Proceedings of DETC02
2002 ASME Design Engineering Technical Conferences
September 29, 2002, Montreal, Quebec, Canada

DETC2002/DAC-xxx

THE EVOLUTION OF A LAYERED MANUFACTURING INTERCHANGE FORMAT

Sara McMains[*]
Mechanical Engineering Department
University of California, Berkeley
Berkeley, California, 94720-1740
Email: mcmains@me.berkeley.edu

Jordan Smith
Carlo Séquin
Computer Science Department
University of California, Berkeley
Berkeley, California, 94720-1776
Email: {jordans | sequin}@cs.berkeley.edu

ABSTRACT

Over the last several years we have developed the Berkeley Solid Interchange Format (SIF) for layered manufacturing data exchange. By building both design software that outputs SIF as well as manufacturing software that processes the SIF input files, we gained insights into the concerns of both sides of data exchange – insights which often led to major changes in successive versions of the format. In this paper, we share some of the most important lessons we learned (many of which are applicable to all geometric data exchange, not merely for layered manufacturing) and explain how they shaped SIF.

INTRODUCTION

Designers who want to make prototypes of solid three-dimensional parts directly from CAD descriptions are increasingly turning to a class of technologies collectively referred to as layered manufacturing (LM) or rapid prototyping. These technologies include stereolithography (SLA), 3-D printing, fused deposition modeling (FDM), selective laser sintering (SLS), and laminated object manufacturing (LOM)(Beaman97). In all these processes, a triangulated boundary representation (b-rep) of the CAD model of the part is sliced into horizontal, 2.5-D layers of uniform thickness. Each cross sectional layer is successively deposited, hardened, fused, or cut, depending on the particular process, and attached to the layer beneath it. (For technologies such as SLA and FDM, a sacrificial support structure must also be built to support overhanging geometry.) The stacked layers form the final part.

The computer representation of the part plays a central role in this process. While a human can easily interpret a shaded 3D surface model or a display of a wireframe model with dashed hidden lines, a solid model that unambiguously defines the region inside of the part is necessary for layered manufacturing. For a b-rep such as the STL format that has become the de facto standard in the LM industry, the boundary must be watertight, oriented, and not self-intersecting. Unfortunately, STL files commonly contain errors such as cracks and penetrating or extraneous faces. Service bureaus that manufacture LM parts typically massage and clean up these files to produce sanitized and consistent models which then are then used as input to the fabrication machine's software. If the original file is highly inconsistent, then the manufacturers have to make an educated guess as to what the intent of the original designer was and what the desired geometry might have looked like. They will then try to approximate that geometry as best possible with a clean STL description.

In this context we developed the Berkeley Solid Interchange Format (SIF) (McMains99) to serve as a replacement interface between designers and fabricators of LM parts. During this project we built both design software that outputs SIF as well as manufacturing software that processes the SIF input files. Building this software helped us gain insights into the concerns of both sides of data exchange – insights which often led to major changes in successive versions of the format. In this paper, we share some of the most important lessons we learned (many of which are applicable to all geometric data exchange, not merely for layered manufacturing) and explain how they shaped SIF.

[*]Corresponding author.

RELATED WORK

In this section we describe several existing interchange formats that have been used for layered manufacturing.

STL

The industry de facto standard for exchanging part descriptions for layered manufacturing is the STL format. STL is a boundary representation that consists of a simple list of triangular facets. The vertex coordinates are specified explicitly for each triangle in which the vertex appears. The vertices are enumerated in counter clockwise order as seen from the exterior of the part. In addition, for each triangle, a surface normal that points to the exterior of the part is specified. Figure 1 shows an example of an excerpt from an STL file for a cube centered at the origin.

STL files come in two types, ASCII and binary. Although the ASCII version uses an organization and keyword choice that suggests the possibility of defining non-triangular facets with multiple loops and grouping them into multiple solids, the binary format has no way of capturing this information. The binary format consists of only a header string, the number of triangles to follow, and a list of four 3-D coordinates for each triangular facet, defining the normal and the three vertices.

```
solid ascii

facet normal 0.0000 1.0000 0.0000    facet normal 1.0000 0.0000 0.0000    facet normal 0.0000 -1.0000 0.0000
outer loop                           outer loop                           outer loop
vertex 1.0000 1.0000 1.0000          vertex 1.0000 1.0000 1.0000          vertex -1.0000 -1.0000 -1.0000
vertex 1.0000 1.0000 -1.0000         vertex 1.0000 -1.0000 1.0000         vertex 1.0000 -1.0000 1.0000
vertex -1.0000 1.0000 -1.0000        vertex 1.0000 -1.0000 -1.0000        vertex -1.0000 -1.0000 1.0000
endloop                              endloop                              endloop
endfacet                             endfacet                             endfacet

...                                  ...                                  ...

facet normal 0.0000 0.0000 1.0000    facet normal 0.0000 -1.0000 0.0000   facet normal 0.0000 0.0000 -1.0000
outer loop                           outer loop                           outer loop
vertex 1.0000 1.0000 1.0000          vertex -1.0000 -1.0000 -1.0000       vertex -1.0000 -1.0000 -1.0000
vertex -1.0000 -1.0000 1.0000        vertex 1.0000 -1.0000 -1.0000        vertex -1.0000 1.0000 -1.0000
vertex 1.0000 -1.0000 1.0000         vertex 1.0000 -1.0000 1.0000         vertex 1.0000 1.0000 -1.0000
endloop                              endloop                              endloop
endfacet                             endfacet                             endfacet
                                                                          end solid
```

Figure 1. *An excerpt from an ASCII STL file for a cube.*

The shortcomings of STL are well known. It is redundant, both in repeating the coordinates of shared vertices in each triangle in which they appear, and in the specification of where the exterior of the part lies (the surface normal, as well as the ordering of the vertices, gives this information). This redundancy not only makes files unnecessarily verbose, but worse, it allows for inconsistency if the two methods of specifying the exterior do not agree. There are no rules associated with STL for resolving such inconsistencies if they arise. Because the vertices are not shared between triangles, gaps can be introduced between vertices that should be coincident. With no information about topology or connectivity included, it is impossible to communicate the designer's original intent when such gaps are present. Units are unspecified in STL, leading some manufacturers to guess the intended units based on the bounding box of the part compared to the machine build volume. There is no way to specify solid or surface properties. And because there is no version specification in the format, it is difficult to update it.

The ACIS .SAT Format

Another popular exchange format is the .SAT save file format (Spatial96) used by the ACIS geometric modeling kernel. The .SAT format is closely tied to ACIS's internal topological data structure, allowing the kernel to quickly rebuild the data structure from saved files. While it makes sense to use this type of "data structure dump" as an internal save file format for applications which use ACIS, it is inappropriate for an LM exchange format. An exchange format should be independent of the internal data structure of any one modeler. Furthermore, since the process of exchanging geometric information consists primarily of writing, transmitting, and reading back in data files, an exchange format should be compact. An internal data structure for a modeler, on the other hand, generally contains redundant information to facilitate real-time analysis and modeling operations; space-time tradeoff considerations will sacrifice a compact data structure for interactivity.

Therefore, it is not surprising that ACIS .SAT files are generally even more bloated than STL files. Yet some of the additional information they contain would in fact be useful for LM applications. For example, the faces that define each connected boundary surface are grouped together into "shells," and the shells that bound a connected piece of solid material are grouped together into "lumps." Surface and solid properties could conceivably be attached to the shells and lumps, respectively.

Other information is redundant and not always useful for LM. For a faceted model, the plane equation of the plane containing each facet and the line equation of the line containing each edge must be specified explicitly. Each vertex is specified separately from the 3-D point that determines its location. Every edge-use, or "coedge," is transmitted separately from its edge and the loop that contains it. Each entity is assigned an index corresponding to its line number in the ASCII version of the .SAT file, and the full radial-edge connectivity (Weiler88) of the entities is recorded using these indices. Figure 2 shows an excerpt from an ACIS .SAT file representing a simple cube.

```
201 0 1 0
7 Unknown 16 ACIS 2.1 Solaris 24 Wed Jul 30 16:51:37 1997
-1 9.9999999999999999547e-07 1.000000000000000036e-10
body $-1 $1 $-1 $-1 #
lump $-1 $-1 $2 $0 #
shell $-1 $-1 $-1 $3 $-1 $1 #
face $-1 $4 $5 $2 $-1 $6 reversed single #
face $-1 $7 $8 $2 $-1 $9 reversed single #
loop $-1 $-1 $10 $3 #
plane-surface $-1 0 0 5 0 0 -1 -1 0 0 forward_v I I I I #

...

vertex $-1 $18 $62 #
straight-curve $-1 5 -5 5 -1 0 0 I I #
face $-1 $-1 $50 $2 $-1 $63 reversed single #
plane-surface $-1 0 -5 0 0 1 -0 -0 0 1 forward_v I I I I #
coedge $-1 $30 $64 $32 $60 1 $20 $-1 #
coedge $-1 $45 $22 $27 $54 1 $12 $-1 #
coedge $-1 $22 $45 $49 $65 1 $12 $-1 #
coedge $-1 $64 $30 $22 $44 0 $20 $-1 #
edge $-1 $57 $66 $43 $67 forward #
coedge $-1 $42 $41 $23 $46 0 $12 $-1 #
edge $-1 $55 $51 $45 $68 forward #
edge $-1 $52 $35 $33 $69 forward #
coedge $-1 $70 $25 $59 $71 0 $50 $-1 #
coedge $-1 $25 $70 $42 $65 0 $50 $-1 #
loop $-1 $-1 $70 $38 #

...

point $-1 -5 5 -5 #
straight-curve $-1 5 -5 -5 -1 0 0 I I #
point $-1 -5 -5 -5 #
straight-curve $-1 -5 -5 -5 0 1 0 I I #
End-of-ACIS-data
```

Figure 2. *An excerpt from an ACIS .SAT file for a cube.*

Alternate LM Interchange Formats

Several alternate interchange formats have been proposed specifically for LM. Stroud and Xirouchakis (Stroud00) proposed extending an STL file with an extra section at the end that lists the faces in the original CAD model and indicates which triangles in the STL file came from which faces. This information about the designer's intent could then be used to clean up inconsistencies in the generated STL. They do not specify the details of how the original face information should be communicated; they used ACIS in their example implementation, but ideally the original faces would be described in a CAD-system independent manner.

Other proposed exchange formats encode their own full topological data structure as well as geometric information. Rock and Wozny (Rock91) developed the "RPI Format" shortly after STL was introduced. A header section specifies the intended manufacturing process, scanning methodology, material, and part name, followed by sections that define the vertices, straight line edges, and triangular faces of the part. Vertices, edges, and faces are stored in indexed array-like lists, allowing triangles that share vertices to reference the same vertex index. In addition to the implicit connectivity information provided by the shared vertex indices, each triangular face explicitly records the indices of the adjacent faces and the edges shared between them. Faces also record outward facing surface normals. Edges record the indices of their endpoints and the adjacent faces. CSG trees can also be built from cuboid, cylinder, cone, sphere and tori primitives using geometric transforms. Like STL, the RPI Format includes redundant surface normal information. In addition, the face connectivity information and the edges are redundant and directly tied to their data structure representation.

Similarly, Jacob et al.'s LMI (Layered Manufacturing Interface) format (Jacob99) is organized to match their own topological data structure choice. It is restricted to 2-manifold objects with a single boundary shell. The faceted version contains triangular facets defined by one loop consisting of three edges. Connectivity information is recorded in the edges, which reference their two endpoints and two adjacent faces. Each facet points to a plane and each edge to a line. In the precise version, edges can point to curves and facets to surfaces. Loops can have any number of edges. Connectivity is again recorded in edges, but this time they are divided into half-edges. Each half edge records one adjacent face, the next half-edge in the loop around that face, and the previous half-edge in the loop around the other adjacent face.

The differences between the organization of ACIS .SAT, the RPI format, and LMI, all of which include similar topological information, reflect the many variations on topological data structures that can represent that information. Clearly a neutral interchange format should not favor a particular data structure. The fact that connectivity shows up in so many different interchange formats, however, reflects the fact that deriving the connectivity is non-trivial.

AN OPERATION-CENTRIC ANALYSIS OF DATA EXCHANGE REQUIREMENTS

In designing a data exchange format it is typical to focus on the *information* that needs to be exchanged. In this paper, we will instead focus on the *operations* to be performed on the data as the organizing principle, and describe how these operations influenced our decisions during the development of SIF.

Figure 3 shows the basic operations involved in transforming a geometric model on a designer's computer into a 3D part. First, the design software must translate from its native solid modeling representation into the representation used by the exchange format. The exchange file must be written out, transmitted over the network, and read back in by the manufacturing software. The manufacturing software then builds its own internal representation of the geometry and validates that it is legal. Next, possibly with user intervention, the manufacturing software may perform operations such as scaling the part, changing its orientation, and positioning it in the build volume. Then the 3D part is sliced into 2D layers from which scanning paths or raster patterns are made to control the cutting/deposition/hardening (depending on the particular process) of each individual layer. Depending on the technology being used, support structure geometry may need to be generated, and various process parameters will need to be set. Finally, the LM machine builds the part.

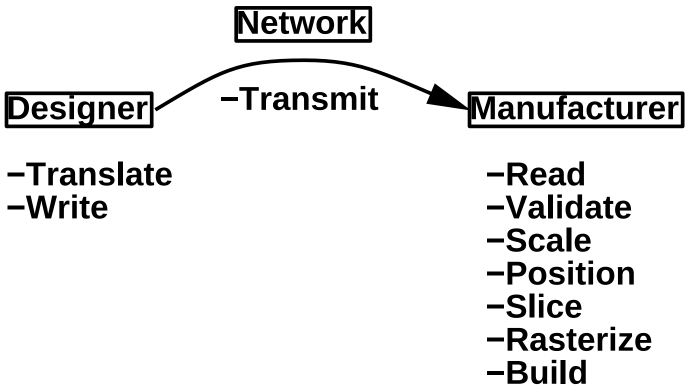


Figure 3. *Basic operations to transform a designer's geometric model into an LM part.*

Translation

The first operation we consider is translating from the CAD modeler's native format into the neutral interchange format. From the point of view of the design software, the ideal interchange format would include all of the design constructs and representations that the CAD software supports, because then translation would just be a simple syntactical mapping from its internal representation. For different design software, this representation might encompass anything from voxels to CSG to b-reps. The b-reps might be as simple as polygonal models or allow faces as complex as trimmed NURBS patches or embeddings in arbitrary analytic surfaces. Design constructs might include parameterized features, rotations, and sweeps of 1D, 2D, or 3D objects.

On the other hand, the manufacturers have to read in the interchange format end and build up a model they can easily manipulate. From their point of view, the simplest possible format is desired. Despite its widely acknowledged problems, .STL is extremely simple, which explains why it is still being used today.

Over the course of the SIF project, members of our group wrote both design sofware (ranging from mathematical sculpture generators (Sequin97; Smith99) to a machining feature-based CAD applet (Sundararajan01)) and layered manufacturing validation, cleanup, and slicing software (McMains99), which gave us a unique insight into *both* sides of the interchange process.

Our initial definition of SIF was driven by the design side. It included many constructs that mapped directly onto the design interface, some of which were redundant from the point of view of the range of geometry that could be expressed. As the design software grew richer, the interchange format became ever more bloated. Eventually we found that even when the same person was writing both the design module that wrote SIF and the manufacturing module that read SIF, the reader module wasn't being kept up to date with all the new features being added. Instead, we would just produce .STL files and read those into our manufacturing software. Clearly a format that we weren't even using internally was not going to be widely accepted.

While some people have proposed using STEP as an interchange format for LM, it suffers from the same problem as this early version of SIF - it was defined by and for designers, who want to be able to exchange information with other designers without losing either accuracy or the semantics of their design process. While the richness of STEP accomplishes this goal to some degree, it does not achieve it entirely; from the manufacturer's point of view, it isn't even the right goal.

In a major overhaul of SIF we reduced it back down to the bare minimum - a triangulated boundary representation much like STL - and only added features that enhanced the clarity of the data exchange process or fit in with downstream processing on the manufacturing side.

Transmission

From the point of view of transmitting files across the network, the best format is the most compact format. This raises the issue of ASCII versus binary exchange formats.

We started the SIF project defining only an ASCII version of SIF. There were several reasons for this decision. First, humans learn about formats much more efficiently from short example files than pages of formal specifications, and ASCII is human-readable. Effectively communicating the essence of a new interchange format to a large number of people requires an ASCII representation. Furthermore, data exchange will always be imperfect, making debugging (by humans) a necessity. Humans would much prefer to examine an ASCII file, even if binary versions are available. Another argument for an ASCII interchange format is that when the original designer typed in coordinate or dimension values to their design software, those were ASCII characters. Software that outputs those values directly will lose no information if the interchange format is also ASCII. And finally, there is no big endian/little endian ambiguity with ASCII decimal values.

On the other hand, there are two major arguments for having a binary format. First, it is far more compact. Second, the exchange is between, and the ultimate processing on, computers, which store and operate on binary numbers. Even simple fractional decimals such as .1 cannot be represented exactly as binary floating point values. If the computers on both ends are storing binary numbers, converting such a number to a finite precision ASCII decimal value to transmit it and then back to binary floating point on the other end can change that value. IEEE floating point representation standards should eliminate any changes in the values during the exchange of binary numbers.

For all of the aforementioned reasons, both an ASCII and a binary version of interchange formats should be defined. The binary version will become the default for mature systems, but the ASCII version will continue to be the first choice for testing rapidly changing systems during development and for introducing an interchange format to new users.

An additional feature of the ASCII version of SIF that we added for the convenience of debugging, at the expense of compactness, was explicit vertex indices at the start of each vertex definition. A computer reading the file can automatically generate indices, but a human cannot. Requiring explicit vertex numbers also immediately reveals the assumptions of the originator of the file about whether the vertex list starts with vertex 0 or vertex 1, as well as making the official choice obvious in a sample file (communicating by example). Our software on the receiving end can check that the vertices are zero-indexed and have no gaps in their enumeration, as SIF specifies, and issues warnings if not, but such files can still be processed successfully if they have no other problems.

Validation

After the manufacturer's software reads in a file, it must validate that the file describes a valid solid model before processing it further. Generally this is accomplished while building up a data structure that will be used during the downstream processing, typically a topological data structure such as a variant of Baumgart's winged edge data structure (Baumgart75) or Weiler's radial edge structure(Weiler88). These data structures capture the connectivity of a b-rep model. Since many design software packages also work off a topological data structure (e.g. any software based on the ACIS kernel), an obvious question to ask is whether the connectivity information should be included in the exchange format.

Our answer is that it should not be. One reason is that the exact topological information exchanged in formats that *do* include connectivity is closely tied to the internal data structure, which tends to be vendor-specific. The ACIS .SAT format, for example, appears to be a direct mapping from their internal data structure. As such, software built on their kernel (such as AutoDesk Inventor) can automatically and consistently output correct ACIS files, while software built on non-ACIS based platforms (such as SolidWorks) often produce incorrect .SAT files. A neutral data exchange format should not favor any vendor, nor should it prescribe an internal data structure on either the sending or receiving end.

For the steps of reading, writing, and transmitting the data, including connectivity information clearly slows down the process. As for rederiving connectivity on the receiving end, we have shown (MMains01) that by using an out-of-core algorithm, a topological data structure can be built very quickly even from huge, unorganized data sets (for example, we can build a complete topological data structure from 1 million triangles, using only 32 MB of RAM, in 5 minutes using a 700 MHz Pentium III). Because connectivity can be so efficiently rederived, there is no need to include it in the interchange format.

On the other hand, we do strongly believe that vertex coordinates should only be specified once, and then all triangles that share that vertex must reference the same definition. This form of connectivity information actually makes the data more compact for transmission. It does put an extra burden on the designer if not all vertices were shared. For example, the part pictured in Figure 4 is made up of six rotated and translated instances of the primitive shown in Figure 5; while each primitive is fully connected with shared vertices, at the seams where they meet round-off error led to small cracks in the STL model because vertices that should be coincident are not. By requiring shared vertices, the burden of determining which vertices are shared between instances and referencing the same vertex in the output falls on the designer, who knows the design intent, rather than the manufacturer, who would otherwise be forced to guess which vertices the designer intended to be shared. Thus, while in some cases it makes outputing SIF more complicated than outputing STL, the gain in clarity is worth the potential extra overhead.

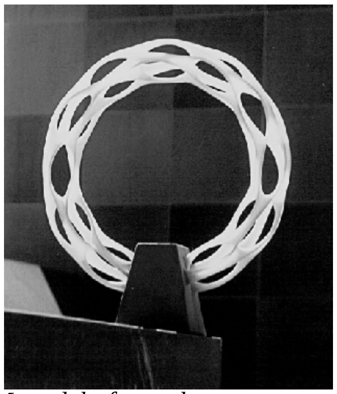


Figure 4. *An LM model of a sculpture composed of six rotated and translated copies of a complex primitive.*

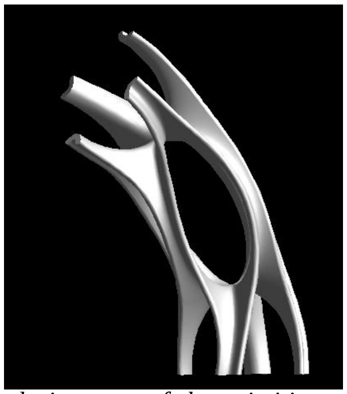


Figure 5. *A single instance of the primitive part used in the sculpture design.*

Scaling

After the manufacturer reads and validates the part description, they must decide on its scale. One of the obvious shortcomings of STL is that it doesn't include units. As a result, manufacturers who only receive an STL file are forced to guess what units the designer intended. Some layered manufacturing software, for example Stratasys's QuickSlice, compare the bounding box of the unit-less STL file to the build volume of the machine in order to determine whether the designer more likely meant inches or millimeters, then scale the part accordingly. (While it can be argued that this approach does in fact *automate* data exchange, it does nothing to ensure that the interpretation of the data is correct!)

In earlier versions of SIF, we took the approach of defining the default unit to be 1 meter if no units were specified. With the default unit so unrealistically large, we reasoned, it would immediately be obvious to the manufacturer if a designer had inadvertently forgotten to redefine the standard default unit in a SIF part description.

We have subsequently reconsidered this decision and concluded that even an unrealistic default unit leaves open the possibility of miscommunication. A small part specified in fractions of an inch might still be a reasonable size with those units interpreted as fractions of a meter. Future commercial processes may also support much larger build volumes or finer details than are common today. Any choice of a default unit, even an unrealistic one, is based upon assumptions about typical parts and processes, and what is "typical" is not static. Thus, in the current version of SIF, a file is not considered valid unless it explicitly specifies whether units are in inches or millimeters.

Adjusting Position and Orientation

The manufacturer must also decide whether to change the orientation of the part and where to position it within the build volume. Faster build times, better accuracy, and/or improved surface finish can often be obtained by re-orienting the input geometry. Most layered manufacturing processes can obtain significant savings of time and/or materials by combining into a single run multiple smaller parts that together fill the footprint of the build volume.

In some cases, however, the designer will not want the manufacturer to arbitrarily rearrange pieces of the geometry. For example, the individual links in a chain to be manufactured cannot be moved very far or they will no longer be interconnected. Therefore, in SIF we include a high-level grouping mechanism, the **constellation**, that allows the designer to communicate that the relative positions of the individual lumps of material in the constellation are fixed. Within a constellation, touching lumps of different materials will be built touching so that they fuse, and the links of a chain will be constructed in the relative positions specified by the designer so that they will interlock. The entire constellation, meanwhile, can be re-oriented freely as a group by the manufacturer to optimize build time or quality.

Slicing

The next step in processing is to slice the geometry into the thin parallel layers that will guide the manufacturing process. We have developed a simple, efficient sweep-plane algorithm for slicing triangulated boundary representations, as described in (McMains99). For some applications, however, the input is initially captured in layers: for example, CT scans or laser scanned data. For such applications, it might initially appear to make sense to manufacture these layers directly, in which case the interchange format should include some way of transmitting this data.

There are two main reasons why transmitting captured layer input to be manufactured directly is generally not advisable. First, the thickness of the layers used for acquisition and manufacturing are seldom identical. Furthermore, some LM technologies support varying the layer thicknesses on different slices to speed up build times. Manufacturers will not want to have to support an interchange format that gives them the resposibility for interpolating the input layers if the transmitted layer thicknesses don't match the manufacturing layer thicknesses. Secondly, the orientation in which the data was captured may not be the best orientation for manufacturing. For these reasons, it is preferable that a boundary representation be reconstructed from the scanned input before transmission to the manufacturer. For volumentric or medical data, the marching cubes method (Lorensen87) or its variants are typically used to construct a surface representation; for scanned data, the Powercrust (Amenta01) algorithm will build a water-tight boundary from densely sampled data points. These faceted reconstructed surfaces that already interpolate the data can then be sent to the manufacturer, who can easily re-orient the geometry to optimize build time and quality.

Rasterization

After slice contours have been created, the final stage in processing is to determine what is the "inside" of the part in each layer, in order to control the scanning paths or raster patterns that determine the cutting, deposition, or hardening of the material. For most input, this rasterization (a.k.a. scan conversion) is a straightforward process. Some data may be ambiguous, however, due to intersections in the input geometry.

For example, the shaded solid model shown in Figure6 may actually be composed of two correctly oriented but intersecting shells, as shown in Figure7. The designer who saw only the shaded model would probably believe that they had designed a part that was the union of the two shells. Yet a slice through the middle of the part as indicated may or may not be rasterized that way. Using the system software that controls the Z-corporation 3D printer (Z-Corp00), the interior of both contours is filled;

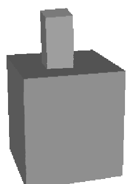


Figure 6. *A shaded solid model.*

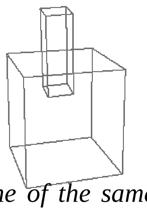
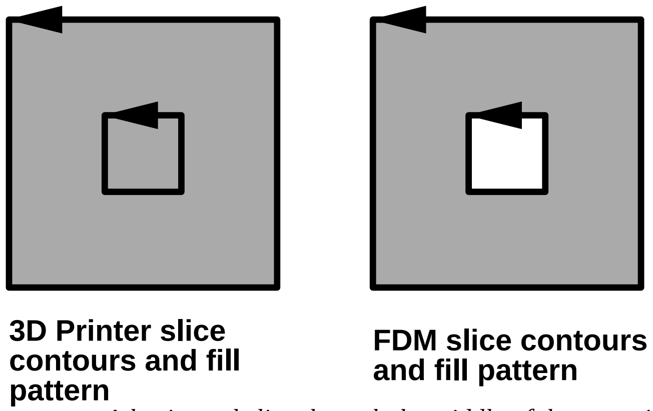


Figure 7. *The wireframe of the same model reveals that it is composed of two intersecting shells.*

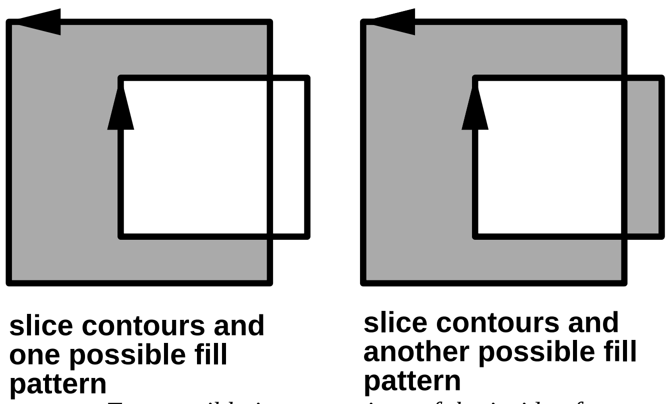
but using the QuickSlice software (Stratasys99) that controls the Stratasys Fused Deposition Modeling (FDM) machine, only the area between the two contours is filled, as shown in Figure 8. The Z-corporation software treats this situation as a union of two shells, while Quickslice ignores the orientation of the inner shell in slices where it is contained within another shell, calculating a difference between the shells in these slices.



3D Printer slice contours and fill pattern
FDM slice contours and fill pattern

Figure 8. *A horizontal slice through the middle of the part pictured in the previous figure will produce two slice contours, oriented as shown, which will be rasterized differently by different manufacturers.*

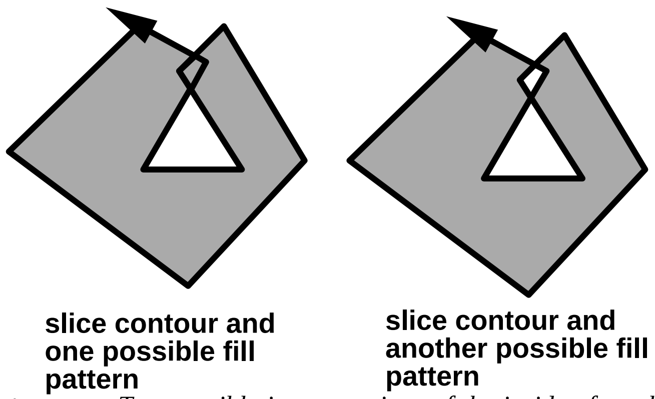
Nested slice contours of opposite orientation can occur even in well-formed files, for example a slice through the middle of a hollow cube. The slice contour through the outer shell will be oriented counterclockwise, while the slice contour through the contained inner contour will be oriented clockwise, unambiguously indicating a hole in the 2D layer. But in an ill-formed geometry, where the inner shell intersects the outer shell, the interpretation is not clear. Two possible interpretations of the slice are shown in Figure 9.



slice contours and one possible fill pattern
slice contours and another possible fill pattern

Figure 9. *Two possible interpretations of the inside of two oppositely oriented slice contours that intersect.*

Finally, a shell (and its slice contours) may self-intersect. Again, there are multiple possible interpretations of what area is “inside” of such a contour (Figure 10.) Such self-intersections may not have been present in the original model, only to be introduced while “healing” geometry that was not initially watertight. This can arise in cases where algorithms that clean up .STL files add additional triangles or perturb vertices without checking that the new or modified triangles don’t intersect existing geometry.



slice contour and one possible fill pattern
slice contour and another possible fill pattern

Figure 10. *Two possible interpretations of the inside of a self-intersecting contour.*

Even if these ambigous cases of intersecting geometry are prohibited in the interchange format (STL certainly doesn’t explicitly allow them), we have seen that manufacturers won't categorically reject them. The interchange format should clearly define the correct interpretation in these ambiguous cases so that all manufacturers given the same part file will interpret it as representing the same geometry.

The interpretation we advocate is to treat overlapping or intersecting geometry as expressing implicit Boolean operations (or implicit "unary" operations in the case of a self-intersecting shell, in the sense of Heisserman's unary shape operators (Heisserman92)). We interpret intersecting shells where both are oriented with surface normals pointing outwards as expressing implicit Boolean unions, intersecting shells of opposite orientations as expressing implicit Boolean differences, and self-intersecting shells as expressing implicit unary unions with themselves. This interpretation treats the orientation of a shell consistently regardless of its context, doesn't produce non-manifold parts that are structurally weak whenever contours intersect, and guarantees that an epsilon change in the geometry of a shell will only effect an epsilon change in the volume of the part.

Furthermore, this interpretation can be easily implemented at the slice level. When scan converting a slice, we simply interpret every region that has a positive winding number (Foley90) to contain material. Performing this calculation and rasterizing accordingly is directly supported by the OpenGL interface (Woo99), which is available for virtually all modern graphics cards.

Once we have implemented the consistent interpretation of implicit Booleans at the slice level, interpreting explicit Booleans is a simple extension. In SIF, we support unevaluated Booleans on any oriented, closed, polygonal shells in the input. The unevaluated Boolean tree is carried through to the slice level and only there resolved, in 2D. While this makes the interchange format slightly more complex, it places very little additional burden on the manufacturer. It can also reduce the number of non-watertight shells transmitted. Since gaps often occur between trimmed surface patches where the trim curves on the two adjacent surfaces approximated their Boolean intersection, by eliminating the need to calculate the intersections in 3D if original watertight solids and the unevaluated Booleans are transmitted instead, such gaps will occur less frequently.

CONCLUSION

In the course of the SIF project, we re-thought many of our initial decisions about what a viable LM interchange format should include. Some of the most important lessons we learned were:

1. A new interchange format should apply the KISS (Keep it simple, stupid!) principle to encourage acceptance on the receiving end.
2. Both ASCII and binary versions of the format are needed.
3. Connectivity in the form of shared vertices should be included in the transmission of a b-rep, but not a full topological data structure dump. Redundancy has no place in an interchange format.
4. Require units to be defined explicitly.
5. Communicate to the manufacturer when the relative position of parts matters.
6. Don't send unprocessed 2D data.
7. Define the interpretation of potentially ambiguous input such as implicit Booleans, even if it is not strictly legal.
8. Many operations that are very difficult to perform robustly in 3D can be postponed to 2D (after slicing) where they become more tractable.

These principles, more so than the semantics and certainly rather than the exact syntax of our interchange format, are the most important contribution of the SIF project.

REFERENCES

Nina Amenta, Sunghee Choi, and Ravi Krishna Kolluri. The power crust. In *Sixth Symposium on Solid Modeling and Applications*, pages 249 – 266, Ann Arbor, MI, June 2001. ACM.

Joseph J. Beaman et al. *Solid Freeform Fabrication : A New Direction in Manufacturing*. Kluwer Academic Publishers, Dordrecht, 1997.

B. G. Baumgart. A Polyhedron Representation for Computer Vision. In *Proceedings of the National Computer Conference*, pages 589–596, 1975.

James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley, Reading, MA, second edition, 1990.

Christoph M. Hoffmann. *Geometric and Solid Modeling : An Introduction*. Morgan Kaufmann, San Mateo, CA, 1989.

J. Heisserman and R. Woodbury. Unary Shape Operations. In *Geometric Modeling for Product Realization*, pages 63–80. North-Holland, Amsterdam, 1992.

Gan G. K. Jacob, Chee Kai, and Tong Mei. Development of a new rapid prototyping interface. *Computers in Industry*, 39(1):61–70, June 1999.

Harvey E. Cline and William E. Lorensen Marching cubes: A high resolution 3D surface construction algorithm. In *Computer Graphics Proceedings, Annual Conference Series*, pages 163–169, Anaheim, CA, July 1987. ACM SIGGRAPH.

Martti Mäntylä. *An Introduction to Solid Modeling*. Computer Science Press, Rockville, MD, 1988.

Sara McMains. The SIF_SFF Page. http://www.cs.berkeley.edu/ũg/sif_2_0/ SIF_SFF.shtml, 1999.

Sara McMains, Joseph M. Hellerstein, and Carlo Séquin. Out-of-Core Build of a Topological Data Structure from Polygon Soup. In *Sixth Symposium on Solid Modeling and Applications*, pages 171–182, Ann Arbor, MI, June 2001. ACM.

Michael E. Mortenson. *Geometric Modeling*. Wiley, New York, 1985.

Sara McMains and Carlo Séquin. A Coherent Sweep Plane Slicer for Layered Manufacturing. In *Fifth Symposium on Solid Modeling and Applications*, pages 285–295, Ann Arbor, MI, June 1999. ACM.

A. A. G. Requicha. Representations for Rigid Solids: Theory, Methods, and Systems. *ACM Computing Surveys*, pages 437–464, December 1980.

Stephen J. Rock and Michael J. Wozny. A Flexible File Format for Solid Freeform Fabrication. *Proceedings Solid Freeform Fabrication Symposium*, pages 1–12, 1991.

Sundararajan, Ahn, Smith, Kannan, D'Souza, Sun, Kim, McMains, Smith, Mohole, Séquin, and Wright. CyberCut: An Internet Based CAD/CAM System. *ASME Journal of Computing and Information Science in Engineering*, 1(1):to appear, 2001.

C. H. Séquin, H. Meshkin, and L. Downs. Interactive Generation of Scherk-Collins Sculptures. In *1997 Symposium on Interactive 3D Graphics*, pages 163–166, Providence, RI, April 1997. ACM SIGGRAPH.

Jordan Smith. SLIDE Educational Rendering System for 3D Interactive Dynamic Environments. http://www.cs.berkeley.edu/˜ug/slide/pubs/masters, 1999.

Spatial Technology, Inc, Boulder, CO. *ACIS Save File Format Manual*, 1996.

Stratasys, Inc., Eden Prairie, MN. *QuickSlice 6.2*, 1999.

I. Stroud and P. C. Xirouchakis. STL and extensions. *Advances in Engineering Software*, 31(2):83–95, February 2000.

Kevin Weiler. The Radial Edge Structure: A Topological Representation for Non-Manifold Geometric Boundary Modeling. In *Geometric Modeling for CAD Applications*, pages 3–36. North-Holland, Amsterdam, 1988.

Mason Woo, Jackie Neider, Tom Davis, and Dave Shreiner. *OpenGL Programming Guide*. Addison-Wesley, Reading, Massachusetts, third edition, 1999.

Z-Corporation, Inc., Burlington, MA. *Z402C 3D Printer*, 2000.