Real-Time Collision Detection using Spherical Octrees : Virtual Reality Application

Costas TZAFESTAS and Philippe COIFFET
Laboratoire de Robotique de Paris
CNRS-UPMC-UVSQ
10-12 Avenue de l'Europe, 78140 Vélizy, France.
E-mail: tzaf@robot.uvsq.fr

Abstract

A method for detecting potential collisions between three-dimensional moving objects is described in this paper. An object-centered, spherical octree representation is defined and implemented for the localisation of potentially colliding features between polyhedral objects. These features are subsequently tested for intersection in order to calculate precisely the actual collision points. Application of the algorithm for the direct manipulation of objects in a virtual scene is considered, to investigate its real-time behaviour. The performance of the algorithm is found to remain linear with respect to the complexity of the colliding objects.

1 Introduction

The detection of collision between two arbitrarily moving objects in a three-dimensional computer-simulated environment plays an important role in various research fields:

a. In the context of ***dynamic computer animation***, where the sequence of object's position has to be dynamically produced by a physical simulation, as the effect of internal, model-derived forces and torques([14]). The problem is usually solved by considering all possible combinations of points and faces for a pair of objects and extensively checking them for interaction. Basic algorithms of this type have a complexity of $0(n^2m^2)$, for n polyhedra and m vertices. Bounding shape techniques are usually employed to ameliorate the computation times by quickly deciding for the non-intersection of two completely separating objects.

A related issue is contact force computation and response once collisions have been detected. Two classes of methods have been mainly used, namely *penalty methods*, allowing interpenatration of objects and introducing virtual springs and elastical deformation models [15], and *analytical methods* treating the case of non-penetrating rigid bodies, and making use of numerical optimisation techniques [2].

b. In the context of ***robot path planning***, where an automatic generation of collision-free robot trajectories has to be based on algorithms performing distance calculation between a moving robot and the obstacles present in its workspace, and determining potential collisions between them. Growth techniques and approximating shapes representation have often been used to assure collision detection and avoidance [13].

c. In the context of ***Virtual Reality Applications*** [5], where the developpement of interactive, realistic, computer-simulated environments is based on algorithms performing fast collision detection between multiple, moving virtual objects. Calculation of virtual contact forces permits the implementation of *force display* techniques using specially designed or general-purpose haptic interfaces [3],[10]. The main difficulty of such applications is the real-time constraints that are imposed. The human operator must interact with a virtual scene in a natural way, moving objects and generating collisions between them. These collisions must be detected and interaction forces computed fast enough (response time less than 100ms), otherwise a sense of incompatibility will be created and the environment will seem unrealistic.

All the collision detection methods can be classified in accordance with the models used to represent the shape of the objects surface. Three major classes of object representations exist:

1. Boundary representations. The object is modelled as a general (convex or concave) polyhedron consisting of a set of vertices, edges and faces. The major drawback of the polyhadral representation is that the resulting precision highly depends on the number of faces used to approximate the object. To determine whether a collision has occured, computation of distance between two convex polytopes can be used. This is usually formulated as a constrained minimisation problem, and numerical optimisation techniques are used to provide measures of the minimum distance. Bobrow [4] used a direct approach which generates a sequence of search directions along the surfaces of the objects in order to obtain the global minimum. The Kuhn-Tucker conditions were used to ascertain whether the global minimum has been reached or not. Gilbert et al. in [6] presented an iterative, descent procedure for computing the distance and finding the nearest points between a pair of convex polytopes. Lin and Canny [11], finally, developped a fast, incremen-

tal, distance calculation algorithm which works, almost always, in constant time. As two objects move in space, this algorithm keeps track of the closest pair of features by performing simple tests, and checking locally the coboundary (neighborhood) of the closest features provided by the previous step.

All these methods need, however, extra computation time to precisely calculate the collision points, when intersection between two objects exists.

2. Analytical surface representation, such as superellipsoids [1]. These modelling techniques provide an analytic inside/outside function that explicitly determines whether a point in space is inside or outside the closed-form surface. Collision detection between two objects has to proceed by testing a large number of representative points for each object. Therefore, a tradeoff between precision and computational efficiency has to be achieved.

3. Constructive solid geometry representations, using volumetric primitives such as cylinders [16] or spheres [17]. In the first case line equations are used giving a very rough approximation of the robot solid geometric model. In the second one, a large number of spheres (over 200) has to be used to achieve a reasonable precision of the representation. Furthermore, collision detection still needs testing all the possible combinations of intersecting spheres which may not always be efficient if we want to increase the precision of contact points computation.

The volumetric primitives (spheres, cubes etc.), used to represent a solid object, can also be organised in *hierarchical tree-structures*. The hierarchy in such a model implies that only intersection testing between nodes that are close to each other is performed, which may accelerate significantly the detection of a potential collision. Liu [12] presents a solid model called hierarchical sphere model which represents a three-dimensional space including an object by a tree structure of spherical cells, using a decomposition of each spherical region into 13 subspheres. Hubbard [8] also presented a form of approximate geometric modelling called 'sphere-trees'. Methods for automatically building these structures were also investigated. The accuracy of collision detection using this approach depends on the number of levels of the tree, which may influence the computation time strongly.

Another hierarchical, tree-structure model that is often used to represent three-dimensional, solid objects is the *octree* [18]. Octrees represent the space occupied by an object using a cubic decomposition of the universe space. The space is recursively partitionned into octants until each octant is completely inside or outside an object, or until the limit of the resolution is reached. The volumetric primitives used in this case are cubes of fixed location and orientation in space. Octree models have already been used for collision detection in a robot programming [7] or virtual workspace [9] systems. In both cases, a special algorithm is needed to update the octree, for arbitrary translation and rotation of the represented objects, which requires modelling the intersection of arbitrary oriented cubes ([18]). This does not seem to be a trivial task, and approximating methods need to be used to achieve the desired computational efficiency.

In this paper we propose a *spherical, object-centered octree* decomposition for the representation of three dimensional, moving objects. This representation has the following characteristics:

a. Each of the oct-nodes corresponds to a spherical region, surrounding a part of the object's surface. Intersection detection between two spheres is trivial, as only positions of their centers and corresponding radii need to be known.

b. The tree structure is fixed with respect to the object's reference frame and moves along with it. This facilitates considerably the computation load needed to represent moving (translating and rotating) objects. The octree can be constructed off-line and easily updated using only simple coordinate transformations.

This spherical octree representation is used to develop a real-time collision detection algorithm. Interfering oct-nodes determine features, of the objects' polyhedral model, that are likely to enter in collision. These features are subsequently tested for intersection to find out which of them are actually colliding and calculate with precision the contact points as well as the corresponding interaction forces. A virtual scene consisting of various objects has been constructed to test the efficiency of the algorithm in real-time applications. Experimental results are presented and analysed in section 4, and concluding remarks given in section 5.

2 General Description of the Method

2.1 Object-Centered, Spherical Octree Representation

Consider an object in 3-dimensional space. Its polyhedral representation consists of a grid of $n \times m$ points in space and is primarily defined by a matrix of vertices. The octree representation of such an object is generally obtained using a cubic, recursive decomposition of the whole space. In the universe-centered definition the volumetric primitives (cubes) of the tree structure are of fixed sizes, locations and orientations. In an object centered representation, on the contrary, the placement of the primitives is determined with respect to the object reference frame. A 'minimal' cube C_0 surrounding the object is first defined and is recursively decomposed into 8 subcubes C_i. Each node T_i of the octree structure corresponds to a cubic region C_i. The whole structure can be created off-line and is considered to move along with the object.

In this paper we propose a spherical, object-centered octree representation, where each node T_i of the tree structure corresponds to a spherical region S_i surrounding part of the object's surface. This is done by inscribing each cube C_i of the original object-centered octree into a sphere S_i. In this way, each node T and its corresponding sphere S can be recursively subdivided into subspheres S_i, the union of which completely surrounds the surface of the object, with an increasing precision as the level of the octree becomes greater. This procedure for a two-dimensional

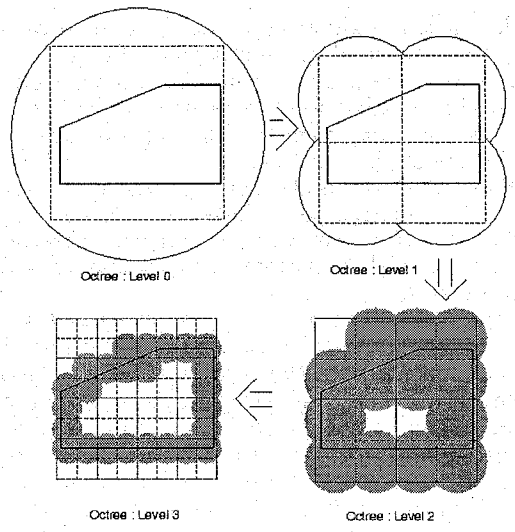

Figure 1: *Spherical, Object-Centered Octree Decomposition*

object is schematically represented in figure 1, until a level 3 for the octree representation is reached. At this point we must note that, as we will see in section 3.1, only nodes intersecting the surface of the object need to be taken into account during the collision detection algorithm, thus reducing the computation load of performing redundant (not necessary) tests.

2.2 Localisation of Colliding Features

Interference between two objects is detected by traversing the octrees surrounding their surfaces. The father-node of the first tree is tested for intersection with the nodes of the second one, starting from the father-node and descending to the children only when interaction is found. Then, in a similar way, each of the second object's leaf nodes that are found to intersect with the father node of the first one, is tested for intersection with all the nodes of the first tree. Each time, the algorithm descends to the children of a node only if interaction of their father is found.

Intersection detection between two nodes is trivial since each node of the octree corresponds to a spherical region. The sphere corresponding to each oct-node surrounds a region of the object's surface, possibly containing or intersecting some of the object's features such as vertices, edges or faces. Two interfering nodes can therefore determine pairs of features that may possibly be in contact. Only potential contact formations have to be tested afterwards, thus eliminating many possible combinations of contacting features.

The pairs of nodes that are found to intersect form the *list-of-intersecting-oct-nodes.* This list is then processed to identify the features of the objects that are actually colliding, the elemental contacts they form as well as their mutual interpenetration. Using this data we can compute the actual force applied between two objects in each one of their contact points.

The method on its whole can be therefore seen as consisting of two stages:

• 1st stage : Detection of potentially colliding features between two objects from the interfering nodes of their octree representations.

• 2nd stage : Identification of the actual contact formations and computation of the total force and moment applied on the centroid of each object.

More details on the algorithm and its implementation are given in the following section.

3 The Algorithm and its Implementation

3.1 Polyhedral and Spherical Octree Representation : Global Data Structures

Every object in the VE is geometrically modelled as a convex polyhedron (or a union of convex polyhedra). Its state in 3D space is described by a vector containing the position of the centroid $P_O^{(w)}$ in world coordinates, a 3x3 rotation matrix R_O^W designating its orientation in space, as well as vectors for its linear and angular velocity v_O, ω_O .

Each object in the VE contains also the following data structures which define its geometrical form:

- a list of vertices (*v-list*).
  Each vertex is characterised by its position (x,y,z) coordinates relative to the centroid, expressed in the objects local reference frame, and its coboundary, that is a list of edges intersecting on the vertex.

- a list of edges (*e-list*).
  Each edge is described by the two vertices *head* and *tail* and also by the two faces *Left-Face* and *Right-Face.* The intersection of those faces is the edge itself.

- a list of faces (*f-list*).
  Each face is parameterised by the position of a point on this face (called center of the face) relative to the object's centroid and expressed in the object's local reference frame R_O, and by its outward normal, also expressed in R_O. It also includes a list of vectors defining the coboundary of the face, that is, vectors originating from the center of the face and being perpendicular to its bounding edges.

All this data is expressed in the object's local reference frame and is therefore computed off-line during the construction of the object's geometrical (polyhedral) model. In our system, this is automatically performed by a procedure which takes as input a nxm matrix of the vertices coordinates and constructs all the necessary data structures (*v-list*, *e-list* and *f-list*) for the correct execution of the collision detection algorithm. The world coordinates of all the necessary vectors are computed on-line using simple coordinate transformations.

We will now describe the data structures defining the object-centered, spherical octree representation of a polyhedral object. This representation consists of a hierarchical structure of nodes. The data structure of each node T contains:

• a field for the relative position (p_x, p_y, p_z) of the corresponding sphere S with respect to the object's centroid P_O and expressed in the object's local reference frame R_O

• a field for its radius r

• an integer number representing the level of the node in the tree structure, which must never overpass the maximum depth of the octree

• a set of pointers to its children oct-nodes $child_i$.

Each node also contains a field *type* which takes as value one of the following possible attributes:

• VERTEX, if the sphere S is found to contain at least one of the object's vertices (from the object's *v-list*).

• EDGE, if the above condition does not hold <u>and</u> the node's corresponding sphere S intersects one of the objects's edges (from *e-list*).

• FACE, if the above does not hold <u>and</u> S intersects one of the objects's faces (from *f-list*).

• NONE, if none of the above conditions hold, which means that the node's corresponding sphere S does not intersect the object's surface. Therefore, this node will not be furthemore decomposed and will not be visited by the collision detection procedure.

The oct-node data mentionned above is computed off-line, once and for all, during the construction phase of each object's spherical octree representation. In our system it is automatically generated by a procedure called *create-oct()* which makes use of the object's polyhedral representation (list of vertices etc. generated beforehand) and gives at its output a pointer to the father-node of the object's octree structure.

Depending on the *type* of the node a new vertex, edge or face list is created belonging to the node's structure itself and containing pointers to the object's features intersected by the corresponding sphere. This means that the data structure of each node also contains its own *v-list*, *e-list* and *f-list*, thus defining a link from the spherical octree surface representation to the polyhedral one. For instance, if the *type* of a node has the atribute EDGE as its value, then the node's *e-list* will contain pointers to all edges intersected by its corresponding sphere. We must, however, remark that only the features lists of the octree leaf-nodes will be used for actual intersection detection between polyhedral objects.

For the description of all the possible ways by which two polyhedral objects can enter in contact, we have chosen to use the two basic contact types presented in [10] (ie: Vertex-Face (VF) and Edge-Edge (EE) contact types). Every contact that may occur between two polyhedral objects can be described by a set of such basic, elemental contacts.

For contact force computation we use a model of stiffness K_c, damping B_c and friction (viscosity coefficient γ_c) associated with every elemental contact point. Interpenetration Δp and relative velocity Δv of the two objects on their contact point are used to compute the force mutually applied on this point:

$$\vec{f_c} = K_c \cdot \vec{\Delta p} - B_c \cdot \Delta\vec{v}_n - \gamma_c \cdot \Delta\vec{v}_t \qquad (1)$$

where

$$\Delta\vec{v}_n = (\vec{\Delta v} \cdot \vec{normal}) \cdot \vec{normal}$$

and

$$\Delta\vec{v}_t = \vec{\Delta v} - \Delta\vec{v}_n$$

For each object the contact forces $\vec{f_{ci}}$ from the existing n_c elemental contact points can be summed up to provide the total force and moment applied on the centroid of the object.

3.2 1st stage : Finding intersecting oct-nodes for the localisation of potentially colliding features

The first stage of the collision detection algorithm consists of the following phases:

Phase 1-1: Detection of interfering oct-nodes by traversing the octrees of the objects (procedure *interoct()*) and creation of the List-of-Intersecting-Oct-nodes (called *Oct-Contacts*). It is based on the distance computation between the centers of the corresponding spheres and, in case intersection is found, recursive calls of the *interoct()* procedure, till reaching the leaf nodes.

To avoid overcharging this stage of the algorithm we have also considered *non-homogeneous octrees*, which means that only some oct-nodes in one level are further decomposed depending on whether we need or not more precision in a particular region of an object, for the localisation of intersecting features.

Phase 1-2: Processing of the list *Oct-Contacts* and identification of potentially colliding primitive features. Creation of the List-of-potential-elemental-contacts (called *Checks*) containing two types of basic features intersection, VF or EE, as described in the previous section.

The list *Oct-Contacts* is traversed one or more times and, depending on the *type* of the nodes intersection, one or more potential, basic contacts are added into *Checks*. In this phase we make use of the lists *v-list*, *e-list* or *f-list* of each oct-node, that are created beforehand (off-line) by the procedure *create-oct()*, as discussed in section 3.1. These lists create a link between the octree and the polyhedral representation of each object and are the ones that determine which pairs of primitive features will be added into *Checks*, and tested for intersection in the following, 2nd stage of the algorithm.

3.3 2nd stage : Determining actually intersecting features

In this stage, the list *Checks*, previously created by phase 1-2 of the algorithm, is traversed twice to determine the pairs of features of the polyhedral objects that are actually colliding.

We start by testing for possible VF contacts (procedure *check-vfs()*), among those included in *Checks*.

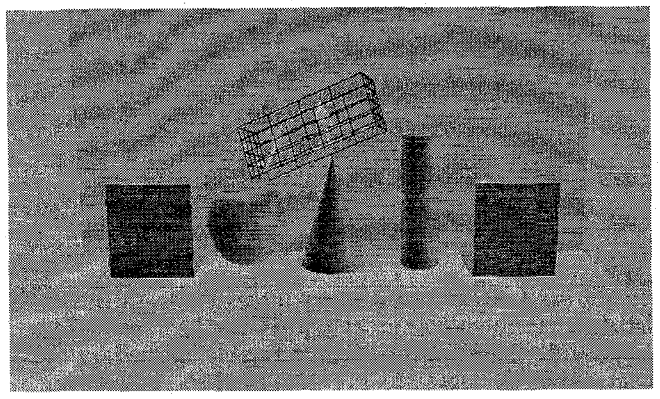

Figure 2: *Snapshot of the Virtual Scene.*

Proposition 1 *Each vertex of a polyhedral object can be considered, at each step, to be in contact with no more than one face for every other* **convex** *polyhedral object. This proposition cannot hold for non-convex polyhedra, which can however be modelled as unions of convex ones.*

To make use of this proposition, the 'history' of previously detected contact configurations has to be maintained at every computation step, and stored in a list called *List-of-Contacts*. At each step, before performing the detection of new actually intersecting features (procedure *detect-contact()*), we must verify the validity of the previously detected basic contacts (contained in the *List-of-Contacts*). This is performed by a procedure we call *verif-contacts()*. In a similar way a procedure called *check-ees()* detects for new currently intersecting pairs of edges (EE basic contacts) from all the possible combinations contained in *Checks* list.

4 Numerical Experiments

To investigate the properties and demonstrate the efficiency of the developped algorithm we have constructed a virtual scene consisting of several three-dimensional objects. One of these objects, called 'FOLLOWER', is manipulated directly by the human operator with the use of a 3D magnetic sensor (Polhemus Isotrack^{TM}), capturing in real-time information about the human hand movements in space. The 'FOLLOWER' is moved arround entering in collision with various objects present in the scene. These collisions have to be detected in real-time in order to respect the motion constraints that are imposed due to the presence of other objects in the virtual scene.

Fig.2 presents an example of a virtual scene constructed to test, in real-time, the efficiency of the algorithm. The small arrows represent contact constraints (unitary forces along normal directions) imposed on the moving object (wireframe polyhedron) due to the presence of collisions with existing obstacles.

The objects used in the experiments with their polyhedral as well as their spherical, octree representations are shown in fig.3.

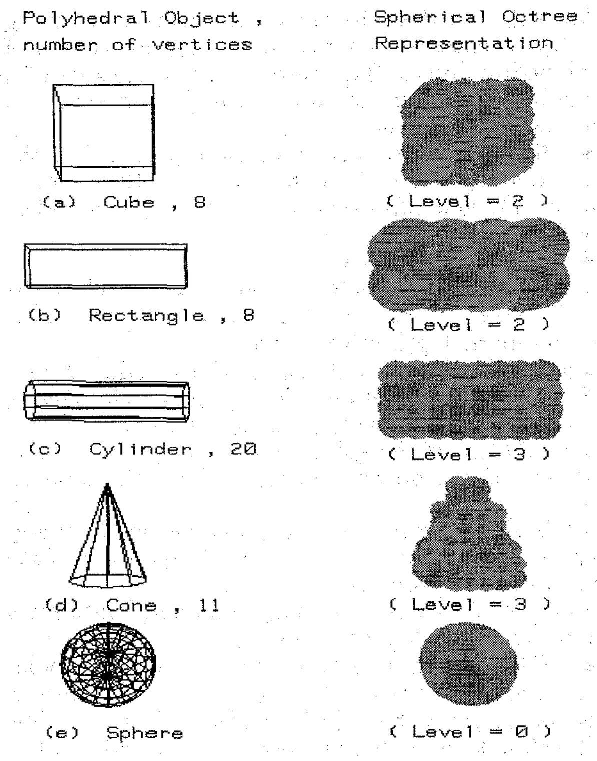

Figure 3: *Polyhedral objects used in the experiments with their spherical octree representations*

A large set of numerical experiments have been performed with these objects entering in various types of collision. Two parameters influencing the efficiency of the algorithm have been mainly studied. The first one is the number of levels in the octrees and the influence its increase may have on the total mean computation time of the algorithm. The second one concerns the complexity of the scene. We investigate the variations of the total execution time for our collision detection and contact force computation algorithm versus the total number of vertices present in the virtual scene. For a virtual scene cosisting of 6 polyhedral objects, as shown in fig.2 we obtain computation times of the order of 20 msecs, which is acceptable for real-time applications.

The algorithm has been implemented in C language and runs on a HP715, 50MHz workstation, equipped with math.coprocessor and graphics accelerator. HP Starbase graphical library provides the necessary procedures for color, 3D image rendering.

4.1 The number of levels in the octrees and its effect on the execution time of the algorithm

The experimental results, obtained for various combinations of colliding objects and different numbers of levels in their octrees, are briefly summarised in table 1.

Objects	Total number of vertices	Average CPU time (in msecs)				
		L=0	L=1	L=2	L=3	L=4
(b)-(e)	-	0.8	0.5	0.8	1.6	9.1
(d)-(e)	-	1.1	0.9	1.2	2.1	9.9
(b)-(a)	16	48	11.1	6.7	7.8	36
(b)-(d)	19	198	62.8	7.1	8.9	27.5
(b)-(c)	28	250	135	59	17.6	60
(d)-(c)	31	700	480	150	34	75
(c)-(c)	40	1720	800	170	41	105

Table 1 : Experimental Results. Mean computation time (in msecs) of the algorithm for various types of colliding objects (refer to fig.3) and increasing number of levels in the octrees.

First of all, we can observe that for each object there is an optimal number of levels for its octree representation which depends on the complexity of its polyhedral representation. For instance, for a rectangular object of type (b) (see fig.3) a number of 2 levels in the octree seems to be sufficient for detecting colliding features. An increase in the number of levels is not counter-balanced by a sufficiently large decrease in the second stage of the algorithm (i.e. testing for intersection all the potentially colliding features supplied by the first stage) due to the small number of vertices present in the object. A number of 3 levels is however needed to sufficiently represent more complex polyhedral objects such as cylinders (object (c)). The mean computation time for all types of colliding objects, versus the number of levels in the octrees is graphicaly represented in fig.4. The performance of our collision detection algorithm for each object will have a form similar to fig.4, presenting an optimality for a particular number of levels in the octree representation. This optimal number of levels increases as the polyhedral model of the objects becomes more complex.

We must also note that intersections between polyhedral and spherical objects are treated separately by a special intersection-testing procedure (*inter-ps()*). A sphere is modelled by an octree of level 0, and its polyhedron (graphical model) is not used for interference detection. Intersection between a vertex, an edge or a face and a sphere is easily detected by simple, specially developped procedures (*check-vs()*, *check-es()*, *check-fs()*).

4.2 Mean Computation Time versus Complexity of the Scene

A classical measure of the efficiency for a collision detection algorithm is to monitor its performance (mean computation time) with respect to the complexity (e.g. total number of vertices) of the colliding objects. Fig.5 shows the evolution of the algorithm's computation time versus the complexity of the objects polyhedral model, for different numbers of levels in the otree representations.

We observe that, as the number of levels in the octree increases, the algorithm presents a behaviour which is *'more linear'* with respect to the total number of vertices of the colliding objects. We therefore

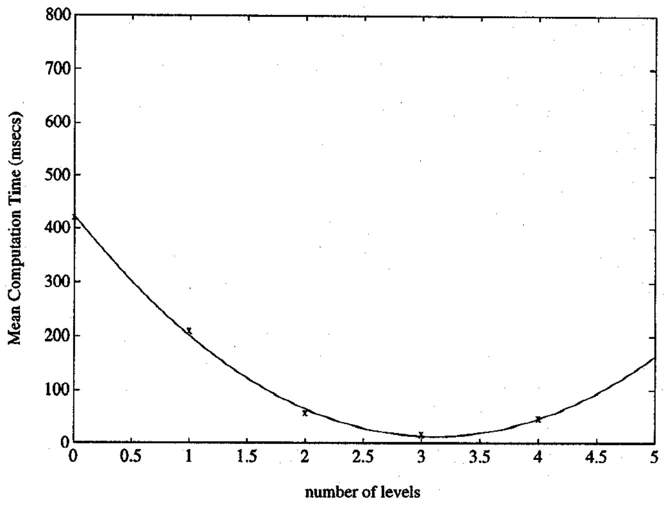

Figure 4: *Mean computation time of the algorithm (in msecs) versus the number of levels in the octrees*

verify our expectation that using the optimal number of levels in the octree representation of each polyhedral object, the performance of the algorithm, for increasing complexity of the virtual scene, remains ***linear***. Moreover, as we can see, the coefficient of linear growth is quite small.

5 Conclusion

An algorithm for the collision detection between two three-dimensional, moving objects has been presented. The algorithm makes use of a spherical, object-centered octree representation to perform localisation of potentially colliding features. This octree decomposition can be easily updated, facilitating the representation of arbitrarily moving objects. The algorithm, in a first stage, performs localisation of pairs of features that are likely to collide. These features are subsequently checked to determine which of them are actually intersecting, and contact points are calculated with precision.

A virtual scene has been constructed to test the efficiency of the algorithm in a real-time application. The human operator directly manipulates a virtual object generating collisions with existing obstacles. The system must detect the collisions, compute and display the interaction forces fast enough to maintain the realistic impression given by the environment. These forces can be used to perform real-time, dynamic animation of virtual objects. Mean computation time has been found to be around 20ms for a scene consisting of several, simple polyhedral objects.

A large set of numerical experiments has been performed with these objects entering in various types of collisions. We have shown that using the optimal number of levels for the octree representation of each object, the performance of the algorithm, with respect

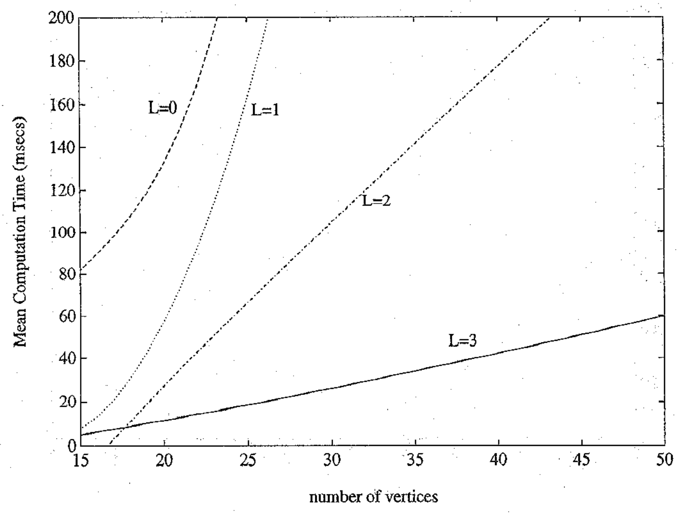

Figure 5: *Mean computation time of the algorithm (in msecs) versus the complexity of the colliding objects (total number of vertices), for different numbers of octree levels*

to the complexity of the colliding objects, remains linear.

References

[1] E.I.Agba, "Objects Interactions Using Superquadratics for Telemanipulation System Simulation", *The International Journal of Robotics Research*, Vol.8, No.3, June 1989.

[2] D.Baraff, "Analytical Methods for Dynamic Simulation of Non-penetrating Rigid Bodies", *SIGGRAPH'89, Computer Graphics*, Vol.23, Number 3, July 1989.

[3] P.J.Berkelman, R.L.Hollis and S.E.Salcudean, "Interacting with virtual Environments using a Magnetic Levitation Haptic Interface", *Proceedings of the 1995 IEEE International Conference on Intelligent Robots and Systems (IROS'95)*, Vol.1, 117-122, Pittsburgh, PA, August 1995.

[4] J.E.Bobrow, "A Direct Minimization Approach for Obtaining the Distance between Convex Polyhedra", *The International Journal of Robotics Research*, Vol.8, No.3, June 1989.

[5] G.Burdea and Ph.Coiffet, "Virtual Reality Technology", John Wiley & Sons, 1994.

[6] 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*, Vol.4, No.2, April 1988.

[7] V.Hayward, "Fast Collision Detection Scheme by Recursive Decomposition of a Manipulator Workspace", *Proceedings of the 1986 IEEE International Conference on Robotics and Automation*, 1044-1049.

[8] P.M.Hubbard, "Collision Detection for Interactive Graphics Applications", Ph.D. Brown University, CS-95-08, April 1995.

[9] Y.Kitamura, H.Takemura and F.Kishino, "Coarse-to-Fine Collision Detection for Real-Time Applications in Virtual Workspace", *Proceedings of ICAT'94: International Conference on Artificial Reality and Tele-existence*, July 14-15, 1994 Tokyo.

[10] T.Kotoku, K.Tanie and A.Fujikawa, "Environment Modelling for the Interactive Display (EMID) Used in Telerobotic Systems", *Proceedings of the 1991 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'91)*, Osaka, Japan, November 3-5, 1991.

[11] M.C.Lin and J.F.Canny, "A Fast Algorithm for Incremental Distance Calculation", *Proceedings of the 1991 IEEE International Conference on Robotics and Automation*, Sacramento, California April 1991.

[12] Y.-H.Liu, S.Arimoto and H.Noborio, "A New Solid Model HSM and Its Application to Interference Detection between Moving Objects", *Journal of Robotic Systems*, 8(1), 39-54 (1991).

[13] T.Lozano-Perez and M.Wesley, "An algorithm for planning collision-free paths among polyhedral obstacles", *Communications of ACM*, vol.22, no.10, 560-570, October 1979.

[14] M.Moore and J.Wilhelms, "Collision Detection and Response for Computer Animation", *SIGGRAPH'88, ACM Computer Graphics*, Volume 22, Number 4, 289-298, August 1988.

[15] D.Terzopoulos, "Elastically Deformable Models", SIGGRAPH'87, *Computer Graphics*, vol.21, Number 4, July 1987.

[16] S.L.Wang, "Collision Detection of Multi-Robots Using Line Geometry", Mechanical Engineering Department, North Carolina AT&T State University.

[17] C.Wang, D.J.Cannon and H.Ma, "A Human-Machine System Integrating Virtual Tools with a Robotic Collision Avoidance Concept using Conglomerates of Spheres", accepted for publication in the *Journal of Intelligent and Robotics Systems, (JINT 1353)*.

[18] J.Weng and N.Ahuja, "Octrees of Objects in Arbitrary Motion: Representation and Efficiency", *Computer Vision, Graphics and Image Processing*, vol.39, 167-185 (1987).