TCR/LSI-98-11-R

Collision Detection: Models and Algorithms

Marta Franquesa
Pere Brunet

Report LSI–98–11–R

Collision Detection: Models and Algorithms

Marta Franquesa Pere Brunet

24th February 1998

Abstract

Collision detection between several objects in a closed space is an interesting research topic in all those areas where objects must not penetrate one another, such as Computer Graphics, Computer Vision, Virtual Reality. In this paper, we review data structures that are useful for collision detection and several methods that treat the collision problem. The paper is focused on collision detection in complex systems with a large number of objects and specially centered in the first step algorithm (the *broad stage*) where relevant regions for collision are detected.

1 Introduction

Whenever two objects attemp to interpenetrate each other, we call it a collision. Collision detection between objects is of fundamental relevance in order to achieve efficiency and realism in several areas such as robotics, motion planning, computer animation, dynamic simulation, virtual reality and in all those applications where objects can not interpenetrate one another. The two main issues involved 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] or [FHA90]. Collision detection is a geometric interference problem that depends on the spatial relationship of objects.

The collision detection problem can be attached 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 and trajectory parameterization. In the present document, we are interested in static collision. For more information about dynamic collision approaches see [JTT98].

In general collision algorithms consist of two main stages. In the first, *broad stage*, an approximate test is performed to identify interfering objects in the entire workspace using

a coarse representation of object shapes (simplified forms of objects such as bounding volumes). In the second, *narrow* or *fine stage*, a tightly representation of object shapes is used to accurately identify any object parts that cause interference and collision. In this last stage the algorithms focus its attention only on objects (or regions) that are likely to collide.

The goal of the broad stage is to restrict the application of the interference test to those object parts at which a collision can truly occur. For this purpose objects are usually modeled by hierarchies of simple shapes (such as bounding boxes or spheres). Hierarchical approximations allow to determine which regions are susceptible of interfering. Then fine interference test need only be applied within the relevant regions.

The remainder of this paper is structured as follows. Section 2 explains the collision problem. Section 3 gives an overview of the main data estructures that are able to attach collision detection. Section 4 reviews the collision–detection literature. Section 5 discusses the methods and algorithmic schemas for collision detection. Finally, section 6 summarizes some conclusions.

2 The problem

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.

Taking into account the output sought we can classify the methods in four different classes:

- 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.

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 involved in. More detailed information about interference between object types can be found in [Mou97] and [Kam93].

- 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 1.

- 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.

3 Data Structures

In this section we describe several data structures that are able to be useful to attach the collision detection problem. 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. The criterion followed to build hierarchical volume representations can be based on object partition or on subregions of the space partition.

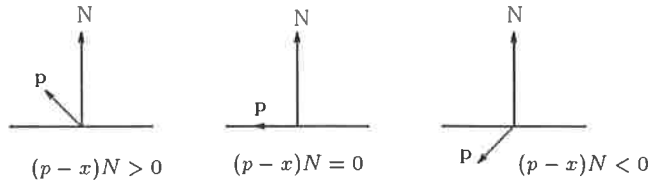


Figure 1: Collision detection with plane and point mass

- Space Partition: In this class hierarchical volume representations based on space partition are considered, such as Bintrees, BSPtrees, Octrees and Hierarchy of space subdivisions.
- Object Partition: In this class hierarchical volume representations based on object partition are considered, such as Sphere–trees, Bounding Boxes, R–trees, Prism–trees. In this class we have also considered the models that represent each object as a partition on subobjects (or primitive–objects) as B–rep and others polyhedral representations.

3.1 Space Partition

3.1.1 Bintrees

A bintree is a recursive spatial subdivision. It can be considered as a generalization of a quadtree and octree (see section 3.1.3). The bintree is a dimension-independent variant of the quadtree an octree representations. The bintree determinates 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.

3.1.2 BSPtrees

Binary–space–partitioning tree (BSP) 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 BSP–tree 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 subdivided, then 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] the algorithms required to perform boolean set operations between two objects represented by BSPtrees are presented.

3.1.3 Octrees

An octree is a tree of degree eight which represents the space occcupied by objects contained in the world space defined by a *universe cube*. 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 gray 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, gray, or white, respectively. This decomposition is carried out recursively for gray 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]. 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], [Fra90]. 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, these nodes are called *face nodes*. In the *Extended–octree*, vertices, edges, and faces nodes are allowed leaves, too.

3.1.4 Hierarchy of space subdivisions

The hierarchy of space subdivisions describes a surface based on a multiresolution geometric models that support a representation and processing of spatial entities at different level of detail. Floriani, Magillo and Puppo present how to build and traverse a surface

at variable resolution in [FMP97]. They propose different methods to construct a *Multi-Triangulation* (MT) to represent free–form surfaces. Once the MT is generated, they consider two basic spatial operations on the multiresotulion surface model. These operations are: extraction of representation of minimum size at a resolution variable over the domain (making use of a thereshold function) and point location at variable resolution. The core of this appropach is the technique used to produce polyhedral approximations of the underlying surface based on suitable selected subsets of the data points. To represent the MT a directed acyclic graph (DAG) is used. Despite this representation seems useful to solve the collision problem, as far as we know, no papers have been published in this sense.

3.2 Object Partition

In this section are included two type of object partition strategies:

- Single object partition: Partition of object on subobjects (*primitive–objects*).
- Partition of the workspace on a set of objects

3.2.1 B–rep

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].

3.2.2 SFOG

Jiménez and Torras [JT96] introduce the *Spherical Face Orientation Graph* (SFOG), a new spherical representation of polyhedra inspired by the *Extended Gaussian Images* (EGI) (for a detailed description of EGI [Hor84]). The representation explicitly captures geometrical and topological information of a polyhedron. Topological relations of adjacency between faces are explicitly depicted with arcs joining nodes that correspond to faces sharing an edge.

3.2.3 R–trees

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. In [BCG+96] can be found a detailed definition of this representation. Variants of R–trees are: R^+–tree, R^*–tree, Cone–tree, Prism–tree.

The R^+–tree representation was introduced by Sellis, Roussopoulos and Faloutsos in [SRF87]. The main apportation of R^+–tree 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.

The Prism–tree representation use truncated tetrahedra to bound objects, and bounding convex polyhedra based on a small fixed set normal directions. Ponce and Faugeras introduce in [PF87] this data structure to model 3D objects. They define the Prism–tree as a "face" representation that describes polyhedra as a ternary tree of prisms as enclosing boxes (truncated pyramids). Each leaf node of the tree represents a triangle of the object surface and it stores geometrical and structural information. The geometric information part is a simple box.

3.2.4 Sphere–trees

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 [Hub95] and [Hub96]. 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.

3.2.5 Bounding–Boxes

A Bounding–box–tree is a hierarchical structure that representate a set of objects by a binary–tree of boxes. The tree is generated in a *bottom–up* fashion: Each object is enclosed by a box (objects´ bounding box). The set of object boxes constitute the leaves of the tree. Then, the BOX–tree 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 [BCG+96].

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. In [GASF94] the terminal boxes represent whole objects. In [BCG+96] the leaf nodes represent facets of a triangulated surface.

Barequet, Chazelle, Guibas, Mitchell and Tal [BCG+96] present the *BOX*-tree data structure for representing triangulated surface in 3D. In their paper, the Bounding–box–tree is a binary tree whose leaves are in bijection with the facets of a triangulated surface. Each leaf of the tree is associated with a box that encloses the corresponding facet. In a similar manner, each internal node is associated with a box that encloses the boxes of its two children.

Variants of Box–trees are:

- *AABB* Based on boxes where the shape of them are *axis–aligned–bounding–boxes*
- *OBB* Based on boxes where the shape of them are arbitrary *oriented–bounding–boxes*

3.2.6 BV–trees

A Bounding–Volume–tree (*BV–tree*) is a treewhere 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 model is used by Held et altri in [HKM+96] and [HKM95] to achieve the collision detection. Due to that this data structure admits different types of bounding volumes it seems to be useful when several object shapes are involved in the collision detection. As it is pointed ou in [HKM+96] the tree construction can be in a top–down or in a botton–up fashion.

3.3 Summary of Data Structures

In this section some tables that classifies the methods depending on the data structures are presented. In the table 1 it is presented a classification of the data structures depending on the criterion followed to model the space. As it has exposed above the structures can be classified in two main groups. One group that subdivide the space (enclosing objects) and the other one that subdivide the objects properly. In the same table there are the data structures that are constructed based on the axis aligned geometry and data structures that are free aligned. The second means that the structure is build having into account the local orientation of objects. In the table 2 it is presented the relationship between the data structure used by methods and the shape of objects allowed by them. Note that the method [HK97] is included in surface objects while it is for a volumetric objects, for that reason it is marked by $*$ (for more information see section 4.3). Table 3 shows the

	Subject of partition	
	Space	Object
Axis Aligned Structure	Octree Foct Bintree	AABB–tree
Object Oriented Struc.	BSPtree	OBB–tree R^+–tree BV–tree
Others	H–Sp–Subdivision	Sphere–tree Prism–tree B–rep SFOG

Table 1: *Classification of Data–Structures depending on the partition subject*

relationship between the data structure used by methods and the kind of objects allowed by them.

	Shape of objects	
	Polyhedral	Surfaces
B–rep	[DK83] [MW88]	
BSPtrees	[TN87] [NAT90]	
Octrees	[KTAK94]	[PG93]
Spheres	[RB79] [Hub96]	
AABB–trees	[PML95]	
OBB–trees	[GASF94] [GLM96]	[BCG+96]
R^+–trees	[SRF87]	
Prism–trees	[PF87]	
H–Sp–Subdivision		[FMP97]
BV–trees	[HKM+96]	
Others	[JT96]	[HBZ90] [SWF+93] [Bar90] [MW88] [HK97] *

Table 2: *Relationship between Data Structures and shape of objets*

	Type of objects	
	general objects	convex objects
B–rep		[DK83] [MW88]
BSPtrees	[TN87] [NAT90]	
Octrees	[PG93]	[KTAK94]
Spheres	[Hub96]	[RB79]
AABB–trees	[PML95]	
OBB–trees	[GASF94] [GLM96] [BCG+96]	
R^+-trees		[SRF87]
Prism–trees		[PF87]
H–Sp–Subivision	[FMP97]	
BV–trees	[HKM+96]	
Others	[HBZ90] [SWF+93] [Bar90] [MW88] [JT96] [HK97]	

Table 3: *Relationship between Data Structures and kind of objets allowed*

4 Collision detection algorithms

In this section a brief description of collision detection algorithms found in the literature are presented. The algorithms have been grouped depending on the model used to represent objects.

4.1 Collision Detection Using Space Partitioning Representations

4.1.1 Collision detection using BSPtrees

In [NAT90] it is presented an algorithm to perform boolean operations between two BSP–trees. As a previous step to compute the boolean operations, they explain how to merge the BSP–trees. 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 new version of one found in [TN87] where it is presented an algorithm for set operations problem. In this second paper, the approach takes a BSP tree as a one operand and a B–rep as a second and produces a new BSP tree 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.

4.1.2 Collision detection using Octrees

Using octree representation, 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 gray 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 gray 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 object´s 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.

Pla–Garcia presents in [PG93] a method to find the approximate intersection region between two smoth surfaces using the variant model of octrees named *Foct* (*Face–octree*). The method is useful when a coarse intersection curve is required as a result of the collision analysis. The contribution of her method is pointed when intersecting homologous *Face* nodes. In this case, the space corresponing to a face node is divided in three different portions: an interior part, an exterior part and a tolerance band. The relation between the homologous face nodes is classified according to the relative position of their tolerance bands. If the intersection of the tolerance bands is empty the two surfaces do not collide in the region corresponding to the current node. If the tolerance bands intersect the current cube, two cases have to be distinguished. If one band goes through the other, the surfaces have to interfere. Otherwise, if no band goes through the other nothing can be assumed about the intersection of the surfaces in the current cube. The paper does not focus on collision detection properly, but the method proposed in [PG93] may be useful for this goal.

4.2 Collision Detection Using Object Partitioning Representations

4.2.1 Collision detection using Polyhedral Representations

Polyhedral shape representation is one of the most common shape representations used to model 3D objects. Collision between polyhedral objects is detected by testing all combinations of faces and edges. The average time complexity for the test is $O(n^2 * e * f)$, where n is the number of objects, e the number of edges, and f the number of faces in the average object [Mou97].

The computational cost is high for a large number of objects. This cost may be acceptable and the method may be useful when the number of faces and edges that are likely to collide is restricted. For that reason many algorithms solve collision problem in several stages using an hybrid representation of objects. The *B–rep* of the objects is only used in the narrow stage when it is known that the objects potencially collide.

Dobkin and Kirkpatrick [DK83] and [Kir83] describe an 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. The method may be useful when only detection is required.

Moore and Wilhelms [MW88] describe two collision detection algorithms. One deals with triangulated surface representations of objects and other that applies to objects modeled as rigid polyhedra. The first algorthim is described in the secction 4.3. The second one is good if only detection is required. It assumes that the two polyhedra are convex, concave polyhedra can be decomposed into collections of convex ones before applying the method. This algorithm works by testing whether representative points of one polyhedron are inside the other one. The algorithm terminates when a single point of interpenetration is found.

Jiménez and Torras [JT96] use the SFOG representation to evaluate collision of two polyhedra. Using the SFOG of the two polyhedra, they compute the applicability constraints to restrict the set of candidates to undergo elementary edge–face intersection tests. Their algorithm is based on the fact that when two initially non–intersecting polyhedra undergo an arbitrary relative translational motion with respect to one another, only certain edge–face intersections can occur first. The applicability constraints allow one to determine the basic vertex–face and edge–edge contacts that are possible between two polyhedra whose relative orientation remains fixed but which are allowed to translate.

4.2.2 Collision detection using R–trees

Using the R–tree representation the collision detection can be performed by checking for interference between boxes. While boxes can overlap in this model, it is difficult to reject collision. No papers we have found that solve the collision detection using R–trees. Using the R^+–tree variant [SRF87] it is possible to attach the problem, due to that the boxes contain disjoint sub–regions of the space.

Ponce and Faugeras [PF87] present a method to detect intersection between two objects modeled as Prims-trees (a variant of R–tree). Their method operates following some properties. If two prisms (truncated pyramids boxes shape) do not intersect, then the underlying surfaces do not intersect either, in this case the associated nodes have a *null intersection*. The converse proposition is, or can be, false in general. Two surfaces can have an empty intersection, although the associated prisms intersect. In fact, it is impossible that surfaces associated to two nodes intersect before the leaf level, as the surfaces are only represented at this level. In this case the intersection is a *possible intersection*. At the leaf node, it is possible to decide if surfaces associated to the two leaves intersect by testing directly each face of one against each face of the other. If any of these couples intersect it is considered that the two surfaces have a *clear intersection*. Otherwise, a *null intersection* is assumed.

4.2.3 Collision detection using Sphere–trees

Making use of Sphere–trees to represent objects, collision detection is attached by traversing the trees in parallel. Descending from the root (object´s 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.

Hubbard presents in [Hub95] and after improve in [Hub96] a method to detect collision between objects using this model of representation. 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 object´s bounding sphere. Subsequent levels are unions of successively more spheres, approximating the object at higher resolutions. 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. O´Rourke and Badler [RB79] present an early algorithm of using xspheres to representate objects and make operations with them. Despite the last paper focus on the descomposition of objects into spheres properly, it has to be mentioned as an introduction to this kind of collision method.

4.2.4 Collision detection using Bounding–Boxes

Starting at the roots of the trees, the collision detection between Box–trees is performed by checking for interference between boxes at the same lebel of the trees, in other words, the test is done by comparing homologous nodes of the two trees. If a collision between boxes is detected, then both children are explored and recurse. This is the same search procedure found in *Octrees*, *BSP–trees* and *R–trees*.

Garcia–Alonso, Serrano and Flaquer [GASF94] design a method to check if two objects interfere. When two objects can collide, the method computes the interference between pairs of 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). Then, If the *minimax test* between the containers of two objects 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.

Ponamgi, Manocha and Lin [PML95] 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.

Gottschalk, Lin and Manocha [GLM96] describe an algorithm for exact interference detection between polyhedra whose surfaces have been triangulated. They use the OBB–tree (*Oriented Bounding Box*) to perform the collision detection test. The process compute the OBB–tree 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 OBB-tree 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. More recently Hudson, Lin, Cohen, Gottschalk and Manocha [HLC+96] developped an algorithm for accelerate collision detection describing its interface for VRML (Virtual Reality Modeling Language).

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.

4.2.5 Collision detection using BV–trees

In [HKM+96] an approximation collision detection method using BV–trees is presented. The bounding volume shapes are convex polytopes (*discrete orientation polytopes or k–dops*). These K–dops have their facets determined by planes whose normals come from a fixed set of k orientations. Thus *AABB* are 6–dops, where the orientation vectors coincide with the positive and negative coordinate axes. The BV–tree approximate the objects by covering it with a hierarchy of bounding volumes. Each node in the hierarchy corresponds to a set of subobjects, together with a bounding volume that covers the subobjects in an

optimal way subject to some function as the *size* of the node. In the paper [HKM+96] there is no the algorithm to attach the collsion problem, only *experimental results* for collision detection between flying objects are exposed.

4.3 Collision detection using other representations

Moore and Wilhelms [MW88] present an algorithm of collision detection for flexible surfaces. Surfaces are modeled as a grid of points connected to form triangles. Collisions between surfaces are detected by testing for penetration of each vertex point through the planes of any triangle not including that vertex. In fact, they use the polygon–polygon collision detection technique for flexible surfaces.

Herzen, Barr and Zatz [HBZ90] give a method for collision detection for time dependent parametric surface that are continuous. The upper bounds of the parametric derivatives make possible to guarantee the successful detection of collision. The method works with many types of surfaces including bicubic patches. More recently, Snyder, Woodbury, Fleischer, Currin and Barr [SWF+93] combine interval analysis with the Newton–Raphson root finding technique providing an accurate detection for collisions involving curved surface. These technique assume knowledge of every object´s exact position throughout simulation time. This information is avalaible to applications that generate prescripted animations off–line and the algorithm works well in that setting.

Baraff proposes [Bar90] a method to speed up the collision algorithm making use of the geometric coherence between successive time steps. This method is based on the fact that a pair of convex objects do not penetrate if and only if a separating plane between them exists. nonpenetration constraint to define a characteristic function ω. This function intuitively defines a distance between two objects near the point of contact. The function ω is chosen for each contact point such that it is twice differentiable. Further, this scalar function is positive if objects are disjoint, zero if objects are in contact and, negative if objects are interpenetrating. In [Bar90] Baraff uses the concept of *extreme distance* to formulate the characteristic function for curved surfaces analogous to one given for polyhedral objects. The *extreme distance* between the objects X and Y near the point of contact is defined as (see figure 2): If X and Y are disjoint, then the *extreme distance* is just the normal minimum distance; If X and Y are in contact, then the *extreme distance* is zero; If X and Y are interpenetrated, then the *extreme distance* is maximum distance between X and Y. Non convex objects must be treated as a union of convex pieces.

He and Kaufman have developped an algorithm to detect collision for volumetric objects [HK97]. A volumetric object is a discrete representation of a predominately heterogeneous object (for a detailed definition [NPT93]). In volumetric terms a collision is considered detected when a voxel address from one object is written into a map cell that has already been occupied by the voxel address of another object. Performing the collision test following this way could consume a high amount of memory. Inherently the interaction between

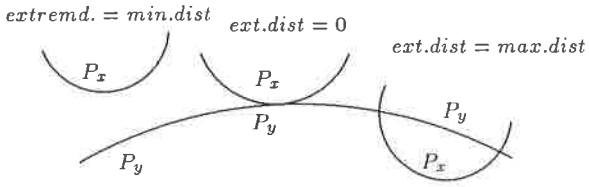


Figure 2: Extreme distance between two objects

volumetric objects is not a binary process. The collision, in this case, can be defined as a non–deterministic process, where a probability of collision is given at each space point to indicate the interaction property between different materials of the objects. They propose a probability model for each object. In this model each continuous point inside a volume is assigned a value in the range $[0, 1]$. This value can be seen as the probability that there exists a *surface* (in volumetric sense of meaning) crossing that point. The probability that two objects interact (that not collide as it is understood) at certain point can be defined as the probability that the *surfaces* in the two objects intersect at that point.

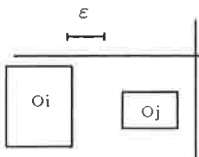
Due to that surfaces in volumetric terms are conceptually distinct than others, in table 2 the method of [HK97] is marked by $*$ to indicate this difference of meaning.

5 Analysis of the methods

5.1 Classification

As it has been introduced in the section 2, algorithms can be classified in different classes depending on the output sought. Four kind of algorithms can be distinguished:

- *ECA*: Exact Collision Algorithims: 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.



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 3: 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

The behaviour of each algorithm class is showed in figure 3. As it can be seen, the ECA algorithms solve the collision detection problem in the *narrow stage*, while others (ST, NCIT and GBA) solve the problem in the *broad stage*. In figure 3 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 objects involved on the collision.

5.2 Algorithmic schemas

In this section general algorithmic schemas are presented. These schemas are based on the classification introduced in the last section. First of all three main data structure types are exposed. The first one corresponds to one hipotetical *ECA* algorithm (or narrow–phase–algorithm), while the last ones correspond to the *ST*, *NCIT* or *GBA* algorithm (or broad–phase–algorithm). After four algoritmic schemas that are representative of the main data structures are presented.

```
type
  object = struct
             tighly = { tighly representation of objects: Brep, CSG,... }
             ch = CH { Convex hull of object: optional }
           endstruct
endtype
```

Type for a NARROW–stage algorithm

In the case of *broad–stage* algorithms, two types of data structures have to be distinguished. One concerning to space–partition representation of objects, and one corcerning to object–partition representation of objects.

```
type
  object = tree_coarse { Hierarchical representation with simple shapes for objects }
  tree_coarse = tree of basic_shape { Octree, BSPtree, ... }
endtype
```

Type for a SPACE–partition BROAD–stage algorithm

```
type
  object = tree_coarse { Hierarchical representation with simple shapes for objects }
  tree_coarse = tree of simple
  simple = simple_shape { sphere for Spheretree, bounding box for OBBtree, ... }
endtype
```

Type for an OBJECT–partition BROAD–stage–algorithm

Due to that the narrow stage collision detection algorithms can be seen as a particular case of intersection algorithms that are widely studied and published, we will center our attention on the algorithms that are focused on the *broad–stage* for reject collision or for detect the relevant regions where collision can truly occur. In this case several algorithmic schemas are presented.

- Schema A: use of AABB–trees. For instance: [PML95]

```
type
  object = struct
             aabb = AABB {Axis Aligned Bounding Box }
             brep = tBrep {B-rep representation of object }
             ch = CH {Convex hull of object }
           endstruct
endtype
procedure ACollision(in o1, o2 : object; out col : boolean; out fs : tBrep)
  var
    overlapaabb : boolean
    colch : boolean
  endvar
  overlappingAABB(o1.aabb, o2.aabb, overlapaabb)
  if
    overlapaabb → col := TRUE
                  if
                    minimum_level → collisionCH(obj1.ch, obj2.ch, colch)
                                    if
                                      colch → intersection_area(o1, o2, fs)
                                              selection(fs) {extract the CH faces }
                                    endif
                    no minimum_level → { seek the collision AABB-tree level }
                                       down_level(o1, o11)
                                       down_level(o2, o21)
                                       Acollision(o11, o21, col, fs)
                                       down_level(o1, o12)
                                       down_level(o2, o22)
                                       Acollision(o12, o22, col, fs)
                  endif
    no overlapp → { no collision between o1 and o2 }
                  col := FALSE
  endif
endprocedure
procedure collisionCH(in ch1, ch2 : CH; out col : boolean)
  col := intersection(ch1, ch2)
endprocedure
procedure overlapingAABB(in aabb1, aabb2 : AABB; out b : boolean)
  b := overlapX(project(aabb1, x), project(aabb2, x)) and
       overlapY(project(aabb1, y), project(aabb2, y)) and
       overlapZ(project(aabb1, z), project(aabb2, z))
endprocedure
```

Note: *minimum_level* is a pre–specified level

- Schema B: use of Sphere–trees. For instance: [Hub96]

```
type
  object = tree_sph
  tree_sph = tree of sphere
  sphere = struct
             center = point3D
             radi = real
           endstruct
endtype
procedure Bcollision(in o1, o2 : object; out b : boolean; out e1, e2 : object)
  if
    not minimum_resolution_level →
                                    b := collision_sphere(o1, o2)
                                    if
                                      b → { seek the collision level }
                                          down_level(o1, o11)
                                          down_level(o2, o21)
                                          Bcollision(o11, o21, b, e1, e2)
                                          down_level(o1, o12)
                                          down_level(o2, o22)
                                          Bcollision(o12, o22, b, e1, e2)
                                        not b → { no collision between o1 and o2 }
                                    endif
    minimum_resolution_level → { for save security }
                               b := TRUE
                               e1 := o1
                               e2 := o2
  endif
endprocedure
```

• Schema C: use of OBBtrees. For instance: [GLM96]

```
type
  OBBtree =  tree of bbox {Object Oriented Bounding Box tree}
  bbox = struct
           { rectangular oriented box }
         endstruct
endtype
procedure Ccollision(in o1, o2 : OBBtree; out col : boolean; out r1, r2 : OBBtree)
  var
    colobb : boolean
  endvar
  if
    minimum_resolution_box → col := TRUE
                             r1 := o1
                             r2 := o2
    no minimum_resolution_box →
                                 BOXcollision(o1, o2, colobb)
                                 if
                                   colobb → { search the OBBtree collision level }
                                            Ccollision(child_o11, child_o21, col, r11, r21)
                                            Ccollision(child_o12, child_o22, col, r12, r22)
                                    no colobb → { no collision }
                                                col := FALSE
                                 endif
  endif
endprocedure
procedure BOXcollision(in obb1, obb2 : bbox; out b : boolean)
  project_center_box(obb1, oriented_axis, center1)
  project_center_box(obb2, oriented_axis, center2)
  compute_radi_interval(center1, rd1)
  compute_radi_interval(center2, rd2)
  b := distance(center1, center2)≤sum(rd1, rd2)
endprocedure
```

- Schema D: use of Octrees. For instance: [KTAK94]

```
type
  object = struct
             brep = tBrep
             octree = tree of node
             node = tnode
           endstruct
  tnode = struct
            typ = {Black:B, White:W or Gray:G }
            coor = coordinates{ coordinates of the cube }
          endstruct
  tBrep = {B-rep representation of object }
endtype
procedure Dcollision(in o1, o2 : object; out colNode : boolean; out fs : tBrep)
  var
    fs1, fs2 : tBrep
    i : enter
  endvar
  if
    o1.node = W and o2.node = W → colNode := FALSE
    o1.node = W and o2.node = G → colNode := FALSE
    o1.node = W and o2.node = B → colNode := FALSE
    o1.node = B and o2.node = B →
                                  extraction_faces(o1.node, fs1)
                                  extraction_faces(o2.node, fs2)
                                  polyhedral_inter(o1.brep, o2.brep, fs)
    o1.node = G and o2.node = G →
                                  for i in [1..8] do
                                      Dcollision(child_i(o1.node), child_i(o2.node), colNode, fs)
                                  endfor
    o1.node = B and o2.node = G →
                                  extraction_faces(o1.node, fs1)
                                  for i in [1..8] do
                                      if
                                        child_i(o2.node) = B → polyhedral_inter(fs1, child_i(o2.brep), fs)
                                      endif
                                  endfor
  endif
endprocedure
procedure extraction_faces(in node : tnode; out faces : tBrep)
  substitute the 8 coordinates of the corresponding cube in the equation of the plane of each face
   of the interfering object obtaining the faces that collide
endprocedure
```

- Schema E: use of OBB (first level only): [GASF94]

```
type
  object = struct
             brep = tBrep {B-rep representation of object }
             obb = OBB {Object Oriented Bounding Box }
             vox = voxelset {Approximation of the solid by voxels }
           endstruct
endtype
procedure Ecollision(in obj1, obj2 : object; out fs : tBrep)
  var
    overlapobb : boolean
    colvox : boolean
    listVOX : list of voxels
  endvar
  overlappingOBB(obj1.obb, obj2.obb, overlapobb)
  if
    overlapobb → collisionVOX(obj1.vox, obj2.vox, colvox, list_col)
                 if
                   colvox → { only for voxels in the list_col }
                            compute_intersection_area(o1.vox, o2.vox, fs)
                 endif
  endif
endprocedure
procedure collisionVOX(in v1, v2 : vox; out col : boolean; out voxset : listVOX)
  col := intersection(v1, v2) {Do voxels collide ? }
  if
    col → calculate_list_voxel_candidates(v1, v2, voxset)
  endif
endprocedure
procedure overlapingOBB(in obb1, obb2 : OBB; out b : boolean)
  b := overlapX(project(obb1, x), project(obb2, x)) and
       overlapY(project(obb1, y), project(obb2, y)) and
       overlapZ(project(obb1, z), project(obb2, z))
endprocedure
```

6 Conclusions

We have discussed several data structures and methods to attach the collision detection problem between objects in a closed space. We have centered the document in the *multiple interference detection* reducing the problem of *dynamic* collision to multiple calls to static interference test, thus we have pointed out in the so called *static collision detection.* From that we have presented and discussed data structures and different strategies to attach the problem. As it has been exposed, collision can be treated in a *broad stage* and in a *narrow* stage.

- When a coarse (broad) 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 (narrow) 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 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. It can be also very useful for solving the *ST* and *NCIT* collision detection problems.

Acknowledgments

Thanks are due to the Graphics Section colleges of the LSI department for their helpful comments.

References

[Bar89] D. Baraff. Analytical methods for dynamic simulation of non–penetrating rigid bodies. *ACM Computer Graphics*, 23(3):223–232, July 1989.

[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 $3D$. In *EUROGRAPHICS Conf. Proc.*, volume 15, pages 387–396. Blackwell Publishers, August 1996.

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

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

[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.

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

[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.

[FMP97] L. Floriani, P. Magillo, and E. Puppo. Building and traversing a surface at variable resolution. In *Proceedings of IEEE Visualization*, volume 1, pages 103–110, 1997.

[Fra90] M. Franquesa. 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.

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

[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.

[HBZ90] B.V. Herzen, A.H. Barr, and H.R. Zatz. Geometric collisions for time–dependent parametric surfaces. In *ACM SIGGRAPH Conf. Proc.*, pages 39–48, August 1990.

[HK97] T. He and A. Kaufman. Collision detection for volumetric objects. In *Proceedings of IEEE Visualization*, volume 1, pages 27–35, 1997.

[HKM95] 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.

[Hor84] B. K. P. Horn. Extended gaussian images. In *Proceedings of the IEEE*, volume 72, pages 1671–1686, December 1984.

[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 Computer Graphics*, 15(3):179–210, July 1996.

[JT96] P. Jimenez and C. Torras. Speeding up interference detection between polyhedra. In *IEEE Proceedings Interna. Conf. on Robotics and Automation*, volume 2, pages 1485–1492, April 1996. Minneapolis (MN).

[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.

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

[Kir83] D.G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM J. COMPUT.*, 12(1):28–35, February 1983.

[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.

[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.

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

[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.

[NPT93] I. Navazo, A. Puig, and D. Tost. Research trends in volumetric modeling. Technical report, Software Dept. Universitat Politecnica de Catalunya, 1993. Ref: LSI–93–35–R.

[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.

[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.

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

[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.

[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.

[SWF+93] J.M. Snyder, A.R. Woodbury, K. Fleischer, B. Currin, and A.H. Barr. Interval methods for multi–point collisions between time–dependent curved surfaces. In *ACM SIGGRAPH Conf. Proc.*, pages 321–334, August 1993.

[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.

Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya

Research Reports – 1998

LSI-98-1-R "Optimal Sampling Strategies in Quicksort and Quickselect", Conrado Martinez, Salvador Roura.

LSI-98-2-R "Query, PACS and simple-PAC Learning", J. Castro and D. Guijarro.

LSI-98-3-R "Interval Analysis Applied to Constraint Feasibility in Geometric Constraint Solving", R. Joan-Arinyo and N. Mata.

LSI-98-4-R "BayesProfile: application of Bayesian Networks to website user tracking", Ramón Sangüesa and Ulises Cortés.

LSI-98-5-R "Some reflections on applying Workflow Technology to Software Process", Camilo Ocampo and Pere Botella.

LSI-98-7-R "Trust Values for Agent Selection in Multiagent Systems", Karmelo Urzelai.

LSI-98-8-R "The use of SAREL to control the correspondence between Specification Documents", Núria Castell and Àngels Hernández.

LSI-98-9-R "Intervalizing colored graphs is NP-complete for caterpillars with hair length 2", C. Àlvarez, J. Diaz and M. Serna.

LSI-98-10-R "A unified approach to natural language treatment", Jordi Àlvarez.

LSI-98-11-R "Collision Detection: Models and Algorithms", Marta Franquesa and Pere Brunet.

LSI-98-12-R "Height-relaxed AVL rebalancing: A unified, fine-grained approach to concurrent dictionaries", Luc Bougé, Joaquim Gabarró, Xavier Messeguer and Nicolas Schabanel.

Hardcopies of reports can be ordered from:

Núria Sánchez
Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
Campus Nord, Mòdul C6
Jordi Girona Salgado, 1–3
08034 Barcelona, Spain
secrelsi@lsi.upc.es

See also the Department WWW pages, http://www-lsi.upc.es/

BIBLIOTECA