Proceedings of DETC'00:
ASME 2000 Design Engineering Technical Conferences and Computers and Information in Engineering Conference
September 10-13, 2000, Baltimore, Maryland

DETC2000/DFM-14031

OCTREE BASED RECOGNITION OF ASSEMBLY FEATURES

Raymond C. W. Sung
Department of Mechanical and Chemical Engineering,
Heriot-Watt University,
Edinburgh, United Kingdom
EH14 4AS, (0)131 4495111
r.c.w.sung@hw.ac.uk

Jonathan R. Corney
Department of Mechanical and Chemical Engineering,
Heriot-Watt University,
Edinburgh, United Kingdom
EH14 4AS, (0)131 4495111
j.r.corney@hw.ac.uk

Doug E. R. Clark
Department of Mathematics,
Heriot-Watt University,
Edinburgh, United Kingdom
EH14 4AS, (0)131 4493234.
derc@ma.hw.ac.uk

ABSTRACT

This paper reviews the nature and use of assembly features. One of the conclusions drawn from this survey is that the majority of assembly features involve sets of spatially adjacent faces. Two principle types of adjacency relationships are identified and an algorithm is presented for identifying assembly features, these are features which arise from these "*spatial*" and "*contact*" face adjacency relationships (known as s- and c-adjacency respectively).

The algorithm uses an octree representation of a B-rep model to support the geometric reasoning required to locate assembly features on disjoint bodies. Once all the adjacent faces which form features have been located, they are used to partition the original faces of the assembly into adjacent and non-adjacent portions. The resulting system can locate and partition spatially adjacent faces in a wide range of situations and at different resolutions. By way of illustration, the algorithm is applied to a trial component.

Keywords: *Octree representation, Assembly features, Feature recognition, Geometric modelling, Assembly planning.*

1 INTRODUCTION

Currently, computers are used in industry during the design, manufacturing and assembly stages of production, and a considerable amount of research is being carried out in order to better utilise ever improving computing technology. Such research aids industry by saving both time and money. One active area of research is focused on improving the efficiency of the assembly planning process by using the notion of "*assembly features*".

The reported research in this area has to date, almost exclusively, used manual identification of assembly features as a precursor to the generation of assembly plans or the location of gripping surfaces. There appear to have been few attempts to identify assembly features automatically from a 3D solid CAD model of a mechanical assembly.

The thrust of the research presented here is that many important types of *assembly features* can be identified automatically by locating several distinct classes of adjacent faces in a 3D CAD model.

There are three challenges inherent in this work:

- The adjacent faces will frequently lie on disjoint components of an assembly model. One consequence of this is that most existing approaches to feature recognition cannot be easily applied.

Only rarely will an entire face be adjacent to another face across its whole extent. More usually, only a portion of a face will be adjacent to another (e.g. one side of a cylinder).

- The separation between adjacent faces in an assembly may vary by orders of magnitude. At one extreme, there may be no effective gap between components (i.e. press-fit), while at the other extreme, there may be a distance of several centimeters.

This paper is structured as follows:

Section 2 begins by describing a selection of assembly feature definitions and applications proposed by a number of researchers before briefly reviewing techniques used to identify features on individual mechanical parts. Section 3 reviews four possible methods for locating assembly features that have been investigated by the authors. Section 4 gives an overview of the different octree applications, properties and algorithms. Section 5 details the adjacent face algorithm and lastly, section 6 draws some conclusions and outlines the direction of future work.

Copyright © 2000 by ASME

2 LITERATURE REVIEW

This section presents a literature review on the topics of assembly features and feature recognition.

2.1 Assembly Features

This review starts by surveying the various definitions of assembly features proposed in the literature, before describing some of the applications they have been used for.

2.1.1 Assembly Feature Definitions

Assembly features have been used by several researchers to improve the efficiency of the assembly planning process, but there have been significant variations in the definition of an assembly feature.

In the paper by Shah and Rogers (1993), assembly features are described as "mating pairs of form features with parameters and compatibility constraints as part of each feature definition".

While according to Anantha et al (1996), *assembly features* are "regions of geometry that are identifiable in the manufacturing sense such as holes and slots", while *primitive features* are "those that are not identifiable in the manufacturing sense, but participate in assembly constraints".

Similar to assembly features are the so-called "mating conditions" mentioned in Mullins and Anderson (1998), these are "relationships that involve contact between parts, as well as relationships in which two parts do not have contact (e.g. clearance conditions)". Furthermore, these mating conditions "define relationships between components that may not hold if the dimensions of the components change".

In the paper by Bronsvoort et al (1995), assembly features were described as "relations between parts of an assembly that are given on a feature level". The assembly features are then subdivided into *connection features* and *handling features*. Connection features are characteristics of connections such as final position, insertion path/point, tolerances, contact faces and some geometrical refinements (e.g. chamfers) that ease assembly operations. Handling features, on the other hand, are "characteristics that give the locations on an assembly component that can be safely handled by a gripper during assembly and the faces that can be used to fix base parts in fixtures".

Another variation on the definition of assembly features is contained in Gui and Mantyla (1994) where "connectors" are used which are a description of features that provide constraints for the joint components to ensure that they perform the required functions. The paper summarises the different connectors available by splitting them into three main categories: fix, motion constraint and motion/force transmission. For the present research, the third category of assembly features will not be relevant because these are of a more complicated form, and hence will be more difficult to identify.

Yet another variant on the concept of assembly feature is the "virtual link", used by Lee and Gossard (1985). These are used to connect two components in an assembly. They contain information on the mating conditions and relationship between the mating pair. These relationships include whether the attachment is rigid or conditional, and whether the constraint is rotational or translational. The *types* of mating conditions available include the mating faces involved and the co-linear centre lines of the assembly components. The main difference between this paper and the other papers mentioned, is that the assembly features used here only occur between two components whereas in the other papers, an assembly feature can occur in two *or more* components.

What all the above definitions have in common is that assembly features are regions of an assembly that help improve the efficiency of the assembly planning process but these regions are frequently poorly defined. In Fig. 1., a simple assembly is shown, and one assembly feature on it could be a hole and the screw that is inserted into it. This assembly feature would be straightforward for a human being to identify, but a computer would have more difficulty because it does not "know" which component is adjacent to which (it is not explicitly recorded in the B-rep data structure).

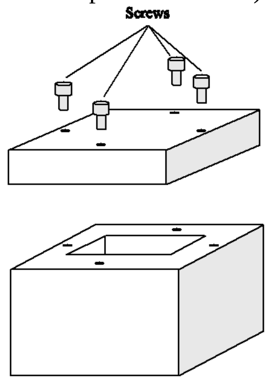

Figure 1 - Typical Assembly

Of course, it would be inefficient for a human to try to locate all the assembly features on an assembly component that contained tens or hundreds of these regions and it would be ideal if a computer could carry out this laborious task. To do this, a computable definition of what constitutes an assembly is needed. Consequently, we propose that with a few exceptions, the majority of assembly features involve sets of spatially opposing faces. It can also be observed that, typically, adjacent faces which are in "contact" create "mating features", while "gripping features" are characterised by a greater degree of separation. In this work, the first type of feature is referred to as being formed by contact adjacency (*c-adjacency*) and the latter by spatial adjacency (*s-adjacency*).

Copyright © 2000 by ASME

2.1.2 Uses of Assembly Features

The previous sections has shown that there are many definitions of assembly features; the uses to which they are put are also varied.

Deneux (1999), describes assembly features have been used in the assembly planning process for the assembling of an aeroplane. The first step involves creating a four by four triangular "relationship matrix" that denotes the set of possible relations between components of the same category (such as fluid circulation or electronic data transmission). From this matrix, the assembly features of the assembly are retrieved by considering typical complex assemblies and analysing the relationships between the components. If during the analysis, a new type of relationship is found, it is given a name and then reported back to the relationship matrix. Finally, by utilising this list of assembly features, possible assembly plans can be created by using a graphical user interface (GUI). This GUI allows the user to make assembly plan changes and then the program shows up any possible obstacles during the assembly process. Once the user has made a change, the whole system automatically updates the geometry and down-stream assembly feature list. Other notable features of the prototype program include the ability to identify several predefined types of components used in the assembly. Also, the system allows the user to create a solution for an attachment or collision problem encountered during the assembly process. The one disadvantage of the system is that the ramifications of each solution given by the user to each problem is hard coded into the program rather than calculated in real time. This means that the range of possible scenarios is limited and adding new alternatives requires the user to program them in.

In Van Holland's thesis (1997), assembly features are used in both assembly modelling and assembly planning to improve their efficiency and this has been implemented in a prototype assembly modelling program called DIAC 1 and also in several assembly planning modules such as grip planning, stability analysis and assembly sequence planning. As mentioned earlier in the paper by Bronsvoort et al (1995), the assembly features used are made up of connection features and handling features. Both these features are predefined as classes in the computer code. In grip planning, assembly features are used to improve the efficiency by reducing the number of calculations needed to be carried out. This involved using the handling features containing information on the addition features because it was proposed that only the addition features (features that add volume to a model) can have an influence on the grip areas. This in turn helps determine which faces of the assembly need to be involved in the calculation and therefore the size of the search space is reduced and efficiency increased. For assembly sequence planning assembly features are used to reduce the size of the search space. This involves simplifying the AND/OR graphs that show all the possible disassembly sequences by utilising the connection features. The connection features are able to give the precedence for component removal by using information on the clusters (components that share some assembly-specific characteristic) and agents (components, whose only purpose is to establish certain connections, for example bolts or screws). Finally, assembly features are used to improve the translational stability analysis process by utilising the connection features of components in the assembly. These connection features define the internal freedom of motion of each component, which, in turn, aids the process of determining whether components will be unstable when assembled to the partial assembly. The use of assembly features in rotational stability analysis is under current investigation.

In this thesis, the assembly features are not automatically located and identified on an assembly CAD model but instead, the assembly features for each generic component of the assembly are manually defined. The disadvantage of this method is that it can only find assembly features contained in its program by comparing the assembly feature found in the predefined assembly features it has listed. This means it does not automatically adapt to new variations of assembly features, which can only be overcome by manually adding the new type to its list.

De Fazio and Whitney (1987), describe a form of assembly feature referred to as "liaisons between assembly components" were used to determine the optimum assembly sequence. These "liaisons" are relationships between the components in an assembly and the user is required to define the specific contact type and precedence relationship for each liaison. One disadvantage of the method is the number of questions asked by the program that the user must answer before the assembly sequence can be calculated. This makes it inconvenient in situations where there is a large assembly containing many liaisons. This also means that the user has to be certain the answers given are correct to enable a correct assembly sequence to be given. Another limitation of the method is that the assembly sequence returned cannot include a change of assembly orientation, which may limit the efficiency of the assembly sequence plan because not all components in a typical assembly can be assembled in one orientation.

The main conclusions drawn from this survey of applications is that unlike machining features, the boundary of an assembly feature does not have to be precisely defined. In applications such as grasp or assembly planning, often only approximate or qualitative information is required. Given the size and complexity of most mechanical assemblies, it will clearly be essential to automatically identify assembly features in CAD models.

2.2 Feature Recognition For Part Programming

The traditional role of automatic feature recognition (AFR) is to identify the machinable features on a 3D CAD model of a mechanical component. Although there are several well established AFR methods, it is important to note that current feature recognition software does not appear to be directly applicable to assembly feature identification because such features are typically formed by the juxtaposition of two or more disjoint components, whereas a classical machinable feature is normally present on only one component.

Feature recognition can be manual, automatic or a combination of both. If feature recognition is carried out manually, it requires human intervention to identify the features. This method is only practicable for simple components. For more complex components (typically consisting of thousands of faces), AFR is a much more realistic option. Furthermore, the automation of feature recognition potentially allows the design and manufacturing stages to be more easily integrated.

2.2.1 Feature Recognition Algorithms

There are a large number of papers on this subject and the following review only provides a very brief outline (for a more complete review, see Marefat and Ji (1997)).

The majority of feature finding algorithms devised to date fall into one of the following categories:

- Graph-Based algorithms (Little et al, 1998)
- Rule-Based algorithms (Vandebrande, 1990)
- Hybrid algorithms (Prabhakar 1990)
- Alternating Sums of Volumes (ASV) algorithm (Kim, 1992 and Woo 1982)

Even though all these methods can automatically identify machinable features on a single mechanical component, they cannot be directly applied to the automatic identification of assembly features.

The problem with applying graph-based algorithms is that the face adjacency graphs commonly used contain no information on adjacent disjoint components. Consequently, it is not possible to locate spatially adjacent faces present in regions containing assembly features. Rule-based algorithms may, in principle, be utilised to find assembly features. However, work in mechanical feature recognition has suggested that comprehensive sets of rules are difficult to create and maintain. In principle, a neural net method should stand a better chance if it has been adequately trained to identify all the different combination of assembly features. However, it is unclear as to what form the input representation should take. In principle, the ASV decomposition technique could be used to locate adjacent faces on a model but the practical details of implementing a robust convex hull algorithm are significant.

It is clear from this brief survey that new forms of geometric reasoning will be required to correctly identify assembly features.

3 POSSIBLE METHODS FOR FINDING ASSEMBLY FEATURES

In order to enable the automatic location of assembly features, adjacent faces on a CAD model of an assembly have to be located first. This section outlines several possible methods for locating adjacent faces that were investigated and describes why the octree method was considered the most promising.

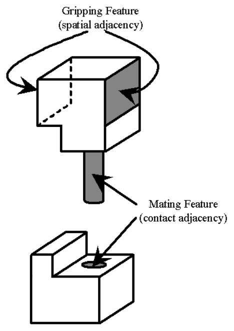

Figure 2 – Examples of Contact and Spatial Adjacency

As mentioned earlier, it should be noted that faces are not always wholly adjacent to each other. For example, with the pair of adjacent (grey) faces belonging to the gripping feature of the model shown in Fig. 2., the visible face is partially adjacent to its opposite face which is hidden in the figure.

3.1 Offsetting and Sweeping.

In order to facilitate the creation of moulds, most kernel modellers provide body offsetting and shelling facilities.

Adjacent faces could, in principle, be found by using offsetting or sweeping API’s in the following way: all the faces on a CAD model of an assembly are offset or swept outwards and, when faces intersect, the implication is that they are adjacent to each other. However, there are several problems with this method:

1. The offset or sweep distance would have to be specified.
2. Modification of the geometry of the solid is computationally expensive.
3. Problems can occur when a feature is made up of concave faces or edges, because these tend to be obliterated by the expanded body. Consequently, the spatial adjacencies of small features are difficult to determine. Furthermore, the resulting non-manifold body frequently causes problems in offsetting and other modelling operations.

3.2 Ray - Firing

Ray firing, is a computer representation of a ray of light, and its properties include a user-defined ray direction and radius.

Adjacent faces can be found using ray firing in the following way: on each surface of the assembly, a ray normal to the surface is fired outwards. If the ray hits a face, as shown in Fig. 3., then the face that the ray has hit is defined to be an adjacent face to the one from which the ray was fired.

Copyright © 2000 by ASME

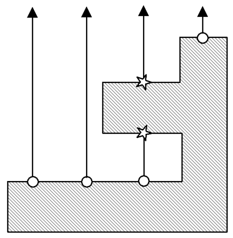
Figure 3 - Ray Firing

However, as with the offsetting method, there are problems:

1. The computational requirements are high because a ray has to be fired from many points on each face
2. The method is very sensitive to the direction of the ray fired and adjacent faces might by missed.
3. It is not clear how this approach could be used to determine *c-adjacency*.

3.3 Tessellating

The final method investigated involved taking the spaces surrounding the components of the assembly and subdividing them into smaller spaces. One form of tessellating is the octree representation method in which the volume that the 3D solid CAD model occupies is subdivided into cubes.

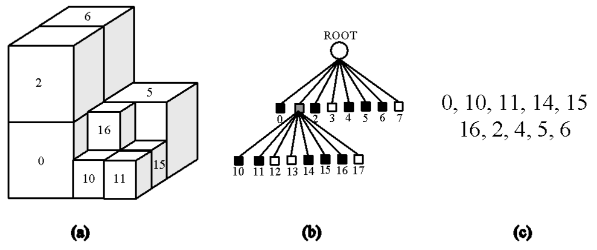

Figure 4 – a) Octants, b) Octree Structure, c) Linear Octree

Each of the cubes has a locational code (see Fig. 4a.) that gives the exact location of each cube in the octree. The relationship between each octant can also be viewed by means of a tree structure (see Fig. 4b.). This resulting data structure can be simplified as a string of numbers (see Fig. 4c.) thereby reducing the amount of space required to store the octree.

3.4 Selection of Best Method

An octree representation method was chosen for this research for the following reasons:

1. Since the representation allows both the space between and inside boundaries to be examined, both "contact" and "spatial" forms of adjacency can be located.
2. An octree based method can function generically, irrespective of whether faces are planar, cylindrical or even spline.
3. The option to vary the resolution of the octree representation.
4. The algorithms used to search for adjacent faces, via the octree data structure, can exploit existing algorithms for neighbour finding.
5. Finally, the data structure itself takes up a relatively small amount of storage space because the 3D CAD model can be represented with a string of letters (i.e. a pointerless octree) rather than a list of component dimensions and physical entities such as faces, edges and vertices.
6. Once generated, the analysis of an octree structure is very fast even for complex models.

The next section describes in more detail, the octree representation method.

4 OCTREE REPRESENTATION METHODOLOGY

As outlined in section 3.3, one way of representing a 3D model is via an octree representation which has as its root, a bounding box around the 3D CAD model. This box is then subdivided into eight octants by halving the bounding box in each axis direction, the resulting octants are known as level one octants.

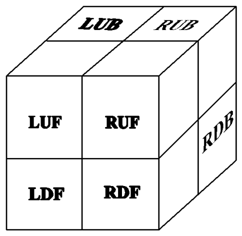

Figure 5 - Octant Identification

Each octant is assigned an identification tag, which signifies the level of the octant (i.e. the length of the tag) and also gives the exact location of the octant in the bounding box (see Fig. 5.). For example, octant [LUF] means the octant is located at the Left Upper Front position of the bounding box and octant [LUB, RDB] is a level two octant because the tag contains two locational codes. At this stage each octant is checked to see whether it is full of solid material, partially full of solid material or if it is empty. If the octant is full or empty, it is not further subdivided any more but if the octant is partially full, it is subdivided to create another level of smaller octants. In principle, the whole process can be repeated until all the octants in the bounding box are either full or empty with no more partially full octants, at which stage, the boundary of the object is obtained. However, the process can also be terminated when some other limit is reached (i.e. maximum level), or if instructed by the user to terminate early. At the end of the process, the linear octree list will contain a list of octants that when viewed graphically, would give an approximate representation of the 3D model.

4.1 Octree Neighbour Finding

The main use of octrees has been to provide a compact representation of a complex 3D image, or object. However, in this work, we exploit another advantage of the octree representation: namely, the fact that the octree structure is a hierarchical data structure so that it is easy to search.

4.2 Adjacent Octant Finding Algorithms

The algorithms developed by Samet (1989) that allow the searching of neighbouring octants in an octree data structure is typical of the methods used to search octrees. One of the algorithms is for neighbour finding in pointerless octrees, and one for octrees using pointers.

Octrees that use pointers contain nodes that have father and a son labels, so that during the search, the octree can be ascended and descended.

Pointerless octrees, on the other hand, consists of a nested list of nodes that are either full, partially full or empty.

The three fields associated with each node are the PATH (path from the root to the node), LEV (level of the node) and COL (colour of the node). Samet's paper suggests that during the neighbour finding process using pointerless octrees, it is not necessary to move through the octree data structure by traversing links, but instead, bitwise manipulation can be used. For a node A, to find its neighbour Q, the path component of the locational code T, of the neighbour has to be constructed. Starting at node A and beginning with the digit position corresponding to the link from A to its father, each directional code is reflected (using a function REFLECT) in direction I until a nearest common ancestor of A and Q is found. The function REFLECT is made up of a table of values containing the reflected node for all the nodes in the octree structure. The purpose of the function is to calculate an octant that is on the same level as A in the octree that also shares an edge, face or vertex with it. A function called ADJ is then used to detect the nearest neighbour by utilising a table of boolean values which define whether octant O is adjacent to the Ith face, edge or vertex of O's containing block. Since reflection occurs while ascending the octree structure, descending the tree is not required.

For example, in Fig. 6., the reflection of LUF, say, in direction R (right) would be RUF and since they are level one octants, there common ancestor ("father") is the bounding box itself.

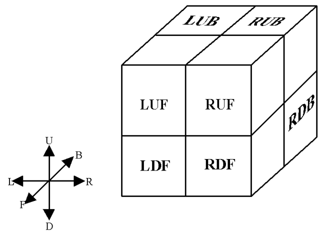

Figure 6 - Reflection of Octants

When T has been located, the algorithm determines if the nearest neighbouring node exists and if so, whether it is white, black or grey. A white node corresponds to an empty octant while a black node is an octant that is completely filled. Finally, a grey node is an octant that is partially filled. To determine whether the neighbouring node exists, the list containing all the locational nodes is searched for to determine if Q is of an equal or greater size to A.

In Fig. 7., an example illustrates that the neighbour octant of [LDF, RUF] in direction R (right) is [RDF, LUF] rather than RDF because it is on a different octant level.

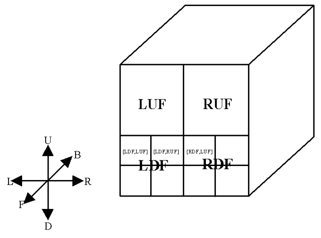

Figure 7 - Nearest Neighbour Octant

4.3 Assembly Applications of Octrees

One way in which octrees can be used to aid the assembly planning process has been described in the paper by Bard et al (1992). This paper uses octrees to carry out *accessibility* and *grasp planning* by taking slices of the octree model that is generated from a reconstructed volumetric representation based on voxels of the object. The next step is to encapsulate each of the connected components of each slice with an ellipse (akin to a bounding box) and then four approach directions (the semi-axis of the ellipse) are associated with each slice. Finally, in each slice, a list of "good" directions and ellipses are chosen which then gives, for each slice, a list of pairs {ellipse, direction}. This list of pairs is then used for the 3-D reconstruction of the object using generalised cylinders by comparing the pairs of slices to find similarities between them. When a similar pair of slices are found they are then joined and the process is repeated with all the slices. At the end of the analysis, the original object is reconstructed with the slices and the access direction of the object will then be given by the direction specified by the list pair of the slices.

The next step is pre-shaping, which determines the contact type, number of virtual fingers, number of real fingers and the opposition type. The contact type corresponds to the contact between the grasping finger and the object: tip, pad and palm. The number of virtual fingers corresponds to the set of real fingers moving together and the number of real fingers are the fingers that are in contact with the object. Finally, the opposition type is the position of the initial fingers for preshaping: 3-digits, 2-digits, side, cylindrical, digit-palm and no-opposition. The contact type is determined by height and width of the generalised cylinders generated earlier and the number of fingers is determined by the thickness of the generalised cylinders. Finally, the type of opposition is determined by the shape of the generalised cylinders.

After pre-shaping has been carried out, a list of the possible grasping configurations is obtained.

At present, this list is not ranked so it is not known which in the list is the optimum configuration which means the current

Copyright © 2000 by ASME

efficiency can be improved upon. The authors acknowledge this problem and state that the list can be ranked if task information is given and this will be the subject of future research.

5 OVERVIEW OF ALGORITHM FOR ADJACENT FACE FINDING

This section describes the algorithms which are applied to the octree data structure to locate adjacent faces. The following sub-sections detail:

1. How an octree representation is generated from a B-rep model of an assembly.
2. The algorithms applied to the octree to locate adjacent faces.
3. The procedure used to map the results of the octree analysis back to the original B-rep model.

5.1 Generation of Octree Representation

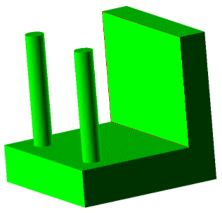

Figure 8 - Original CAD Model

A pointerless octree representation is generated by recursively sub-dividing the assembly model's bounding box. The kernel modeller's boolean API's are used to label each octant as:

Type 0:	An octant located outside the solid model.
Type 1:	An octant partially inside the solid model of one body.
Type 2:	An octant located inside the solid model.
Type 3:	An octant partially inside the solid model of two or more bodies.

A vector is associated with every type 1 octant which reflects the approximate orientation of the solid boundary contained by the octant. This is obtained by:

1. Calculating the mid-point of the octant.
2. Using the kernel modeller's API to determine the closest face on the model to the octant's mid-point.
3. Using the kernel modeller's API to determine the closest point on the closest faces to the octant mid-point. Note the success of this operation is independent of which side of the face the mid-point lies.
4. Using the kernel modeller's API, the face's surface normal at the closest point is found and associated with the octant.

The level of the subdivision is currently determined by the user.

Figure 9 shows the octree representation of the simple assembly shown in Fig. 8. (Two cylinders inserted into a pair of holes in an L-shaped bracket).

The red regions depict the solid octants while the transparent grey regions represent the partially full octants. Since the empty octants are not visible, they are omitted from the VRML file used to generate the illustration.

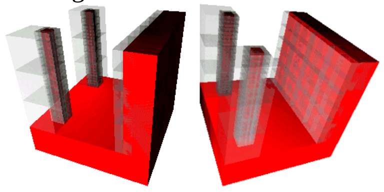

Figure 9 - VRML Representation of Octant List

5.2 Adjacent Face Identification

The octree representation is used to locate two types of adjacent faces:

1. Those portions of faces which are *c-adjacent* (i.e. they effectively touch within the tolerance of the octree).
2. Those portions of faces which are *s-adjacent* to a nominated face.

The procedures adopted for these two cases can be visualised in terms of two schematic representations of the octree generated from the assembly shown in Figure 8.

5.2.1 Contact Adjacency Detection

The level A section (see Fig. 10.) illustrates the detection of *c-adjacency* octants.

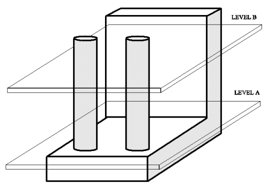

Figure 10 – Two Sections Through the Assembly Model

C-adjacent octants (shaded octants shown in Fig. 11.) are determined by locating the contiguous regions of type 3 octants and then applying the mapping procedure detailed in section 5.3.

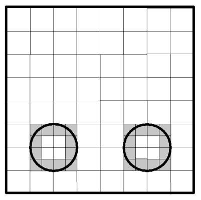

Figure 11 - *C-adjacent* Octants

5.2.2 Spatial Adjacency Detection

Having located those portions of the assembly model's faces which are in "contact" with other areas of solid, the next problem is to identify potential gripping surfaces of features which can be used to extract or insert components. Gripping features are frequently defined by portions of anti-parallel faces (see Figure 2.). The suitability of potential gripping features are

Copyright © 2000 by ASME

determined, in part, by their proximity to other parts of an assembly (e.g. are there sufficient clearances to allow a robot gripper, or human hand, to locate and release the part in its mating hole?). Consequently, it is necessary to search both sides of a face when searching for *s-adjacency* relationships.

In addition to location of contact adjacency, the octree data-structure can also be used to detect spatial adjacency. Ultimately, the spatial adjacency of all non-*c-adjacency* faces will be investigated. However, in this prototype implementation, the search is restricted to planar faces, aligned with the axis of the octree. Obviously, an exhaustive approach will not be feasible in the case of assemblies and heuristics will be required to limit the search. This issue is discussed further in section 6.

Given a starting face (the process is illustrated with the vertical wall of the L-shaped bracket), the algorithm locates the octants that intersect it (shown as grey octants in Fig. 12.). This is achieved by using the kernel modeller's API to test whether individual octants intersect the face.

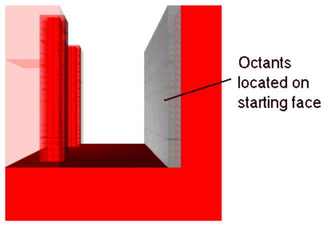

Figure 12 - Starting Face on Model

Each of these octants is used as the starting point for a search process that traverses the octree once in the direction of its face normal and once in the direction anti-parallel to the face normal, using an algorithm based on Samet's (1989) and described in section 4.2 (see Fig. 13.).

In this way, the adjacency of faces separated by both solid and air can be established. The subsequent description and illustrations deal only with air traversal but the process is equally applicable for solid traversal.

```
*Face Traversing Algorithm*
OctList = Set of octants lying on start face
Octant = Member of OctList
Norm = Normal of start face
OctNeighbour = Locates Neighbour of Octant in direction of Norm
**for each** member of OctList
  Neighbour = OctNeigbour (Octant, Norm)
  Continue Until Neighbour is non-empty
  Add final Neighbour to Adjacent Octant List
**end for**
```

Figure 13 – Face Traversing Algorithm Pseudo Code

The level B section (see Fig. 10.) illustrates the process of detecting *s-adjacency* octants.

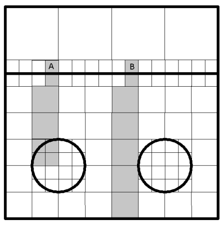

Figure 14 - Possible Traversal Paths

The traversing continues through empty space until a solid or partially full octant is reached, at which point the search terminates.

During the search process it is common for the level (i.e. depth) at which a neighbouring octant is located to change. For example, if the traversing begins at octant A shown in Fig. 14., then the size of the octants will increase during the traversal. The reason for this is that at each stage, the algorithm attempts to locate a face neighbour octant that is the same size as the starting octant, but when none are available, the algorithm uses the *smallest* octant present. However, once the depth at which neighbouring octants are being located has been reduced, it cannot be increased at a later stage. Therefore, when the octants on the adjacent face are reached, they are not always of a similar size to starting octants and can contain smaller octants. This is the case in Fig. 14., where the search terminates with the location of a type 1 octant containing smaller octants.

Sometimes, this is not a problem, as when the traversing begins at octant B, for example, in Fig. 14. Here, no adjacent octants will be found containing material and the search terminates at the edge of the octree.

Hamet's algorithm has been modified to detect the boundary of the octree since the original returned octant paths without checking if they lie outside the object's boundary.

At the end of the search process, all the octants surrounding the two vertical cylinders of the test object are found (shown in Fig. 15.). Note that the traversal process has located type 1 octants surrounding the cylinders at a depth in the tree which is well above the smallest octants. Consequently, some of the octants located contain portions of the face orientated away from the face which initiated the search.

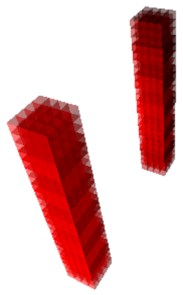

Figure 15 - Octants Found Adjacent to the Starting Face (i.e. enclosing the cylinders)

Copyright © 2000 by ASME

5.3 Locating Anti-Parallel Octants

This step takes the list of octants found by the traversal process and discards those that are lying on non-adjacent portions of the face (i.e. those that are anti-parallel to the search direction). This is done by comparing the orientation of the octants with that of the octant used to initiate the search. Those found to be parallel or orthogonal are deleted from the list leaving only anti-parallel octants. Figure 16 shows the algorithm pseudo code, while Fig. 17. shows the results of applying this procedure.

```
Octant Discarding Algorithm
OctList = List of octants surrounding adjacent Face
OctList2 = List of octants on adjacent face
Oct = Member of OctList
OctNormal = Surface normal of Oct
TraDir = Traversing direction
for each member of OctList
  if ((OctNormal (Oct) = - TraDir) and (Oct Intersects face))
    Add Oct to OctList2
  end if
end for
```

Figure 16 - Octant Discarding Algorithm Pseudo Code

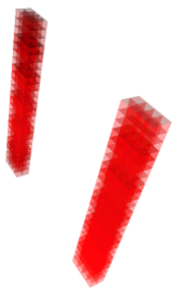

Figure 17 – Adjacent Octants with Anti-Parallel Orientation to the Search direction

5.4 Partitioning of Feature Faces

Using the kernel modeller, the list of octants lying on both *s-adjacent* and *c-adjacent* faces are converted to solid blocks and then united to create a number of solids. These solids are then imprinted with the original assembly model. The imprinting process partitions the faces into portions which explicitly represent the adjacency relations in the model.

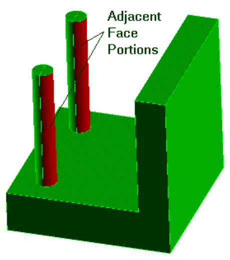

Figure 18 - Test Model with Faces Adjacent to the Starting Face Highlighted

In Fig. 18. above, the regions in red represent the adjacent faces of the starting face opposite to it.

Facilities within the kernel modeller allow an attribute to be attached to each solid cube created from an octant. These attributes record the face from which the adjacency search started. Mechanisms within the modeller allow the attributes to be passed on to the partitioned faces by the boolean imprinting process.

6 CONCLUSIONS AND FUTURE WORK

The results show that, at least with the test model used, the algorithm has been successful in locating the pair of *s-adjacent* and *c-adjacent* faces on the assembly.

It has been found that the execution times for the algorithms presented in section 5 are in the region of 1-10 minutes using a computer with a 400MHz CPU. The most time consuming algorithm, not surprisingly, is the generation of the locational codes for the level five octant representation of the test model.

However, one advantage of the current algorithm is that each step can be carried out separately so the whole process does not have to be executed in one single stage. Another advantage is that the resolution of the octree representation can be varied to suit the computing hardware being used and the complexity of the test object. In addition, how detailed the adjacent face finding process is to be can be controlled, depending on the user's requirements.

The current implementation is written in Scheme and uses the ACIS kernel API (through its Scheme interface). Future work will involve implementing the approach in C++, in order to improve the resolution of the octree that can be generated.

Testing of the algorithm will continue with more varied and complicated test assembly objects. One known problem occurs in the case where the model boundary is coincident with the octree, so transitions in the octree between solid and air with no partially full octants at the boundary present will be checked for. In order to tackle more complex assemblies, work will be required to:

1. Develop heuristics which will reduce the number of searches for *s-adjacency* relationships.
2. Allow searches to be initialised from non-planar faces.

Once this has been accomplished, the next goal will be to further classify the basic types of mating and gripping features which have been identified. This task will be more complicated because the adjacent face patterns specific to each assembly feature will have to be searched for. Finally, the manner in which the results can be exploited to help generate an assembly plan for components will be demonstrated.

ACKNOWLEDGMENTS

The authors would like to express their gratitude to the UK's Engineering and Physical Sciences Research Council for their support under award number 98315497.

Copyright © 2000 by ASME

REFERENCES

Anantha R., Kramer G.A. and Crawford R.H., 1996, “Assembly modelling by geometric constraint satisfaction”, *Computer-Aided Design*, Vol. 28, No. 9, pp 707-722.

Bard C, Troccaz J and Vercelli G., 1992, “Shape analysis and hand preshaping for grasping”, *Proceedings — IEEE/RSJ International Workshop on Intelligent Robots and Systems*, No. 1992, pp 64-69.

Bronsvoort W.F. and van Holland W., 1996a, “Assembly features and sequence planning”, *Product Modelling for Computer Integrated Design and Manufacturing – Proceedings TC5/WG5.2 International Workshop on Geometric Modelling in Computer Aided Design, 19-23 May 1996, Airlie, USA*, Pratt M, Sriram R.D & Wozny M.J.(eds), Chapman & Hall, London, pp 275-284.

Bronsvoort W.F. and van Holland W., 1996b, “Extracting grip areas from feature information”, *CD-ROM Proceedings of the ASME 1996 Design Engineering Technical Conferences and Computers in Engineering Conference, 19-22 August, Irvine, USA*, McCarthy J.M. (ed), ASME, New York.

Bronsvoort W.F., van Holland W. and Jansen F.W., 1995, “Feature modelling for assembly”, *Graphics and Robotics*, Straszer W. and Wahl F. (eds), Springer-Verlag, Berlin, pp 131-148.

Chan H. W. and Kwok W. C., 1993, “An enhanced octree structure for representing the spatial/boundary decomposition of 3-D objects”, *Proceedings of the 1993 IEEE Region 10 Conference on Computer, Communication, Control and Power Engineering*, Part 2, pp 1017-1020.

De Fazio T. L. and Whitney D. E., 1987, “Simplified generation of all mechanical assembly sequences”, *IEEE Journal of Robotics and Automation*, RA-3, No. 6.

Deneux D., 1999, “Introduction to assembly features: an illustrated synthesis methodology”, *Journal of Intelligent Manufacturing*, Vol. 10, pp 29-39.

Gui J. K. and Mantyla M., 1994, “Functional understanding of assembly modelling”, *Computer-Aided Design*, Vol. 26, No. 6, pp 435-451.

Henson B. W. and Juster N. P., 1997, “Information requirements for the support of assembly mating conditions”, Proc DECT, *ASME Design Engineering Technical Conferences*.

Kela A., 1989, “Hierarchical octree approximations for boundary representation-based geometric models”, *Computer-Aided Design*, Vol. 21, pp 355-362.

Kim Y. S., 1992, “Recognition of form features using convex decompostion”, *Computer-Aided Design*, Vol. 24, pp 461-476.

Lee K. and Andrews G., 1985, “Inference of the positions of components in an assembly: part 2”, *Computer-Aided Design*, Vol. 17, No. 1, pp 20-24.

Lee K. and Gossard D. C., 1985, “A hierarchical data structure for representing assemblies: part 1”, *Computer-Aided Design*, Vol. 17, No. 1, pp 15-19.

Li R., 1991, “An algorithm for building octree from boundary representation”, *Intelligent Design and Manufacturing for Prototyping*, ASME, 50, pp 13-23.

Little G., Clark D. E. R., Corney J. R. and Tuttle J. R., 1998, “Delta-volume decomposition for multi-sided components”, *Computer-Aided Design*, Vol. 30, No. 9, pp 695-705.

Marefat M. M. and Ji Q., 1997, “Machine interpretation of CAD data for manufacturing applications, *ACM Computing Surveys*, Vol. 24, No. 3.

Mullins S. H. & Anderson D. C., 1998, “Automatic identification of geometric constraints in mechanical assemblies”, *Computer-Aided Design*, Vol. 30, No. 9, pp 715-726.

Prabhakar S., 1990, “An experiment on the use of neural nets in form feature recognition”, M.S, Arizona State University.

Samet H., 1989, “Neighbour finding in images represented by octrees”, *Computer Vision, Graphics and Image Processing*, 46, pp 361-386.

Shah J. J. & Rogers M. T., 1993, “Assembly modelling as an extension of feature-based design”, *Res. Engng Des.* 5, pp 218-237.

Vandebrande J. H., 1990, “Automatic recognition of machinable features in solid models”, Electrical Engineering Department, University of Rochester, Rochester, NY.

Van Holland W., 1997, “Assembly Features in Modelling and Planning”, Ph.D Thesis, Delft University of Technology, Holland.

Woo T. C., 1982, “Feature extraction by volume decomposition”, *Proc. of CAD/CAM Technology in Mechanical Engineering*, Cambridge, Massachusetts, USA, pp 76-94.

Yang S. N. & Lin T. W., 1990, “A new 3D-border algorithm by neighbour finding”, *Proceedings — IEEE Computer Society’s International Computer Software & Applications Conference*, No 1990, pp 353-358.

Copyright © 2000 by ASME