

Generating Topological Information from a “Bucket of Facets”

Stephen J. Rock
Michael J. Wozny

Rensselaer Design Research Center
Rensselaer Polytechnic Institute
Troy, New York 12180

Abstract

The STL *de facto* data exchange standard for Solid Freeform Fabrication represents CAD models as a collection of unordered triangular planar facets. No topological connectivity information is provided; hence the term “bucket of facets.” Such topological information can, however, be quite useful for performing model validity checking and speeding subsequent processing operations such as model slicing. This paper discusses model topology and how to derive it given a collection of unordered triangular facets which represent a valid model.

1 Introduction

Computer Aided Design (CAD) model data is frequently passed to various Solid Freeform Fabrication processes using the STL polygonal facet representation [1]. Facet models represent solid objects by spatial boundaries which are defined by a set of planar faces. This is a special case of the more general Boundary Representation which does not require object boundaries be planar [2]. In general, the term facet is used to denote any constrained polygonal planar region being used to define a model boundary; however, in the Solid Freeform Fabrication (SFF) community the term facet is typically understood to mean *triangular* facet. Representing models using triangular facets has both good and bad points [3]. Facets do provide a “greatest common denominator” geometrical form for data exchange between many CAD systems and SFF processes. Non-CAD scalar field data, such as that from CT imaging, can also be used to generate facet models [4]. However, facet models are generally only an approximation of mathematically precise CAD models.

Precise CAD models must be tessellated, where defining model surfaces are subdivided into planar facets, to create polygonal facet models [5]. As model precision demands become more stringent, the number of facets required to adequately approximate a model will increase. Model tessellation should yield a set of facets which define a closed region representing the material boundary of a part. Unfortunately, many commercial CAD system model tessellators are not robust, and sets of facets which do not define closed regions result. This missing facet problem is particularly prevalent where surfaces intersect in the original CAD models. A set of facets which, when assembled, forms a solid object with holes in its surface is incomplete and is termed an *invalid* model. In addition to missing facets, other causes of model invalidity exist. They include errors due to numerical round-off, missing data, altered data, and sometimes the presence of extraneous or redundant data.

The *de facto* industry standard STL model representation defines models as a set of triangular facets [1]. Unfortunately, these facets are stored independently, as if each facet were created and tossed into a bucket with no particular ordering and without information relating a given facet to any other facets in the bucket. Since many CAD systems fail to generate valid facet model tessellations, it is necessary to perform model validity checking before subsequent processing operations are undertaken. Given only the data in an STL file, performing model validity checking is computationally expensive. Attempting to

determine the relationships, or topology, between model facets from the “bucket of facets” is the first step in performing validity checking. The resulting topological information is important for use in subsequent processing operations such as model repair, model slicing, and finally during the scan conversion operation.

2 Topology

Topology describes the connectivity relationships between various geometric entities [2]. A facet can reference the three edges which bound it. Each edge can reference the two vertices which define it. Topological connectivity relationships are not limited to individual facets. For instance, a facet can reference the three facets which share edges with it. An edge can reference not only the two vertices which define it but also the two facets which share it. Vertex points can contain connectivity information to all edges or faces which share it. Such references are all examples of topological connectivity information.

It is important to consider two topological classes of boundary representation models: manifold (two-manifold) and non-manifold. A two-manifold is defined as a two-dimensional, connected surface where each point on the surface has a neighborhood topologically equivalent to an open disk [6]. In a two-manifold, every edge in the model is shared by two and only two facets. This is the case for most facet models where only the facets representing a part’s spatial boundaries exist. One side of a facet is directed toward part material, and the other is directed away from it. The spatial boundaries of a facet model are expected to have a distinct “inside” and “outside” which is consistent across all facets defining the model boundary; such a model is termed *orientable* [2]. Non-manifold conditions occur where, for example, two distinct enclosing volumes share one facet or a set of facets as a common boundary [7]. In this case, the shared facet no longer has a clear “inside” or “outside”. Both sides of the facet are surrounded by part material. This typically occurs when facets representing multiple solids which are tangent along some boundary are not properly delimited as belonging to individual solids in an STL file.

3 Benefits of Topological Information

The STL format represents facet models with nearly the minimal information necessary to define a solid object. Each facet, along with its normal, is specified explicitly and no topological connectivity information is provided. This “bucket of facets” approach to model representation has many limitations, both with respect to ensuring valid models and subsequent processing.

3.1 Validity Checking

When no topological information is provided, model validity checking involves computationally expensive searching operations. If model topology were available, validity checking would be a much simpler and efficient operation.

3.2 Model Repair

When invalid models are encountered, topological information is useful for attempting to repair the models. Such information makes it readily apparent when greater than two facets share a single model edge. In the case of model holes, it is important to know how the facets surrounding a hole are connected and this too can be easily determined given topological information.

3.3 Subsequent Processing

Facet model slicing performance can also benefit from topological information which makes it possible to march from facet to neighboring facet performing simple edge/plane intersection calculations [8]. This same topological information can be passed to subsequent processing phases, such as scan conversion, which occur after slicing.

3.4 Model Representation

Finally, model topology generation capability has facilitated the development of a richer facet model representation format [9]. Storing topological information with a facet model, although increasing the information content of the model, reduces the volume of data required when compared to an equivalent model represented in STL format. The net result is a more robust representation with less data.

4 Topology Reconstruction Concepts

Reconstructing model topology given a “bucket of facets” is basically a searching operation. Entity relationships must be found by searching the unordered model data. These relationships, or topology, must then be stored for later use. The conceptual steps for producing such topology will now be discussed; however, implementation details, such as the searching algorithms or structures used, will be dealt with separately after the topology reconstruction concepts are understood.

4.1 Vertex Merging

Each facet is defined by three vertices whose coordinates are explicitly specified. The first operation performed when reconstructing model topology is *vertex merging*. Here, equivalent, explicitly specified vertices are replaced by a single entry in a list of unique vertices. Each face then references three vertices in the list instead of being defined by actual vertex values. This removes significant redundancy present in the model representation. It also allows vertex comparisons to be made based on vertex references without comparing actual floating point coordinate values. Figure 4.1 illustrates the savings realized by using model topology and representing each vertex uniquely.

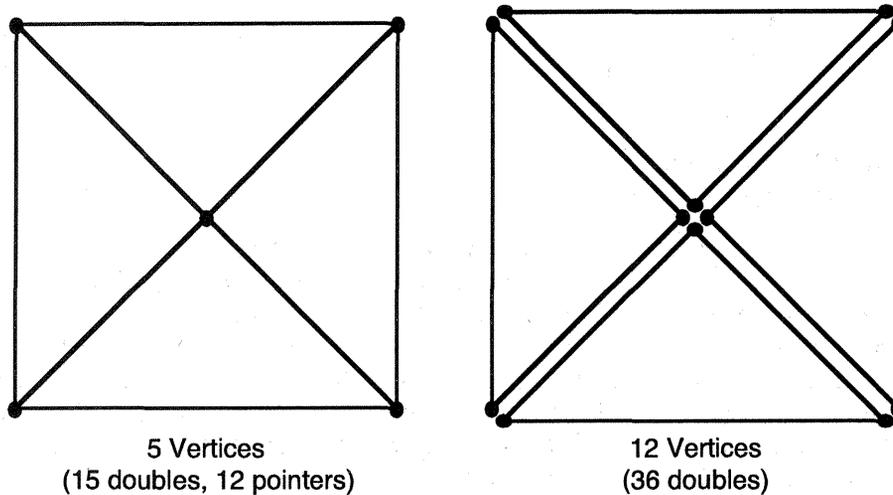


Figure 4.1 - Storage Reduction with Topological Information

Without topological connectivity information, the vertex shared by the four facets shown would be represented four times. Assuming a vertex is specified by three double precision values, and a double precision value requires eight bytes, the four facets shown could be represented by 288 bytes (4 facets x 3 vertices/facet x 3 doubles/vertex x 8 bytes/double). By using topological information, a unique definition of each vertex can be referenced by the facets using pointers. If a pointer consumes four bytes, this will reduce the memory required to represent the four facets shown to 168 bytes (5 vertices x 3 doubles/vertex x 8 bytes/double + 4 facets x 3 pointers/facet x 4 bytes/pointer). This figure shows a significant storage savings where only four faces meet. However, in a real model which is closed and likely more complex, many facets will share each single vertex. This will result in even greater savings by using topology and representing each vertex uniquely.

The final benefit of vertex merging is that vertices within a predetermined numerical round-off tolerance of each other can be easily merged, and this can be used to overcome errors introduced by inconsistent numerical round-off. Such errors can occur when a slightly different sequence of mathematical operations is used to calculate the same vertex value. For example, given three finite precision binary numbers A, B, and C, the operation $A + B + C$ may not produce the same result as the operation $A + C + B$ due to rounding or chopping errors [10]. Figure 4.2 provides a two-dimensional example of how vertex merging can also remove facets smaller than the size of the numerical round-off tolerance.

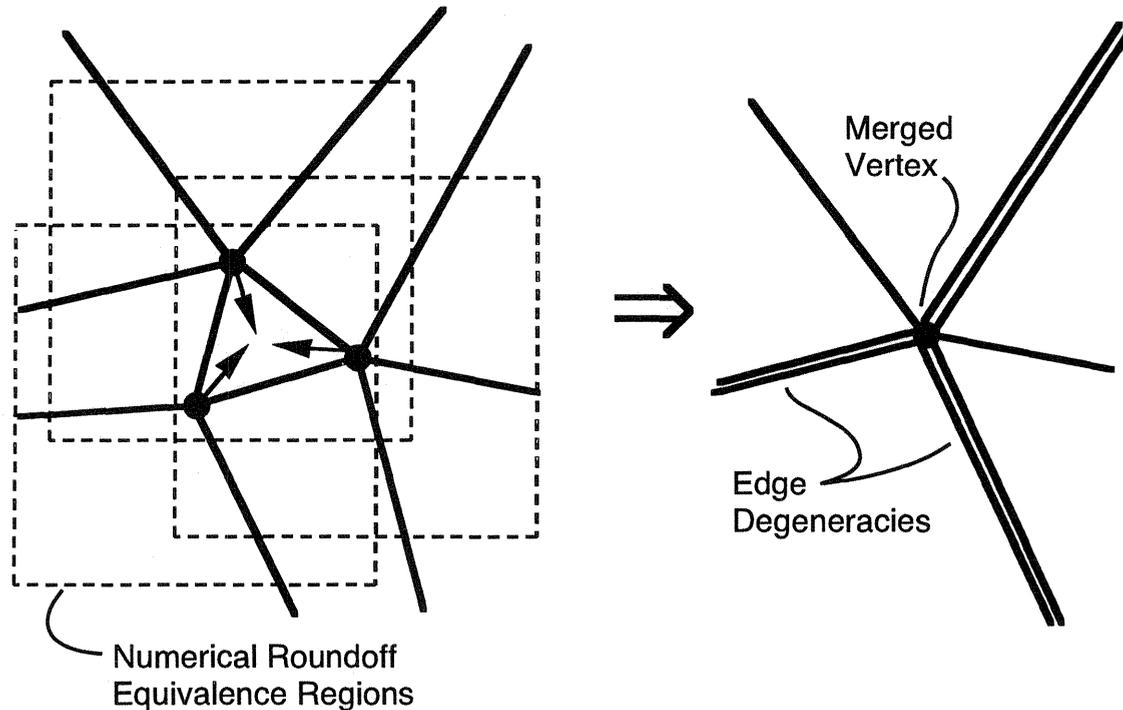


Figure 4.2 - Facet Removal by Vertex Merging

The facet defined by the three vertices shown is smaller than the specified numerical round-off tolerance. The squares drawn with dashed lines and centered on each of these vertices indicate the equivalence regions defined by the numerical round-off tolerance setting. Since the vertices are within the equivalence regions, they are considered equivalent and should be merged into one vertex. This is illustrated by the three arrows pointing toward the center of the left hand figure. Notice that as the three facet edges collapse, the remaining edges of the three adjacent facets will approach each other.

The figure on the right shows what happens to the model after the three equivalent vertices have been replaced, by merging, with one new vertex. The three facets which were adjacent to the facet removed by vertex merging now have two identical vertices (the vertex resulting from the merging operation). Note that this creates a degenerate condition where three of the facets surrounding the merged facet have effectively collapsed to an edge. The facet topology is still present; however, two facet vertices are the same in each of the degenerate facets.

4.2 Face & Edge Creation

A face entity must be created to represent each facet in the model. Since vertices are represented uniquely, each face maintains references to the three vertices which define it. Each face entity also carries with it a facet normal. This aids in determining which side of the face is inside the model; however, this information could also be computed if it were not given, provided the right-hand rule for vertex definition is adhered to [1].

Similarly, it is possible to create edge entities to represent model edges. Each edge can reference the two vertices which define it and the two faces which share it. Edge information is not provided by STL format files, but it can have utility when slicing facet models and can be represented using alternative file formats [3]. The following section will show that it is appropriate to wait until face relationships have been determined before creating model edges.

4.3 Determining Face & Edge Relationships

When face entities are created, they not only contain references to the defining vertices and normal, but they also contain unassigned references to the three adjacent faces and corresponding edges. Figure 4.3 provides a graphical depiction of adjacent face and edge references.

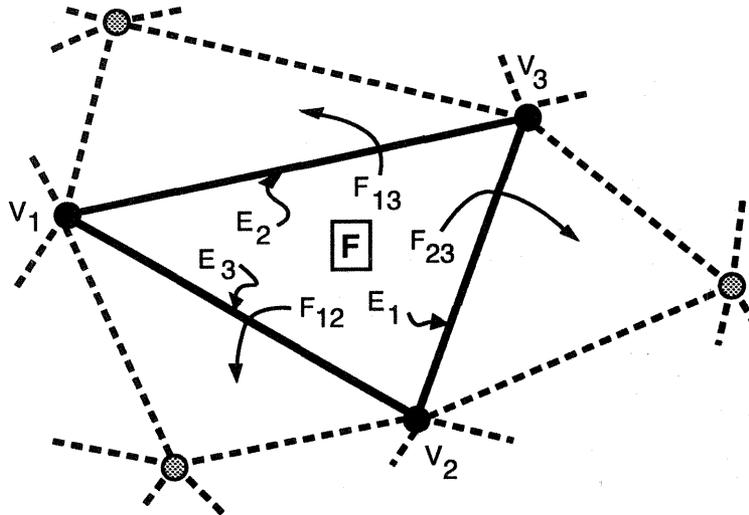


Figure 4.3 - Adjacent Face and Edge References

These references will be assigned as model topology is determined. After all face relationships are determined, the relationships between model edges can be determined using the face relationship information.

Face relationships are determined by searching. For each face in the model, searching is performed to determine what other faces share two common vertices. The existence of a pair of such faces defines an edge. When such a face is located, the adjacent face references of each face corresponding to the shared edge are cross-referenced. This establishes one topological relationship between two faces. This process is repeated until all adjacent face references are set. Notice that this assumes each model edge is shared by exactly two faces; invalid models exist where this is not the case.

After defining all face adjacency relationships, it is possible to define model edges. The three vertices which define each face and the three adjacent faces of each facet are known. Edge topology can be derived by using this information. The three edges of each face, defined by the face's vertex pairs, can be added to an edge list if they do not already exist. When an edge is added, the vertices which define it are known, so its vertex references can be set accordingly. One of the faces which shares the edge is immediately known, as this is the face being used to reference the edge's defining vertices. The other face which shares the edge is also known after the face adjacency relationships have been determined. By sequencing through all faces in the model after the face adjacency relationships are known, complete edge topology can be created without additional searching.

5 Data Structures & Algorithms

The fundamental operations used to reconstruct model topology have been discussed, but few operational details were provided. If brute force approaches to vertex merging and adjacent facet searching are used, computational cost will quickly become prohibitive as model complexity increases. Consequently, care must be taken to ensure the data structures and algorithms employed facilitate efficient searching.

Facets are read sequentially from a data file with no particular ordering. Each time data representing a facet is read, a face entity is created. Three vertex entities, each of which may already exist or need to be created, are referenced by each face entity. A linear list, although a poor structure for sorting and searching, can be used to store the face entities because they are not searched directly. Vertex entities must, before they are created, be tested to ensure an equivalent vertex does not already exist. If it does, this equivalent vertex should be referenced by the newly instantiated face entity instead of creating a new vertex entity and referencing it. A searching operation is required to perform this equivalence testing, and it is repeated for each model vertex. Repeatedly searching, and maintaining in sorted order, a linear list of vertex entities is a costly proposition. An alternative to the linear list data structure should be used.

5.1 The AVL Tree for Vertex Merging

With the goal of vertex merging in mind, a linear list is clearly an unacceptable solution for storing the unique vertices. Each time a vertex entity is added, the complete list would have to be sorted and/or searched. Recall that one requirement for vertex merging is that all *equivalent* vertices be merged. This suggests that for a vertex entity to be added, there must be no existing equivalent vertices. Consequently, a range of vertex values in the neighborhood of the search key must be tested. This makes a hash table an unattractive storage structure. A sorted array could be used, but because vertex data is continually being added as each model facet is read, repeated insertions into the array and the block moves which result make this an unattractive solution. A binary search tree could be used; however, the input data ordering is unknown. In the degenerate case, this could be equivalent in performance to searching a linear list of vertex values, and this is unacceptable.

A balanced binary search tree, termed an AVL search tree, overcomes these limitations [11]. It never degenerates to the equivalent performance of searching a linear list because its balance is maintained as each element is inserted in it. It also meets the requirements for being able to traverse through the data and efficiently allow range searching, which is used to determine equivalence within a set numerical round-off tolerance after an initial search value is located.

As each vertex defining a face is read, it is added to an AVL search tree if an equivalent vertex does not already exist in the tree. If the vertex is added, the corresponding vertex reference for the face being added is set to this vertex element in the AVL tree. If an equivalent face exists, the corresponding vertex reference for the face being added is set to the equivalent vertex already in the AVL tree and the vertex data read is discarded. The AVL search tree data structure works well to provide an efficient equivalence detection mechanism for vertices being added. After vertex merging is completed, face adjacency relationships must be determined.

5.2 Searching for Face Adjacencies

Face comparisons are performed in an attempt to locate face adjacencies. Recall that face adjacencies are determined for each face in the model by sequentially stepping through a list of model faces, searching for adjacent faces, and cross-connecting references when adjacencies are located. A linear list of faces provides the ability to sequentially process faces, but it is clearly not the solution for an efficient searching structure.

The structure imposed to effect efficient adjacency searching is simply a list of face pointers attached to each vertex in the model. As new vertices are created or existing vertices are referenced by a given face, the reference value of this face is added to a list of face references for each vertex which defines the face. Consequently, each vertex knows of every face which references it after all model faces have been read from the data file. Searching for adjacencies is greatly simplified using this structure. Instead of searching every face in the model for a face which shares two vertices with a test face, now only the faces contained in the face lists corresponding to the three vertices defining a test face need to be searched.

If for each edge of a test face, all other model faces in a list structure were searched to determine adjacency relationships, this would be of order $O(3n^2)$ where n denotes the number of facets in the model. However, by maintaining for each vertex a list of faces which share it, searching complexity is reduced to an order $O(3nk^2)$ problem where n is again the number of facets in the model and k is the average number of faces sharing each model vertex. Figure 5.1 depicts the data available for a given face and is useful in understanding how adjacencies are determined.

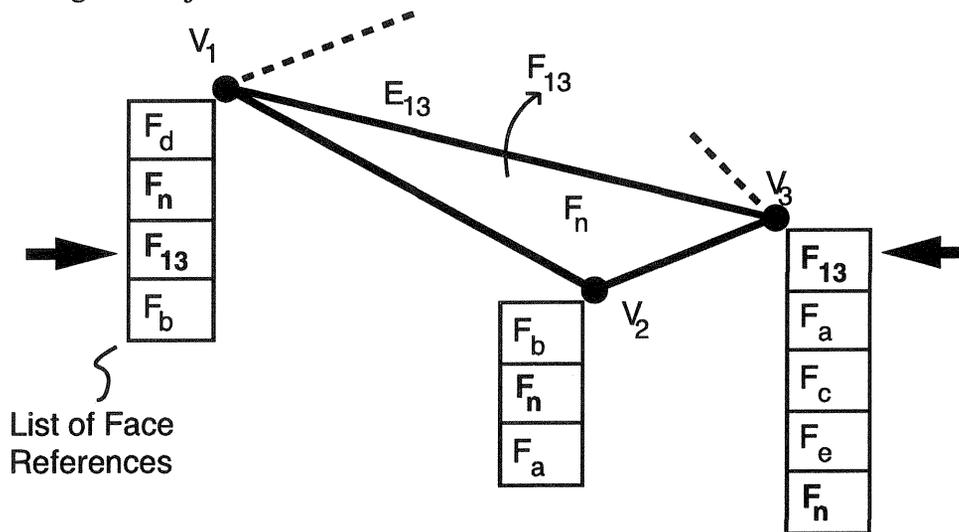


Figure 5.1 - Face Data Used for Adjacency Searching

Consider the searches necessary to locate the facet adjacent to face F_n which shares edge E_{13} . Instead of searching all facets in the model, the search is confined to the lists of face pointers maintained by vertices V_1 and V_3 . The reference to face F_n should appear in the three vertex lists; however, there should be one and only one other identical face reference contained in both the face lists for V_1 and V_3 . This should reference the face which shares edge E_{13} with the facet F_n under test and is labeled F_{13} in the figure. This provides a useful way to bound the search space which must be traversed each time an adjacency must be sought. Since in most cases, the number of facets in a model far exceeds the number of facets sharing a particular model vertex, this provides a significant reduction in the number of elements which must be searched throughout the adjacency determination process. It is also important to note that only face references, not actual vertex coordinate triples, must be tested during searching. This too contributes to the efficiency of this algorithm.

6 Results & Conclusions

There is a definite up-front cost associated with generating topological information from a "bucket of facets" model representation such as the STL *de facto* standard.

However, an investment in such pre-processing adds information which can be used to realize a healthy dividend in later model processing operations.

6.1 Results

Topology generation, given only a “bucket of facets”, can be computationally prohibitive if a brute force approach is taken. Appropriate data structures and algorithms must be used to make topology generation an profitable exercise. An important lesson illustrated by this work is that a marginal increase in storage requirements can effect a significant increase in processing performance.

Storing additional face reference lists for each vertex increases memory requirements; however, it provides a very necessary performance increase. By using an $O(3nk^2)$ algorithm instead of an $O(3n^2)$ algorithm, processing time for model topology generation is significantly reduced. For a model with approximately 82,500 facets, 8.6 CPU hours on a Sun SPARCstation2 were required to generate its topology when the brute force $O(3n^2)$ algorithm was employed. However, the $O(3nk^2)$ algorithm performed the same task in only 9.1 CPU minutes on the same machine. It is important to note that these performance figures are for an ASCII format STL file. Preliminary evaluation of an optimized version of the parser for binary format STL files suggests that the contrast between the brute force and AVL tree approaches is even larger because parse time contributes significantly to the 9.1 CPU minute figure cited above[12].

6.2 Conclusion

Model topology is clearly important for successful model processing, and generating it can be a costly endeavor. Unfortunately, the SFF community’s STL *de facto* standard for representing facet models does not support the definition of model topology. It must instead be reconstructed from an unordered “bucket of facets” provided by the STL representation. This requires significant searching through model facet data which is typically very voluminous for real part models. The approach presented in this paper realized better than an order of magnitude performance improvement when contrast to a brute force topology generation approach.

Ideally, topological information should be generated during model tessellation which is most likely performed by a CAD system. This would not only prevent the need to reconstruct it later, but it would also make data transfer more robust and concise. The RPI format demonstrated a significant redundancy reduction while increasing the information content in a file by utilizing topological information [9]. While facet models may fade as the primary representation for future SFF data exchange, topological information and some of the lessons learned from this work will remain important for higher-order geometrical entities such as parametric surface patch models.

Acknowledgments

This research was supported by NSF Grant DDM-8914212 as a subcontract through the University of Texas Solid Freeform Fabrication program, the New York State Center for Advanced Technology, the Office of Naval Research, and other grants of the Rensselaer Design Research Center (RDRC) Industrial Associates Program. Any opinions, findings, conclusions, or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation, the New York State Center for Advanced Technology, the Office of Naval Research, or any of the industrial sponsors.

We would like to thank Dick Aubin, Pratt & Whitney (a division of United Technologies), for providing a number of industrial STL models. A special thanks to James Miller for all the valuable comments and ideas on early drafts of this paper, and to Jan Helge Bøhn for his review and helpful comments.

References

1. "Stereolithography Interface Specification," 3D Systems, Inc., June 1988.
2. Michael E. Mortenson, *Geometric Modeling*, John Wiley & Sons, Inc., 1985.
3. Stephen J. Rock, "Solid Freeform Fabrication and CAD System Interfacing," M.S. Thesis, Rensselaer Polytechnic Institute, Troy, NY, Dec. 1991.
4. James V. Miller, "On GDM's: Geometrically Deformed Models for the Extraction of Closed Shapes from Volume Data," M.S. Thesis, Rensselaer Polytechnic Institute, Dec. 1990.
5. Donald Hearn and M. Pauline Baker, *Computer Graphics*, Prentice-Hall, Inc., 1986.
6. Kevin Weiler, "Edge-Based Data Structures for Solid Modeling in Curved-Surface Environments," *IEEE Computer Graphics and Applications*, Vol. 5, Num. 1, pp. 21-40, Jan. 1985.
7. Kevin Weiler, "The Radial Edge Structure: A Topological representation for Non-Manifold Geometric Boundary Modeling," in: *Geometric Modeling for CAD Applications*, M. J. Wozny, H. W. McLaughlin, J. L. Encarnacao (eds.), North-Holland, pp. 3-36, 1988.
8. Stephen J. Rock and Michael J. Wozny, "Utilizing Topological Information to Increase Scan Vector Generation Efficiency," in: *Solid Freeform Fabrication Symposium Proceedings*, H.L. Marcus, J. J. Beaman, J.W. Barlow, D.L. Bourell, and R.H. Crawford (eds.), The University of Texas at Austin, Aug. 1991.
9. Stephen J. Rock and Michael J. Wozny, "A Flexible File Format for Solid Freeform Fabrication," in: *Solid Freeform Fabrication Symposium Proceedings*, H.L. Marcus, J. J. Beaman, J.W. Barlow, D.L. Bourell, and R.H. Crawford (eds.), The University of Texas at Austin, Aug. 1991.
10. Kendall Atkinson, *Elementary Numerical Analysis*, John Wiley and Sons, Inc., 1985.
11. Daniel F. Stubbs and Neil W. Webre, *Data Structures with Abstract Data Types and Pascal*, Brooks/Cole Publishing Company, 1989.
12. Jan Helge Bøhn, Personal Communication, June 30, 1992.