Submitted to the Archives of Computational Methods in Engineering

Discrete Element Simulation and The Contact Problem

*John R. Williams*and Ruaidhrí O'Connor†*
Intelligent Engineering Systems Laboratory (IESL),
Department of Civil and Environmental Engineering,
Massachusetts Institute of Technology
Cambridge, MA 02139, U.S.A.

Abstract

This paper addresses the problem of contact detection in discrete element multibody dynamic simulations. This paper presents an overview of the problem and a detail description of a new object representation scheme called the discrete function representation (DFR). This representation is designed to reduce the computational cost of both contact detection and the more difficult problem of contact resolution. The scheme has a maximum theoretical complexity of order $O(N)$ for contact resolution between bodies defined by N boundary points. In practice, the discrete element method constrains overlap between objects and the actual complexity is approximately $O(\sqrt{(}N)$ giving a speedup of nearly 2 orders of magnitude over traditional algorithms for systems with more than 1000 objects. The technique is robust and is able to handle convex and concave object geometries, including objects containing holes. Examples of relatively large discrete element simulations in three dimensions are presented.

Key Words: Object Representation; Contact Detection; Object Intersection; Physically Based Modeling;Discrete Element Methods

1 Introduction to the Contact Problem

In this section we give an overview of the contact detection problem as implemented in a discrete element simulation. We start with the general N-body problem and then focus on the more restricted problem of geometric contact between bodies with specific geometric shape.

*Associate Professor of Civil & Environmental Engineering.
Director, Intelligent Engineering Systems Laboratory.
E-mail: john@iesl.mit.edu
†Research Assistant. E-mail: roryoc@iesl.mit.edu

1.1 Discrete Elements and Multi-Body Systems

There are several classes of problem associated with automatic reasoning about interactions in N-body systems. The most general problem involves long range forces typical of gravitational systems. The central difficulty in analyzing these systems is the approximation of the long range *all-to-all* coupling between bodies. This can be efficiently accomplished by using clustering techniques and/or multipole expansions first proposed by Greengard in 1988 [Gre88].

The second class of N-body system, in which interactions are medium range, are those associated with molecular dynamics. Recently, there have been several advances in the development of efficient parallel N-body applications that allow researchers to consider systems containing as many as $10^6 - 10^8$ bodies [WS93, BL95]. However, these analyses were conducted for very few time steps. The systems analyzed are usually characterized by central force fields with a high degree of symmetry so that each object can be represented as a point charge or spherical potential field.

The third class of N-body system, in which interactions are short range, is typified by mechanical contact and impact problems. In these problems the number of bodies directly influenced by contact with a given body is usually less than 20. The core difficultly in these analyses is in determining the geometric details of the contact between the bodies. Table 1 illustrates the time a typical discrete element code spends in its various functions. We note that the time spent by the code in dealing with geometric reasoning, here called contact detection, ($\geq$ 80%) far exceeds that involved with solving the equations of Newtonian physics ($\leq$ 10%) or penalty force generation ($\leq$ 3%)). The contact detection process is known to be the principal cost in DEM simulation, scaling as a function of both the number of objects being simulated [Wil88, Can88] and the complexity of each object's surface geometry, [Hah88, MW88, Pen91, Bar90, WO95a]. In the next sections we discuss in detail the problem of contact algorithms and geometric object representation in discrete element analysis.

Function Name		Time in Self [msec]	% of Total	Function Calls
simulation housekeeping		77.20	4%	1000
CONTACT DETECTION	spatial search	1197.89	60%	2056000
	resolve geometry	422.90	21%	8482677
	penalty force	71.33	3.6%	981745
Newtonian physics		156.72	8%	1157239

Table 1: Simulation Profile for Analyzing 1000 Two Dimensional Objects

1.2 The Discrete Element Pipeline

The four principal components of a general discrete element simulation system are considered to be a) Object Representation b) Contact Detection c) Physics d) Visualization, These components are represented graphically in the DEM analysis pipeline shown in Figure 1. In this paper we focus our attention on the *object representation* and *contact detection* phases of the DEM pipeline.

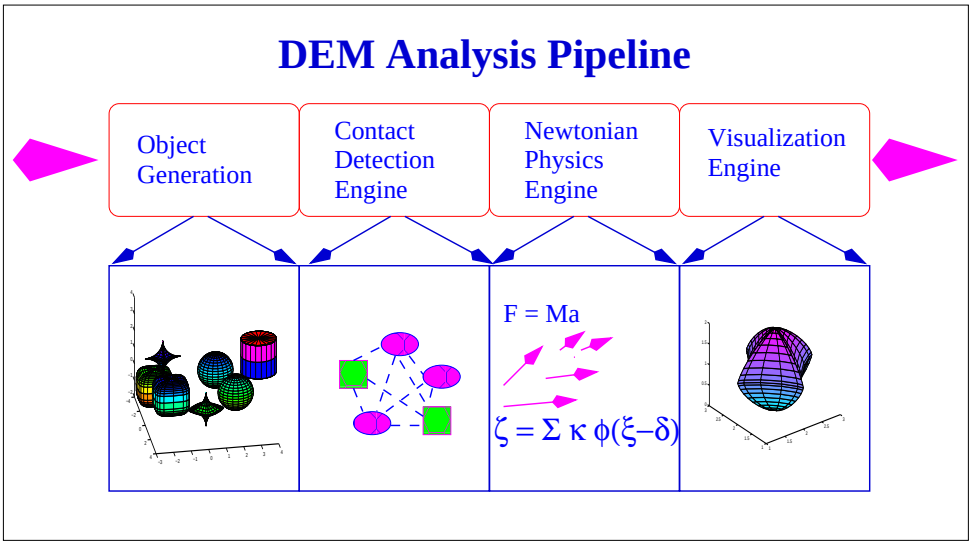

Figure 1: Analysis Pipeline

1.3 Definition of Spatial Sorting and Contact Resolution

A review of present contact detection schemes is contained in Williams, 1995 [WO95b] and Hogue, 1996 [**?**]. Contact detection consists of two main phases, **spatial sorting** and **contact resolution** (Table 2). The algorithms in each of these two phases have different orders of complexity.

CONTACT DETECTION	spatial search
	detect and resolve contact geometry

Table 2: Phases of Contact Detection

Spatial sorting refers to the identification of pairs of objects that are spatially local to one another at some instant in time. First the set of objects are spatially ordered using an appropriate sorting algorithm. During this phase we can use a very high level abstraction for the object geometry, such as a bounding sphere or bounding box. Next the sorted data set is *searched* for objects lying close to one another. The cost of spatial sorting is proportional to the number of objects, M. For systems where no assumptions can be made about the spatial coherence of the objects, sorting requires $O(M \log M)$ operations.

Contact resolution determines whether pairs of objects actually intersect one another and calculates a detailed description of the contact region (the geometry of the common

interface, depth of penetration or the area/volume of overlap). These details are then stored for later use in the calculation of the normal and frictional forces at the contact interface[1].

The order of algorithmic complexity for contact between two objects depends on the way in which the geometry is represented. For unstructered boundary representations, such as typical finite element, it is $O(N^2)$ [Wil88, CH89, MW88]. For schemes such as Binary Space Partitioning (BSP) trees, [Jr.94, MO93, MOW94], this cost can be reduced to $O(N \log N)$ for convex objects. We note that in order to judge the actual efficiency of an algorithm the theoretical estimates of complexity should be accompanied by timings or operation counts for a single check of a point on one body against another body.

1.3.1 Point Classification

Consider now a typical two dimensional finite element representation (Figure 2). To test if a point lies inside a polygonal region it must be checked against *every* edge of the boundary. A simple analogy helps to show why this number of checks are necessary. Consider a square room with one-way mirrors on each wall and assume that the reflective face of each mirror is on the inside of the room. If a person is standing *inside* the room they can confirm this fact by checking for a reflection of themselves when they look to the left (mirror 1), in front (mirror 2), to the right (mirror 3), and behind (mirror 4). This idea is shown in Figure 2.a. Note that it is necessary to see a reflection in *all* mirrors to guarantee that they are inside. If the person is outside the room, say below the lower righthand corner they will only see a reflection of themselves in mirrors 1 and 2. In this case the *lack* of a reflection in *either* mirror 3 or 4 is sufficient to indicate that the viewer is *outside* the room.

Mathematically the inside-outside tests are performed by replacing the inward facing mirrors with inward pointing edge normals. For example, consider the polygonal region shown in Figure 2.b, and the point $\vec{R}$, both described with respect to a common coordinate system. To test if the point $\vec{R}$ lies in the interior, the following construction is used. We note that any point on the extended line $\vec{P}\vec{Q}$ is given by the vector equation $\vec{X}.\vec{n} = d_0$ where $d_0 = \vec{P}.\vec{n}$, the perpendicular distance from the origin to the line.. Now if $\vec{R}.\vec{n} = d_1 < d_0$ then the point is inside, and if $\vec{R}.\vec{n} = d_1 > d_0$ then the point is outside. To confirm that $\vec{R}$ is inside the complete region it is necessary to perform the same construction with *all* of the other edges describing the boundary of the region.

The same requirement extends to the classification of lines against convex polygons. Finding that one edge intersects the boundary of the other object simply confirms that contact might have occurred but is not sufficient to fully determine what portion of the line is contained.

The main point to note is that while the component operation of the classification test is relatively cheap, the cumulative cost of the *all-to-all* requirement becomes the computational bottleneck in pairwise contact checking. Two polygonal objects represented with 100 boundary edges would require 20,000 edge classification tests in the worst case.

[1] The calculation of the contact forces can be associated either with the contact detection or with the *physics* stage of the analysis pipeline and is not addressed in this paper. Typically the penalty function method for force generation requires less than 4% of the total computational effort.

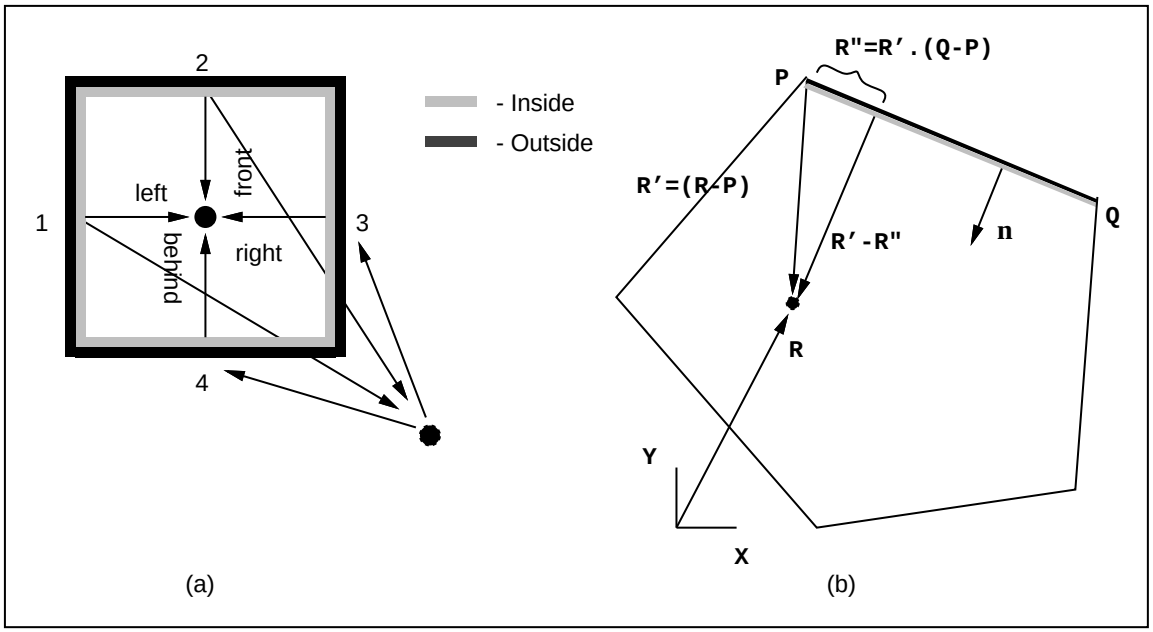

Figure 2: Point Classification

1.4 Implicit Functions and Superquadrics

In order to avoid the *all-to-all* problem of unstructured representations, we now investigate implicit functional representations of geometry. While the boundary representation breaks the surface of each body into facets or patches and interpolates within the patch, the implicit representation, represents the whole geometry as a single function, eg $f(x, y) = 0$ (where for example $f = x^2 + y^2 - R^2 = 0$). This representation is popular in DEM for analysis of disks and spheres because of the speed with which intersections can be checked and the compactness of the representation [CS79, TM92, TP92]. It has been extended to two dimensional ellipses by Ting [TKMR93a, TKMR93b], to three dimensional ellipsoids by Ng [NL93, NF95a, NF95b].

The implicit representation has been generalized to a wider range of two and three dimensional shapes by this author and others using superquadrics [Bar81, PW89, WP89, WP92]. In the case of superquadrics the intersection check relies on the sampling of the surface at discrete points. Each point on the superquadric of one body is checked against the implicit function representing the other body. The order of the contact algorithm is then $O(N)$ where N is the number of points per body. However, the calculation of the *inside-outside* function and the surface normals (Figure 3), is relatively expensive, involving trigonometric and exponential functions.

While the order of the complexity gives important information about how the algorithm scales, it does not take into account the number of calculations required for each check. As we have noted, for ellipses and superquadrics the calculation of the *inside-outside* function is expensive. Similarly, the root finding for quadratic implicit equations (eg. ellipses) is also expensive.

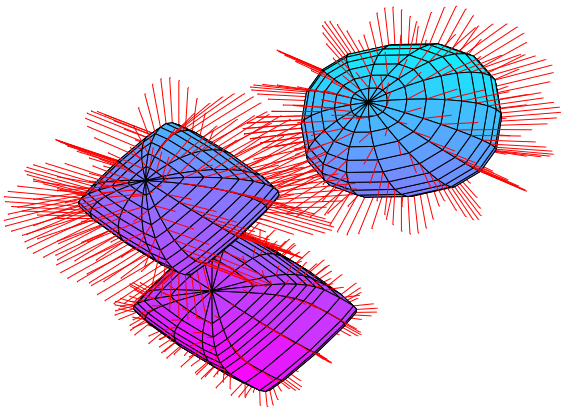

Figure 3: Intersection Checks Using Superquadric Normals

1.4.1 The Ideas of Discrete Functions

In order to combat such expensive calculations, look-up tables were adopted by Pentland and Williams [WP92] in which the value of the implicit function was precalculated for a cloud of points around the object. This led to more efficient algorithms for superquadrics.

The formulization of the idea of a discrete version of the implicit function was developed by O'Connor et al. [WO95a] and used to generate what is now called the Discrete Function Representation or DFR. In Section 3 we describe the DFR algorithm and its associated hierarchical geometric representation for 3-D objects. We believe this approach is both efficient and of general applicability in DEM simulations. DFR based contact detection between a pair of objects theoretically exhibits an upper bound of $O(N)$ computational complexity, where N is the number of surface points used to represent each object. In practice the scheme yields a complexity dependent on the number of points involved in the overlap which is empirically closer to $O(\sqrt{N})$.

In Section 2 we describe the spatial sorting phase of contact detection. In Section 3 we describe the details of contact resolution and in Sections 4 and 5 the DFR data structure and algorithm followed by timings and algorithmic complexity measures.

2 Spatial Sorting

In the most general situation no *a priori* information is known about the shape and spatial distribution of the objects. To ensure a comprehensive examination of the system requires that each object is checked against every other object. For a system containing M objects, the exhaustive enumeration of the pairings is said to require *of the order of* M^2 contact tests, denoted $O(M^2)$. The algorithm then has a *worst case* performance requiring a number of operations that asymptotically approaches M^2 operations, as M becomes very large. In this section we describe the heapsort algorithm which we use to reduce the complexity of the *object to object* checks from $O(M^2)$ to $O(MlogM)$.

In general it is necessary to reason about a system of bodies (an ensemble) that is

evolving with time. The efficiency of almost any contact algorithm is improved if one assumes 1) *a priori* knowledge of the evolution of the ensemble geometry or, 2) that there is coherence of the configuration between time steps. These assumptions allow one to predict subsequent geometries based on the present ensemble state. For example, a rock mass under small strain will probably evolve so that any new contacts are formed only with neighboring objects where all neighbors can be defined from the initial configuration. In the case of an avalanche, the system will evolve erratically, and any coherence with the initial condition will be destroyed after a very short time.

2.1 Classifying Ensemble Evolution

Algorithms assuming *a priori* knowledge of the system evolution can be extremely effective but are limited in the range of problems which can be tackled. Below we give a brief list of some of the kinds of a priori knowledge which can give rise to fast problem specific algorithms.

Fixed Topology

If the objects do not vary their relative position, as is the case for FEM elements, then one can set up an object connectivity once at the start of the problem which needs no further modification during problem execution. The data structures associated with these algorithms tend to be simple arrays.

Slowly Varying Topology

If objects move only a small amount according to some metric then we can conclude that bodies can make new contacts only with near neighboring objects. We can then arrange for each object to keep track only of a limited number of objects as potential contactors. One possible assumption is that no object further away than $d = nstep * \Delta t * velocity_{max}$ can make contact with the object within the next $nstep$ time steps. In order to make such a scheme robust one must continuously monitor the characteristic of the system (in this case maximum velocity) on which the assumption depends.

Spatially Sparse Systems

If the density of bodies in the problem space is sparse such that many time steps ellapse between contacts, then it may be beneficial to project ahead the path of each body. Thus each body must reside within a cylinder or cone in space-time. By intersecting these cones the position and time of the next collision can be calculated and the system can be updated to take advantage of this, i.e. we may proceed directly to the time station of the next collision without checking for new collisions at intermediate time steps. This scheme has been used successfully by Dworkin [Dwo94] for animation applications.

2.1.1 Exhaustive Spatial Sorting Schemes

Contact schemes which make no a priori assumptions about the problem evolution and reason based only on the present state of the geometry are called *exhaustive*. Exhaustive schemes are more general and robust, and therefore potentially slower than non-exhaustive schemes. The DFR contact resolution algorithm which we present here is combined with a **heapsort** spatial sorting algorithm which is exhaustive (See Figure 4). The DFR algorithm

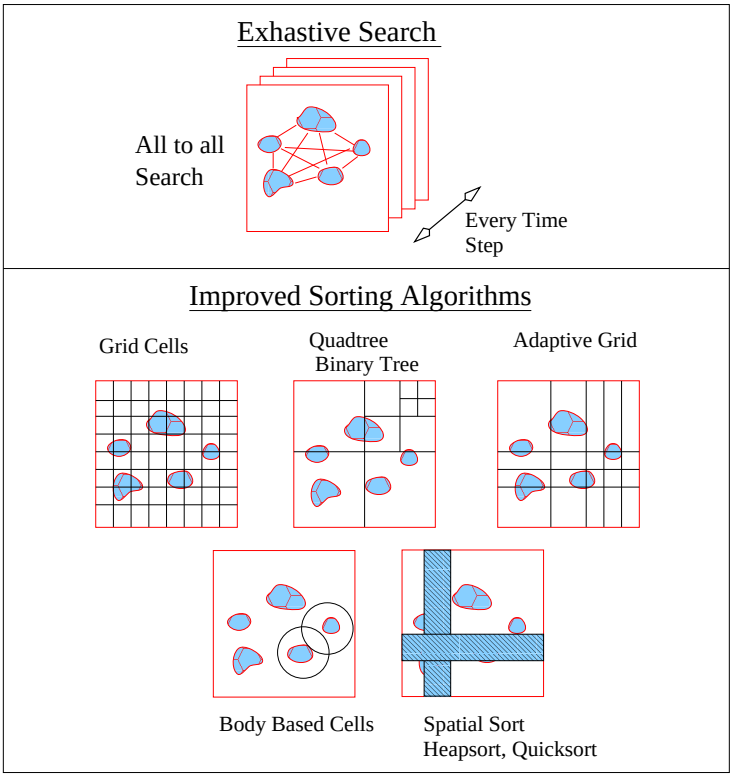

Figure 4: Illustration of Various Sorting Strategies

can also be combined with faster but more restrictive non-exhaustive schemes if we assume a priori knowledge of system evolution, as described above.

2.2 Classifying Spatial Sorting Schemes

Figure 4 illustrates the various schemes for the spatial ordering of objects. All these schemes can be used in either exhaustive or non-exhausive search algorithms. The usual method of implementing a non-exhausive search scheme is to assume that the information in the sorting arrays remains valid over a number of time steps. Non-exhaustive schemes are what might be termed *iteresting* but dangerous. They give excellent performance, and algorithmic complexities, but are liable to *blow up* on any new problem for which they have not been tuned.

Grid Subdivision The grid subdivision method uniformly discretizes the simulation volume into recti-linear cells, each grid cell enclosing zero or more objects. Objects are associated with each cell based on their coordinates, typically based on the coordinate extents of a bounding hull or bounding sphere enclosing the object. In order to account for objects near the cell boundary it is necessary to implement a buffer zone so that such object may be assigned to more than one cell. The performance of this method depends of a tradeoff between cell size and the number of objects per cell. In order to gain significant advantage from this stratagem, the objects should be fairly evenly distributed amongst the cells. If all the objects are clustered in one cell then no advantage acrues from this method. The method is therefore sensitive to the homogeneity of the spatial distribution of objects.

Adaptive Grid Methods Adaptive cell methods can be adopted to avoid the problems associated with spatial heterogeneity, albeit at the extra cost of managing multiple cell dimensions. In this technique the simulation volume is subdivided by cutting planes parallel to the principal Cartesian planes. This is done such that there is the approximately the same number of objects on either side of the cutting plane. However, the scheme suffers where the state of the ensemble evolves from a heterogeneous to homogeneous distribution, where the work involved in maintaining uniformity will exceed that of the static uniform grid approach.

Octree Method The *Octree* method is conceptually one of the most elegant technique for spatial reasoning. Derived from extending a binary sort/search technique in 2D, it provides a flexible and general algorithm. The technique again involves treating the simulation volume as consisting of rectilinear cells, however, in this case only those cells which contain objects are maintained in the tree structure.

It should be noted that the search is heavily dependent on how the octree is first constructed. By minimizing the depth, hence, balancing the branches of the tree, the search can be minimized. Knuth suggests that insertion into the tree should be done in a random fashion, to ensure a relatively uniform distribution of objects, that is, a balanced tree. The time to create the octree is of order $O(M \log M)$ and the time to search this tree is also of order $O(M \log M)$ (for a balanced tree and where cell size is small compared to the full space).

Body Based Cells

Another alternative is to base the center of the search cell on the object's centroid, so that each object has a cell surrounding it. Any objects lying within the cell are deemed to be potential contacts and are stored in a *neighbor list*. The cells can be either taken as circular or retilinear. The multipole algorithm of Greengard and Rohklin, [Gre88], make extensive use of a similar approach in conjunction with octrees, and is applied to molecular dynamics problems. This method is often coupled with a non-exhastive update scheme.

Spatial Heapsort

Spatial sorting provides an improvement to the body based cell strategy by improving the assignment objects to cells. Heapsort is one of several algorithms which can be used to sort the objects by some key into an ordered list. In the case of spatial sorting the sort key is the coordinate along one or more global axes. For speed of search the list is stored in a tree structure, such as a binary tree. The basis for this method is a well described sorting algorithm, [Knu73, TV88, Sed90], and has been applied to smooth particle hydrodynamics (SPH) problems by Swegle, [Swe93]. We describe below the **Heapsort** algorithm and the extensions we have made for sorting two and three dimensional objects.

2.3 Spatial Sorting Using Heapsort

The method chosen to manage the spatial sorting of the objects in the simulation is called *Spatial Heapsort*. The basis for this method is to create lists of objects sorted by increasing ordinates along the principal axes of the simulation volume. Heapsort is known to perform in a time of order $O(M \log M)$ when applied to sorting an unordered collection of m objects. The algorithm was chosen for the following reasons:

- Implementation is straightforward in software, with no special data structures necessary.
- The behavior of the algorithm is insensitive to the spatial distribution of the objects to be sorted (i.e. the relative clustering/grouping of the objects in the simulation volume is unimportant). [2]
- Searching for spatially localized sub-sets of the objects is straightforward and, with suitable choice of sort key, can be performed efficiently.
- The space requirements are typically small, being of order $O(M)$ *words* of memory.

2.4 Comparision of Heapsort and Octree Algorithms

By comparing the heapsort and the octree method it was found that:

1. Octree implementation and the related utilities to manipulate the tree required considerably more development effort.
2. The space requirements of octree were approximately 10 times that of the spatial heapsort.
3. The runtime recursion overheads that the octree method imposed were quantifiable larger.
4. For octreees the average search time can be significantly longer and is also directly dependent on the distribution of the objects. Heapsort does not suffer from this dependency.

3 Contact Resolution

The result of the spatial sorting algorithm are pairs of bodies which possibly intersect. We now focus on the problem of deciding 1) if they intersect and if they do, 2) what are the details of the intersection.

We use a hierarchical representation of the geometry to decrease the computational work. By representing each object using a simple shape, such as its bounding box, or its bounding sphere, we can implement simple checks to cull the number of possible contacts.

3.1 A Multilevel Representation For Contact Resolution

Our goal is to represent an object at multiple levels of detail and then perform contact resolution at the appropriate level of approximation. The scheme presented in this paper uses 4 levels of representation (shown in Figure 5):

1. Bounding Sphere $\mathcal{S}$: An orientation invariant approximation to the region $\mathcal{R}$. The radius of the sphere is taken to be the maximum extent $\mathcal{R}$ measured from an arbitrary center point (typically the center of volume of $\mathcal{R}$).

[2] The Quicksort algorithm, [Knu73], while known to perform faster in many implementations and applications, degrades to an $O(M^2)$ algorithm when applied to existing ordered sets.

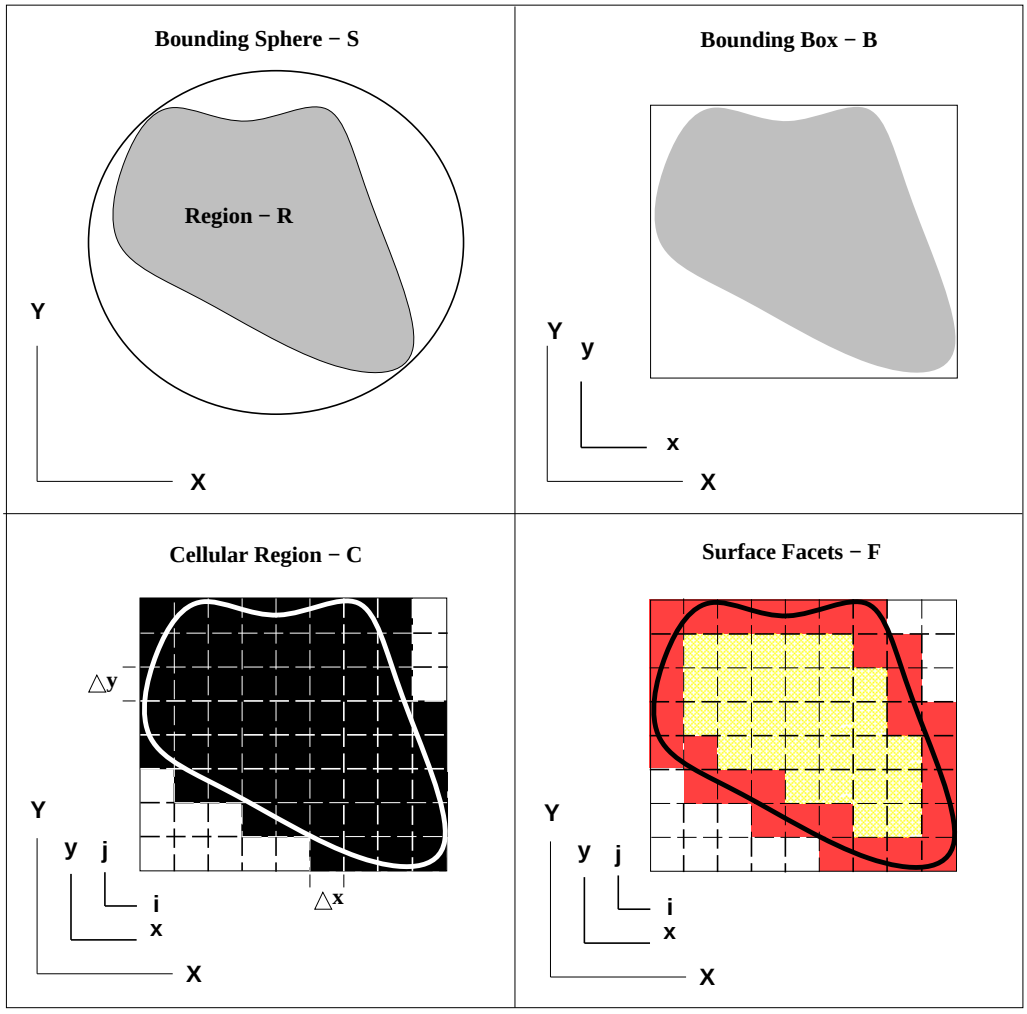

Figure 5: Multiple Levels of Respresentation

2. Bounding Box $\mathcal{B}$: A rectilinear approximation to $\mathcal{R}$. $\mathcal{B}$ is alligned with a body centered coordinate system (x, y, z). The dimensions of $\mathcal{B}$ are calculated from the max/min spatial extents of $\mathcal{R}$ along each of the local coordinate axes. While no longer orientation invariant, $\mathcal{B}$ better captures the aspect ratio of $\mathcal{R}$ giving a tighter bounding region.

3. Cellular Region $\mathcal{C}$: In this representation we discretize the bounding box $\mathcal{B}$ into a number of cells of fixed dimension $(\triangle x, \triangle y, \triangle z)$, along the respective coordinate axes. Cells are indexed locally using discrete coordinates (i, j, k), with $\triangle x, \triangle y, \triangle z$ as the units along each axis. Any point $P(x, y, z)$ lying inside the region $\mathcal{B}$ can be uniquely associated with a cell $\mathcal{C}(i, j, k)$ under the mapping:

$$u = \lfloor \frac{q}{\triangle q} \rfloor, \quad q \in \{x, y, z\}, \; u \in \{i, j, k\} \tag{1}$$

Each cell in $\mathcal{C}$ is designated as either *containing* part of of $\mathcal{R}$ or lying completely outside. In Figure 5 the shaded cells are considered to contain the region $\mathcal{R}$

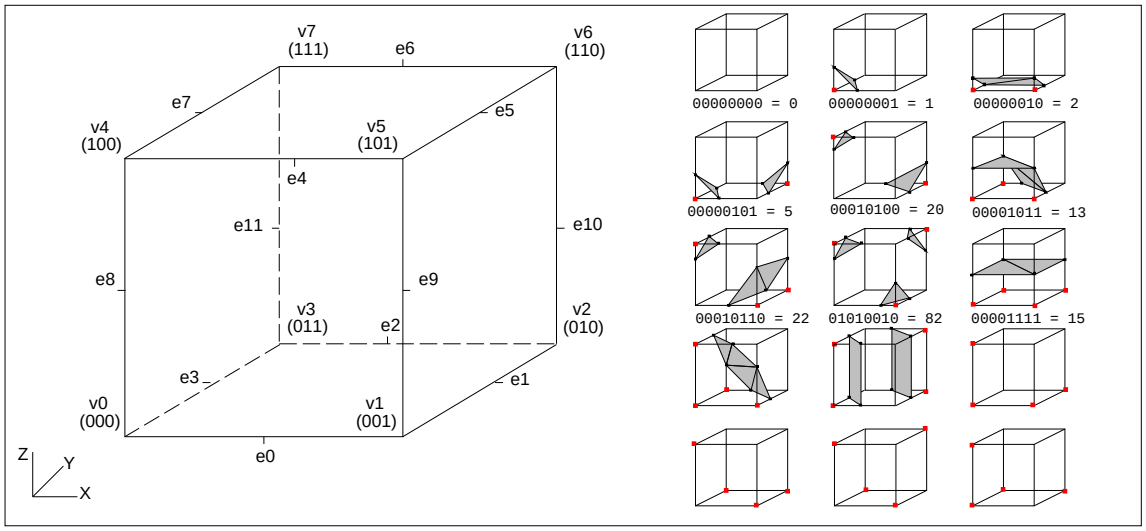

Figure 6: Voxel Labeling and Tesselation Topologies

4. Surface Facets $\mathcal{F}$: This is the highest resolution representation for $\mathcal{R}$. Within each cell $\mathcal{C}(i,j,k)$, the boundary is of $\mathcal{R}$ is approximated by a set of facets $\mathcal{F}(i,j,k)$ 6. Facets are lines in 2D and triangles in 3D. The summation of the facets $\sum_{i,j,k}\mathcal{F}$ over the domain bounded by $\mathcal{B}$ is then a surface tesselation of $\mathcal{R}$.

4 Discrete Function Representation in 3D

In this section the *discrete function representation* (DFR) is developed. Consider an implicit function written in the form $f(x,y,z) = 0$. We now form a discrete sampling of this function on a regular lattice such that the value of the function at the lattice point (x_i, y_j, z_k) is given by $f(i,j,k) = f(x_i, y_j, z_k)$. This discretized function can be viewed as a sampled scalar potential field throughout space. We note that the technique of volume visualization which was derived as a means of visualizing tomographic data from medical MRI scans produces exactly this kind of data. In the case of MRI the scalar potential is a measure of tissue density.

The discretely sampled function $f(i,j,k)$ can be derived in several ways from the original continuous function. For clarity we consider the case of a one dimensional function.

$$f(x) = \sum_k f(k)\phi(x-k); \qquad f(k) = \int f(x)\phi(x-k)dx \tag{2}$$

The function $f(x)$ is projected onto the space spanned by the basis functions $\phi(x-k)$. One choice of basis function which gives a piecewise constant approximation is the Haar

function.

$$\phi(x) = \{ \begin{array}{ll} 1 & 0 \leq x < 1 \\ 0 & otherwise. \end{array} \quad (3)$$

In the case of volume visualization the function is not explicitly defined and its use is implicitly assumed. Other choices such as the following are possible:

- $\phi(x) = e^{i\omega x}$ - Fourier
- $\phi(x) = e^{-x^2/a}$ - Gaussian (also known as *splatting*)
- $\phi(x)$ - any wavelet scaling function.

We note that the choice of basis function leads to many possibilities which have not been fully explored in the context of object representation. For example, wavelets lead naturally to a hierarchical multiscale representation of geometry.

Our approach here is to use a Haar function supplemented at the lowest level by a bit code which determines the position and orientation of a planar surface through the voxel.

4.1 The DFR Data Structure for 3D

This section describes the structure of the 3D-DFR data set and how it is generated from input data as part of the tesselation process. The function of the data structure is to produce an efficient mapping between an arbitrary point in the problem space and the geometric properties of that point. If the point is associated with an object we also seek relationships between the point and other points on the object surface.

At the coarsest level the 3D-DFR corresponds to a cage of cells that completely enclose a 3D surface. The 3D-DFR data structure uses only the subset of the cells that intersect the surface. We refer to the cage of cells as the *discrete bounding hull* (DBH) of the data set. For each body we use the sampling grid as a local coordinate system with the cell side as the metric. We use the following nomenclature:

cell - the fundamental building block in discrete space.
prism - a line of cells or 1D section of a **slice**.
slice - a 2D region of colinear prisms i.e. a plane of cells

We call the axis perpendicular to the slice plane the **slice axis**, the axis perpendicular to the prism line the **prism axis**, and the remaining axis the **cell axis**. The discrete bounding hull of a sphere represented in this manner is shown in Figure 7.
The 3D-DFR data structure is a ragged 3D array of cell indices called the **stencil** and a set of ragged arrays containing *offset/run-length* (ORL) pairs to describe the **stencil** layout. This structure is shown in Figure 8. The indices contained in the **stencil** refer to pointers to the data structure of the tessellated surface (not shown) which is stored separately.
The offset/run-length (ORL) pair arrays are defined as follows:

1. **slice map** - a single ORL pair. The first element of the ORL pair is the offset in **cell** units to the first **slice** of the DBH from the origin of the discrete coordinate system. The second element is the run-length (the number of slices) that follow.
2. **prism map** - a 1D array of ORL pairs. The length of the array corresponds to the number of slices. The first element in each ORL pair denotes the offset to the first **prism** of the respective slice. The second element specifies the number of **prism**s in that slice.

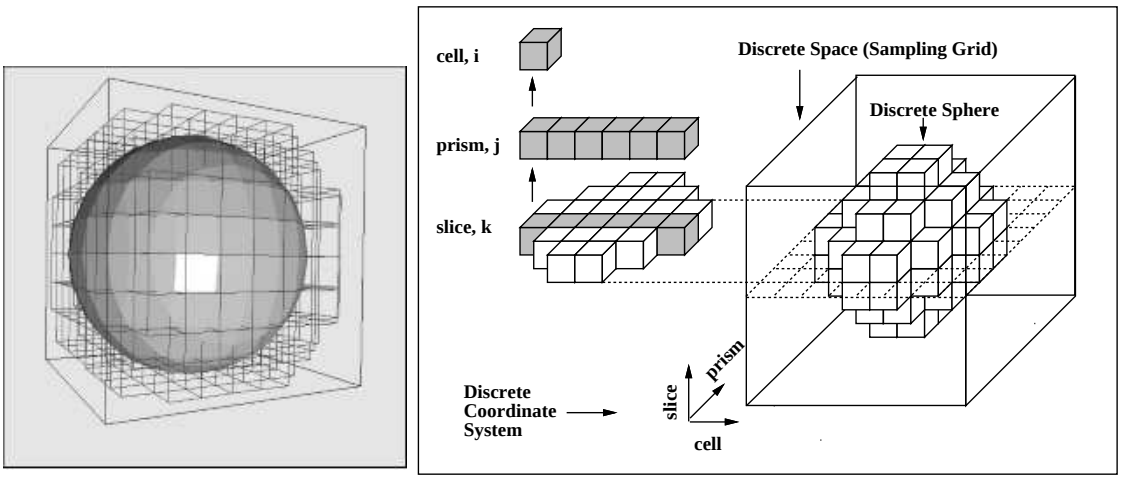

Figure 7: Example Object and 3D-DFR Elements

3. **cell map** - a ragged 2D array of ORL pairs aligned with the slice and **prism** axes. Each element of the array is indexed by the offset in **cell** units to the current slice and **prism**. The first element of each ORL pair is the offset to the start **cell** of that **prism**. The second entry is the number of cells in that **prism**.

4. **stencil** - a ragged 3D array of *cell descriptor table* indices that demarcate the region contained by the DBH. The ragged array topology is configured to be equivalent to the aggregation of cells making up the DBH of the surface. Internally, the ragged array is described by the ORL maps described in Items 1, 2 and 3 above.

The memory layout for the data structure is *parallel* to the discrete coordinate system. There exists a one-to-one topological mapping such that each cell maps to an unique location in the 3D **stencil**.

Consider the example discrete sphere of Figure 8. The ORL arrays describe the layout of the 3D **stencil** as being a discrete sphere in address space. This analogue allows the tessellation algorithm to easily identify the cells in which to store facet information. The tessellation process stores an index [3] in each field of the 3D **stencil**, which points to an entry in the *cell descriptor table*. This entry in the *cell descriptor table* describes the surface facets of the cell.

The mapping from the local to the discrete coordinate systems is achieved by a translation and a scaling. The scaling factors are the ratios of the sampling grid dimensions

[3]Native language pointers or integer indices are most efficient but space considerations may dictate that single word or bitwise indices should be used. We use integer indices for simplicity

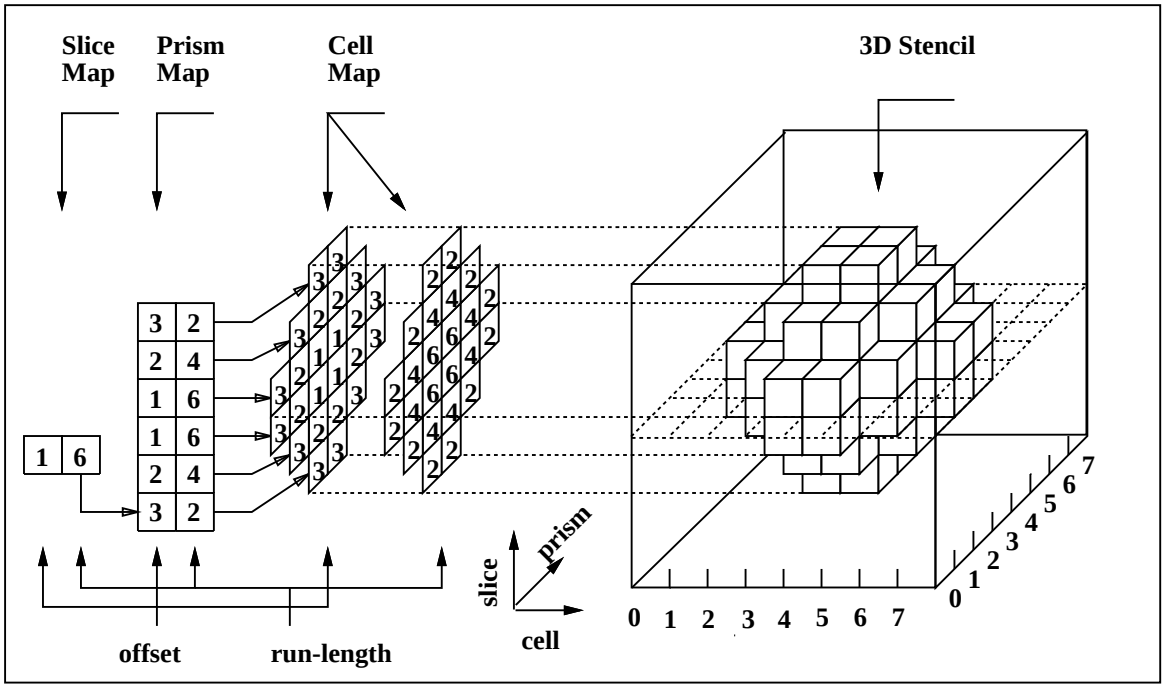

Figure 8: 3D Ragged Array Storage for Discrete Sphere

and the resolution of a **cell**, measured along each axis of the local coordinate system. The translation terms are the offsets of the origin of the local coordinate system to the discrete coordinate system. This relationship allows 3D coordinates in the local coordinate system to be mapped to a coordinate triple in the discrete space. Once transformed, the discrete space coordinates can be treated directly as indices into the ORL arrays and in turn to a **stencil** entry.

In order to distinguish the cells referenced by the 3D **stencil** entries, we classify them into 3 groups: *boundary* (surface) cells, *internal* (completely enclosed) cells and *external* (empty) cells. The first grouping are cells that correspond to legitimate **stencil** index entries that point into the *cell descriptor table*. A legitimate **stencil** index refers to a **cell** with some portion of the surface geometry passing through it. We differentiate the second and third groups (internal, external cells) using reserved index values for their **stencil** values. Cells that these indices refer to, act as spatial placeholders without requiring a cell description to be maintained in the *cell descriptor table*. This is possible because a cell's dimensions are uniform over the sampling grid and therefore completely empty or completely enclosed cells will have the same geometric description. We will also see that many of the external (empty) cells that occur can be discarded as each **prism** is completely processed. We show this next in a cell classification example for a 2D region.

A 2D example with a detailed view of cell classification for a *discrete bounding hull* is shown in Figure 9. The shaded area represents the closed region which we wish to enclose in a *discrete bounding hull*. The hull is built incrementally as the grid of cells is processed in scan line order (i.e. cell by cell along a prism, prism by prism over the 2D section).

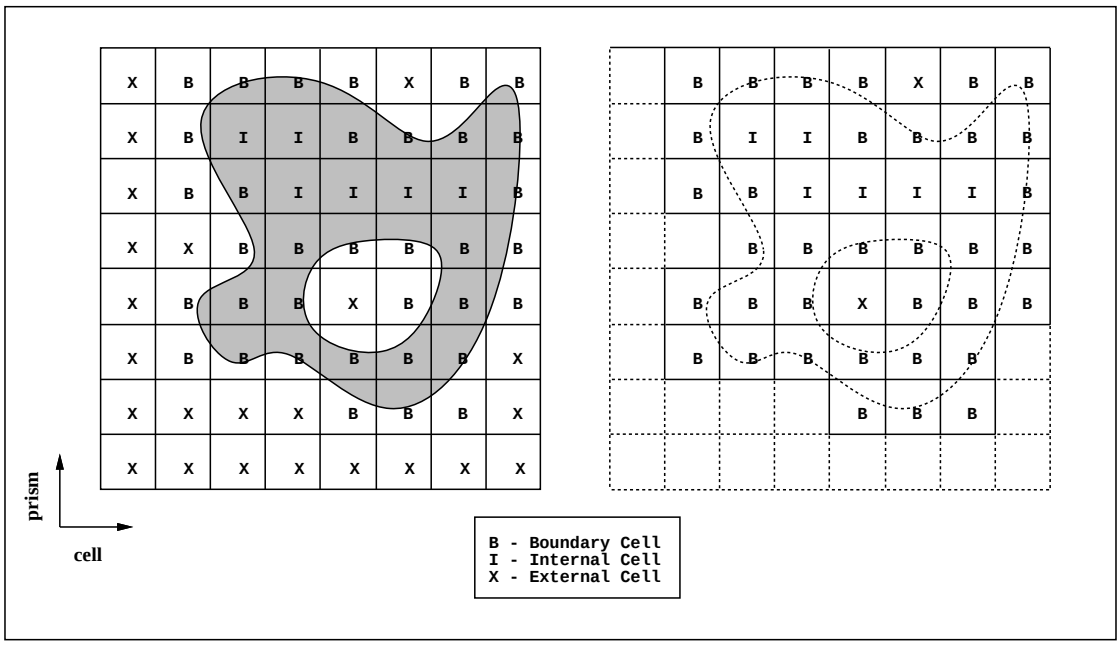

Figure 9: a) Cell Code Assignment, b) Discrete Bounding Hull (2D)

Each cell is assigned a cell code (corresponding to an index) with the following meanings:

B - a cell that straddles the boundary of the region.
I - a cell completely internal to the region.
X - a cell lying completely external to the region.

Trailing external cells (**X**) must be retained for the complete traversal of a prism. At the end of each pass, trailing external cells can be trimmed from the list of cells retained[4]. We see this in Figure 9.b which shows the region superimposed on the grid after discarding external cells not required for the bounding hull. We note that the discrete bounding hull need not be strictly convex[5], rather that it encapsulates the entire region (including holes and concavities).

For 3D objects the complete DFR construction consists of the tessellation operation previously shown in Figures 5, and 6, and the simultaneous construction of the discrete bounding hull and internal DFR storage as shown in Figure 4.1. Figure 4.1.a shows the DBH of the spherical surface. As the surface is convex, all visible DBH cells will be boundary (**B**) cells. Figure 4.1.b shows the wireframe Discrete Bounding Hull (DBH) with

[4] *Leading* external cells (**X**) that occur before the first boundary cell is found, are simply ignored.

[5] Should the region be convex, the boundary cells describe a convex hull and no internal cells would actually need to be retained. For generality we chose to include the possibility of non-convex closed regions.

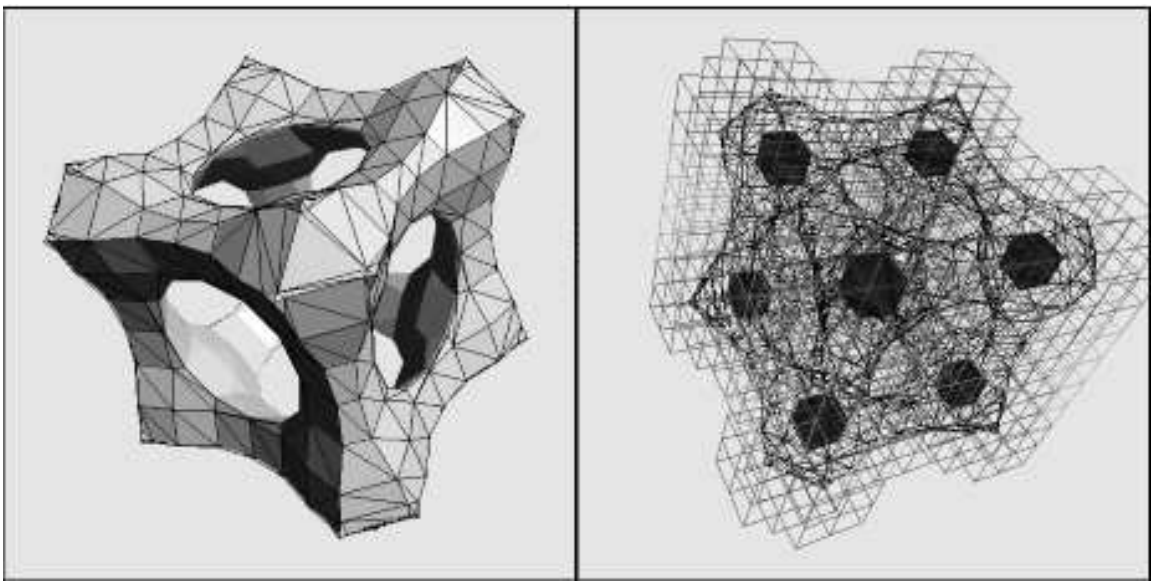

Figure 10: (a)Discrete Bounding Hull, (b) Internal Cells With Embedded Mesh

internal (**I**) cells filled, viewed through a wireframe outline of the isosurface.

5 DFR Based Contact Detection

A general algorithm for contact detection contains some or all of the following tasks. Initial steps seek to further cull nearby objects from consideration using a relatively crude but fast bounding box test. Intermediate steps attempt to reduce the amount of local geometric detail to be analyzed. The ultimate steps examine the higher resolution details of the intersection. In 3D-DFR based contact detection we follow this general outline and show the hierarchy of steps in Figure 11. First we start with an overview of the algorithm.

5.1 Algorithm Description

The DFR contact detection scheme determines whether pairs of objects intersect (**contact detection**) and calculates a precise geometric description of the contact region (**contact resolution**). Each object maintains a reference frame describing the position and orientation of the object's local coordinate system. The local frame is in turn defined with respect to a fixed global reference frame. The geometric description of each object is expressed using the DFR scheme, in addition to a bounding box of the surface, both aligned with the local frame.

5.1.1 Zone Clipping

The first formal step in the contact detection process is to check for *bounding box* overlap. Since this operation occurs high up in the 3D-DFR contact detection hierarchy we expect

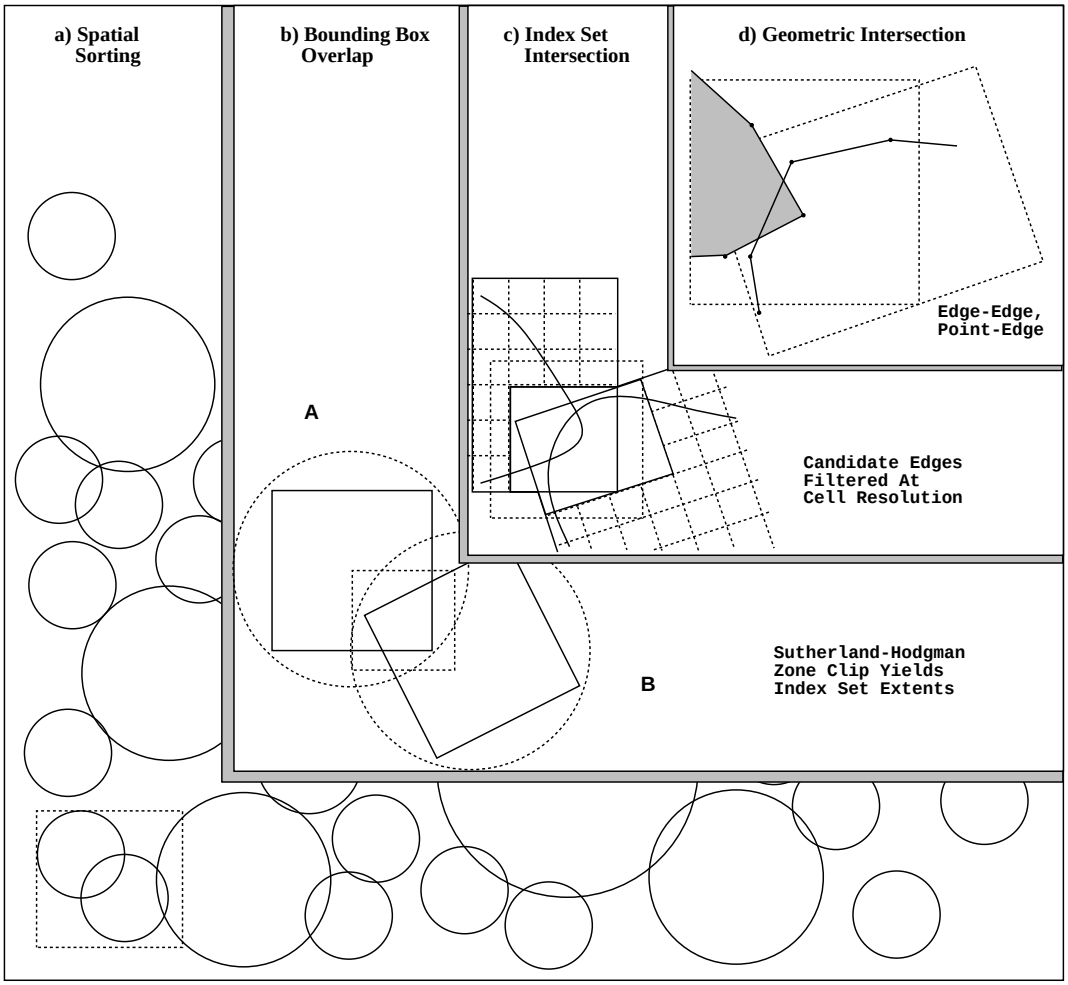

Figure 11: 3D-DFR Contact Detection as a Hierarchical Process

it to be performed relatively frequently. For this reason we use an optimized *Sutherland-Hodgman* polygon clipping algorithm, [Rog85], to determine the geometry of the overlap region between two bounding boxes. The optimization takes the form of hard coding the clipping operations to reflect that the clip volume is a rectangular parallelepiped, and that each polygon to be clipped to this region will be a rectangle. We show an example of the optimized algorithm for the 2D case in Figure 12.

The algorithm takes the polygon (P1, P2, P3, P4) and loads the end points into an *edge table* reflecting the connectivity of each edge of the input polygon. For the 2D example the input edges are (P1, P2), (P2, P3), (P3, P4), (P4, P1). The edges are then checked against the boundaries of the clip region (one boundary per iteration) by testing if an edge trivially crosses the extended boundary line and calculating the point of intersection when appropriate. Points that lie on the extended boundary or are potentially inside the clip region boundaries are cached in an intermediate buffer which is used to form the input *edge table* for the next iteration. For the first level of the input *edge table* the intermediate crossing points with the `Xlow` boundary are P2′, P3, P4, P4′. The points of intersection

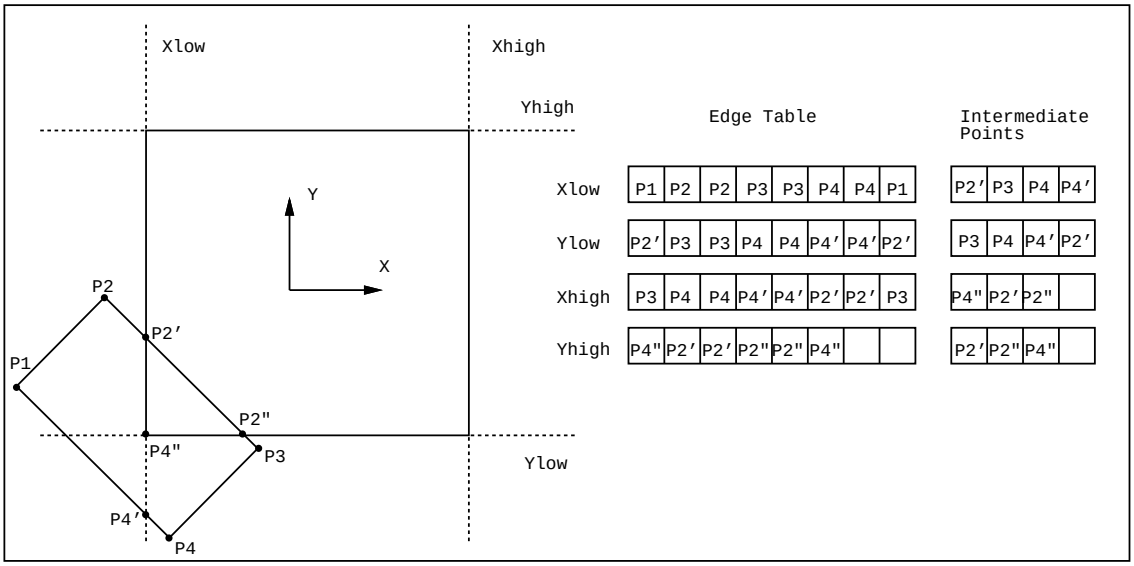

Figure 12: Zone from Sutherland-Hodgman Clipping

are buffered as intermediate points and then used as the input points to form the next level of the *edge table* which is (P2′, P3), (P3, P4), (P4, P4′), (P4′, P2′). We continue applying these steps with each subsequent *edge table* by comparing the edges with the remaining extended clip region boundaries. Finally a set (possibly empty) of points describing the edges of the overlap region are produced. For the example shown in Figure 12 the overlap region is described by the set of points P2′, P2″, P4″.

Where bounding box overlap occurs, the upper and lower bound extents of the overlap region parallel to the local coordinate axes are calculated for each object and stored as a pair of 3D vectors, i.e. $(x, y, z)_{min}, (x, y, z)_{max}$. We refer to each vector pair as a *zone* of the object and show a possible *zone* pair from a bounding box overlap test in Figure 13. The *zones* now bound the region of each object that needs to be further considered in the contact detection process.

5.1.2 Zone Index Sets

Next the *zones* are transformed into discrete coordinates. To map to the discrete coordinate system each coordinate is first scaled by dividing by the unit cell dimension in that direction and then taking the *floor* of the resulting value. In discrete space a coordinate triple corresponds to the location occupied by a finite cell, and so the discrete coordinates retain the property of enclosing the overlap region, but now at the discrete level. The transformed coordinates of the *zone* then delineate the extents of a set of index triples refered to as the *zone index set*. Geometrically the *zone index set* describes a localized region in discrete space which needs to be further checked for overlap.

The *zone index set* for a 2D overlap example is shown in Figure 14. The *zone boundaries* enclose the region of overlap between the two bounding boxes. The *zone index set* is then found by enumerating the cells that the *zone boundaries* contain. For this example the

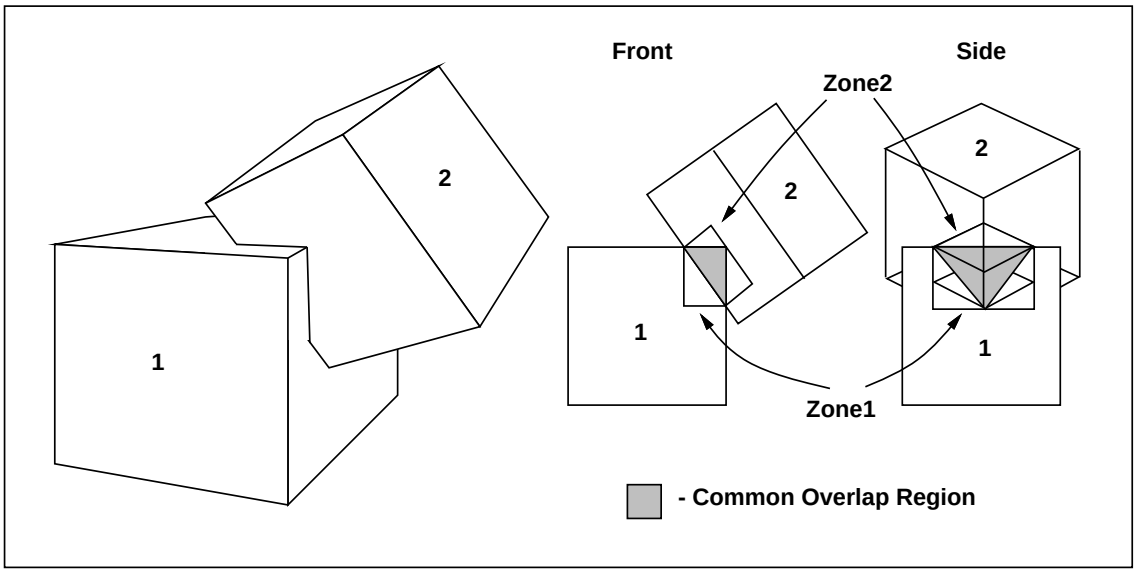

Figure 13: Zone From Bounding Box Overlap

zone index set for object **A** would be: (6,4), (6,5), (6,6), (6,7), (7,4), (7,5), (7,6), (7,7), and for object **B**: (0,0), (0,1), (0,2), (1,0), (1,1), (1,2).

5.1.3 Zone Index Set Filtering

We can further reduce the scope of the overlap tests by taking the intersection of the *zone index set* with the ORL arrays describing the DBH of the object. The *zone index set* intersection yields a subset of index triples to be retained for detailed checks, and the complement can be simply ignored. The set intersection takes the form of an iteration over the *zone index set*, checking each index triple for a corresponding entry in the ORL arrays. A matching entry is an index triple that refers to an existing ORL entry in the 2D **cell map**, and in turn to a legitimate **stencil** entry that points into the *cell descriptor table*. For legitimate **stencil** entries, the surface data for this **cell** is then extracted from the *cell descriptor table*.

Cells that pass this discrete scale filtering are then examined at the highest resolution, the surface facets obtained from the tessellation process. The checks at this final level are the necessary *point-edge* and *edge-edge* containment tests, [Wil88, Can88, BJ91].

5.1.4 Surface Intersection

The data extracted from the *cell descriptor table* corresponds to a portion of the *source* object's surface that we need to check against the *target* object's surface. The *source* surface data (edges) are next transformed into the *target* object's frame.

While neither end-point of a candidate edge necessarily lies inside the discrete space, the *edge* may cross through the space containing cells with valid ORL array entries. The general case of an edge passing through a portion of the *target* object DBH is shown in Figure 15. The edge (P1,P2) is to be checked against the *target* object i.e. the lower triangular prism. The *start* and *end* points of each candidate edge are clipped to the bounds

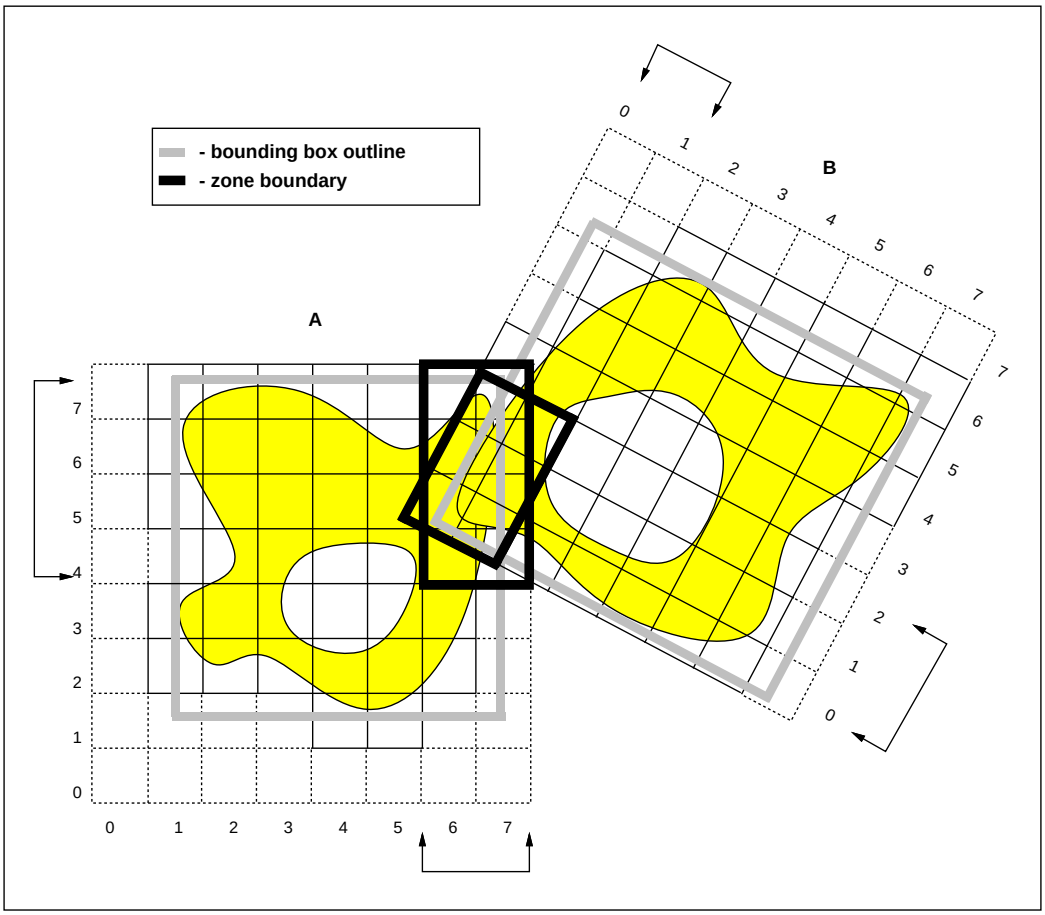

Figure 14: Zone Index Set From Clip Zone

of the *target* object, i.e. the bounding box of the target object's DBH. This classification is equivalent to clipping a line segment to a region (in this case a rectangular parallelpiped) to determine the visible portion therein. Having clipped the segment, we now have (possibly new) *start* and *end* points that lie on or inside the *target* object's discrete space and can be mapped to index triples in the *target* object's discrete coordinate system.

The index triples will either both refer to a single cell or individually to a pair of independent cells. Where a single cell is identified, it is first checked for a corresponding ORL array entry to verify some portion of the target object's surface exists there. This being the case, the edge can be directly tested for *edge-edge* intersection with the surface facets in that cell.

If the index triples map to different cells then we are required to check the *edge* against both of these and some number of the intermediate cells between them. For the example *edge-edge* contact situation in Figure 15 the edge (P1, P2) spans several cells. The problem here is to determine which cells the edge passes through before attempting to perform the individual *edge-edge* tests. On the right of Figure 15 we show a plan view of the *edge-edge* contact with the tessellated geometry (triangular facets) outlined. The cells the line segment (P1, P2) passes through are shaded for clarity.

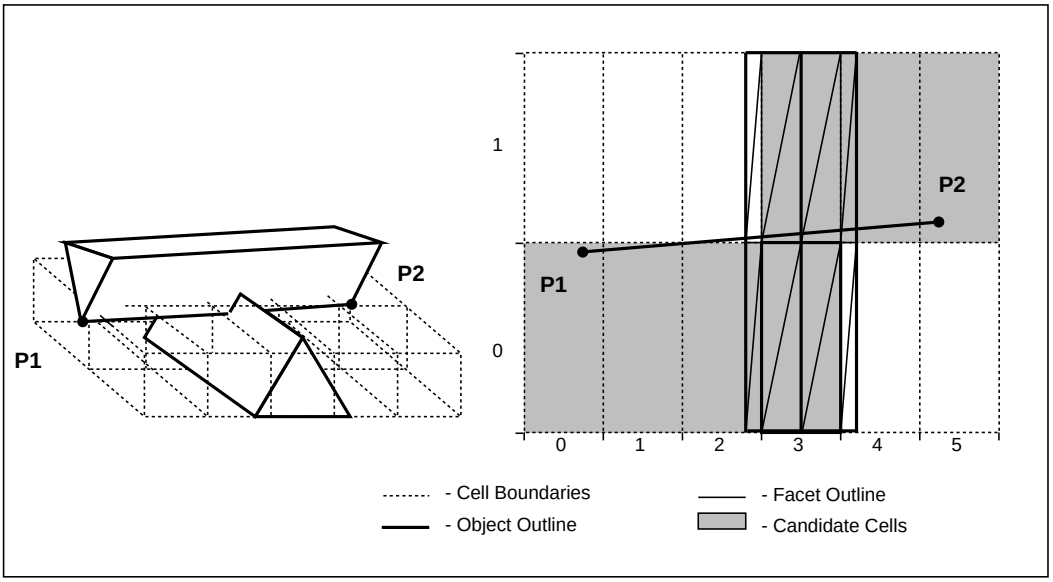

Figure 15: Discrete Edge - Edge Contact

We choose one of the end-points as a starting point and map this point to a discrete space cell. Next clip the edge (P1, P2) to that cell's boundary using the *Sutherland-Cohen* algorithm, [Rog85]. The clipping identifies an intermediate end-point for the edge that is also in the current cell. To complete the *edge-edge* test in the current cell, the intermediate line segment is checked against the portion of the *target* object's geometry contained in the **cell**.

The scan then continues on to the next cell that the original *edge* (P1, P2) spans. The next cell to visit is identified from the calculation of the previous intermediate end-point. This simultaneously identified which face the edge exits from the cell, and therefore the face of the neighboring cell it will next enter. Once determined, we can apply the previous two steps to all remaining cells spanned by the edge.

This completes the algorithm overview for 3D-DFR contact detection algorithm. We next describe the algorithm in terms of a set of operations more ammenable to implementation. Following this we detail some of the operations and transformations most easily expressed in a mathematical manner.

5.2 Algorithm Steps

The steps of the 3D-DFR contact algorithm are:

1. Transform a copy of the bounding box of each object into frame of the other object. Intersect each transformed bounding box with each local bounding box. If intersection points occur calculate the *zone* vector pair description and go to the next step, otherwise exit.
2. Transform the *zone* coordinates into the discrete coordinate system of the object to obtain *zone index set*. The local coordinates to discrete space coordinates transfor-

mation are calculated using:

$$w_u = \lfloor d_i + l_i/c_i \rfloor \quad (4)$$

These are the indices of the cells that bound a *zone* of candidate *source* cells to be checked against the *target* edges.

3. Perform cell index filtering by iterating over the *zone index set*. For each cell specified by the *index set* triples:

   (a) Map discrete space coordinates of cell (w_u) to ORL indices (ORL_u) using:

$$\begin{aligned} ORL_{slice} &= d_{slice} - o_{slice} \\ ORL_{prism} &= d_{prism} - (o_{prism})_{slice} \\ ORL_{cell} &= d_{cell} - ((o_{cell})_{prism})_{slice} \end{aligned} \quad (5)$$

   i - local (real) axes x, y, z
   u - discrete axes $cell, prism, slice$
   w_u - discrete ordinate along discrete axis u.
   d_i - offset to local origin from discrete space origin along local axis i
   c_i - cell dimension parallel to axis i.
   l_i - ordinates of point in local space along axis i.
   $\lfloor \rfloor$ - denotes the *floor* of a real value.
   o_u - offset to start cell along discrete axis u

   (b) If the index triple maps to an ORL index with a valid *cell descriptor table* entry then continue to the next step, otherwise exit.

   (c) For each edge in the referenced cell of the *source* object's geometry:

      i. Transform the edge into the frame of the *target* object.
      ii. Clip the edge to each *target* cell that it spans.
      iii. For each intermediate edge:
         A. Map real space coordinates of edge to cell coordinates using Eqn. 4.
         B. If the cell coordinate matches a valid ORL index triple then perform the geometric intersection operations, otherwise exit.

5.3 Example Geometries

The DFR algorithm can be applied to complex geometries including objects containing holes. Figure 16 shows a single DFR object of a porous body containing tortuous holes.

5.4 Point Classification Timing Tests

A series of point classification tests were timed for both the DFR scheme and the Cyrus-Beck algorithm. The Cyrus-Beck algorithm is the 3D extension of the 2D case. The tests check whether a point is contained inside a bounded region represented using an unstructured list of polygonal facets (Cyrus-Beck), and then for the DFR scheme. Each test is applied for 10,000 iterations and the elapsed time averaged to give a mean time of execution. The timing results for polyhedra containing various numbers of triangular facets are shown in Table refpt-class-times. $T_{Cyrus-Beck}$ is the average time required to

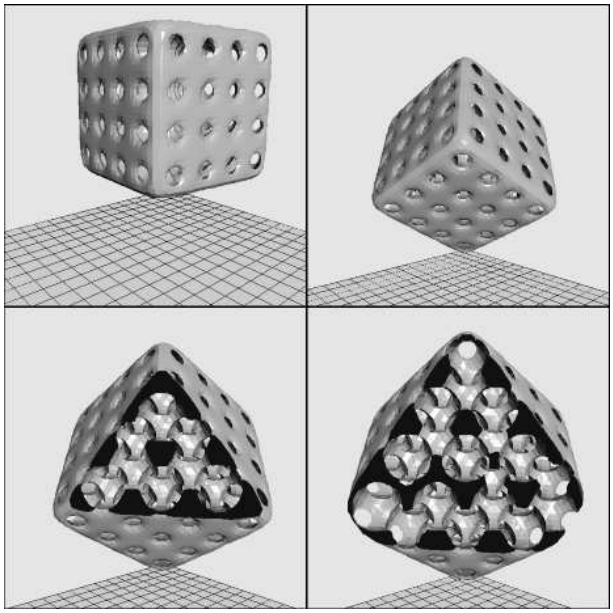

Figure 16: Porous Object With Holes

test a point using the Cyrus-Beck algorithm, similarly T_{DFR} is the average time required using the DFR representation. The tests were performed on an IBM RS-6000 320H with a 20MHz clock speed. The speedup using the DFR scheme over the Cyrus-Beck algorithm is shown in Figure 17. These results relate to a single point contact test. The potential benefit of using the DFR scheme for resolving the multiple contact points which usually occur are much greater. (The code χmal distributes the contact force over the several points in the contact area.)

Finally we show the application of the algorithm to a full three dimensional dynamics simulation involving approximately 5,000 randomly shaped grains falling under gravity into a box (Figure 18). Two sides of the box on which some particles have come to rest have been removed for rendering purposes.

5.5 Summary and Discussion

An object representation scheme that permits efficient contact detection has been presented. The performance of the algorithm in theory and in practice indicates that it possesses significant benefits over certain traditional methods. The speed of the algorithm derives from the spatial ordering imposed on the boundary points, achieved by aligning

Number of Facets	$T_{Cyrus-Beck}$ [sec]	T_{DFR} [sec]	Speedup = $\frac{T_{Cyrus-Beck}}{T_{DFR}}$
8	0.000071	0.000027	2.6
16	0.000114	0.000028	4.1
104	0.000562	0.000025	22.5
296	0.001553	0.000025	62.1
536	0.003017	0.000025	120.7
1052	0.006359	0.000024	265.0
2060	0.013071	0.000025	522.8
2312	0.014920	0.000025	596.8

Table 3: Point Classification Times and Speedup

the address space of the data set with the local coordinate system of the object.

The algorithm for *contact detection* requires $O(N)$ operations, where N is the number of data points used to describe the surface geometry of the object. The testing of a point on one surface with the surface of the other body can be treated as a constant time operation. The is because the test involves a transformation, projection to map to a cell index, and a bounds test for inclusion against the surface represented in a single cell of the underlying structure.

The number of surface points intersecting depends on the resolution of geometry describing the two objects and their relative position. In discrete element analysis the contact constraint dictates that objects may only overlap by a small amount. Consequently, the number of overlapping points is significantly smaller than the total number per object.

References

[Bar81] Alan Barr. Super quadrics and angle preserving transformations. *IEEE, Computer Graphics and Applications Vol 1, No.1*, 1981.

[Bar90] David Baraff. Curved surfaces and coherence for non-penetrating rigid body simulation. In *Proceedings of ACM SIGGRAPH*, volume 24, 1990.

[BJ91] William J. Bouma and George Vanecek Jr. Collision detection and analysis in a physically based simulation. In *2nd Eurographics Workshop on Animation and Simulation*, Vienna, 1991.

[BL95] David M. Beazley and Peter S. Lomdahl. Large-scale molecular dynamics on mpps. *SIAM News*, 28(2), 1995.

[Can88] John Canny. *The Complexity of Robot Motion Planning, ACM Doctoral Dissertation Award 1987.* MIT Press, Cambridge, MA, USA, 1988. ACM Distinguished Dissertation Series.

[CH89] Peter A. Cundall and Roger D. Hart. Numerical modeling of discontinua. In *Proceedings of the 1st U.S. Conference on Discrete Element Methods (DEM), see [MHH89]*, 1989.

[CS79] P.A. Cundall and O.D.L. Strack. A distinct element model for granular assemblies. *Geotechnique*, 29:47,65, 1979.

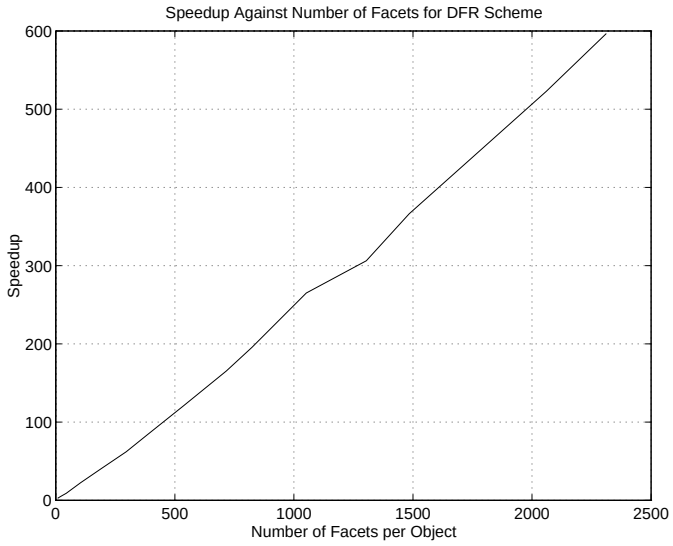

Figure 17: 3D Point Containment Speedup Using DFR Over Cyrus-Beck

[Dwo94] Paul Dworkin. Effcient collision detection for real-time simulated environments. Master's thesis, Media Lab, Massachusetts Institute of Technology, 1994.

[GHM87] J.R. Williams G. Hocking and G.G.W. Mustoe. Dynamics analysis for three dimensional contact and fracturing of multiple bodies. In *Proceedings of NUMETA 1987, Numerical Methods in Engineering, Theory and Applications*, Rotterdam, 1987. Balkema.

[Gre88] Leslie F. Greengard. *The Rapid Evaluation of Potential Fields in Particle Systems, ACM Distinguished Dissertation 1987.* MIT Press, 1988. ACM Distinguished Dissertation Series 1987.

[Hah88] James K. Hahn. Realistic animation of rigid bodies. *ACM Computer Graphics*, 22(4):299,306, 1988.

[Jr.94] George Vanecek Jr. Towards automatic grid generation usnig binary space partition trees. Technical report, Department of Computer Science, Purdue University, West Layfayette, IN 47907, 1994.

[Knu73] D.E. Knuth. *The Art of Computer Programming, Volume 3, Sorting and Searching.* Addison Wesley, Reading, Mass, USA, 1973.

[MHH89] G.G. Mustoe, M. Henriksen, and H-P Huttelmaier, editors. *Proceedings of the 1st U.S. Conference on Discrete Element Methods (DEM)*, Colorado School of Mines, Golden, CO, 1989.

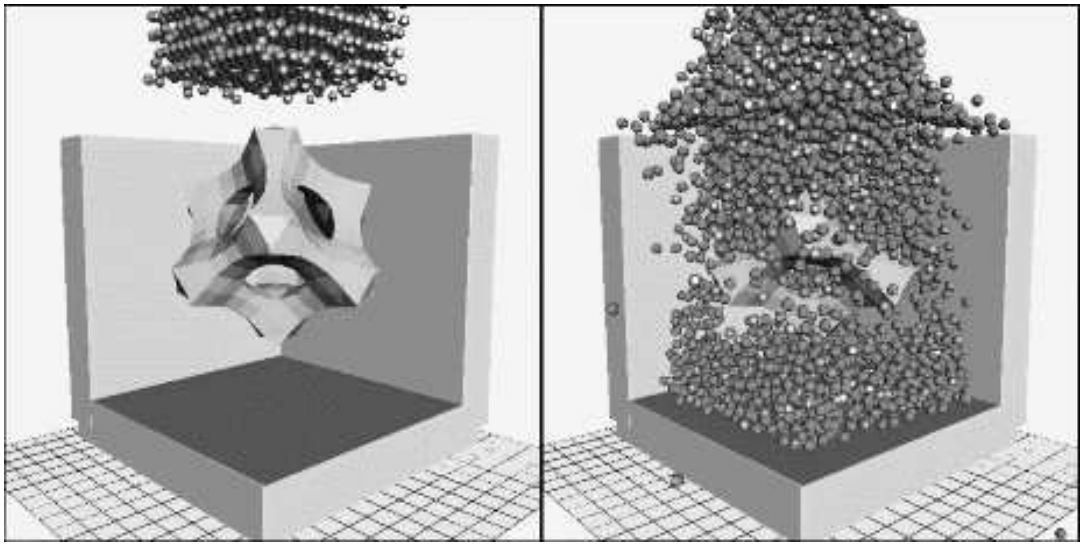

Figure 18: Dynamic Impact Simulation for 3D Objects Falling Under Gravity

[MO93] Ante Munjiza and D.R.J. Owen. Discrete elements in rock blasting. In *Proceedings of the 2nd International Conference on Discrete Element Methods (DEM)*, pages 287,300, 1993. see [WM93].

[MOW94] Ante Munjiza, D.R.J. Owen, and J.R. Williams. On a rational approach to rock blasting. In H.J. Siriwardane and M.M. Zaman, editors, *Proceedings of the 8th International Conference on Computer Methods and Advances in Geomechanics*, volume 1, pages 857,862, 1994.

[MW88] Matthew Moore and Jane Wilhelms. Collision detection and response for computer animation. *ACM Computer Graphics*, 22(4):289,297, 1988.

[NF95a] Tang-Tat Ng and H. Eliot Fang. Cyclic behavior or arrays of ellipsoids with different particle shapes. In *Proceedings of Joint ASME Applied Mechanics and Materials Summer Conference, Mechanics of Materials with Discontinuities and Heterogeneities Symposium*, volume AMD-Vol. 201, pages 59,70, UCLA, Los Angeles, 1995.

[NF95b] Tang-Tat Ng and H. Eliot Fang. Cyclic behavior or arrays of ellipsoids with different particle shapes. In *Proceedings of Joint ASME Applied Mechanics and Materials Summer Conference, Mechanics of Materials with Discontinuities and Heterogeneities Symposium*, volume AMD-Vol. 201, pages 59,70, UCLA, Los Angeles, 1995.

[NL93] Tang-Tat Ng and Xiaoshan Lin. Numerical simulations of naturally deposited granular soil with ellipsoidal elements. In *Proceedings of 2nd International Conference on Discrete Element Methods (DEM)*, pages 557,567, 1993. See [WM93].

[Pen91] Alex P. Pentland. Computational complexity versus simulated environments. In *ACM SIGGRAPH Computer Graphics, Volume 24 Number 2*, pages 185,192, March 1991.

[PW89] Alex P. Pentland and John R. Williams. Good vibrations: Modal dynamics for graphics and animation. *ACM Computer Graphics*, 23(3), 1989.

[Rog85] David Rogers. *Procedural Elements for Computer Graphics.* McGraw Hill, 1985.

[Sed90] Robert Sedgewick. *Algorithms in C.* Addison Wesley, 1990.

[Swe93] Jeffrey W. Swegle. Search algorithm. Technical report, Sandia National Laboratories, Solid and Structural Mechanics Dept., Albequerque, New Mexico, 87185, 1993. External Distribution Memo.

[TKMR93a] John M. Ting, M. Khwaja, L. Meachum, and J. Rowell. An ellipse-based discrete element model for granular materials. *International Journal for Numerical and Anlytical Methods in Geomechanics*, 17:603,623, 1993.

[TKMR93b] John M. Ting, M. Khwaja, L. Meachum, and J. Rowell. An ellipse-based discrete element model for granular materials. *International Journal for Numerical and Anlytical Methods in Geomechanics*, 17:603,623, 1993.

[TM92] B.C. Trent and L.G. Margolin. A numerical laboratory for granular solids. *Engineering Computations*, 9:191–197, 1992.

[TP92] L.M. Taylor and Dale S. Preece. Simulation of blasting induced rock motion using spherical element models. *Engineering Computations*, 9(2), 1992. See also [MHH89].

[TV88] William H. Press Brian P. Flannery Saul A. Teukolsky and William T. Vetterling. *Numerical Recipes in C.* Cambridge University Press, 1st edition, 1988.

[Wil88] John R. Williams. Contact analysis of large numbers of interacting bodies using discrete modal methods for simulating material failure on the microscopic scale. *International Journal of Computer Aided Engineering - Engineering Computations*, 5(3), 1988.

[WM93] John R. Williams and Graham G.W. Mustoe, editors. *Proceedings of the 2nd International Conference on Discrete Element Methods (DEM)*, Dept. of Civil & Environmental Engineering, Massachusetts Institute of Technology, 1993. IESL Publications.

[WO95a] John R. Williams and Ruaidhrí O'Connor. A linear complexity intersection algorithm for discrete element simulation of arbitrary geometries. *International Journal of Computer Aided Engineering - Engineering Computations*, 12(2), 1995. Special Edition on Discrete Element Methods.

[WO95b] John R. Williams and Ruaidhrí M. O'Connor. A linear complexity intersection algorithm for discrete element simulation of arbitrary geometries. *Engineering Computations*, 12:185,201, 1995.

[WP89] John R. Williams and Alex Pentland. Superquadric object representation for dynamics of multi-body structures. In *Proceedings of ASCE Structures*, San Francisco, CA, 1989.

[WP92] John R. Williams and Alex Pentland. Superquadrics and modal dynamics for discrete elements in interactive design. *International Journal of Computer Aided Engineering - Engineering Computations*, 9(2), 1992.

[WS93] Michael S. Warren and John K. Salmon. A parallel hashed oct-tree n-body algorithm. In *Proceedings of Supercomputing '93*, Los Alamitos, IEEE Comp. Soc., 1993.