grids of Boxes

This representation model is based on a grid of equal sized boxes over the space, which is assumed to be a unit cube (also called *voxel*) [HKM95b, MPT99] or a parallelipipeds [GASF94]. Thus, objects are represented by a 3D matrix of voxels *voxelization* (Fig. 9 shows an example). In this kind of representation is easy to find which voxel contains a point, this operation is done in constant expected time $O(1)$. Lawlor and Kal [LK02] present an algorithm to compute collisions between many moving objects making use of this technique. Borro [Bor03] use the voxelization representation to compute interference between static n–bodies. As pointed out in [Bor03] the selected number of voxels to permit to subdivide the space is not a trivial problem. Another problem is the memory occupancy when the number of voxels increase to represent an object and, in many cases most of voxels are empty of solid. In his work, Borro presents a solution to this problem based on hierarchical methods and using, also, hashing techniques.

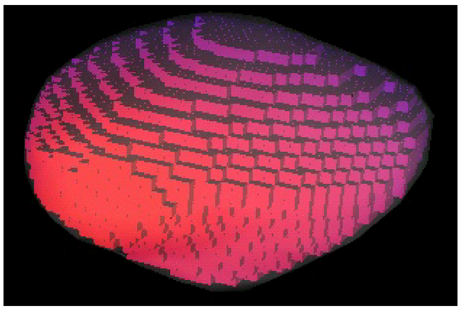


Figure 9: Image of a voxelization of an object (courtesy: The University of Hull)

Garcia–Alonso, Serrano and Flaquer [GASF94] design a method to check if two objects interfere. The two objects are bounded by *containers*. One container of one object is the smallest box that contains it, whose faces are parallel to the object's local reference frame (*OBB* oriented–bounding–box). When two objects can collide, the method computes the interference between their containers. Then, If the *minimax* *test* between them does not report a noninterference status, the program applies the interference among voxels test. If the test of voxels can not reject the collision between the objects, the program move on to the last test, the test between pairs of facets. They solve the three last phases of the global collision detection problem (see sect. 2.3).

McNeely et al. [MPT99] present a method where dynamic objects are represented by a set of surface point samples, plus associated inward pointing surface normals, collectively called a *Point Shell*. The environment of the static objects is collectively represented by a single space occupancy map, called *Voxmap*, which is a regular grid of voxels (see fig. 10). In order to explain how the data structure is computed consider a solid object such as the teapot of fig. 11(a). It partitions the space into regions of free space, object surface and object interior. Now tile this space into a volume occupancy map *voxmap*, fig. 11(b). The collection of center points of all surface voxels constitutes the *point shell* needed by the depth of interpenetration calculation. How the original object representation (polygonal model) is not used after voxelization, this representation has an error of $\varepsilon \leq \sqrt{3}s/2$. Renz et al. presented an improved method to minimize the approximation error [RPP+01].

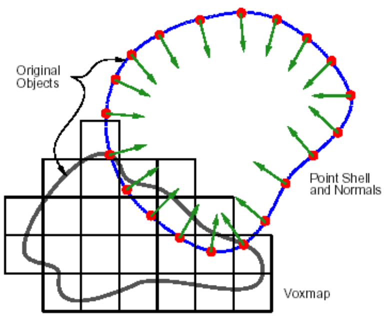


Figure 10: Voxmap representation of an scene object colliding with a Point Shell representation of a moving object [MPT99]

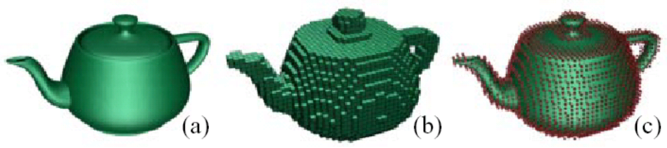


Figure 11: Teapot: (a) Polygonal model. (b) voxel model. (c) point shell model. [MPT99]

Varadhan et al. [VKK+03] present techniques to compute the distance under max-norm (see equation of section 1) between a point and a wide class of geometric primitives. They use the max-norm distance computation algorithm to design a reliable voxel-intersection test to determine whether the surface of a primitive intersects a voxel and They use this test to perform voxelization of solids and generate adaptive distance fields that provides a Hausdorff distance guarantee between the boundary of the original primitives and the reconstructed surface. Figure 12 shows an example of performing voxelization using their voxel-intersection test. They introduce a method called *voxel–intersection–test*. This method uses an exact test based on computing the max–norm distance between the center of a voxel and the primitive. The test is based on the fact that a voxel is intersected by the surface if the max–norm distance at the center of the voxel is less than half the voxel size. They demonstrated that the above statement is valid even when the voxels are not regular-sized cubes.

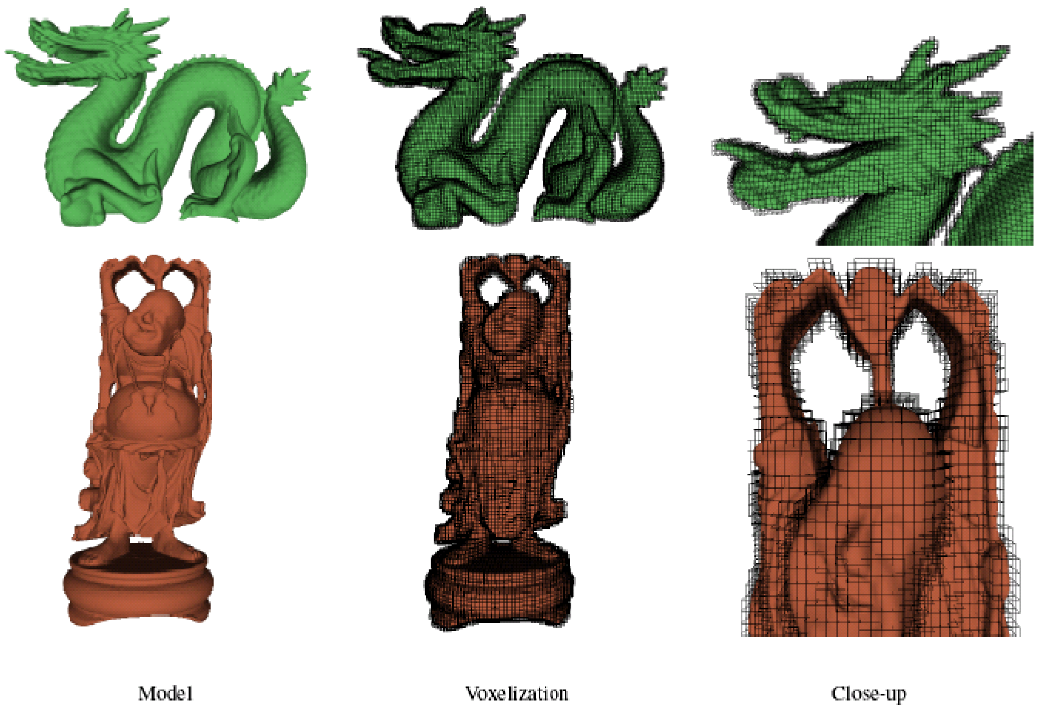


Figure 12: Voxelization algorithm applied to the Dragon and Happy Buddha models. The two models consists of 871,414 and 1,087,716 triangles respectively. The middle and right column of figures shows an entire view and a close-up view of the voxelization superimposed on the model. The voxels occupied by the model have been rendered in wireframe [VKK+03]

3.3.2 Octree

An octree [Sam90] is a tree of degree eight which represents the space occcupied by objects contained in the world space defined by a *universe cube* (AABB). If this cube is empty or completely filled by objects, then a single (root) node marked *white* (empty) or *black* (full), respectively, constitutes the octree representation. Otherwise, the universe cube is decomposed into eight octants, which are represented by eight children of the root node. The root node is marked grey to indicate the partial occupancy of the workspace. Each octant is tested to see whether it is completely occupied by objects, is partially occupied by objects, or is completely contained in free space. The octree node corresponding to the octant is given a color black, grey, or white, respectively. This decomposition is carried out recursively for grey nodes. The decomposition halts either when all leaf nodes are found to be black or white, or when a prespecified level of resolution is reached. The model obtained by this way is known as *classical octree*. In a [BJN92] can be found a detailed description and operations involving Octrees.

Even though the most used shape of *octree–octant* is a cube, there exists also models where spheres are used [Hub93, PG95] (see fig. 15). In fact, an algorithm that builds an octree representation of an object can also build a *sphere–tree* by circunscribing each occupied octant with a sphere. Liu, Noborio and Arimoto [LAN91] not only use spheres to node–representing volumes but also the non–disjoint subdivision of the space is performed by following a different technique where the branching is 13 instead of 8. The change between cubes for spheres is due to that spheres are invariant in rotation operations.

Pointing out in the information associated to leaf nodes, we find variants of octree model too. Among them we can quote: Foct (Face Octrees) [PG93] and Extended Octrees [BN90, FNB90, FB97]. The principal feature of them consist in the kind of leaf nodes that are allowed in the model. In the *Face–octree* model leaves information about the faces of the objects is included, those nodes are marked as *face nodes*. In the *Extended–octree*, vertices, edges, and faces nodes are allowed leaves, too. Figura 13 shows an example of the extended octree of an object.

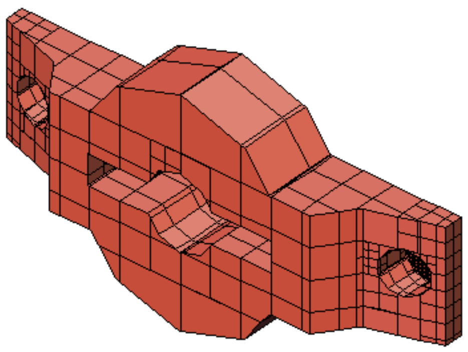


Figure 13: Extended octree of an object. Courtesy of Marc Vigo

Using octree representation [HH96, Hay86], collision among objects is evaluated by traversing theirs octrees in parallel and comparing homologous nodes. When *black* nodes overlap it means that a collision is detected. At the lowest level, the grey nodes are usually consider that interfere for save security. The time complexity to evaluate if octrees collide is proportional to the number of nodes visited. The average time for detecting interferences among n objects is of order $O(n * N)$, where N is the average number of nodes in the tree [FB97].

Kitamura, Takemura, Ahuja and Kishino present in [KTAK94] an hybrid collision detection method that performs an approximate test to identify interfering objects in the entire workspace and then perfoms more accurate test to identify the objects parts causing collisions by using octree and polyhedral representations, respectively. Interference in the entire workspace is detected by traversing the object octrees in parallel.

If black nodes exists whose corresponding node in another tree is also black, these nodes are considered to be interfering (as explained above). The traversal is performed down to a pre–determined lowest level. At this lowest level any pair of corresponding grey nodes is considered to be interfering. After the traversal is finished, all interfering nodes and corresponding objects are identified. The faces of objects that intersect with their octree nodes are extracted. For each such interfering node in an objects octree, the coordinates of the eight vertices of the corresponding cube C are substituted in the equation of the plane T of each face F of the interfering object. If all the vertices do not lead to the same sign for the value obtained after substitution, then a face F is assumed to be possibly causing the interference detected by octree representation. To determine if the face actually cause interference, the polygon S of the intersection between C and T is found. A two–dimensional interference detection is then done for S and F. If an intersection is found, F is labeled as interfering, otherwise, it is labeled as noninterfering.

Ganovelli et al. [GDO00] introduce the *BucketTree* to represent objects as a soup of freely moving primitives. The method uses un octree to partition the space and place buckets as leaves where geometrical primitives can be placed. At every time step of a simulation, the models primitives are assigned to an appropriate bucket. Then the intersection tests between any two models are done recursively by testing the nodes AABBs as their trees are traversed.

3.3.3 Bintrees

A bintree is a recursive space subdivision. The bintree is a dimension-independent variant of the quadtree and octree representations. The bintree determines explicitly which parts of space are solid and which empty. A bintree represents discrete solid objects of arbitrary dimensionality by a binary tree defining a recursive subdivision of space and recording which parts are empty (white) and which are solid (black). In [ST85] is presented an algorithm to construct the bintree model from the *constructive solid geometry* (CSG) one, which a cost of order $O(N)$, where N is the number of bintree nodes. Even though this representation seems useful to solve the collision problem, as far as we know, no other papers have been published in this sense.

3.3.4 BSPtree

A Binary–Space–Partitioning tree (BSPtree) is a representation model that recursively divides space into pairs of subspaces, each separated by a plane of arbitrary orientation and position. Thibault and Naylor [TN87] introduced the use of BSPtrees to represent arbitrary polyhedra. Each non–leaf node of the BSPtree is associated with a plane and has two child pointers, one for each side of the plane. Assuming that normals point out of an object, the left child is behind or inside the plane and, the right child is in front of or outside the plane. If the half–space on a side of the plane is not homogeneous then it is subdivided and its child is the root of a subtree. If the half–space is homogeneous, then its child is a leaf, representing a region either enterely inside or enterely outside the polyhedron. The BSPtree is not restricted to be axis–aligned and geometric transformations can be computed by applying the transformation to each hyperplane, without rebuilding the whole representation. In [NAT90] algorithms required to perform boolean set operations between two objects represented by BSPtrees are presented.

The main disadvantage of BSPtrees is to find the planes to well cut the scence objects keeping on a minimum depth $O(n^2)$ (where n is the objects number in the scene). Sigal et al. [SGA02] defined and implemented a *deferred self-organizing BSP* that is a BSPtree built throughout the walk in walkthrought applications (see fig. 14).

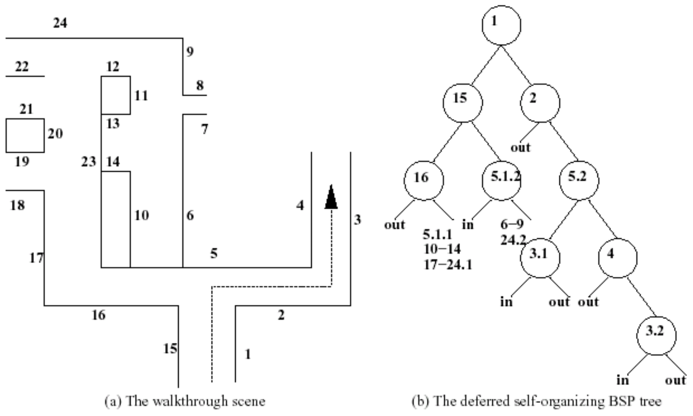


Figure 14: The deferred self-organizing BSP as built throughout the walk [SGA02]

Collision detection between two objects using BSPtrees is basically reduced to tree traversal, or search. This approach rejects a lot of geometry early, so in the end, we only have to test the collison detection against a small number of planes. As we have seen in the previous section, finding a separating plane between two objects is sufficient for determining that those objects do not intersect. So, we can recursively traverse a world's tree and check whether separating planes intersect the bounding volume (see previous section). The easy way to perform this check is to test whether all parts of the object are on the same side of the plane. This computation is simple and fast using a plane equation $ax+by+cz+d=0$ and determine the side of the plane upon which the point set lie. In [NAT90] it is presented an algorithm to perform boolean operations between two BSPtrees. As a previous step to compute the boolean operations, they explain how to merge the BSPtrees. It is accomplished by subdividing one of the tree depending of the other one. The boolean operation is performed when the cell level is reached by classifying the regions (cells). This method is a improved version of one found in [TN87] where it is presented an algorithm for set operations problem. In this second paper, the approach takes a BSPtree as a one operand and a B–rep as a second and produces a new BSPtree determined by the set operation via modification of the original tree. Despite the paper [NAT90] focus on the set boolean operations between BSPtrees properly, the approach can be used for collision as well, since collision is equivalent to a non–empty intersection. From the collision point of view, their algorithm can be improved in the sense to reject interference as soon as this situation is detected.

3.3.5 KdTree

A KdTree, firstly described by Bentley [Ben75], is a binary space partition tree (a kind of BSPtree) whose cuts are chosen orthogonal to the coordinate axes. Thus, KdTree data structure is a multidimensional search tree in K dimensional space. Levels of the tree are split along successive dimensions. The KdTree method, is one of the methods used by Held et al. in [HKM95a]. The root node is associated with the workspace (box). Each node of the tree is associated with a box and implicitly with the set of objects that intersect that box. Each non–leaf node is also associated with a cut plane. To define the children of a node, we must to take two decisions: 1) Will the cut be orthogonal to the x, y or z axis? 2) At what value of the chosen coordinate axis will the partition take place? If the number of objects intersecting a box is less than a pre–specified threshold, then the box is not partitioned and the corresponding node is a leaf. With each leaf is explicitly stored the list of objects that intersect its box. In [Zac97], uses this method, with the name of *Boxtree*, making use of OBB as bounding volumes to regions associated to each node. After, in [Zac02] presents a *minimal hierarchical collision detection* method using *restricted boxtree*. The *restricted boxtree* is a tree of boxes that enclose the previous OBB to speed the collision test between boxes with AABB. His boxes are axis–aligned in object space. Unlike AABB trees, in the *restricted boxtree* the two children of a box cannot be positioned arbitrarily.

Held et al. [HKM95a] consider the problem of preprocessing a scene of polyhedral models to perform collision detection for an object that moves amongst obstacles. They presented a comparative analysis between several scene objects representations with KdTree, Rtrees and Grid of boxes (voxels). They implemented each method and give results from comparison.

To compute collision detection using KdTrees, we start by comparing the moving object M with the cut plane of the root. If M lies entirely on one side of the cut plane, then we visit recursively that corresponding child, otherwise, we visit both children. When we visit a leaf node we check each object belonging to the leaf with M.

3.4 Bounding Volume Hierarchies Methods

A Bounding–Volume–tree, *BV–tree*, is a tree where each node is associated with a subset of the input primitive objects, together with a bounding volume that approximates this subset with a smallest containing instance of some specified class of shapes, such as boxes, spheres, polytopes of a given class, etc. This class of tree is commonly used in the narrow phase at the progressive refinement levels. The tree construction can be in a top–down or in a botton–up fashion.

In this section we introduce several BVtrees that are the most commonly used. Before, we expose the R-tree, firstly introduced by Guttman [Gut84], because it provide the theoretical basics for bounding volume hierarchies which are mostly used to generate hierarchical representations of complex meshes, where the BV entity is some kind of box.

Using the R–tree representation the collision detection can be performed by checking for interference between boxes. R–tree is an organization of a set of geometric objects into a tree with each node associated with a subset of the objects, for which a minimal amount of geometric information is stored at the node (e.g. the axis–aligned bounding box of the subset of objects). The tree is generated in a *top–down* fashion. A leaf node contains some small constant number of primitive objects (e.g. triangular facets). The root node corresponds to the full set of objects. The children of an internal node represent a partitioning of the objects associated with that node. Variants of R–trees are: R^+–tree [SRF87], R^*–tree, Cone–tree, Prism–tree [PF87]. In the R^+–tree representation, introduced by Sellis, Roussopoulos and

Faloutsos in [SRF87], the main apportation with respect the R–tree, is that in the R^+–tree not overlap boxes are allowed. It means that if an object is inside two boxes in R–tree model, in the R^+–tree the object is truncated in two parts and then it is represented by two boxes containing one fragment each.

3.4.1 SphereTree

The model is based on representation of the objects by sets of spheres. It produces multiple levels of detail arranged in a hierarchy. To place the spheres the process can use the *medial–axis* surface which represents an object in skeletal form. The *medial–axis* surface guides a process that matches the spheres to the object shape [Qui94, Hub95, Hub96, OD99, BO03]. Another way to construct the model can be done by fitting spheres to a polyhedron by anchoring big spheres to points on the polyhedron and shrinking them until they just fit inside the polyhedron [RB79]. Spheres are rotationally invariant, so for a rigid object, the hierarchy is built once by a preprocess, a running application applies to this hierarchy the same linear transformations it applies to the object (fig. 16 shows an example of Sphere–tree).

Making use of Sphere–trees to represent objects, collision detection is solved by traversing the trees in parallel. Descending from the root (objects bounding sphere) to leaves. When spheres of homologous nodes overlap, a potencial collision between objects is detected. Detecting interference between two spheres is simply performed by comparing the distance between their centers and the sum of their radii [PSL92, Hub95, PG95, Hub96].

Hubbard presents in [Hub95] and after improve in [Hub96] a method to detect collision between objects using this BV tree. The algorithm assumes each object is approximated by a hierarchy of spheres which represents the object at multiple levels of detail. Level 0 is the objects bounding sphere. Subsequent levels are unions of successively more spheres, approximating the object at higher resolutions (see fig. 16). The detection algorithm uses these hierarchies to implement progressive refinement. The algorithm tests collisions between the level 0 of the hierarchies. If a collision between them is detected, then next phase descends one level in the hierarchies. The algorithm checks the colliding spheres at the current level of the two hierarchies to see if their children collide. If no spheres collide at the current level, then the pair of objects need no furher processing. Otherwise, the algorithm returns the colliding spheres. If a more detail of result is demanded, the algorithm proceed with the next refinement step. The algorithm can continue these steps as long as the hierarchies have levels and the application can spare the time. Quinlan [Qui94] presents an algorithm to construct an sphere–tree in a bottom–up fashion by beginning at leaves and using a binary tree, see figure 17.

3.4.2 AABBtree

Based on boxes where their facets are axis–aligned. Van Der Bergen [Ber97] makes use of this kind of BVtrees. He implemented a system to collision detection called *SOLID*. The tree can be built in a top–down way by partitioning along longest axis or, in a bottom–up way by joining boxes up to the root in a pre–processing step. The AABBs are refitted during simulations. The AABBtrees are suitable for most types of deformations. Larsson et al. in [LAM01] used AABBtrees in an hybrid refitted AABBtrees for continously deforming bodies. The method does arbitrary vertex repositioning. They improves making

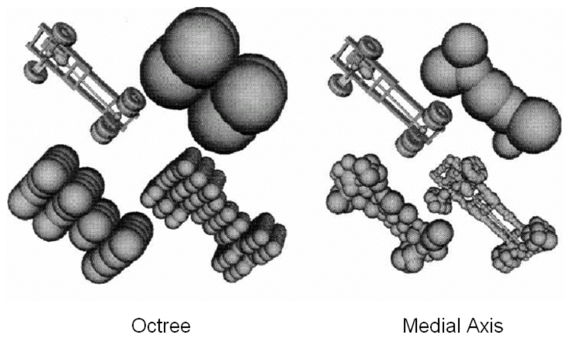


Figure 15: Example of sphere tree construction. Left: from octree–based. Right: From medial–axis–based. [Hub96]

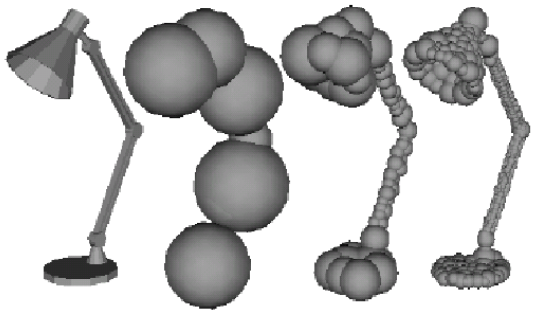


Figure 16: A desktop lamp and three levels of its sphere–tree built from the medial axis–surface [Hub95]

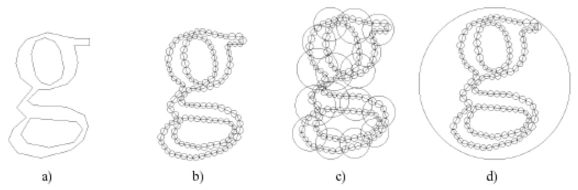


Figure 17: Bounding sphere–tree of an object [Qui94]

fast the refit technique than [Ber97].

Held et al. [HKM95a] consider the problem of preprocessing a scene of polyhedral models to perform collision detection for an object that moves amongst obstacles. They presented a comparative analysis between several scene objects representations with KdTree, Rtrees (that are AABBtree) and Grid of boxes (voxels). They implemented each method and give results from comparison.

Klein and Zachmann [KZ03] present a framework for hierarchical collision detection that can be applied to virtually all bounding volume (BV) hierarchies, they say. It allows an application to trade quality for speed. Using a probabilistic analysis, the algorithm yields an estimation of the quality, so that applications can specify the desired quality. In a time-critical system, applications can specify the maximum time budget instead, and quantitatively assess the quality of the results returned by the collision detection afterwards. Their framework stores various characteristics about the distribution of the set of polygons with each node in a BV hierarchy. They have implemented their approach using AABB trees and present performance measurements and comparisons with previous algorithms (figure 18 is an example of their results)

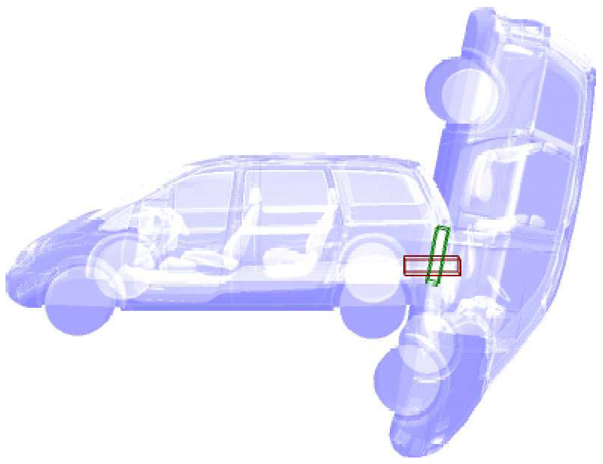


Figure 18: A snapshot of the Klein and Zachmann benchmarking procedure (objects are rendered transparent). The two boxes show one pair of BVs that contain at least one collision cell with high probability. [KZ03]

3.4.3 OBBtree

An Oriented Bounding Box tree [GLM96, BCG$^+$96, Got00] is a hierarchical structure that representate a set of objects by a binary–tree of boxes. Each object is enclosed by an oriented box (objects bounding box). The set of object boxes constitute the leaves of the tree. Then, the OBBtree [BCG$^+$96] is obtained by joining pairs of "neighboring" leaf nodes, and so on. The root node of the tree corresponds to the box that encloses the full set of objects. Different bounding–box–trees can be derived depending on the criteria for deciding pairs of neighboring nodes. The leaf nodes of the tree can represent a whole object or parts of it. For intance, the boxes associated to the leaves may represent facets of a polyhedral object, or may represent surface parts of object.

Gottschalk, Lin and Manocha [GLM96] describe an algorithm and implemented a system (*RAPID*) for exact interference detection between polyhedra whose surfaces have been triangulated. They use an OBB-tree to perform the collision detection test. The process computes the OBBtree getting a representation of objects and, then perform the intersection test between OBBtrees. The algorithm traverses two such trees and tests for overlaps between boxes based on a *separating axis theorem.* The intersection test is achieved by projecting the centers of the boxes onto the axis and computing the radii of the intervals. If the distance between the box centers as projected onto the axis is greater than the sum of the radii then the intervals (and the boxes as well) are disjoint, if it is less then the objects can potencially intersect. Generating the OBBtree from an original model with n triangles can be accomplished in $O(n^2 * log^2\ n)$ time if the convex hull is used and $O(n * log\ n)$ if it is not.

Hudson, Lin, Cohen, Gottschalk and Manocha [HLC+96] developped a method and a system *V–COLLIDE* for accelerate collision detection describing its interface for VRML (Virtual Reality Modeling Language). Figura 19 shows an example of an OBBtree.

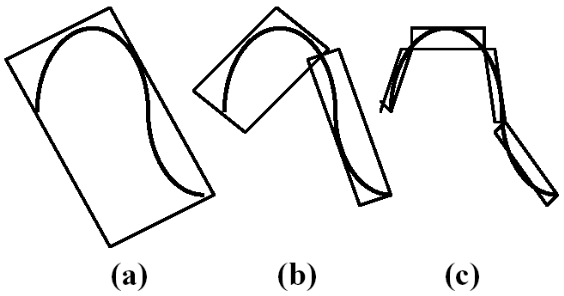


Figure 19: Example of three levels of an OBBtree [GLM96]

As it is pointed out in [JTT98] when the number of objects to check for collision is high in comparison with the workspace (i.e. when the density of objects is high), OBB representation is better than AABB or spheres as they do fit more tighly to the objects and then less interferences between bounding volumes are reported.

3.4.4 kDOPtree

Hierarchies of discrete oriented polytopes (kDOPtree) have been used by several authors [HKM+96, KHM+98, Zac98, MKE02]. Klosowski et al. [KHM+98] developped a system called *QuickCD* to built a kDOPtree in a top–down fashion. Mezger et al. [MKE02] used kDOPs to collision detection between deformable and flat shaped meshes like textiles (fig. 20 is an example of 18–DOPtree). Zachmann [Zac98] uses a DOPtree to demonstrate that is a faster and less memory occupancy that OBBtree used in the RAPID system [GLM96].

3.4.5 Convex–Hull–tree

Ehmann and Lin [EL01] describe an algorithm for construct a convex–hull tree based on vertex insertion which They use to decompose the surface of an object and compute convex–hulls of groups of objects.

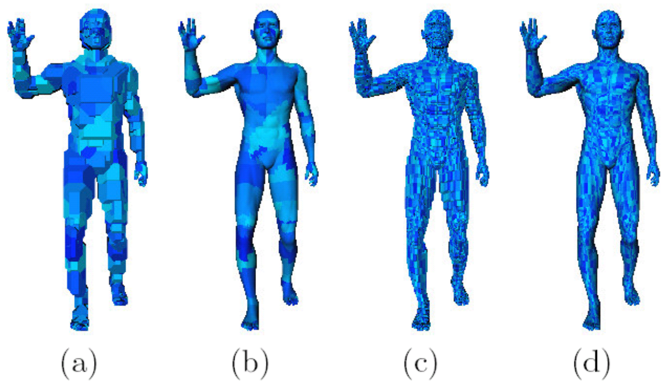


Figure 20: Two levels of an 18-DOP-hierarchy. (a) and (c) show the 18-DOPs, (b) and (d) the corresponding regions on the surface [MKE02]

They implement an automatic partitioning and grouping algorithm to group nearby neighbors when constructing the convex hierarchy. In the convex–hull hierarchy, each face is given one of three classifications *original, contained, free* (see fig. 21). Where *original* are the faces of the orginal surface, *contained* are faces created when converting patches to pieces by convex–hull and, finally, *free* are faces created when constructing the hierarchy. These classifications are the face relationship between a bounding volume at an internal node and the convex pieces at its leaves.

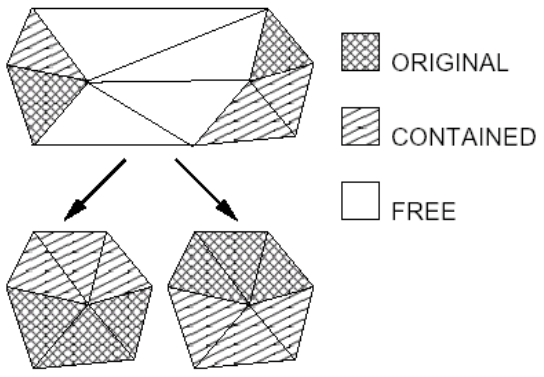


Figure 21: Example of convex–hull–tree [EL01]

Varadahn et al. [VKK$^+$03] compute the *max–norm* distance between a point and a wide class of geometric primitives. When computing the distance in a complex models they use the convex–hull–tree. To compute the distance for a non–convex polyhedron P they construct a hierarchical bounding volume of P culling away unnecessary triangles by traversing the hierarchy. For the hierarchical representation they employ a surface convex decomposition scheme similar to Ehmann et al. [EL01], but here, a leaf node in the BVH is created by decomposing P into a collection of convex surface patches P_i and computing its convex hull.

Then, the entire BVH is recursively built by merging children nodes in the hierarchy and computing their convex hull in a bottom–up fashion.

3.4.6 Swept–Sphere–Volume–trees

A Swept–Sphere–Volume–tree *SSVtree* [LaMLM99] is a hierarchy where the BVs at the nodes of a BVH belong to a family of three different swept sphere volumes. They correspond to a sphere (also known as point swept sphere or *PSS*), and more complex volumes obtained by sweeping along either an arbitrarily oriented line (line swept sphere or *LSS*) or along an arbitrarily oriented rectangle (rectangle swept sphere or *RSS*). These volumes are generated by using a single primitive and the Minkowski sum with an sphere. They provide algorithms to compute distance between any combination of these BVs based in computing the distance between the primitives and substract the swept sphere. Figures 22, 23 show one example of SSV.

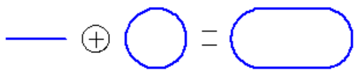


Figure 22: Right: Example of SSV building [LaMLM99]

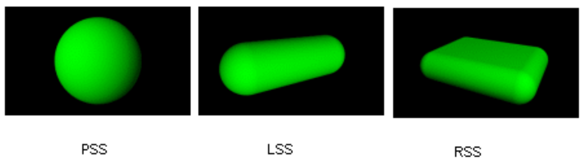


Figure 23: PSS Point swept sphere, LSS Line swept sphere, RSS Rectangle swept sphere [LaMLM99]

3.4.7 Computational Cost of Collision Query using Bounding Volume Hierarchies

Given two large input models and hierarchies built to approximate them, the time required to perform a collision query can be quantified as [GLM96]:

$$T = N_v C_v + N_p C_p$$

Where T is the total cost function for collision detection, N_v is the number of pairs of bounding volumes tested for overlap, C_v is the cost of testing a pair of bounding volumes for overlap, N_p is the number of pairs of primitives tested for contact and, finally, C_p is the cost of testing a pair of primitives for contact.

Klosowski [KHM+98] add to this equation the cost of updating the hierarchy as the moving object rotates. Thus, he proposed that for collision detection in motion simulation the total cost will be:

$$T = N_v C_v + N_p C_p + N_u C_u$$

Name	Type of entity	Some References
OcTree	3D cube or sphere	[SW98, Hub93, KTAK94]
BSPTree	hiperplanes	[TN87, NAT90, SGA02]
RegularGrids	voxel	[GASF94, MPT99, RPP+01, Bor03]
KdTree	ortoghonal hiperplanes	[SW98]
BucketTree	*Bucket* of polygons	[GDO00]

Table 2: Types of hierarchy space partition representations

Where T, N_v, C_v, N_p, C_p have been defined above, N_u is the number of nodes that must to be updated, and C_u is the cost of updating each such node.

As pointed out by Zachmann [Zac00] the performance of any collision detection based on hierarchical bounding volumes depends on two conflicting constraints:

1. The tightness of the BVs, which will influence the number of BV (N_v) test and,
2. The simplicity of the BVs (N_p), which determines the efficiency of an overlap test of BVs

Queries using simple AABBtrees or Spheretrees, exhibit large N_v and small C_v, while those using more complex bounding volumes, as Convex–hull or OBB have smaller N_v and larger C_v. Also, using simple BVs tends to increase N_p, since the leaf nodes of simple BVs are less likely to bound the models apart than those of complex BVs. Looking at this equation it is difficult to choice the best BV given the opposite tendencies of C_v and N_v. Thus, no single BV is the best choice for all ocasions and applications may perform best with different BV types. Therefore, the cost of a collision query depends not only on the input and output size, but also on the nature and degree of the proximity of the models. Input and output size are easily to quantified. However, proximity is not so easily quantified, since it should capture the shape complexity of the models and how those shapes spatially interact [Got00].

3.5 Summary of the Section

Table 2 and table 3 summarize the types of space partition representations and the bounding volume hierarchies respectivelly.

4 Summary of the Paper

In this paper the collision and proximity detection problem has been stated. A collection of approaches to achieve the problem from several point of views has been described. Solutions that are conceived to achieve different phases of the general problem have been presented and different approaches to solve the problem from different environments have been exposed. Figure 24 shows an example of the phases and shows

Name	Type of Bounding Volume	Some References
SphereTree	sphere	[PG95, Hub96, OD99, BO03]
AABBTree	axis-aligned bounding box, AABB	[HKM95b, Ber97, LAM01, KZ03]
OBBTree	oriented bounding box, OBB	[GLM96, BCG$^+$96, Got00]
kDOPTree	discrete oriented polytope defined by k vectors	[HKM$^+$96, KHM$^+$98, Zac98, MKE02]
ConvexHullTree	convex polytope	[EL01, OL03]
ConvexPieceTree	based on OBB	[FWG03]
restricted BoxTree	based on AABB	[Zac02]

Table 3: Types of bounding volume representations

how different data structures are used to solve different collision detection phases. The data structures that are appearing in the figure have been selected to illustrate the example any other combination of them could be used.

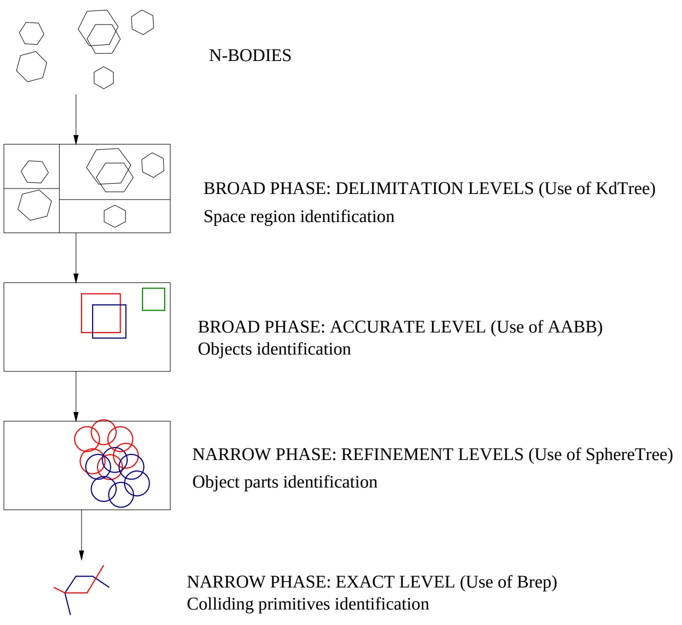

N-BODIES
BROAD PHASE: DELIMITATION LEVELS (Use of KdTree)
Space region identification
BROAD PHASE: ACCURATE LEVEL (Use of AABB)
Objects identification
NARROW PHASE: REFINEMENT LEVELS (Use of SphereTree)
Object parts identification
NARROW PHASE: EXACT LEVEL (Use of Brep)
Colliding primitives identification

Figure 24: Example in 2D of the collision detection test and phases.

5 Conclusions

In this paper, the problem of collision and proximity queries amongst moving objects has been described from several point views. We have reviewed several data structures and methods that achieve the problem that are in the literature. From that we have presented different strategies to solve the problem. The phases to solve the global problem of collision detection have been stated. Thus, collision can be treated in a *broad phase* and in a *narrow phase* and their subphases.

- When a non–enumerative or exact solution for collision is sought, hierarchical representations allow to point out on regions where interference is most likely to occur faster than using a tighly representations. This is true specially when the density of involved objects is low (no false collision detection between containers are reported).
- When a fine solution for collision is sought, or the density of objects on the scene is high, a more tightly representation of objects shapes is needed to identify which object parts cause the interference.

We can also conclude that, even, in the second case a coarse phase is always a benefit to reduce the number of regions where collision can truly occur, speeding up the whole interference test procedure.

As far as we know, and after having reviewed the literature on collision detection techniques it seems that no hierarchy structures that include different types of space patitioning and bounding volume shapes have been taken into account. This kind of structure can be very useful when the object shapes in the scene to evaluate collision are substantially different, such as ship or car design, where involved objects are a set of many different pieces.

References

[AdBG+01] P.K. Agarwal, M. de Berg, J. Gudmundsson, M. Hammar, and H.J. Haverkort. Box–trees and r–trees with near–optimal query time. In *7th Annual Symposium on Computational Geometry (SCG2001)*, pages 124–133, 2001.

[Bar90] D. Baraff. Curved surfaces and coherence for non–penetrating rigid body simulation. In *ACM SIGGRAPH Conf. Proc.*, pages 19–28, August 1990.

[BCG+96] G. Barequet, B. Chazelle, L.J. Guibas, J.S.B. Mitchell, and A. Tal. *BOXTREE*: A hierarchical representation for surfaces in 3*D*. In *EUROGRAPHICS Conf. Proc.*, volume 15, pages 387–396. Blackwell Publishers, August 1996.

[Ben75] J.L. Bentley. Multidimensinal binary search trees used for associative searching. *Communications of the ACM*, 18:509–517, 1975.

[Ber97] G. Van Der Bergen. Efficient collision detection of complex deformable models using aabb trees. *Journal of Graphic Tools*, 2(4):1–14, 1997.

[BJN92] P. Brunet, R. Juan, and I. Navazo. Octree representations in solid modeling. *Progress in Computer Graphics*, 1:164–215, 1992.

[BKOS00] M. Berg, M. Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry. Algorithms and Applications.* Springer-Verlag, 2000. ISBN 3-540-65620-0.

[BN90] P. Brunet and I. Navazo. Solid representation and operation using extended octrees. *ACM Transactions on Graphics*, 9(2):170–197, April 1990.

[BO03] G. Bradshaw and C. O'Sullivan. Adaptative Medial–Axis Approximation for Sphere–Tree Construction. *ACM Transactions on Graphics*, 22(4), 2003.

[Bor03] D. Borro. *Colisiones en Estudio de Mantenibilidad con Restitucin de Esfuerzos sobre Maquetas Digitales Masivas y Compactas.* PhD thesis, Universidad de Navarra, 2003.

[Cam90] S. Cameron. Collision detection by four-dimensional intersection testing. In *Proceedings of International Conference on Robotics and Automation*, pages 291–302, 1990.

[Cam97] S.A. Cameron. Enhancing GJK: Computing minimum penetration distances between convex polyhedra. In *Proc. of the Int. Conf. on Robotics and Automation*, pages 3112–3117, 1997.

[Can86] J. Canny. Collision detection for moving polyhedra. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(2):200–209, March 1986.

[CLMP95] J. Cohen, M. Lin, D. Manocha, and K. Ponamgi. $I - COLLIDE$: An interactive and exact collision detection system for large–scaled environments. In *ACM Int. 3D Graphics Conference*, pages 189–196, 1995.

[CW96] K. Chung and W. Wang. Quick collision detection of polytopes in virtual environments. In *Proc. of ACM Symposium on Virtual Reality Software and Technology*, 1996.

[DK83] D.P. Dobkin and D.G. Kirkpatrick. Fast detection of polyhedral intersection. *Theoretical Computer Science*, 27:241–253, 1983.

[DK90] D. P. Dobkin and D. G. Kirkpatrick. Determining the separation of preprocessed polyhedra – A unified approach. In *In Proc. 17th Internat. Colloq. Automata Lang. Program. Lecture Notes in Computer Science,*, volume 443, pages 400–413. Springer–Verlag, 1990.

[EL01] S.A. Ehmann and M.C. Lin. Accurate and fast proximity queries between polyhedra using convex surface decomposition. In A. Chalmers and T.M. Rhyne, editors, *Computer Graphics Forum. Eurographics'2001 Procedings*, volume 20(3), pages 500–510. Blackwell Publishing, 2001.

[FB97] M. Franquesa and P. Brunet. Analysis of methods for generating octree models of objects from their silhouettes. In *Proceedings of the MICAD*, volume 12, pages 33–65, 1997. ISBN 0298–0924.

[FDF$^+$93] J.D. Foley, A.Van Dam, S.K. Feiner, J.F. Hughes, and R. L. Phillips. *Introduction to Computer Graphics.* Addison-Wesley, 1993. ISBN 0-201-60921-5.

[FHA90] A. Foisy, V. Hayward, and S. Aubry. The use of awareness in collision prediction. In *IEEE Proceedings Interna. Conf. on Robotics and Automation*, pages 338–343, 1990.

[FNB90] M. Franquesa-Niubo and P. Brunet. Generating extended octree models of $3D$ real objects from a sequence of images. In *Proceedings of the MICAD*, volume 2, pages 752–772, February 1990. ISBN 2–86601–219–4.

[FWG03] Z. Fan, H. Wan, and S. Gao. Ibcd: A fast collision detection algorithm based on image space using obb. *To be appear in Journal of Visualization and Computer Animation*, 2003. URL: http//www.cad.zju.edu.cu/home/Zwfan/papers/ibcd02.pdf.

[GASF94] A. Garcia-Alonso, N. Serrano, and J. Flaquer. Solving the collision detection problem. *IEEE Computer Graphics and Applications*, pages 36–43, May 1994.

[GDO00] F. Ganovelli, J. Dingliana, and C. O'Sullivan. Buckettree: Improving collision detection between deformable objects. Spring Conference in Computer Graphics (SCCG2000), 2000. Bratislava.

[GJK88] E.G. Gilbert, D.W. Johnson, and S.S. Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal of Robotics and Automation*, 4(2):193–203, April 1988.

[GLM96] S. Gottschalk, M.C. Lin, and D. Manocha. *OBBtree*: A hierarchical structure for rapid interference detection. In *ACM SIGGRAPH Conf. Proc.*, pages 171–180, August 1996.

[Got00] S. Gotschalk. *Collision Queries using Oriented Bounding Boxes.* PhD thesis, University of North Carolina. Department of Computer Science, 2000.

[Gut84] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *In Proc. ACM SIGMOD Int. Conference on Management of Data*, pages 47–57, June 1984. Boston, MA, USA.

[Hay86] V. Hayward. Fast collision detection scheme by recursive decomposition of a manipulator workspace. In *IEEE Int. Conf. on Robotics and Automation. Sn Francisco (CA)*, volume 2, pages 1044–1049, July 1986.

[HH96] K. Hamada and Y. Hori. Octree–based approach to real–time collision–free path planning for robot manipulaiton. *ACM96–MIE*, pages 705–710, 1996.

[HKM95a] M. Held, J.T. Klosowski, and J.S.B. Mitchel. Evaluation of collision detection methods for real reality fly–throughs. In C. Gold and J.M. Robert, editors, *Proc. seventh Canadian Conf. on Computer Geometry*, pages 205–210, August 1995.

[HKM95b] M. Held, J.T. Klosowski, and J.S.B. Mitchell. Speed comparison of generalized bounding box hierarchies. Technical report, Applied Math, SUNY Stony Brook, 1995.

[HKM$^+$96] M. Held, J.T. Klosowski, J.S.B. Mitchell, H. Sowizral, and K. Zikan. Real-time collision detection for motion simulation within complex environments. Technical report, Applied Math, SUNY Stony Brook, 1996.

[HLC$^+$96] T. Hudson, C. Lin, J. Cohen, S. Gottschalk, and D.Manocha. V-collide: Accelerated collision detection for vrml. http://www.cs.unc.edu/v_collide, 1996. manocha@cs.unc.edu.

[Hub93] P. M. Hubbard. Interactive collision detection. In *Proc. IEEE Symp. on Research Frontiers in Virtual Reality*, volume 1, pages 24–31, October 1993.

[Hub95] Philip M. Hubbard. Collision detection for interactive graphics applications. *IEEE Transactions on Visualization and Computer Graphics*, 1(3):218–230, September 1995.

[Hub96] Philip M. Hubbard. Aproximating polyhedra with spheres for time–critical collision detection. *ACM Transactions on Graphics*, 15(3):179–210, July 1996.

[JTT98] P. Jimenez, F. Thomas, and C. Torras. *Book: Robot Motion. Planning and Control. Chapter: Collision Detection Algorithms for Motion Planning*, pages 305–343. Springer-Verlag, 1998. ISBN 3–540–76219–1.

[JTT01] P. Jimenez, F. Thomas, and C. Torras. (3d) collision detection: A survey. *Computers and Graphics*, 25(2):269–285, August 2001.

[Kam93] V.V. Kamat. A survey of techniques for simulation of dynamic collision detection and response. *Computers and Graphics*, 17(4):379–385, 1993.

[KHM^{+}98] J. T. Klosowski, M. Held, J. S.B. Mitchel, H. Sowizral, and K. Zikan. Eficient collision detection using bounding volume hierarchies of k–dops. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):21–36, january-march 1998.

[KR03] B. Kim and J. Rossignac. Collision prediction for polyhedra under screw motions. In *Proc. of the ACM SM'03*, pages 4–10, June 2003. Seatle, Washington.

[KTAK94] Y. Kitamura, H. Takemura, N. Ahuja, and F. Kishino. Efficient collision detection among objects in arbitrary motion using multiple shape representation. In *Proceedings 12th IARP Inter. Conference on Pattern Recognition*, pages 390–396, October 1994.

[KZ03] J. Klein and G. Zachmann. Probability–guided collision detection. Technical report, University of Paderborn, 2003. tr–ri–03–242.

[LAM01] T. Larsson and T. Akenini-Mller. Collision detection for continously deforming bodies. Eurographics Conference. Short presentations, 2001.

[LaMLM99] E. Larsen, S. Gottshalk ans M. Lin, and D. Manocha. Fas proximity queries with swept sphere volumes. Technical report, Dept. Computer Science, University of North Carolina, 1999. TR99–018.

[LAN91] Y-H. Liu, S. Arimoto, and H. Noborio. A new solid model *hsm* and its application to interference detection between moving objects. In *J. Robotic Systems*, volume 8, pages 39–54, 1991.

[LC91] M.C. Lin and J.F. Canny. A fast algorithm for incremental distance calculation. In *Proc. of the IEEE Inter. Conf. on Robot and Automation. Sacramento. CA*, volume 2, pages 1008–1014, 1991.

[LK02] O.S. Lawlor and L.V. Kal. A voxel–based parallel collision detection algorithm. In *Proc. of the Int. Conf. on Supercomputing*, pages 285–293, June 2002. New York.

[LM03] M.C. Lin and D. Manocha. *Handbook of Discrete and Computational Geometry Collision Detection*, chapter 35. CRC Press LLC, 2003. To appear.

[Mir98] B. Mirtich. V–Clip: Fast and robust polyhedral collision detection. *ACM Transactions on Graphics*, 17(3):177–208, July 1998.

[MKE02] J. Mezger, S. Kimmerle, and O. Etzmuß. Improved Collision Detection and Response Techniques for Cloth Animation. Technical Report WSI–2002–5. ISSN 0946–3852, Universität Tübingen, 2002. colisions.

[Mou97] D. M. Mount. *Book: Discrete and Computatinal Geometry. Chapter: Geometric Intersection*, pages 615–630. CRC Press LLC, 1997. ISBN 0–8493–8524–5.

[MPT99] W.A. McNeely, K.D. Puterbaugh, and J.J. Troy. Six–degrees–of–fredom haptic rendering using voxel sampling. In *Proceedings of SIGGRAPH 99*, pages 401–408, August 1999. ISBN: 0–20148–560–5.

[MW88] M. Moore and J. Wilhelms. Collision detection and response for computer animation. *ACM Computer Graphics*, 22(4):289–298, August 1988.

[NAT90] B.F. Naylor, J. Amanatides, and W. Thibault. Merging *bsp* trees yields polyhedral set operations. In *ACM Computer Graphics*, volume 24, pages 115–124, August 1990.

[OD99] C. O'Sullivan and J. Dingliana. Real–time collision detection and response using sphefe–trees. In 15th Spring Conference on Computer Graphics, April 1999. ISBN: 80–223–1357–2.

[OL03] M.A. Otaduy and M.C. Lin. Clods: Dual hierarchies for multiresolution collision detection. Eurographics Symposium on Geometry Processing (2003). Aachen, Germany, 2003. coliosions.

[O'S99] C. O'Sullivan. *Perceptually–Adaptive Collision Detection for Real–time Computer Animation.* PhD thesis, University of Dublin, Trinity College Department of Computer Science, June 1999.

[PF87] J. Ponce and O. Faugeras. An object centered hierarchical representation for $3D$ objects: The *Prism* tree. *Computer Vision Graphics and Image Processing*, 38:1–28, 1987.

[PG93] N. Pla-Garcia. Boolean operation and spatial complexity of *FaceOctrees*. In *EUROGRAPHICS Conf. Proc.*, volume 12, pages 153–164. Balckwell Publishers, 1993.

[PG95] I.J. Palmer and R.L. Grimsdale. Collision detection for animation using sphere-trees. *Computer Graphics Forum*, 1995.

[PML95] M. Ponamgi, D. Manocha, and M.C. Lin. Incremental algorithms for collision detection between solid models. In *In Proceedings of the Third ACM Symposium on Solid Modeling and Applications*, pages 293–304, May 1995.

[PML97] M.K. Ponamgi, D. Manocha, and M.C. Lin. Incremental algorithms for collision detection between solid models. *IEEE Transactions on Visualization and Computer Graphics*, 3(1):51–67, 1997.

[PS85] Franco P. Preparata and Michael I. Shamos. *Computational Geometry, An Introduction.* Springer-Verlag, 1985. ISBN 0-387-96131-3.

[PSL92] A.P. Del Pobil, M.A. Serna, and J. Llovet. A new representation for collision avoidance and detection. In *IEEE Int. Conf. on Robotics and Automation (Nice)(France)*, volume 1, pages 246–251, May 1992.

[Qui94] S. Quinlan. Efficient distance computation between non–convex objects. In *Proceedings of the IEEE Int. Conf. on Robotics and Automation*, pages 3324–3329, 1994. San Diego, CA.

[RB79] J.O. Rourke and N. Badler. Decomposition of three–dimensional objects into spheres. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI–1(3):295–305, July 1979.

[Rit90] J. Ritter. An efficient bounding sphere. Graphics Gems I. Academic Press, 1990. http://www.acm.org/pubs/tog/GraphicsGems/gems/BoundSphere.c.

[RPP$^+$01] M. Renz, C. Preusche, M. Ptke, H.P. Kriegel, and G. Hirzinger. Stable haptic interaction with virtual environments using an Adapted Voxmap–PointShell algorithm. In *Proceedings of the Eurohaptics Conference.Birmingham, UK*, 2001.

[Sam90] H. Samet. *The Design and Analysis of Spatial Data Structures.* Addison-Wesley, 1990. ISBN 0-201-50255-0.

[Sei90] R. Seidel. Linear programming and convex hulls made easy. In *In Proc. 6th Ann. ACM Conf. on Computational Geometry*, pages 211–215,, 1990. Berkeley,California.

[SGA02] A. Sigal, M. Gil, and T. Ayellet. Deferred, self-organizing bsp trees. *EUROGRAPHICS*, 21(3), 2002. http://www.ee.technion.ac.il/ ayellet/Ps/paper107.pdf.

[SRF87] T. Sellis, N. Roussopoulus, and C. Faloutsos. The R^+–tree: A dynamic index for multidimensional objects. In *Brighton 13th.*, pages 507–518. VLDB Conf., 1987.

[ST85] H. Samet and M. Tamminen. Bintrees, *CSG* trees, and time. In *ACM SIGGRAPH Conf. Proc.*, pages 121–130, July 1985. Computer Graphics vol. 19 num. 3.

[SW98] H. Sammet and R.E. Webber. Hierarchical data structures and algorithms for computer graphics. *IEEE Computer Graphics and Applications*, 8(3):48–68, May 1998.

[TN87] W.C. Thibault and B.F. Naylor. Set operations on polyhedra using binary space partitioning trees. In *ACM Computer Graphics*, volume 21, pages 153–162, July 1987.

[VKK$^+$03] G. Varadhan, S. Krishnan, Y.J. Kim, S. Diggavi, and D. Manochal. Efficient max-norm distance computation and reliable voxelization. Eurographics Symposium on Geometry Processing (2003), 2003. To appear.

[Zac97] G. Zachmann. Real–time and exact collision detection for interactive virtual prototyping. In *DETC'97. ASME Design Engineering Technical Conferences. Sacramento. California*, pages 14–17, September 1997.

[Zac98] G. Zachmann. Rapid collision detection by dynamically aligned dop–trees. In *Proc. of IEEE Virtual Reality Annual Inter. Symposium*, March 1998. Atlanta, Georgia.

[Zac00] G. Zachmann. *Virtual Reality in Assembly Simulation– Collision Detection, Simulation Algorithms, and Interaction Techniques.* PhD thesis, Dem Fachbereich Informatik der Technischen Universitat Darmstadt engereichte, 2000.

[Zac02] G. Zachmann. Minimal hierarchical collision detection. In *Proc. ACM Symposium on Virtual Reality Sotfware and Technology (VRST)*, pages 121–128, Hong Kong, China, November 2002. ISBN:1–58113–530–0.