A Flexible File Format for Solid Freeform Fabrication

Stephen J. Rock
Michael J. Wozny

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

Abstract

A flexible file format for Solid Freeform Fabrication data is presented which significantly improves on the de-facto industry standard *STL* format. The new format removes the redundancy present in *STL* files and can contain topological information. Its specification flexibility allows users to balance storage and processing costs. Since facet boundary models currently provide the greatest common denominator for data exchange between many CAD systems, they are supported by this format. Additionally, representation of CSG primitives is provided, as are capabilities to represent multiple instances of both facet and CSG solids. Format extensibility, without obsoleting existing programs, is made possible by interleaving the format schema with the data. User data can be added to existing entities, or new entities can be created. This allows the addition of NURBS based geometries in the future.

1 Introduction

No standard has been formally adopted for transfering Solid Freeform Fabrication (SFF) data. The STL facet model format specified by 3D Sytems has become the de-facto industry standard in the SFF community [1].

1.1 STL Format Limitations

Solid objects are represented as an unordered collection of triangular planar facets, and each facet has an outward directed facet normal associated with it. Multiple solids are allowed, although it is unclear from the interface specification whether these are allowed to intersect other solids in the model.

Given only the data in an STL file, ensuring a model is physically realizable and is not missing any information (model validity checking) is computationally expensive. Infering model topology from this minimal information is also costly and the results are subject to ambiguity caused by numerical imprecision. This suggests the need for an alternative representation which carries greater information content and can reduce model preprocessing demands. STL files also carry a significant amount of redundant information. For each facet, vertex coordinates are specified explicitly. Process attributes are mentioned in the specification but never developed. Both ASCII and Binary formats exist and it is possible to translate between them. Additional constraints are placed on model data. Ordinate values can never be zero or negative, and all ASCII format keywords must contain only lower-case characters.

1.2 Higher Order Approximations

The need for an improved SFF model format and representation has been recognized [2]. Arguments were made in support of using surface approximations, termed "precise geometry" by the authors, instead of facet model approximations. Use of a single modeling entity was also recommended. These are valid suggestions; however, to make

use of truly precise geometry, information must not be lost by approximation when transfering CAD model data to the SFF process. Representing models by surface patch approximation will certainly be more compact than planar facet approximation. A surface patch approximation will also experience slower data volume growth as the geometry approximation error tolerance is decreased.

While surface approximations are representationally more efficient than their facet equivalents, other considerations must be analyzed before defining surface patch approximations as the new and only SFF modeling entity. Model validity checking will increase in complexity. Model slicing, while having fewer surface patches to intersect, will also increase in complexity. Surface patch approximations must be generated, and this too incurs computational cost. Triangular facet tessellations are also used in the field of computer graphics [3] and algorithms tailored to SFF are beginning to emerge [4]. Code development to support surface patch approximation will be more expensive than that for facet approximation.

1.3 IGES & PDES/STEP

CAD system data transfer standardization provides another representational alternative, and some standards exist. The Initial Graphics Exchange Specification (IGES) began as a standard for representing graphical information [5]. It has since evolved to include limited solid model representation capability with CSG primitives supported in Version 4.0 and boundary representation support in Version 5.0. Unfortunately, these representations are verbose [6]. The Standard for Exchange of Product Model Data (STEP) standard is under development and is to represent the complete product model, not just the geometry [7]. The generality of this representation is expected to make it verbose with capabilities beyond those needed here.

1.4 2D Slice Contours

The representational alternatives discussed provide greater information content that the STL Format file. Another alternative has been proposed which provides only slice contour data to SFF processes [8]. Two dimensional slice contours are expected to hold less information than a three dimensional model, and any compression gained by correlation in the third dimension will be lost. However, each slice contour can be provided as a precise representation. This moves the slicing process into the CAD system and eliminates the need to tessellate solid models into approximate facet representations. Unfortunately, this approach limits flexibility at the process controller and will likely increase data transfer volume. New slicing orientations and part nesting arrangements will be difficult to select at the process controller. Slicing in the CAD system effectively moves process control to the processor running the CAD system. Spending design engineering time dealing with SFF process control issues may not be cost effective. This approach also fails to recognize that certain non-CAD data sources, such as reconstructed models from medical imaging data, will likely continue to generate three dimensional facet model data [9].

1.5 Facet Models

Facet models currently serve as the greatest common denominator between the multitude of CAD systems in existence because many CAD vendors have developed translators capable of generating facet model representations. Facet representations are also very common when transforming scalar fields into three dimensional geometry (medical imaging, remote sensing, etc.)[10, 11]. Facet approximation tolerance thresholds can be decreased to achieve required model precision. While this increases storage demands, the trend of continually decreasing storage cost devalues the consequences.

Facet models are not proposed as the optimum solution to the SFF data exchange problem. They are only one alternative but should continue to be supported in future representation schemes because of their greatest common denominator property, both between CAD systems and other sources of geometrical models. CSG primitives are an

attractive representational addition to a new format because they are well defined. More general standards tend to be verbose. This suggests a data exchange format tailored to SFF applications is appropriate. In this paper, we present a flexible, extensible model format which supports both facet solids and CSG solids. It supports multiple instancing of a single solid object definitions as well as a number of SFF specific entities. Topological information is provided for facet models to increase validity checking and processing efficiency.

2 RPI Format Design Considerations

A number of design considerations for an improved SFF format, termed *RPI format*, are presented in this section. The RPI format must be derivable from currently accepted STL format data. A radical change which fails to maintain such compatibility will gain poor acceptance. The RPI format must provide an extensible framework for evolution. This includes provisions for adding new geometric entities, adding data to existing entities, and adding non-geometric process data. The new format must also allow flexibility, with respect to both format conformance and the amount of data stored. It is also important to eliminate the redundancy present in STL format files.

2.1 Upward Compatibility

Upward compatibility from STL format to RPI format is possible. The RPI format is capable of representing facet solids, but it includes additional information about facet topology. Topological information is maintained by representing each facet solid entity with indexed lists of vertices, edges, and faces. Instead of explicitly specifying the vertex coordinates for each facet, a facet can reference them by index number. This contributes to the goal of overall redundant information reduction.

2.2 Topological Information

Given an STL model, topological information can be inferred at some computational expense. This is also true for the RPI format and is the mechanism by which data volume flexibility is provided. Many possible combinations of topological data can be maintained for facet solids in an RPI format file. In the simplest case, only a vertex list and face list are provided. The vertex list is referenced directly by index values in the face list. When normal data is included, this is informationally equivalent to the STL format. It is not even necessary to specify facet normals as these can be computed based on vertex ordering assuming the right hand rule is followed. In its most complete form, a facet model can be represented by a vertex list, edge list, and face list. The edge list can reference the two vertices and two faces which define each edge. The face list can reference each faces vertices, edges, and three neighboring faces. Normals can also be specified for each face. Any range of topology specification between these extremes is permissible. When less topological information is provided, more CPU time will be required to generate it. This flexibility allows storage and processing costs to be appropriately balanced.

2.3 Conformance and Extensibility

Providing format conformance flexibility and format extensibility are tightly coupled objectives. Both are achieved by imbedding the format schema within the data. The schema defines the data which represents each entity. It not only defines variable names and defines the order in which they occur, but it also specifies the data types for each of these variables. This allows unrecognized variables to be skipped without rendering the remaining data unreadable. It is not always desirable to omit unrecognized data, as this could indicate a model error; however, assuming valid data, there are cases where omitting unrecognized data is very useful. For example, the graphics technique of Phong Shading utilizes vertex normal information which could be stored using the RPI format [12]. This same file should be readable by an SFF processor which is completely unaware of the existence of, or variable names used for, vertex normal data in the file. By associating a data type with each variable name in the file, data corresponding to

unrecognized keywords can be read and discarded. This mechanism also allows entire unrecognized entities to be skipped; a feature of the public domain Interchange File Format (IFF) specification [13]. This allows the format to evolve without rendering existing interpreters useless with each new format addition.

Imbedding format schema information in the data also allows the order in which data is presented to be altered. Instead of enforcing specific data record ordering, the variable names found in each schema definition determine the order in which data is presented. Similar flexibility also exists at the entity level. Most model entities can appear in any order, and it is the interpreter's job to derive the necessary information. This provides conformance flexibility while placing a greater burden on the RPI format interpreter.

New entities can be added to the format and new variables can be added to exiting entities. Data types, however, can neither be added nor modified while preserving the backward compatibility designed into the format. If an interpreter reads an unknown variable, it can successfully continue processing the data because it knows what to read and discard. If an unknown data type is used, the interpreter will not know how much information to read and is unable to continue processing. Consequently, data type selection during format specification is critical to ensure the format's successful evolution. The data types provided closely resemble those available in the C programming language [14]. Integers, single and double precision floating point values, strings of characters, and boolean operations are supported. Additionally, a binary type is defined which supports binary values of any length.

2.4 CSG Primitives

In addition to facet solids, the RPI format supports cuboid, cylinder, cone, sphere, and tori CSG primitives. These are common in mechanical design and one study found that more than 90% of an industrial sampling of parts could be constructed from them [15]. The primitive representation is also much more efficient than the corresponding facet approximation. Transformation entities are provided so that actual solid coordinates do not have to be individually repositioned. This is useful when orienting CSG primitives or when attempting to nest larger collections of solids. The same solid or collection of solids can be instanced multiple times without replicating the data. This serves to reduce data redundancy and provide flexibility. Finally, non-geometrical entities are provided. These allow process data to be maintained within the RPI format file. Included are entities which represent the process type, scanning methodology, and provide notes for the machine operator.

The benefits encompassed in the RPI format are not without cost. Developing an interpreter which processes a format as flexible and extensible as the one described is non-trivial. However, from a global perspective on effort, it seems complicating interpretation for a few SFF producers is worth the benefit of simplifying format conformance, and increasing storage and processing efficiency, for many times that number of end-users.

3 The RPI Format

An abridged specification of the RPI format is provided to illustrate the concepts previously discussed. The format was developed in ASCII to facilitate cross-platform data exchange and debugging, but attention has been given throughout format design to ensure a reasonable binary representation mapping is possible. An RPI format file is composed of a collection of entities, each of which internally defines the data it contains. Each entity conforms to the syntax defined by the syntax diagram shown in Figure 3.1[16].

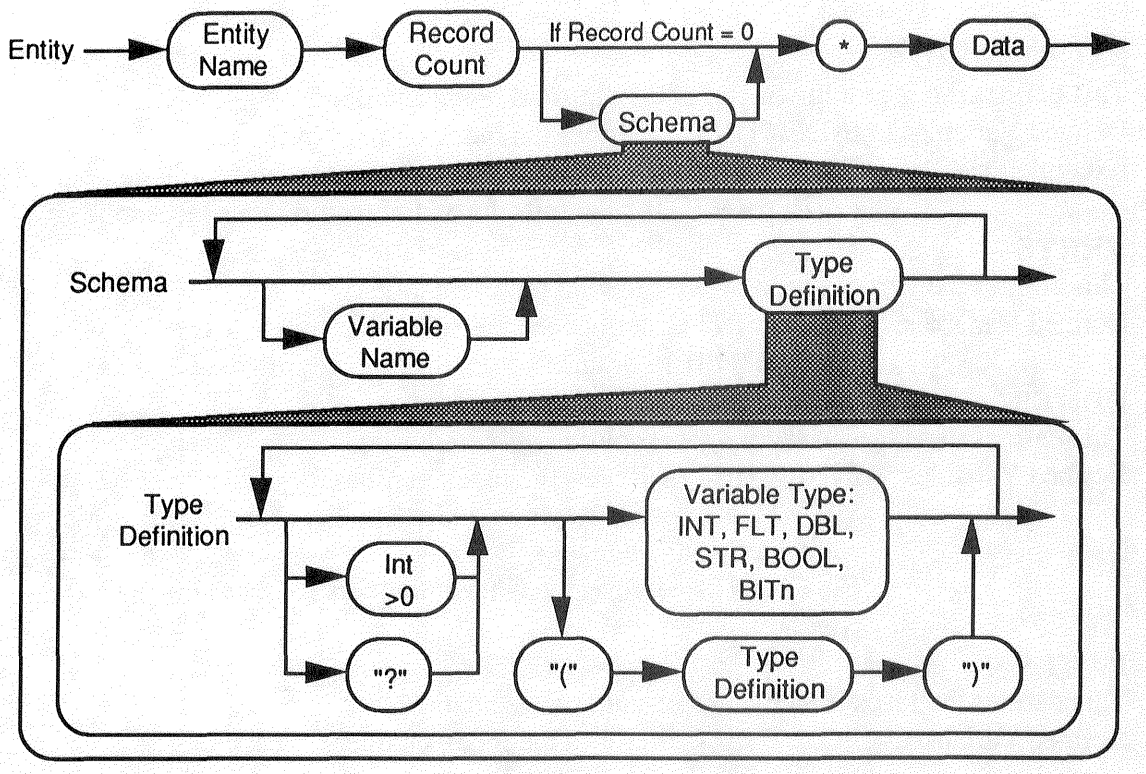

Figure 3.1 - RPI Format Entity Syntax Diagram

Each entity is composed of an entity name, record count, schema definition, schema termination symbol, and the corresponding data. The data is logically subdivided into records which are made up of fields. Each record corresponds to the definition provided by the Schema. Each field corresponds to one variable type in the Type Definition.

Variable types include integer (INT), single precision float (FLT), double precision float (DBL), string (STR), boolean (BOOL), and binary representation n bits in length (BITn) where n is a positive integer. All fields are separated by white space (space, tab, newline, etc.), so strings with spaces must be enclosed in quotes. Strings may include any of the standard escape sequences (\...) defined by the C programming language [14]. Any variable type, or type definition enclosed in parenthesis, can be preceded by either an integer or the "*?*" character. This represents a repeat count for the definition which it precedes. An integer specifies a constant repeat count, while the "*?*" character specifies a repeat count which varies from record to record. Each variable repeat count specification causes the data it defines to be preceded by one integer field which identifies the number of subsequent fields to be read for that particular record. Flexibility is increased by allowing variable names to be omitted; however, there use is encouraged. This feature necessitates that variable names be reserved words. The "*" character has been designated as the schema termination symbol, and data records are expected to immediately follow it. The syntax diagram for the Data is not shown because of its dependency on the Schema definition. C style comments (/* ... */) are supported and are valid everywhere except in quoted strings where they will be treated as part of the string.

3.1 Entity Definitions

Entity definitions exist for specifying facet solids, CSG solids, operations and transformations on these solids, and process specific data. Standard variable name definitions also exist for each entity. Space limitations dictate that only brief descriptions be given for each entity, and some features are not discussed. Some examples regarding facet model specification are omitted and can be found in section 3.2.

FACETSOLID
This defines the beginning or end of a facet solid representation.
Defined Variables: INDEX (INT) - Defines the index number of the facet solid.
Example(s): FACETSOLID 1 INDEX INT * *i*, where *i* is the solid index number.
FACETSOLID 0 * delimits the end of the previous facet solid.

VERTS
This defines the vertices used by a facet solid representation.
Defined Variables: INDEX (INT) - defines the index number for each vertex.
X, Y, Z (DBL) - defines vertex location.coordinates.

EDGES
This defines the edges used by a facet solid representation.
Defined Variables: INDEX (INT) - defines the index number for each edge.
V1, V2 (INT) - defines vertex index numbers for the two vertices which define an edge.
F1, F2 (INT) - defines face index numbers for faces meeting at the edge.

FACES
This defines the faces used by a facet solid representation.
Defined Variables: INDEX (INT) - defines the index number for each face.
V1, V2, V3 (INT) - defines vertex index numbers for the three vertices which define a face.
E1, E2, E3 (INT) - defines edge index numbers for the three edges which define a face.
F12, F23, F31 (INT) - defines face index numbers for faces adjacent to the face being defined. The numbering convention indicates the which edge a neighboring face is sharing.
NX, NY, NZ (DBL) - define the normal direction vector for the face.

CUBOID, CYLINDER, CONE, SPHERE, & TORI
These entities define the five CSG primitive types supported.
Defined Variables: INDEX (INT) - defines the index number for each primitive.
CX, CY, CZ (DBL) - defines the center of a primitive.
D1, D2, D3 (3DBL) - defines the direction vectors.
L1, L2, L3 (DBL) - defines lengths.
R, R1, R2 (DBL) - defines radii.
Obviously, not all are used in any single primitive definition.

CSGOP
Defines a boolean operation on CSG and facet solids.
Defined Variables: INDEX (INT) - defines the index number for the resulting solid.
I1, I2 (INT) - defines the index numbers for each solid (either a facet solid, CSG solid, TRANSform, or other CSGOPeration index number) to be operated on.
OP (STR) - defines the operation. Defined operations include *union*, *inter*section, *diff*erence, and *inv*ert. Invert uses only I1.

TRANS
Defines a transform operation which may be applied to any solid index number.
Defined Variables: INDEX (INT) - defines the index number of the transform result.
MAT44 (16DBL) - defines 16 entries of a 4 x 4 transform column by column.

CSGROOT
Defines the root index value of a hybrid CSG tree which can include facet models.
Defined Variables: ROOT (INT) - defines the index number of the root entity.

PROCESSDATA
Defines SFF process information.
Defined Variables: PART_NAME (STR)
PROCESS (STR)
MATERIAL (STR)
SCAN_METHOD (STR)

INCLUDE
Allows inclusion of other RPI Format data files containing non-conflicting entity index values.
Defined Variables: FILE (STR) - defines the filename, including path, for inclusion.

3.2 A Facet Model Example

An example of facet model definition using the RPI Format is provided. The data provided is an excerpt from the 7160 facet model referenced in Figure 4.1.

```
/* The following file is a .RPI Flexible Format Facet File */
PROCESSDATA 1 PART_NAME STR * tooth.rpi
PROCESSDATA 1 PROCESS STR * SLS
PROCESSDATA 1 MATERIAL STR * Polycarbonate
PROCESSDATA 3 SCAN_METHOD STR * boundary halflap orientmod
FACETSOLID 1 INDEX INT * 1
VERTS 3582 INDEX INT X DBL Y DBL Z DBL *
1 83.6787 121.7 27.7256
2 84.3622 121.356 28.2707
3 84.2698 122.023 27.5492
...
3580 125.606 100.531 81.6959
3581 122.387 95.68 83.1261
3582 122.708 106.524 82.3616
EDGES 10740 INDEX INT V1 INT V2 INT F1 INT F2 INT *
1 3483 3482 1 1792
2 3482 3480 1 1793
3 3483 3480 1 1791
...
10738 11 18 7147 7151
10739 6 2 7149 7158
10740 3 2 7150 7160
FACES 7160 INDEX INT V1 INT V2 INT V3 INT
E1 INT E2 INT E3 INT F12 INT F23 INT F31 INT
NX DBL NY DBL NZ DBL *
1 3483 3482 3480 1 2 3 1792 1793 1791 0.755175 0.596069 0.272787
2 3350 3348 3347 4 5 6 1794 1795 1796 0.378247 0.844803 -0.378467
3 2387 2594 2592 7 8 9 1797 1798 1799 0.101944 -0.879524 -0.464806
...
```

```
7158 6 4 2 5370 10719 10739 7125 1790 7149 -0.083132 -0.760829 -0.643605
7159 5 1 4 5368 10720 10734 7139 1790 7127 -0.148119 -0.821666 -0.550387
7160 3 2 1 5369 10732 10740 7150 1790 7138 0.184947 -0.709709 -0.679785
FACETSOLID 0
```

The entities currently defined do not make full use of the syntax flexibility provided for defining schemas. None utilize the variable repeat count facility; however, now unused features should not be ignored when developing parsers for the RPI format. If an application needed to reference each face sharing a particular vertex, for example, variable repeat count capability would be very useful.

4 Results

A conversion capability has been developed which allows RPI format files to be generated from STL format data. This is a computationally expensive task because model topology must be inferred from the unordered STL model facets. All work has been done with ASCII format files. Comparisons drawn should remain valid when binary formats are used because the ASCII and Binary representations do not differ significantly in organization. Figure 4.1 contrasts the file size of STL Format files to corresponding RPI Format files.

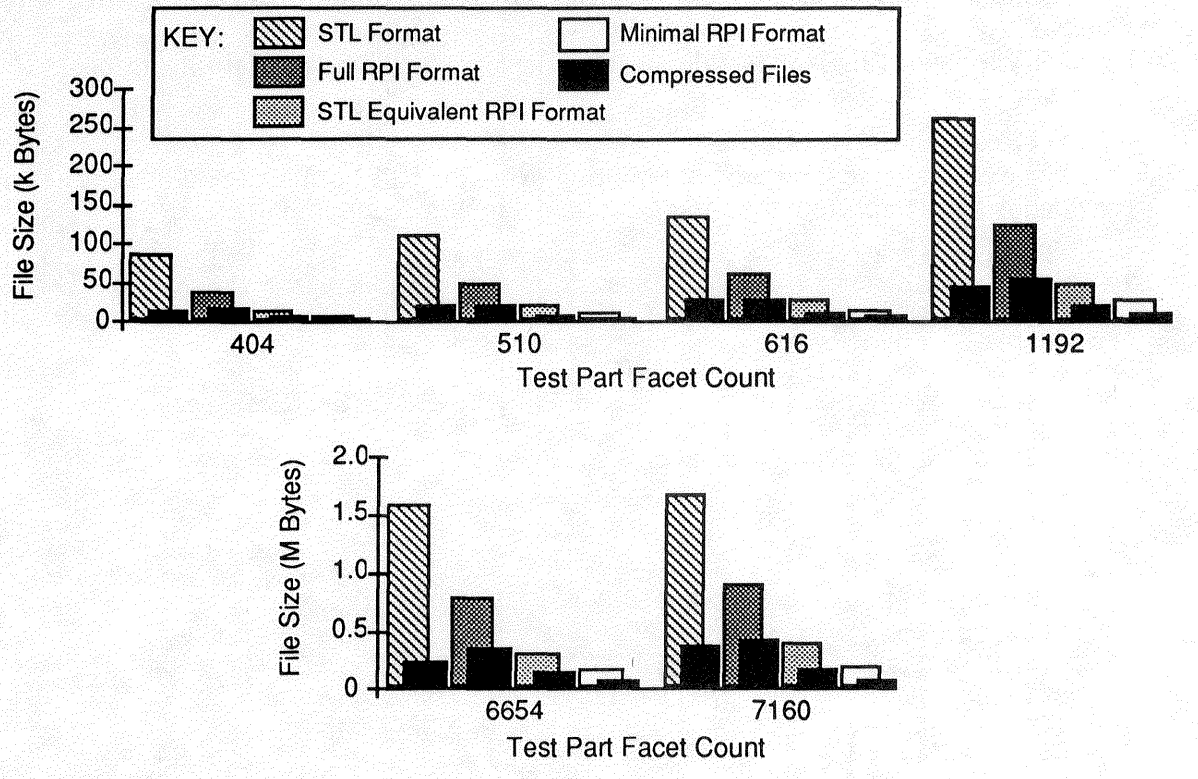

Figure 4.1 - STL Format/RPI Format File Size Comparisons

The Full RPI Format file contains the full range of topological connectivity information needed to make model slicing more efficient with the topological slicing approach of [17]. The experimental topological slicer implementation interprets a sub-set, supporting only single facet solids, of the RPI format specification previously defined to avoid the costly step of inferring model topology from an STL file. The STL Equivalent RPI Format file contains information content equivalent to that of an STL file, and the Minimal RPI Format file contains the minimal information necessary for model representation. Only the Full RPI Format file is suitable for active use; however, since it can be generated at some computational expense from one of the other formats, this allows

a balance between storage and processing costs. The Minimal RPI Format file should prove useful for archival storage.

The UNIX Compress utility was used to compress each of the files using adaptive Lempel-Ziv coding [18]. The amount of compression provides some indication of how much redundancy is present for each of the formats. Notice that while the Full RPI Format file averages approximately 50% the size of the corresponding STL Format file, the compressed STL file is typically marginally smaller than the compressed Full RPI Format file. This highlights the redundancy present in STL Format files. Also note that the compressed STL Equivalent RPI Format file is smaller than the compressed STL file in all experimental cases shown. This suggests it may be suitable for archival purposes.

5 Conclusion

5.1 Conclusions

The RPI Format offers a number of features unavailable in the STL Format. The format can represent CSG primitive models as well as facet models. Both can be operated on by the boolean union, intersection, and difference operators. Provisions for solid translation and multiple instancing are also provided. Process parameters, such as process type, scan method, material, and even machine operator instructions, can be included in the file. Facet models are more efficiently represented as redundancy is reduced. The flexible format definition allows storage and processing costs to be balanced.

While these features are useful, perhaps the largest benefit of this format is its extensibility. It provides for backward compatibility and allows users to include additional non-SFF data, or new SFF data, in a file. Emphasis was placed on RPI Format design considerations, not on specification of the defined entities, because this paper does not attempt to define a new standard. A sub-set of the format specified has been used to our benefit; however, it is by no means complete. The goal of this work is to provide an extensible format which can evolve with new representational enhancements. This should lessen the number of distinct file formats in use and eliminate the translators necessary to convert between such formats.

5.2 Future Work

In the near term, a binary format equivalent to the ASCII RPI format presented must be developed. Model precision requirements should be investigated to determine what representational precision is necessary in a binary format. This is less of an issue in ASCII format because any precision can be represented. It would also be useful to develop a method for creating new data types which do not disable backward compatibility.

Research must be conducted to identify a surface patch suitable for solid approximation. It is also important to develop basic algorithms which can approximate CAD system entities with these surface patches. This resurrects the issue of primitive level CAD system interfacing and the cost associated with developing such an interface. As the PDES/STEP standard is developed, it should be evaluated as the potential primary representation for SFF data transfer. It may be appropriate to allow a variety of entry points to SFF processes; from high level CAD data through scan vector input to the process controller. Advancements in controller technology may also influence the direction of future SFF model formats.

Acknowledgements

This research was supported by NSF Grant DDM-8914212 as a subcontract through the University of Texas Solid Freeform Fabrication program, 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 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. Research by James Miller, RDRC, generated the facet model of the tooth nerve cavity from CT scan data provided by William Lorensen, General Electric Corporate Research and Development. A special thanks to Charles Gilman and James Miller for all the valuable comments and ideas on early drafts of this paper.

REFERENCES

1. "Stereolithography Interface Specification," 3D Systems, Inc., June 1988.

2. James Darrah and Martin Wielgus, "A New CAD Model Format for SFF Machines?," Soild Freeform Fabrication Symposium Proceedings, University of Texas at Austin, Aug. 1990.

3. Reed D. Clay and Henry P. Moreton, "Efficient Adaptive Subdivision of Bezier Surfaces," *Eurographics '88*, Elsevier Science Publishers B.V., 1988.

4. X. Sheng and U. Tucholke, "On Triangulating Surface Model for SLA," Second International Conference on Rapid Prototyping, Dayton, OH, 1991.

5. "The Initial Graphics Exchange Specification (IGES) Version 5.0," National Institute of Standards and Technology, Gaithersburg, MD, 1990.

6. P.R. Wilson, I.D. Faux, M.C. Ostrowski, K.G. Pasquill, "Interfaces for Data Transfer Between Solid Modeling Systems," *IEEE Computer Graphics and Applications*, Jan. 1985

7. "Product Data Exchange Specification (PDES) Testing Draft," National Institute of Standards and Technology, Gaithersburg, MD, 1988.

8. Richard J. Donahue and Robert S. Turner, "CAD Modeling and Alternative Methods of Information Transfer for Rapid Prototyping Systems," Second International Conference on Rapid Prototyping, Dayton, OH, 1991.

9. 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.

10. William E. Lorensen and Harvey E. Cline, "Marching Cubes: A High Resolution 3D Surface Construction Algorithm," *Computer Graphics*, Vol. 21No. 4, July 1987.

11. Lori Scarlatos and Theo Pavlidis, "Hierarchical Triangulation Using Terrain Features," Visualization 90 Conf. Proc., Oct. 1990.

12. J.D. Foley, A. vanDam, S.K. Feiner, J.F. Hughes, *Computer Graphics Principles and Practice*, second ed., Addison-Wesley Publishing Co., 1990.

13. "'EA IFF 85' Standard for Interchange Format Files," Commodore-Amiga Technical Reference, pp. I-5 - I-24, 1985.

14. Brian W. Kernighan and Dennis M. Ritchie, *The C Programming Language*, Prentice Hall, Englewood Cliffs, NJ, 1988.

15. M.J. Pratt, "Solid Modeling and the Interface Between Design and Manufacture," *IEEE Computer Graphics and Applications*, Jul. 1984.

16. Derick Wood, *Theory of Computation*, John Wiley & Sons, Inc., 1987.

17. Stephen J. Rock and Michael J. Wozny, "Utilizing Topological Information to Increase Scan Vector Generation Efficiency," to appear in Soild Freeform Fabrication Symposium Proceedings, University of Texas at Austin, Aug. 1991.

18. "SunOS Reference Manual," 800-1751-10, Revision A, Sun Microsystems, Inc., 1988, p.83.