JSc-EUSL(2003) **Vol.3 No.1**, p 55-84

ISSN 1391-586X; ©2003 Published by Eastern University, Sri Lanka

COLLISION HANDLING OF POLYHEDRAL OBJECTS - *A SURVEY*

S.Kanaganathan

Department of Mathematics, Eastern University, Chenkalady, Sri Lanka

(Received December 2001: Accepted September 2003)

Abstract

A survey of collision detection and collision response algorithms in Virtual Reality environment is given. Data structure to represent objects in Virtual Reality environment are also discussed. A parallel implementation of collision handling algorithm is given and an application of it to pulping industry is described.

keywords: Collision, Virtual Reality, Polyhedral, Parallel implimentation, MPI

1 Introduction

Collision handling involves two phases:

- Collision Detection.
- Collision Response.

The detection algorithm identifies whether objects collide. The response algorithm determines appropriate action based on the contact points and impact dynamics when there is a collision.
Collision handling has may applications:
Physically based simulation, where moving objects are simulated. In order to determine their behavior over time, the most basic information needed is the time and position of collision together with the exact point of collision. Only if this information is known exactly, the collision response can determine how object will react, according to their mass, mass distribution, velocity etc.

Animation is one of many applications of physically based simulation. An animation system does not really need absolutely correct physical simulation; usually it suffices that the behaviour and paths of objects look realistic. An animator specifies weight, initial speed etc, and has the simulation module compute the paths of the objects. Then he compares those with the path he had in mind and verifies that they look realistic. If so, he's done; otherwise he change some parameters of the objects until the output matches with his intention.
Another area is *robotics.* A common application is *path planning.* Given the geometry of an object and the start and end point of the motion of it, the task is to find a path among several obstacles so that the object will never hit any obstacle.
In *Virtual Reality* collision detection can be used to facilitate intuitive interaction, natural manipulation of the environment, any kind of physically based simulation, and modeling. In general, collision detection with appropriate collision response can make a virtual reality application look more believable.[15].

1.1 Object Representation

The internal representation of graphical objects has great impact on the choice of algorithms. Several different approaches to object representation have emerged over the past decade. We can classify them as *boundary based vs volume based.*
Boundary based object representations are the classical B- reps, free from surfaces "argumented" octrees[34, 35, 36] and hierarchical B-reps.
Volume based object representations are the well known octree [20, 21], BSP [22] and CSG.
Octrees and BSP trees are well suited for intersection computation of polyhedral objects.
BSPs represent an object exactly by intersections of half spaces. Octrees approximates an object by a hierarchical decomposition into cubes. Thus, octrees cannot provide for exact collision detection algorithms.
Neither BSPs nor octrees seem to be suitable for objects which change their geometry, because in this case their BSP-or octree- representation would have to be re-computed. Also, octrees have to be recomputed when the object moves, which is particularly expensive, because the motion includes rotation (octrees are based on axis aligned cubes). In any case, a B-rep representation seems to be needed in addition to a BSP or octree representation. Thus, in an integrated system, two representation would have to be maintained, which is always prone to inconsistencies, increases memory requirements and might destroy any efficiency gain.
The advantage of B-reps is that they can easily hold topological information about the geometry, like adjacency and incidence of features(vertices, edges, polygons). Octrees are better suited for computing mechanical properties like mass, volume, inertial tensors etc.

The considerations above favour the use of B-reps solely for polyhedra and build any additional data structures, which might be needed for speed up, on top of these B-reps.
For other representations -refer, [28] for sphere trees, [24] for boxtrees , [14] for OBB-Tree (Oriented Bounding Boxtree)

1.2 Types of Collision Detection

Collision detection algorithms can be classified by several criteria:

(i) *Approximate vs exact:* Making some sort of geometry simplification or using probabilistic algorithm.

(ii) *Time as a fourth dimension vs purely three-dimensional timeless geometries:* Approaches which take time into account are Hubbard [28,15], Canny [37]. The time dimension can be used to compute the exact time of a collision, or it can be used to exploit time coherancy in order to speed up the collision detection procedure.
Timeless approaches consider all objects only at a certain time, but they do keep in mind that objects probably move-unlike approaches in computational geometry. If timeless approaches have to provide the time of collision more accurately they will use some kind of back-tracking method.

(iii) Speed up by *object hierarchies*[38] (above object level or on intra object level), by space subdivision, or by plane sweep[39].

(iv) *Restriction of domain;* mostly, the class of polyhedra is restricted to convex ones. Other possible restrictions could be closed objects or polyhedra consisting of convex polygons.

(v) *Flexible vs rigid objects;* the issue of collision detection between flexible (also called "soft") objects [41] which change their geometry with time, complicates the problem significantly, because either no pre-computation can be used at all or the pre-computation has to be updated with every change of geometry. Self intersections might have to be checked for, too[16]. Some algorithms are based on parametric object representations.

(vi) *On-line vs off-line;* many application can do without on-line collision detection, because the applications is not driven by real-time input like in VR environments, for example, path planning in robotics or physically based simulation for animation.

(vii) *Incremental vs "from - scratch";* incremental methods try to exploit results of an earlier collision query. This is a form of exploiting time coherency.

With robotics, collision detection usually does not have to be real-time or exact; usually path planning can be done off-line, and it suffices if the path makes sure that there won't be any collision. However, if there are moving obstacles whose motions are not known in advance, path planning will become more like on-line collision avoidance in a not fully predetermined environment. In virtual reality, the requirements are most severe. Under all circumstances, the collision detection must be real-time in order to attain the effect of immersion. For physically based simulation within VR environment, it is also highly desirable to have exact and accurate collision detection, because there won't be a second chance to tweak if the output of the simulation module is not satisfactory. Although intersection in virtual environments might not really need the exact point of collision, a detection too inaccurate (e.g only bounding box tests) disturbe the impression of realism.

1.3 Common Difficulties with collision detection

There are several difficulties that commonly arise when the ultimate goal is real-time exact collision detection:

- *pairwise tests;* on the object level as well as on the edge/face level; a naive algorithm has to test all possible pairs of edges and faces, and also all possible pairs of objects.

- *Discrete time;* this makes it hard to compute the exact time of collision.

Dynamic graphical systems display all objects at certain time interval, usually as soon as the application is done with all the computation, gathering input data, simulating the environment, moving objects etc. If the collision detection module "sees" the environment only at these time steps without any further information about the future, then it can only check whether there is a collision or not.
If speeds (translational and rotational) and may be acceleration are provided, too, then the module can also compute the exact time of collision(or an approximation) either by computing the next collision in the future or back-tracking and recursive interval bisection (of the time interval).

- *concave* polyhedra or even worse, polyhedra which do not consist of convex polygons and which are not closed.

There are many possible ways to tackle the collision detection problem with convex polyhedra; for non-convex polyhedra, very few algorithm seems to be known. For closed polyhedra, we can still resort to algorithms for convex polyhedra only, by partioning them into convex pieces; if they are not closed there is little we can do.

1.4 Class of polyhedra

Here we only consider the class of polyhedra which can be represented by B-reps, ie we do not consider curved surfaces.

- A *collection of polygons;* in this class, we do not require any thing but plane polygons.
- *Closed polyhedra;* in this case, an "inside" and "outside" can be defined and exploited.
- Polyhedra consisting of only *convex polygons;* these polyhedra may still be non-convex.
- *Convex polyhedra;* this class is probably the smallest reasonable one. It seems to provide the greatest advantage for incremental algorithms[42,11]

1.5 How to speed up collision detection

There are several features that can be exploited to speed up collision detection, some of them at a pairwise collision detection level, some of them at the global level, which is concerned with multiple moving objects:

- All kinds of *bounding volumes* for objects as well as on the polygonal level. The most common bounding volumes are axis aligned boxes, others are Bounding spheres and non-axis aligned bounding boxes.
- *Space coherency* use the fact that usually large regions of the space are occupied by only one object or non at all. This is the basis for speed-up on the global, multiple-object level by space indexing method.
- *Time coherency* exploits the fact that moving objects usually move on a continuous path. Also, every interactive system will try to maximize the frame-rate, so that objects move rather slowly compared to the frame rate. If the path is even C^2 continuous and the acceleration can be estimated in advance, this can be exploited by the use of space time bounding of volumes[15].
- Restrictions of the input class of polyhedra.
- *Pre-processing:* by augmenting geometry data with additional data structures, faster algorithms are made possible.
- *Approximate representations* of the object geometry. Thus, collision detection algorithms have to deal only with very special "geometries" which make them very fast. e.g. Approximating the objects by spheres. see Hubbard[27,28].

- *Parallization;* there are several possibilities: parallelization on the pairwise edge-face interaction level(fine-gain); parallelization on the global, multiple collision level(coarse-grain) and the concurrent run of the application and the collision detection module.

1.6 Requirements of collision detection

In general, it is highly desirable that collision detection have the following qualities:

- Fast(if possible, real time).
- Suitable for a class of polyhedra as large as possible.
- Exact(ie the module reports a collision if and only if there is an intersection of surfaces)
- Able to report a witness ie an edge and/or polygon, when the two objects collide(if possible, report all collision points.)
- Able to handle many moving objects.

2 Collision Detection Algorithms

2.1 Space Partitioning Methods.

Highly dynamic environments (eg virtual environments), where many objects should be checked against each other for collision, pose the *all-pairs problem* on the global level-just like the all pair problem on the edge-face level. Even if only a few objects are to be checked against many other statics ones, we don't want to check all possible pairs. Before doing any collision check between two objects, we first check their bounding volumes for intersection. Still the complexity is quadratic ie with **N** moving objects we have to do $\sim \mathbf{N^2/2}$ bounding volume check.

The basic idea with all space partitioning methods is to exploit *space coherence*: most region of the "universe" are occupied by only one object or empty. Consequently, each object has a very small number of "neighbours"; only these neighbours have to be tested for collision with the object itself. The difference among approaches is the data structure, which is maintained during run-time.

The serious constraint is the dynamic nature of the environment. While the only criterion with static environments is *fast retrievability or fast neighbour finding*, the criterion with dynamic environment is, in addition, *fast updating*, for moved objects.

2.2 Bounding Volumes

Even without any space partitioning method, one does want to do a bounding volume pre-check before doing any further collision detection. In the context of space partitioning, we want to deal only with bounding volumes, too, because dealing with the object themselves is usually too expensive.
In order to gain any speed, the bounding volume must be much simpler an object than the object it bounds. At the same time, this is the general disadvantage of bounding volumes: depending on the geometry of the object "inside", they could contain very much "empty" space.
There are a few very simple, commonly used bounding volumes:

- Simplest of all is the *axis-aligned bounding box.* Its faces are always parallel / perpendicular to the world coordinate system. Many pre-checks using axis-aligned bounding boxes are just comparisions of coordinates, which make them very fast.
  They have some disadvantages. The first one depends on the method how they are computed: either they are computed from scrach every time the object has rotated (ie do a whole pass over all vertices in world coordinates), or we transform all eight vertices of the box, and then put an axis -aligned box around those. The problem with the first approach is that it is very expensive; the second one is fast, but the boxes generated by it are up to 2 times as large in volume as the original bounding boxes.
  The second disadvantage depends on the geometry of the object "inside" : the more "spherical" the object is the less tight the bounding box can be. A box around a sphere is $6/\pi \sim 2$ times large in volume than the sphere.
- *Spheres are second in popularity.* The geometry of the spheres is the simplest one, which makes them attractive. The transformation of spheres is very simple, too, since we only have to transform one point; there is no problem with axis-alignment.

An overlap test between two spheres is not quite an inexpensive as an overlap test between two axis-aligned boxes, but it is cheaper than a test between two arbitrarily oriented boxes. Overlap test between spheres and axis-aligned boxes are fairly cheap too.
Computing the optimal bounding sphere is not nearly as easy as computing the bounding box. The algorithm usually chosen is *linear or quadratic programming.*

- Other bounding volumes are slabs, cylinders etc.

Some of the desirable characteristics of bounding volumes are:

(a) Easy to compute.

(b) Little memory requirements.

(c) Fast transformable.

(d) Simple overlap test.

(e) Tight fitting.

2.3 Data Structures

2.3.1 *BSP*

Binary Space Partitioning was developed to partition a whole scene(a bunch of polygon) so as to solve the hidden surface problem.
The idea is to cut space recursively into two halves by a plane; we start with the whole "universe", cut it in half, and continue with each of the halves. See Figure 1 below.

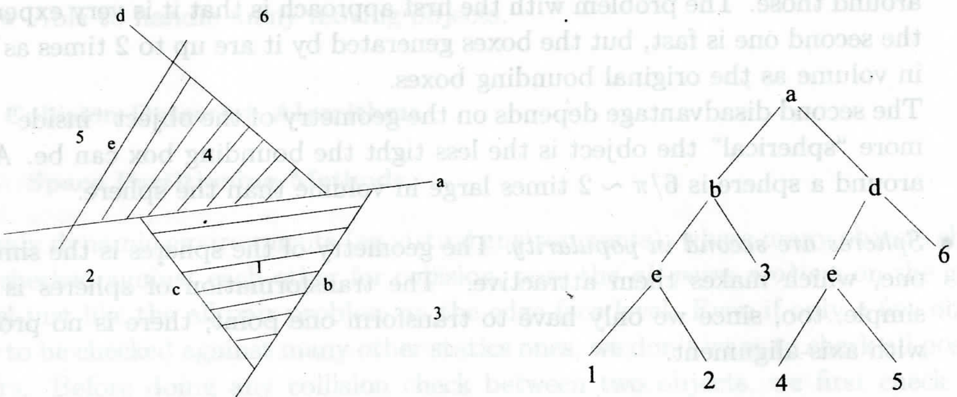

Figure 1: A BSP tree of a set of polygons

BSP-tree partitions each object into groups of parts which are close together in binary space.

2.3.2 Cell Subdivision

There are two basic classes of cell decompositions of space:

- *Uniform;* associated algorithms are very simple and as such prety fast. However, these data structures cannot adopt to large, entirely occupied or empty region or space.

- *Hierarchical;* advantages and disadvantages compared to the uniform structures are swapped. In addition, they usually use less memory.
  In contrast to BSPs, cell subdivisions are not "object oriented" but space oriented ie the data structure itself does not depend on the arrangement of objects. (with BSPs, the choice of partitioning planes depends heavily on the arrangement of objects). Instead, this data structure is built once at start-up time; later on, cells are just "filled" with objects they contain.

2.3.3 Grids

Almost always regular grids are used ie the cells are rectangular boxes. This makes them very simple; hence, algorithms operating on regular grids are very simple too.

2.3.4 Octrees

The basic idea with any cell decomposition in the context of collision detection is: whenever we want to know which other objects a given object could collide with, we don't have to consider object pairs which do not share a cell ie there is know cell which does not overlap (partially) with the two object's bounding box at the same time.
One can think of an octree as tree of cubes within cubes. But, the size of the cube varies depending on the number of objects occupying that region. A sparsely populated region is covered by one large cube, while a densly occupied region is divided into more smaller cubes. Each cube can be divided into 8 smaller cubes if necessary. So, each node in the tree has 8 children(leaves).
In order to find all object pairs which might intersect, we make a pass overall objects; for each object we look at all the cells it occupies. Then we have to do an exact collision detection only with those objects which are in one of these cells.
However there is a little problem: two objects might occupy several cells at the same time. With the nave approach, the same object pair would be handed to the exact collision detection several times in one frame. To prevent this from happening, we need to mark them some how. See[23] for a technique using time-stamp.

2.4 Distance Based Algorithms

A few algorithms use the notion of *distance* between polyhedra. Clearly, two polyhedra do not intersect each other if their distance is greater than zero. All of the algorithms presented in the literature can handle only convex polyhedra.
Gilbert, Johnson and Keerthi(GJK) [19] presented an algorithm to compute the distance between convex polyhedra with approximately linear complexity. An enhancement of this algorithm is given in [31].
Given two polyhedra,GJK searches for a simplex, defined by vertices of the Minkowski difference polydedra, that, either encloses or is nearest to the origin. If the origin is not enclosed, the distance between the origin and the nearest simplex of the difference polyhedra is equal to the distance between the original polyhedra. If the origin is enclosed, the polyhedra are penetrating and measure of the penetration is available.

2.5 Lin-Canny Algorithm [11,12,9]

The idea is to maintain a pair of closest features (vertex, edge,or face) with every pair of objects. Since in general, objects moves slowly compared to the frame rate, these features are probably still the closest ones in the next frame or if not, the closest features are in the neighborhood of the old ones.
The polyhedra have to be pre-processed in order to be able to maintain the closest features quickly. To do this, the *Voronoi* region to each pair is constructed. The *Voronoi region* associated with a feature is a set of points exterior to the polyhedron which are closer to the feature than any other[25]. Let $(\mathbf{a}, \mathbf{b})$ be a pair of features of polyhedra A and B resp; and $(\mathbf{P_a}, \mathbf{P_b})$ two points, with $\mathbf{P_a} \in \mathbf{a}$ and $\mathbf{P_b} \in \mathbf{b}$. Then $(\mathbf{a}, \mathbf{b})$ together with $(\mathbf{P_a}, \mathbf{P_b})$ realize the distance of A and B $\Leftrightarrow$ $\mathbf{P_a}$ is in the *Voronoi* region of $\mathbf{b}$ and $\mathbf{P_b}$ is in the *Voronoi* region of $\mathbf{a}$.
If two features $(\mathbf{a}, \mathbf{b})$ are given, we can easily check whether or not they do realize the distance of A, B. If they don't, we trivially know another feature (either of A or B whichever feature fails the test) which is closer to other one. Eventually, we must terminate with a closest pair, since the distance between each candidate feature pair decreases, as the algorithm steps through them. The test of whether one point lies inside of a *Voronoi* region of a feature is called " applicability test" There are three basic applicability criteria. These are (i) point-vertex, (ii)point-edge, and (iii) point-face applicability conditions. Each of these applicability criteria is equivalent to a membership test which verifies whether *the point* lies in the *Voronoi* region of the *feature.* If the *nearest points* on two features both lie inside the *Voronoi* region of the other feature, then the two features are closest feature pair and the points are closest points between the polyhedra. See [9] for details. This algorithm is used to implement a collision detection library called I-COLLIDE [10]. This is available in the web.

2.6 Separating planes

If we consider a convex polyhedron to be the convex hull of its vertices, we can reformulate the condition for collision detection as follows: Polyhedra P and Q do not intersect if and only if there is a plane H such that all vertices of P are on one side of H, and all vertices of Q are on the other side. See Figure 2.

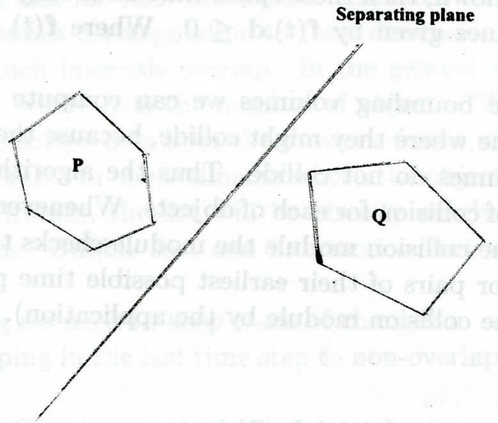

Figure 2: Separating Plane

Such a plane will be called a *separating plane*, and P and Q will be called *linearly separable*. H is said to be a *witness* to the separation of P and Q.
If no separating plane can be found, then polyhedra P and Q must be interpenetrating. Baraff [2,3] describes the use of a cache of witnesses. Each pair of polyhedra will have a cache of prior witnesses that can be checked for a possible valid separating plane whenever the current witness fails. The majority of time, the separating plane from the previous time step will still be the separating plane for the current step. In the case where non of the cached witness is a valid separating plane, the adjacent faces and edges are checked as potential separating planes. If a separating plane is found, then the polyhedra are not inter-penetrating. However, a further test needs to be made to decide whether they are disjoint or in contract with each other with no inter-penetration. GJK algorithm can be used to test this.
In[18] a "quick" algorithm to find the separating plane is described. This algorithm has been used to implement a collision detection library called Q-COLLIDE. This library is a modification of I-COLLIDE. Because of this, Q-COLLIDE is not available, at present, on the web.(due to legal procedings). He claims, that Q-COLLIDE is more efficient than I-COLLIDE.

2.7 Space-Time Approach

The idea with this approach is to consider a dynamic environment in 4 dimensions: space and time See[15]. Thus we can bound objects not just by volumes in space but also in time if their velocity is known. If we also know the upper bound on the acceleration, then we can bound objects by a parabolic cone in space-time. These cones are further bounded by axis-aligned $4-D$ trapizoid. If also the direction of the acceleration is known, then these space-time bounding volumes can be further bounded by hyper-planes given by $\mathbf{f}(t).\mathbf{d} \leq 0$. Where $\mathbf{f}(t)$ is the acceleration at time t.
With these space-time bounding volumes we can compute for any given pair of objects the earliest time where they might collide, because they can't collide as long as their bounding volumes do not collide. Thus the algorithm first calculates the earliest possible time of collision for each of objects. Whenever the application issues a collision query for the collision module the module checks the time and computes only exact collisions for pairs of their earliest possible time past (current time has to be announced to the collision module by the application).

2.8 Collision Detection for Multiple Objects

Virtual environments contain both stationary and moving objects. For example, the human participants may walk through a building where the tables, chairs etc remain stationary. In such an environment, there are N moving objects and M stationary objects.Each of the N objects can collide with the other moving objects, as well as the stationary one. Keeping track of $N^2\mathrm{C_2} + \mathrm{NM}$ pairs of objects at every time step can become time consuming as N gets large. To achieve interactive rates, we must reduce this number before performing pairwise collision.

2.8.1 Sweep and Prune Approach

Ponamgi et.al,[12] and Baraff [2,3] described a sweep and sort method to reduce the pairwise collision tests. Assume that each object is surrounded by some 3-dimensional bounding volume. We would like to sort these bounding volumes in 3-space to determine which pairs are overlaping. We would then sweep over these sorted bounding volume, pruning out the pairs that do not overlap. We only need to perform exact pairwise collision tests on the remaining pairs. They used a *dimension reduction* approach to sort the bounding volumes. If two bodies collide in $3-D$ space, their orthogonal projections into xy, yz and zx- planes and x, y, z- axes must overlap. [12] used axis-aligned boxes as the bounding volumes.

2.8.2 One Dimensional Sweep and Prune

The one dimensional sweep and prune algorithm begins by projecting each three -dimensional bounding box onto the x, y, z- axes. Because the bounding boxes are axis-aligned, projecting them onto the coordinate axes results in intervals. We are interested in overlaps among these intervals., because a pair of bounding boxes can overlap *if and only if* their intervals overlap in all three dimensions.
Construct three lists, one for each dimension. Each list contains the values of the end points of the intervals corresponding to their dimension. By sorting these lists, we can determine which intervals overlap. In the general case, such a sort would take $O(n \log n)$ time, where n is the number of objects. We can reduce this time bound by keeping the sorted lists from the previous frame, changing only the values of the interval end points. In environments where the objects make relatively small movements between frames, the list will be already nearly sorted, so we can sort in expected $O(n)$ time. Bubble sort and intersection sort work well for previously sorted lists.
In addition to sorting we need to keep track of changes in overlap states of interval pairs(ie from overlapping in the last time step to non-overlapping in the current step and vice versa).
This can be done in $O(n + e_x + e_y + e_z)$ time, where e_x, e_y, e_z are the number of exchanges along the x, y, $,z-$axes. This is also expected linear time due to coherence, but e_x, e_y, e_z can be $O(n^2)$ with an extremely small constant.
In computational geometry literature several algorithms exist that solve the static version of determining bounding box overlaps in $O(nlog^{d-1}n + s)$ time, where d is the dimension of the bounding boxes and s is the number of pairwise overlaps [43,44,45]. For $3 - D$, sweep and prune algorithm may reduce this to $O(n + s)$ by using coherence.

2.8.3 Two Dimensional Intersection Tests

This algorithm begin by projecting each $3 - D$ axis- aligned bounding box onto the $x - y$, $x - z$ and $y - z$ planes. Each of these projections is a rectangle in $2 - D$ space. Typically there are fewer overlaps of these $2 - D$ rectangles than of the $1 - D$ intervals used by sweep and prune technique. This results in fewer swaps as the objects move. They used interval trees to find the overlaps in $2 - D$.

3 Collision Response Algorithms

More and Wibhelms [16] give one of the earliest treatments of two fundamental problems in dynamic simulation: Collision detection and Collision response. They described two methods for collision response: using springs and using analytical solution.

3.1 Using Springs

When a collision is detected, a very stiff spring is temporarily inserted between the points of closest approach(or deepest interpenetration)of the two objects. The spring law is usually K/d, or some other functional form that goes to infinity as the separation d of the two objects approaches 0 (or the interpenetration depth approaches some small value), K is the spring constant controlling the stiffness of the spring. The spring force is applied equally and in opposite directions to the two colliding objects. The direction of the forces such as to push the two objects apart(or to reduce their interpeneration)
The spring method is easy to understand and easy to program. It applies equally well to rigid bodies (articulated or not) and to flexible bodies. The main problem with this method is that it is computationally expensive,; stiffer springs mean stiff differential equations, which require smaller time steps for accurate numerical integration [46]. The numerical effort required goes up with the violence of the collision: as the springs are compressed more and more, the equations become stiffer and stiffer and smaller and smaller time steps are needed.

3.2 Using Analytical Solutions

More and Wibhelms treatment

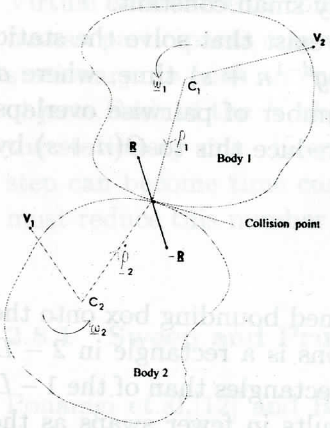

Figure 3: Collision of two objects

Each object has a linear velocity vector $\mathbf{v}_i$, an angular velocity vector $\underline{w}_i$. Mass of the object m_i , center of mass C_i, and inertial tensor matrix I_i which is relative to center of mass. All of these quantities, for both objects, are expressed in the same inertial reference frame. Each object has a vector ρ_i which points from its center of mass to the collision point. See Figure 3.

Three orthogonal vectors **i**, **j**, **k** defines the "collision frame"; **k** will be perpendicular to the plane of collision and **i** and **j** will be in that plane. If the vertex of one object is colliding with a face of the other, then that face defines plane of collision. If an edge of one object is colliding with an edge of the other, these two edges define the plane of collision. If two vertices are colliding, **k** is directed along the line joining. It is reasonable to assume that whenever two objects collide in the real world, there is one point at which they collide first. Thus, a collision detection algorithm must furnish a single collision point with two objects.
The solution involves solving a set of 15 linear equations in 15 unknowns. The fifteen unknowns are:

i. ($\mathbf{V_1}, \mathbf{V_2}$) ———————new linear velocities

ii. ($\underline{\Omega}_1, \underline{\Omega}_2$) ———————new angular velocities

iii. **R** ————impulse vectors

Because the collision is assumed to occur in a negligible time (approximately instantaneous), only the collision impulse itself matters; any other forces being applied to the objects will be too small to have an effect.

Applying *Impulse* = *change in momentum*

$m_1\mathbf{V_1} = m_1\mathbf{v_1} + \mathbf{R}$
$m_2\mathbf{V_2} = m_2\mathbf{v_2} - \mathbf{R}$

Applying $\mathbf{r} \times \mathbf{F} = \Delta(I\underline{w})$ gives
$I_1\underline{\Omega}_1 = I_1\underline{w}_1 + \rho_1 \times \mathbf{R}$
$I_2\underline{\Omega}_2 = I_2\underline{w}_2 - \rho_2 \times \mathbf{R}$

The last three equations come from assumptions about collisions conditions:

(a) the elasticity $\epsilon = 0$(two colliding objects come to rest relative to each other) and surfaces are friction less (impulse perpendicular to the collision plane).

Then we have

$\mathbf{R} \cdot \mathbf{i} = 0$
$\mathbf{R} \cdot \mathbf{j} = 0$
and ($\mathbf{V_2} + \underline{\Omega}_2 \times \rho_2 - \mathbf{V_1} - \underline{\Omega}_1 \times \rho_1$) $\cdot\, \mathbf{k} = 0$

These equations can be solved by standard Gauss elimination algorithm.

(b) elastic collision ($\varepsilon \neq 0$)

In this case new collision impulse $\mathbf{R}_{\text{actual}}$ is given by

$$\mathbf{R}_{\text{actual}} = (1 + \varepsilon_{\text{actual}})\mathbf{R}\,.$$

This new collision impulse is then plugged back into the defining equations above, to solve for $\mathbf{V}_i$ and $\underline{\Omega}_i$.
For other cases of collision conditions such as friction etc refer [16].

3.3 Baraff's [1, 3] treatment

He starts by labeling the polyhedra as A and B. For vertex/face contact the polyhedron whose vertex is in contact is labeled A and the other is labeled B. For edge/edge they are labeled arbitrarily.
Point of contact vector is $\mathbf{p}$ and $\mathbf{p}_a$ and $\mathbf{p}_b$ are the points on A and B respectively, which will come into contact at $\mathbf{p}$ at time t_c.

i.e. $\mathbf{p}_a(t_c) = \mathbf{p}_b(t_c) = \mathbf{p}$.

Then $\dfrac{d\mathbf{p}_a}{dt} = \mathbf{v}_a(t) + \mathbf{w}_a(t) \times (\mathbf{p}_a(t) - \mathbf{r}_a(t))$.

And $\dfrac{d\mathbf{p}_b}{dt} = \mathbf{v}_b(t) + \mathbf{w}_b(t) \times (\mathbf{p}_b(t) - \mathbf{r}_b(t))$.

Then the relative velocity between the two polyhedron is given by

$$\mathbf{v}_{rel} = \hat{\mathbf{u}} \cdot \left(\frac{d\mathbf{p}_a(t_c)}{dt} - \frac{d\mathbf{p}_b(t_c)}{dt} \right)$$

where $\hat{\mathbf{u}}$ is the unit vector along the collision normal.

If $|\mathbf{v}_{rel}| > 0$ then the polyhedra are moving away from each other and no further action is necessary.
If $|\mathbf{v}_{rel}| = 0$ then the polyhedra are in resting contact and a *contact force* should be applied to keep them from inter-penetrating.
If $|\mathbf{v}_{rel}| < 0$ then the polyhedra are in colliding contact and an *impulse* should be applied to keep them from inter-penetrating.

(i) *Colliding Contact*

In this case an impulse $\mathbf{J}$ must be applied immediately to change the direction of travel.

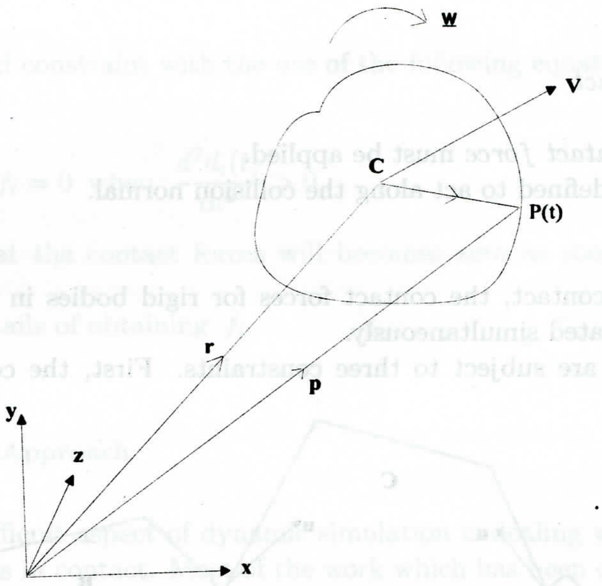

Figure 4: Polyhedron after applying impulse

$$\mathbf{J} = \lim_{\substack{|\mathbf{F}| \to \infty \\ \Delta t \to 0}} \mathbf{F} \Delta t$$

Also $\mathbf{J} = \Delta(\mathbf{P})$ (1)

This gives $\mathbf{J} = \Delta(M\mathbf{v}) \Rightarrow \Delta\mathbf{v} = \dfrac{\mathbf{J}}{M}$ (2)

An impulse applied to a rigid body will create an impulse torque $\mathcal{T}$

Then we have $\mathcal{T} = \left(\mathbf{p} - \mathbf{r}(t)\right) \times \mathbf{J}$.
So change in angular momentum will be

$\Delta\mathbf{L} = \mathcal{T}$ (3)

i.e. $\Delta\left(I\mathbf{w}\right) = \mathcal{T} \Rightarrow \Delta(\mathbf{w}) = I^{-1}(t)\mathcal{T}$.
Baraff[1] defines $\mathbf{J}$ to be

$\mathbf{J} = j\hat{\mathbf{u}}$.
Where j is a scalar representing the magnitude of $\mathbf{J}$.
For a derivation of j refer [1, 3].
If we know j, we can calculate the new linear, angular velocities and momentum using (1), (2) and (3).

(ii)*Resting Contact*

In this case a *contact force* must be applied.
Contact force is defined to act along the collision normal.
i.e. $\mathbf{F} = f\hat{\mathbf{u}}$

Unlike colliding contact, the contact forces for rigid bodies in resting contact all need to be calculated simultaneously.

Contact forces are subject to three constraints. First, the contact forces must

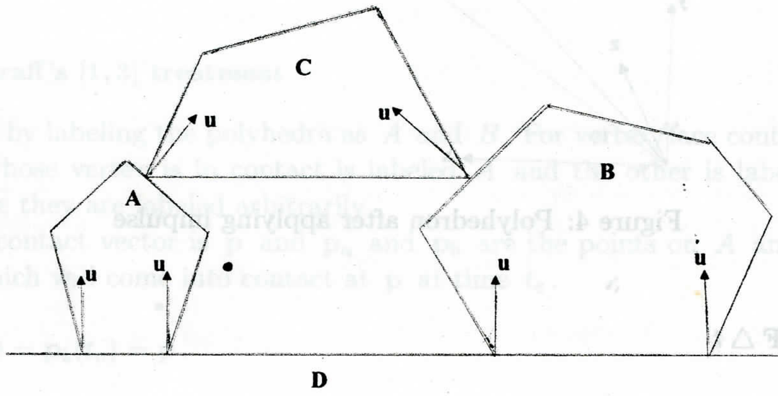

Figure 5: Resting Contact

prevent inter-penetration. Secondly, the contact forces should be repulsive, that is they act to push bodies apart not "glue" them together. Lastly, a contact force must vanish to zero once the bodies begin to separate.
Baraff [1, 3] defines the function $d_i(t)$ to be the distance between the two rigid bodies involved in i^th contact at a time t.
i.e. $d_i(t) = \hat{\mathbf{u}} \cdot \left(\mathbf{p}_a(t) - \mathbf{p}_b(t)\right)$ with $d_i(t_c) = 0$.

The goal is to keep $d_i(t) \geq 0$ for values of $t > t_c$.
$\dfrac{d^2 d_i(t)}{dt^2}$ gives how fast the two bodies are accelerating away from each other.
By restricting this acceleration to be non negative it will be possible to guarantee the first constraint.
For the second and third constraints, Baraff [1, 3] defines the quantity f_i to be the force present at the i^{th} contact point. He shows [1, 3] how to meet the second constraint by restricting f_i to be non-negative. This guarantees that the contact force will be repulsive and will not act to keep the bodies together.

He satisfies the third constraint with the use of the following equation:
$f_i \dfrac{d^2 d_i(t)}{dt^2} = 0$
and requiring that $f_i = 0$ when $\dfrac{d^2 d_i(t)}{dt^2} > 0$.
This guarantees that the contact forces will becomes zero as soon as the bodies begin to separate.
See [1, 3] for the details of obtaining f_i.

4 Impulse Based Approach

One of the most difficult aspect of dynamic simulation is dealing with the interactions between bodies in contact. Most of the work which has been done in this area falls into the category of constraint-based methods [1, 2, 4, 47, 48, 16].
Brain Mirtich and John Canny [5, 6] proposed a different approach-*impulse based approach*. Consider a ball rolling along a table top. The normal force which the table exerts on the ball is a constraint force that does not do no work on the ball, but only enforces a non-penetrating constraint. In a constraint based approach, this force is not modeled explicitly but is instead accounted for by a constrain on the configuration of the ball.(in this case, the ball's z-coordinate is held constant). The problem with this method is that as a dynamic system evolves, the constraints may change many times eg. The ball may roll off the table, may hit an object on the table etc. Determining the correct equations of motion for the ball means keeping track of all these changing constraints, which can become complicated. Moreover it is not clear what type of constraint should be applied. Finally impacts are not easily incorporated into the constraint model, as they generally give rise to impulses, not constraint forces present over some interval. These collision impulses must be handled separately as in [1].
In impulse-based method, no constraints are imposed on the configurations of the moving objects; when the objects are not colliding, they are in ballistic trajectories. The key advantage of the impulse-based method is the unification of all types of contact under a single model. The model used for collisions between objects can also be used for continuous contact situations in which one object is resting, sliding or rolling on another object. Consider for example a block resting on a table. Under impulse-based simulation, the block is actually experiencing many rapid, tiny collisions with the table, each of which can be resolved as any other collisions. During this small, vibrating motion, different corners of the block will collide and rebound with the table. These small, frequent collision, between objects in continuous contact are called *microcollisionas* [6].

4.1 Computing Collision Impulses

When two bodies collide, an impulse **p** must be applied to one of the bodies to prevent inter-penetration; an equal and opposite impulse-**p** is applied to the other. Once **p** and the point of application is known, it is a simple matter to compute the new center of mass and angular velocities for each body. After updating these velocities, dynamic state evolution can continue, assuming ballistic trajectories for all moving objects. The point of application is computed by the collision detection system and hence the central problem in collision resolution is to determine the collision impulse **p**.

There are three assumptions central to Mirtich and Canny analysis:

1. Infinitesimal collision time
2. Poisson's hypothesis
3. Coulomb friction model.

Since the frictional force is dependent on the relative sliding velocity of the bodies in contact, and this velocity is not constant during a collision, the dynamics of the object must be analyzed during the collision.
Let **u** be the relative velocity between the two bodies at the contact point, and $\Delta\mathbf{u}$ be the change in this quantity over the course of the collision. If **p** is the collision impulse imparted by one body on the other, they obtained
$\Delta\mathbf{u}(\gamma) = M\mathbf{p}(\gamma)$.
Here, $\gamma = p_z$, the normal component of the impulse delivered to the body 1, and, M is a 3×3 matrix dependent only upon the masses and mass matrices of the colliding bodies, and the locations of the contact points in the body frames. We can compute **p** by tracking **u** during the collision, and then inverting the above equation.
Using the Coulomb sliding friction law, one can derive a differential equation for **u** [6]. The equations are integrated by the collision response subsystem to track the evolution of **u** during a collision. If sticking occurs during this integration, ($u_x = u_y = 0$), the model changes and a simpler set of equations governs the evolution of **u**.

In [7] Mirtich described a simulator called *Impulse* which combines Constraints and Impulses.

5 Collision Handling Procedure for Convex Polyhedra

Here I will describe a procedure by Mirtich and Canny [6]. There are three main modules in this procedure:

(i) Dynamic state evolution

(ii) Collision Detection

(iii) Collision resolution

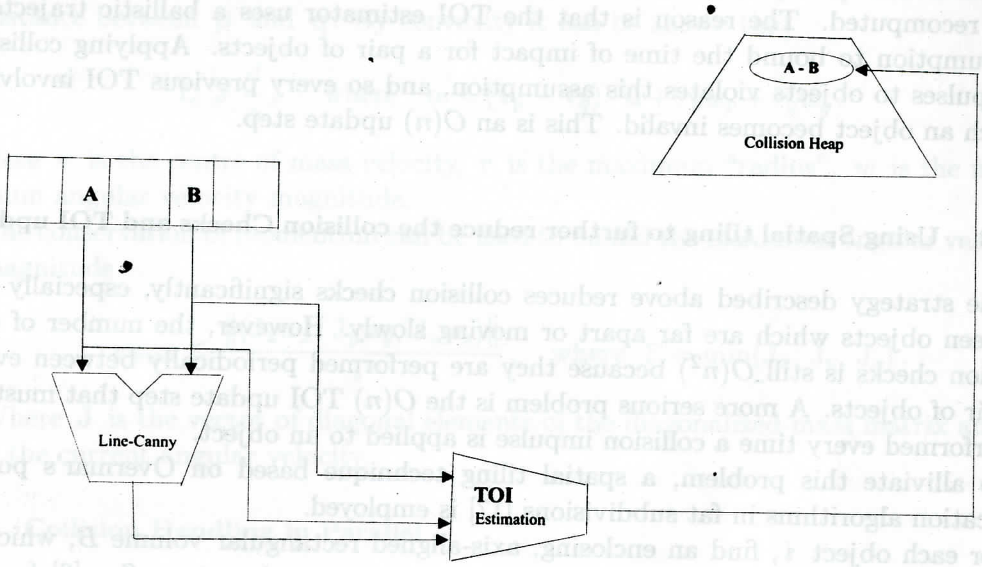

Figure 6: Collision Handling modules

5.1 Prioritizing Collision

Obviously, checking for possible collisions between all pairs of objects after every integration step is too inefficient. Instead, collision are prioritized in a heap. For each pair of objects in the simulation, there is an element in the heap, which also contains a lower bound on the time of impact(TOI) for the given pair of objects. The heap is sorted on the TOI field, thus the TOI field of the top heap element always gives a "safe" value for the next collision free integration step. After an integration step, the distance between the objects on the top of the heap (call A and B; see figure above) must be recomputed. Lin-Canny algorithm can be used for this purpose. This is an extremely efficient algorithm. Collisions are declared when the distance between objects falls below some threshold ε_c. First suppose the distance between A and B lies above ε_c. In this case, the dynamic states of A and B along with the output of the Lin-Canny algorithm are used to compute a new conservative bound on the time of impact of A and B. The $A-B$

heap pair is updated with this new value, possibly affecting its heap position, and the integration is ready for another step.
If the distance between A and B is less than ε_c, a collision is declared. The collision resolution system computes and applied collision impulses to the two objects, changing their dynamic state. At this point the TOI is recomputed for these objects as before. TOI between all objects pairs of the form $A - x$ and $B - x$ must also be recomputed. The reason is that the TOI estimator uses a ballistic trajectory assumption to bound the time of impact for a pair of objects. Applying collision impulses to objects violates this assumption, and so every previous TOI involving such an object becomes invalid. This is an $O(n)$ update step.

5.2 Using Spatial tiling to further reduce the collision Checks and TOI update

The strategy described above reduces collision checks significantly, especially between objects which are far apart or moving slowly. However, the number of collision checks is still $O(n^2)$ because they are performed periodically between every pair of objects. A more serious problem is the $O(n)$ TOI update step that must be performed every time a collision impulse is applied to an object.
To alliviate this problem, a spatial tiling technique based on Overmar's point-location algorithms in fat subdivisions [17] is employed.
For each object i, find an enclosing, axis-aligned rectangular volume B, which is generated to contain the object during the next integration step. See [6] for the construction of this box.
The idea is to keep track of which objects are near each other, by keeping track of which bounding boxes overlap. To do this, the physical space is partitioned into a cubical tiling with resolution ρ. Under this tiling, coordinates in physical space are mapped to integers under the tiling map σ:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \xrightarrow{\sigma} \begin{pmatrix} x/\rho \\ y/\rho \\ z/\rho \end{pmatrix}$$

Let S_i be the set of tiles which i intersects. We store i in a hash table multiple times, hashed on the coordinates of each tile in Si. Clearly objects i and j can only possibly collide during the next integration step if i and j are both present in the same hash bucket. Only in this case do we keep object pair $i - j$ in the collision heap. Furthermore, if object i experiences a collision impulse, TOIs need only be recomputed for object pairs $i - k$ where object k shares a hash bucket with object i.

5.3 TOI estimation

TOI estimator takes the current dynamic state (position and velocity) of two objects as well as the closest points between them and returns a lower bound on the time of impact for those two objects.
Let $\mathbf{p}$ and $\mathbf{q}$ be the current closest points between two objects P and Q on a collision course. Let $\hat{\mathbf{u}}$ be a unit vector in the direction of $\mathbf{p}-\mathbf{q}$ and d be the distance between $\mathbf{p}$ and $\mathbf{q}$. By convexity it can be shown that

$$t_c \geqslant \frac{d}{\alpha}, \quad \text{where} \quad \alpha = (\mathbf{v}_p - \mathbf{v}_q)\cdot\hat{\mathbf{u}} + r_p w_p + r_q w_q .$$

Here $\mathbf{v}$ is the centre of mass velocity, r is the maximum "radius", w is the maximum angular velocity magnitude.
The conservation of momentum can be used to bound the maximum angular velocity magnitude.

$$w_{max} \leqslant \frac{\|(\mathrm{J}_x w_x, \mathrm{J}_y w_y, \mathrm{J}_z w_z)\|}{\Gamma}, \quad \text{where } \Gamma = \min(\mathrm{J}_x, \mathrm{J}_y, \mathrm{J}_z).$$

Where $\mathbf{J}$ is the vector of diagonal elements of the diagonalized mass matrix and $\boldsymbol{w}$ is the current angular velocity.

6 Collision Handling in Parallel

Here we give a procedure described in [8].

(a) Initialize all of the various data structures that will be used.

(b) Objects are read in and distributed across the processors. Each processor is in charge of maintaining the state of a smaller subset of objects. Fixed objects are local to all processors as well as static data about all of the objects such as their polygonal models, tiling resolution etc. So, objects in collision with fixed objects or with objects that are local to the same processor achieve near linear parallel speed up.

(c) Next the closest feature table is initialized. This table is used by Lin- Canny algorithm to find the next closest pair between two objects. This simply provides a starting pair for the algorithm. This table can be quite large for many objects with large numbers of facts. This portion of the initialization step is done only once at the beginning and is performed entirely on one processor. The closest feature table is a global structure visible to all processor.

(d) Then compute initial boundary volumes of the objects and place them into a data structure.

Here all of the processors compute the bounding volumes of their local objects and place them into the global hash tiling. The bounding volume calculations all occur in parallel for the objects. Access to the hash tiling structure is via atomic access routines to the hash bins. Objects that don't hash to the same spatial hash tiling will not be in contention for the same block of data and hence will proceed to operate in parallel.

(e) Initialize the collision heap.
This is ordered by the estimated time to impact of each object pair. See section 5.1 for details. Since this portion is done only at the beginning, this can be done entirely on one processor.

(f) Removing collision heap.
Since the collision heap is an ordering on possible collision object pairs by soonest expected time to collision, we simply remove the top heap element. We are guaranteed that no such collision can occur before this top heap.
The heap is local to one processor. Therefore, that processor simply removes the top heap element and broadcasts to all of the other processors both the pair involved in the collision and the time step for this simulation loop.

(g) Computation of swept bounding volume.
During the initialization process, we pre-compute a maximum radius for each object from its centre of mass. We use this radius and the object's initial and final position for this time step to compute a worst case swept bounding volume.
This computation occur in parallel for the objects on all processors. Since each processor maintains a distributed subset of all of the objects, all computations are local for determining the swept volume.

(h) Updating spatial hash tiling.
After parallel computation of the swept volumes for all of the objects, we determine the new hash entries that need to be inserted into the spatial tiling. If an object's bounding volume has not changed since it was last updated, no work is required and no modification to the hash tiling are made for that object. When new hash entries need to be inserted and old ones removed, we use atomic operations for accessing the bin. Access to the tiling data structure can be performed in parallel when processors are making updates to different spatial hash tiling bin. We only require atomic access to the bin for updates and that we only make updates when an object enters or exists a spatial hash tiling bin.

(i) Evolve Ballistic trajectory.
This proceeds in parallel on all processors. No communication is necessary

since all information is local to each of the processors.

(j) Collision Handling.
If the collision is between a moving object and a fixed object, we perform the collision calculations on the processor local to the moving object. This will insure that the entire collision operation can occur locally since fixed objects are local to all processors. We may also have a collision between two objects that are on the same processor, in which case collision calculations can proceed without communication. In the worst case the two objects will be on separate processors. If so, we simply choose one of the processor to handle the collision and spend time in communicating a few of the necessary pieces of data about the remote object to the controlling processor. Then collision and impulse calculations can proceed locally.
After the impulses have been applied to the two objects, we need to perform the necessary updates to the heap. Since we have reduced the heap size to include only object pairs that are close to one another, we will only need to perform heap updates and new time to impact estimations for objects that are close, as defined by the spatial hash tiling, to one of the recently colliding objects. Rather than performing all the necessary communication to this single processor, we simply broadcast flags and the necessary object data necessary to processors with at least one of the objects in the pair to be updated as local.

(k) Recording state of the system.
Since, again, each processor maintains a subset of all the objects, all of the processors can, in parallel, record the objects state to an internal array or to a graphics file. Since the objects are local to each processor, this proceeds in parallel without any need of communication.

6.1 An application

Here we consider the modeling of a refiner in a pulping factory using the above parallel procedure.
The refiner consists of a rotor and stator. Rotor rotates with a constant angular velocity. Wood chips are released with certain initial velocity towards the rotor. We want to model the behaviour of these wood chips.
Here the rotor and the stator could be meshed into tetrahedral elements. These tetrahedra could be considered as objects in the simulation. Only external faces and edges should be taken into account for the application of the Lin-Canny algorithm. Elements in stator are stationary and could be considered as stationary objects in the simulation. Elements in the rotor are moving in a fixed circular path. We can exploit this special property:

For a particular value of the angular velocity of the rotor, the tetrahedral elements will be at a predefined position in space at a particular time of the simulation. Therefore before beginning the simulation, we calculate the bounding boxes of the tetrahedra in the rotor mesh for all time steps. We distribute equal number of tetrahedra to each processor and do the computation of bounding boxes in parallel. After this computation every processor will broadcast their bounding box data to every other processor. Now every processor will have all the bounding boxes necessary for the overlap test locally during the entire simulation period. Since the wood chips are rectangular blocks we do not need to compute the bounding boxes for these. Swept bounding boxes for the tetrahedra also calculated at the start of the simulation for all time steps.

If we change the angular velocity or time step we need to recompute the bounding boxes. Also note that we have to test the collision between tetrahedra and wood chips and among wood chips. We do not check the collision among tetrahedra.!.

7 Conclusions

Collision Detection is the bottleneck in real time multibody simulation software. We need efficient algorithms to detect collisions. In this respect Lin-Canny algorithm seem to be an efficient algorithm, at present, for the distance-based collision detection. I-COLLIDE, V-COLLIDE libraries are based on this algorithm. Recently Mirtich [30] published an algorithm, which is an enhancement of Lin-Canny algorithm and performs better than Lin-Canny and Enhanced GJK[31]. He implemented a library called V-clip based on this algorithm.

Kelvin in his thesis[18], described a collision detection library called Q-COLLIDE which uses separating plane and an enhancement of GJK. He reported that Q-COLLIDE performs better than I-COLLIDE.

Impulse[7] is one of the collision handling library. It consists of over 27000 lines of C code written by Brian Mirtich. It is based on Lin-Canny and Impulse based collision resolution algorithm.

More recent work seems to have focused on tighter-fitting bounding volumes.

Gottschalk et.al, [14, 49] have presented a fast algorithm and a system, called RAPID, for collision-detection based on oriented bounding boxes (OBBs) which approximate geometry better than do axis-aligned bounding boxes.

Very few parallel implementations are reported in the literature. Zachmann [23] has reported some results in his thesis. He used *fine-grain* and *coarse-grain* parallelization. Recently Eric[8] has parallelized *Impulse* using *coarse-grain* parallelization. He used split-C to parallelize the original serial code.

To achieve real-time performance in an environment like virtual reality, parallelization would be beneficial. In large scale environment with many moving bodies, coarse-grain parallelization using MPI would be worth considering.

References

[1] Baraff, David, 1989. "Analytical Methods for Dynamic Simulation of Non-penetrating Rigid Bodies", In Computer Graphics Proceedings, Annual Conference Series, 1989, by ACM SIGGRAPH, pp. 223-232.

[2] Baraff, David, May 1992. "Dynamic Simulation of Non-Penerating Rigid Bodies" PhD. thesis, Cornell University.

[3] Baraff, David,1995. "Rigid Body Simulation" In An Introduction to Physically Based Modelling, Course Notes by ACM SIGGRAPH, 1995, G1-G68.

[4] B.Aude Charles, December 1996. "Computer Simulation of Rigid Body Motion", MSc. thesis, Eastern Washington University.

[5] Mirtich Brain and Canny John 1995. "Impulse-Based Dynamic Simulation". IN K.Goldberg, D. Helperin, J.C Latombe, and R. Wilson, editors. The algorithmic foundations of robotics. A.K. Peters, Boston, MA, Proceedings from the workshop held in February, 1994.

[6] Mirtich Brain and Canny John, 1995. "Impulse-Based Simulation of Rigid Bodies". In Symposium on Iteractive 3D Graphics, New York, ACM press.

[7] Mirtich Brain "Hybrid Simulation: Combining Constraints and Impulses",WWW, mirtich@cs.berkely.edu

[8] WWW, Eric Paulos/paulos@robotics.eecs.berkelly.edu

[9] Lin M.C, December 1993. "Efficient Collision Detection for Animation and Robotics" PhD thesis, Department of Electrical Engineering and Computer Science, University of California, Berkeley.

[10] Cohen J, Lin M, Manocha.D and Ponamgi 1995. "I-COLLIDE": An Interactive and exact collision detection system for large-scale enviornments. In Proc. of ACM Interactive 3D graphics Conference, pp. 189-196.

[11] Lin.M.C and Canny.J.F, 1991. "Efficient algorithms for incremental distance computaion". In IEEE conference on Robotics and Automation, pp. 1008-1014.

[12] Ponamgi M, Monocha D and Lin M, 1995. "Incremental algorithms for collision detection between solid models" Proceedings of ACM/ Siggraph Symposium on Solid Modeling, pp. 293-304.

[13] LIn M.C and Manocha D 1996. "Efficient contact determination between geometric models", International Journal of Computational Geometry and Applications.

[14] S.Gottschalk, M.Lin and D.Manocha. 1996. "Obb-tree : A hierarchical structure for rapid interference detection". In Proceedings of ACM Siggraph', **96:** 171-180.

[15] Hubbard P.M, October 1993. "Interactive collision detection". In Proceedings of IEEE symposium on Research frontiers in Virtual Reality .

[16] Moor M and Wilhelms, 1988. "Collision detection and response for computer animation", Computer Graphics , **22(4)** 289-298.

[17] Overmas M.H,1992. "Point location in fat Subdivisions", Inform.Proc. Lett. , **44** 261-265.

[18] Kelvin C.T.L, 1996. "An Efficient Collision Detection Algorithm for Polytopes in Virtual Environment" M.Phil. thesis. The University of Hong Kong.

[19] Gilbert, E.G, Johnson D.W and Keerthi S.S, 1988. "A fast Procedure for Computing the Distance Between Complex Objects in Three-Dimensional Space". In IEEE Journal of Robotics and Automation, pp. 193-203.

[20] Samet H, 1990. "The design and Analysis of Spatial Data Structures" Addison-Wesely, Reading, MA.

[21] Samet H, 1990. "Applications of Spatial Data Structures" Addison- Wesely, Reading, MA.

[22] Thibault W.C and Naylor B.F, July 1987. "Set Operations on Polyhedra Using Binary Space Partitioning Trees" In Baureen C. Stone, editor, Computer Graphics SIGGRAPH 87 Proceedings, **Vol.21** 153- 162.

[23] Zachmann G, 1994. "Exact and Fast Collision Detection", Diploma thesis, Beckman Institute.

[24] Zachmann G and Felger W, 1995. "The box-tree:Enabling real-time and exact collision detection of arbitrary polyhedra" Procedings of SIVE'95.

[25] Preparata F.P and Shamos M.I, 1995. "Computational Geometry" Springer-Verlag, New York.

[26] Hubbard P.M, July 1995. "Real-Time Collision Detection and Time-Critical Computing", from the workshop on Simulation and Interaction in Virtual Enviornment.

[27] Hubbard P.M, July 1996. "Approximating Polyhedra with Spheres for Time-Critical Collision Detection", ACM Transactions on Graphics , **Vol. 15, No. 3** 179-210.

[28] Hubbard P.M, October 1994. "Collision Detection for Interactive Graphics Applications", PhD thesis, Department of Computer Science, Brown University.

[29] Hudson T.C, Lin M.C, Cohen J, Gottschalk S and Manocha "DV-COLLIDE:Accelerated Collision Detection for VRML", http://www.cs.unc.edu/~ geom/collide.html

[30] Mirtich Brian, 1997. "V-Clip:Fast and Robust Polyhedral Collision Detection", ACM Transaction on Graphics.

[31] Cameron S, April 1997. "Enhancing GJK: Computing minimum penetration distances between convex polyhedra", In International Conference on Robotics and Automation, IEEE.

[32] Chung K "Quick collision detection library". www.cs.hku.hk/~tlchung/collision_ library.html

[33] Mirtich Brain, December 1996. "Impulse-Based Dynamic Simulation of Rigid Body Systems" PhD thesis, University of California, Berkely.

[34] Fujimura K and Kunii T.L, 1985. "A Hierarchical space indexing method". In Kunii T.L, editor, Computer Graphics Visual Technology and Art:Proceedings of Computer Graphics Tokyo '85, Springer-Verlag, pp. 21-33.

[35] Carlbom I, Chakravarty I, and Vanderschel D, April 1985. "A Hierarchical Data Structure for Representing the Spatial Decomposition of 3-D Objects". IEEE Computer Graphics and Applications **5(4)** 24-31.

[36] Navaso I, Ayala D and Brunet P, June 1986. "A geometric Modeller based on the Exact Octree Representation of Polyhedra" Computer Graphics Forum, **5(2)** 91-104.

[37] Canny J, March 1986. "Collision Detection for Moving Polyhedra", IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-**8(2)** 200-209.

[38] Youn Ji-Hoon and Wohn K, 1993. "Realtime Collision Detection for Virtual Reality Applications" In IEEE Virtual Reality Annual International Symposium, **September 18-22** 415-421.

[39] Mehlhorn K and Simon K, 1985. "Intersecting two polyhedra one of which is convex". In Budach L editor, Proc. Found. Comput. Theory, **Vol. 199** of Lecture Notes in Computer Science, Springer-Verlag, pp. 534-542.

[40] Agarwal P.K and Sharir P, 1990. "Red-blue interaction detection algorithms, with applications to motion planning and collision detection". SIAM J. Computer, **19:** 297-321.

[41] Snyder J.M, Woodbury A.R, Fleischer K, Currin B, and Barr A H, August 1993. "Interval Method for Multi-point Collision Between Time-dependent Curved Surfaces". In Kajiya J.T, editor, Computer Graphics :SIGGRAPH '93 Proceedings, **Vol. 27** 321-334.

[42] Lin M.C and Canny J.F, September 1992. "Efficient Collision Detection for Animation".

[43] Hopcroft J.E, and Schwartz J.T and Sharir M, 1983. "Efficient detection of intersections among spheres" Internat. J. Robot. **Res.,2(4)** 77-80.

[44] Six H.W and Wood D, March 1982. "Counting and reporting intersections of d-ranges". IEEE Trans. On Computers, **C-31** (No.3).

[45] Edelsbrunner H, 1983. "A new approach to rectangle interactions, Part I", Internat. J. Comput. Math., **13** 209-219.

[46] Gear C.W, 1971. "Numerical Initial Value Problems in Ordinary Differential Equation", Prentice-Hall, Engle wood Cliffs, NJ.

[47] Cremer J.F and Stewart A.J, May 1989. "The architecture of Newton, a general - purpose dynamics simulator", In International Conference on Robotics and Automation ,IEEE, pp. 1806-1811.

[48] Witkin A, Gleicher M and Welch W, March 1990. "Interactive Dynamics" Computer Graphics, **24(2)** 11-22.

[49] RAPID interference detection system http://www.cs.une.edu/ geom//OBB//OBBT.html.