A Combinatorial Parametric Engineering Model for Solid Freeform Fabrication

J.C. Boudreaux
NIST/Advanced Technology Program

Abstract: Fabricated parts are often represented as compact connected smooth 3-manifolds with boundary, where the boundaries consist of compact smooth 2-manifolds. This class of mathematical structures includes topological spaces with enclosed voids and tunnels. Useful information about these structures are coded into level functions (Morse functions) which map points in the 3-manifold onto their height above a fixed plane. By definition, Morse functions are smooth functions, all of whose critical points are nondegenerate. This information is presented by the Reeb graph construction that develops a topologically informative skeleton of the manifold whose nodes are the critical points of the Morse function and whose edges are associated with the connected components between critical slices. This approach accurately captures the SFF process: using a solid geometric model of the part, defining surface boundaries; selecting a part orientation; forming planar slices, decomposing the solid into a sequence of thin cross-sectional polyhedral layers; and then fabricating the part by producing the polyhedra by additive manufacturing. This note will define a qualitative and combinatorial parametric engineering model of the SFF part design process. The objects under study will be abstract simplicial complexes K with boundary ∂K. Systems of labeled 2-surfaces in K, called *slices*, will be associated with the cross-sectional polyhedral layers. The labeled slices are mapped into a family of digraph automata, which, unlike cellular automata, are defined not on regular lattices with simple connectivities (cells usually have either 4 or 8 cell neighborhoods) but on unrestricted digraphs whose connectivities are irregular and more complicated.

1. Introduction

Engineering designs are complicated data sets which contain such essential information about industrial products as CAD data , material specifications, dimensioning and tolerancing data, surface condition specifications, and much else besides. The complexity of designs mirrors the structural complexity of products. Products are properly modeled as *networks of interacting components*, each of which separated from the others by a (physical or conceptual) boundary, and all of which are interconnected by links such that outputs of one component may be passed as inputs to others. Products are also *stratified structures*, that is, a component at one level (a parent) may be refined into networks of components (the children) at the next-lower level. In the stratification process, the links of the parent component are preserved: every input of the parent must be an input to at least one child and every output an output of at least one child.

Stratification also leaves the parent's boundary in tact, allowing it to become a container of the boundaries the children. Boundaries drawn in this manner impose a nest structure on the product: at the highest level is the product itself, followed by its immediate descendants, and so on, until all of the boundaries have been accounted for. In the product-as-network model, stratification boundaries mark off key components which can be then be identified by symbolic "tags" or labels. This symbolic structure permits stratified product networks to form a framework with respect to which the *design features* of components are defined. Features of one component may be either synthesized from those of its descendant components in a defined stratification or inherited by virtue of its links to other (in some cases, all other) components. The multiplicity of these relational ties goes a long way in explaining the observed conceptual diversity among feature

categories. For example, *functional* features are inherited from above by considering the constraints imposed by the component's children. Specifically, the functional capabilities and operational behavior of components may be modeled by *transfer functions* which, given time-sequences of inputs (and possibly information about the current *state* of the component), produce time-sequences of outputs (and possibly a new state) after a specified time lag. *Mating* features are both synthesized from below and inherited from above because of the requirements imposed by assembly and joining processes. *Form* features are those through which all of the (often) contending requirements of the part are finally reconciled and by which the connectivity-induced constraints are resolved.

Lurking in the background is the difficult notion of *design completeness*, which might be roughly defined as data sets which are sufficient to determine all of the essential properties of the components which instantiate them. However, since no fixed meaning can be attributed to this notion, a few illustrative examples might suggest lines for future study. First, given a model of the "local" context of the component in the product network, a complete design for that component will need to be elaborated to the point that it possible to decide with confidence that its instances would satisfy all of the requirements and constraints active in that context. Second, a complete design and an inventory of available manufacturing capabilities and plant layouts should yield either manufacturing process plans and material flow routes or a determination that the component lies beyond the available productive capabilities. These two examples show why the notion of complete designs is as difficult as it is!

In any event, constructing engineering designs is a difficult and costly undertaking. It has often been proposed that the thing to do is to reuse designs by "instrumenting" them by defining a system of parameters for them. This has proven to be a difficult undertaking in its own right. *Parametric engineering is based on the premise that an adequate inventory existing designs can be used to create active design templates and that by "plugging" in reasonable values for the embedded parameters, it is possible to automatically generate novel variant designs.*

In an earlier note **[4]**, it has been argued that it is reasonable to model SFF-eligible parts as *3-manifolds with boundary.* The boundary consists of a possibly disconnected set of compact smooth *2*-manifolds (*topological surfaces*). For example, the *2*-sphere is the single boundary of the closed 3-ball. In the general case, the class of structures includes more complicated objects that have enclosed *voids* and *tunnels*. But the fact that every compact *3*-manifold *M* can be *triangulated* by an *abstract simplicial complex K* whose geometric realization in Euclidean 3-space, written $|K|$, is homeomorphic to *M* suggests that it is possible to construct a parametric engineering model directly on discrete combinatorial structures. *From the parametric engineering point of view, geometric part definitions are massively overdetermined.*

Moreover, this note will suggest that a major part of the burden can be carried by even simpler mathematical structures, called *directed graphs,* shortened to *digraphs*. A digraph is defined as a pair $(\mathcal{V},\mathcal{E})$, where $\mathcal{V}$ is an unempty set of *vertices,* $\mathcal{E}$ is a set of *edges*, together with a pair of functions $start,stop:\mathcal{E}\rightarrow\mathcal{V}$ such an edge *h* is interpreted as an *arrow* from the vertex *start(h)* to the vertex *stop(h)*. This definition admits a very large class of structures. For example, digraphs may have many (even infinitely many) distinct *parallel* and *antiparallel* edges between the same two vertices. Digraphs may also have *loop* edges for which the *start* vertex is the same as the *stop* vertex.

Stratified product networks are complicated enough to illustrate the technical usefulness of digraphs. The first step is straightforward: the *vertices* of the *product diagraph* are symbolically "tagged" components; and the arrows consist of both structural links which connect parents with their children and general links between components. The second step handles repeated stratifications and is only a bit more complicated. Suppose that $\mathcal{A}$ is any arbitrary digraph whose vertices are distinct from those of $\mathcal{V}$ and that v is any vertex of $\mathcal{V}$, then the desired stratification may be obtained by replacing v by $\mathcal{A}$ and adding one or more arrows to parallel those which originally connected v. In effect, the addition of $\mathcal{A}$ induces a three-fold partition of the edges of the new digraph: *interior* edges whose *start* and *stop* vertices are both in $\mathcal{A}$, *exterior* edges whose *start* and *stop* vertices are both not in $\mathcal{A}$, and *boundary* edges one of whose *start* or *stop* vertices is in $\mathcal{A}$ and other is not in $\mathcal{A}$. This procedure shows that products may be represented by series of digraphs linked to one another by successive stratifications.

The full power and potential of the parametric engineering model is strongly apparent in the context of SFF. This style of manufacturing is based on additive manufacturing processes which may resolved into the following steps: (1) create a *solid geometric model* of the part, defining surface boundaries and identifying interior (material) regions from the exterior (void) regions; (2) select a part orientation and then form *planar slices*, decomposing the solid into a sequence of thin cross-sectional *polyhedral* layers; and (3) *fabricate* the part by producing all of the polyhedra by any of several methods, including: stereolithography, selective laser sintering, laminated object manufacturing, fused deposition modeling, and three dimensional printing. *In classical manufacturing work is applied to the surface of the part either through deformation process or by material removal. But the primary advantage of SFF is that it reduces the 3-space shape of compact objects to a sequence of 2-space slices (and possibly some mild "skinning" conditions).*

The necessary topological details are sketched in Sections 2 and 3 (see **[11]** for details). In Section 4 below, I will describe how the notion of a digraph can be applied to define a class of automata, called *digraph automata*, which I suggest will provide a natural computational framework for a parametric engineering model for SFF.

2. Simplicial Complexes, Pseudomanifolds, and Edge Paths

An *abstract simplex* is a finite unempty set of elements, called *vertices*. If s is a simplex containing $q+1$ vertices, then s is a simplex of dimension q, that is, a *q-simplex*. Simplexes have a clear geometric meaning: a *0-simplex* is a set containing a single vertex representing a point (with the usual mild abuse of notation, points and vertices are identified); a *1-simplex* is an unordered pair of vertices representing an edge; a *2-simplex* is an unordered triple of vertices representing a triangle; a *3-simplex* is an unordered quadruple of vertices representing a tetrahedron; and so on.

Every unempty subset of a q-simplex s is a p-simplex, for some dimension $p \leq q$, which is called a *face* (resp., a *p-face*) of s. For example, if s is a *3*-simplex, then s contains four distinct *2*-faces, six *1*-faces, and four *0*-faces. Every (proper) subset of s is called a *(proper) face* of s.

An *abstract simplicial complex* (henceforth, *complex*) *K* is a set of abstract simplexes such that if simplex *s* is an element of *K* then all faces of s are elements of *K*. A complex is determined by the simplexes that it contains. Every complex is partially ordered by the inclusion relation between its faces. Thus, the pair $(K,\subset)$ is a digraph whose vertices are the simplexes of *K* and whose arrows link each simplex with its proper faces. Those simplexes of *K* that are not proper faces of other simplexes are called *maximal faces* of *K*. The set of all maximal faces completely determines the complex, which may be generated by closing the set of maximal faces under the formation of subsets. For example, if *{1,2}* is a maximal face, then let *[1,2]* be the set of its unempty subsets, that is, *{{1,2},{1},{2}}*.

If *K* is a complex, then its *dimension (*written *dim K*) is -1 if *K* is empty, and *q* if *K* contains at least one *q*-simplex but no *(q+1)*-simplex. Suppose *K* is a complex, then the *q-dimensional skeleton* (henceforth, *q-skeleton*) of *K*, written K^q , is the set of all of the *p*-simplexes of *K* such that $p \leq q$. The *q*-skeleton of a complex is itself a complex. In particular, the *1-skeleton* of a complex consists of its vertices and edges.

A *simplicial mapping* is a function from the vertices of one simplicial complex to to the vertices of another such that the images of the simplexes of the first are simplexes of the second. Two simplicial complexes are *isomorphic* if there is a bijective simplicial mapping from one to the other such that the inverse mapping is also simplicial. Given a Euclidean *n*-space of suitably high dimension, the *convex hull* every set of *(k+1)* pointwise independent points is a *geometric k-simplex*. Every finite abstract simplicial complex is isomorphic to a geometric simplicial complex, called its *geometric realization*.

Since exotic complexes will have no role in this note, some additional restrictions on complexes will be imposed. For example, only those complexes that have finitely many elements will be considered. These complexes are not only of finite dimension but are also *locally finite* in the sense that each of their simplexes is a face of at most finitely many other component simplexes. Suppose that *K* is a finite simplicial complex, then:

(**R1**) $dim\ K \leq 3$.
(**R2**) *K* is a *pure complex* if all maximal faces of *K* are of equal dimension, that is, *K* is a *q*-complex and every simplex *s* in *K* is a face of at least one *q*-simplex in *K*.
(**R3**) *K* is a *q-pseudomanifold (with boundary)*, that is, *K* is a pure q-complex and every *(q-1)*-face of *K* is the face of at most two distinct *q*-simplexes of *K*.

The first two restrictions insure that the complexes under study have no isolated or dangling points, edges or triangles. A simple example shows that restriction (**R3**) can be violated even if both (**R1**) and (**R2**) are satisfied. Suppose that $K^{\#}$ is the 3-complex whose maximal faces are *3*-simplexes *[4,1,2,3]*, *[5,1,2,3],* and *[6,1,2,3]* whose vertices are distinct. Then $K^{\#}$ is both a finite and a pure *3*-complex, thereby satisfying the first two restrictions, but obviously fails to satisfy the third, since the *2*-simplex *{1,2,3}* is a proper face of each of them. $K^{\#}$ is mildly pathological in the sense that it cannot be realized as a polyhedron in 3-space, which is one of the pathologies which (**R3**) is designed to remedy.

Let *K* be a *3*-pseudomanifold and *s* be any arbitrarily chosen *2*-simplex of *K*, then there are only two possible cases: either *s* is common face of two distinct *3*-simplexes or *s*

is a face of exactly one *3*-simplex in *K*. In the first case, the *2*-simplex is said to be an *interior face* of *K*. In the second case, the *2*-simplex and all of its unempty subsets are said to be *boundary elements* of *K*. Let ∂K be the set of all such boundary elements. Thus, the restrictions induce a partition on the 2-simplexes of *3*-pseudomanifolds. The restrictions also imply that ∂K is a 2-complex, that is, that ∂K is a finite pure *2*-complex, and that every *1*-simplex (edge) in ∂K is a common face of precisely two *2*-simplexes, that is, ∂K is a *2-pseudomanifold without boundary*.

Imagine a walk through a *3*-pseudomanifold: start at a vertex, say *v*, then step across any of the finitely many edges containing that vertex, say *{v,w}*, to the companion vertex *w*, at which point either stop or repeat the process with *w* as the new starting point. Such a walk can be unambiguously encoded by a list of ordered pairs of vertices <*v,w*> representing the case in which one started at *v*, stepped across *{v,w}*, and then landed on *w*. A list of ordered pairs generated in this manner is called an *edge path*. If *K* is a *3*-pseudomanifold, then let $\Pi(K)$ be the set of all edge paths through *K*. This list is contained in the *1*-skeleton of *K* and has important algebraic properties. For example, every edge path has an *inverse*, that is, a path which starts from the original stop vertex and traverses the list of ordered pairs in the reverse order, stepping across <*w,v*> whenever the original stepped across <*v,w*>, and stops at the original start vertex. A *composition operator* can be defined for any pair of edge paths such that the stop vertex of the first is identical the start vertex of the second. The resulting algebra is called the *edge path groupoid* (see **[11]**, 134-139).

If there is at least one edge path between every pair of vertices of *K*, then *K* is said to be *(edge) connected*. If a *q*-pseudomanifold is not connected, then it resolves into finitely many *component q*-pseudomanifolds which are connected. Henceforth, all pseudomanifolds will be assumed to be edge connected.

Other types of paths, each having an associated definition of connectedness, may also be defined. For example, there are walks through *3*-pseudomanifolds in which one starts an edge and then steps across any one of the *2*-faces containing that edge to another edge of that *2*-face. The associated path structure is a natural extension of the one defined for edge paths.

More generally, let *P* be any one of these paths, then *P* may be represented as a digraph whose vertices are the *(q-1)*-faces encountered along the path and whose arrows are the *q*-faces across which one steps.

3. Slices, Patches, and Reeb Graphs

Given the importance of thin cross-sectional polyhedral layers in SFF-style manufacturing, the next step is to build a combinatorial mechanism to represent them:

A *slice* of a *3*-pseudomanifold *K* is a *2*-pseudomanifold with boundary *X* such that
(**S1**) *X* is a *subcomplex* of *K* and ∂X is a subset of ∂K;
(**S2**) *X* decomposes *K* into upper and lower subcomplexes $K_u(X)$ and $K_l(X)$ whose union is *K* , whose intersection is *X*, and whose vertices are disjoint except for the shared vertices of *X*; and
(**S3**) if *v* is any vertex of $K_u(X)$ and *w* of $K_l(X)$, every edge path in $\Pi(K)$ from *v* to *w* contains at least one vertex in common with *X*.

If X is a proper subcomplex of ∂K consisting of one or more *2*-simplexes and their faces, then X is a slice of K and in this case $K_u(X)=X$ and $K_l(X)=K$. All sets X of this kind will be called *boundary slices* of K. Boundary slices may be composed of multiple components, each of which is a connected 2-complex with boundary. These components will be called *(boundary) patches*. Thus, every boundary slice resolves into one or more patches. Since *3*-pseudomanifolds are required to be finite, there must be *minimal* patches, specifically all boundary *2*-pseudomanifolds consisting of a single *2*-simplex and its faces. Let $\mathcal{P}$ be any unempty set of boundary slices of K closed under set intersection. Then every unempty element of $\mathcal{P}$ is itself a boundary slice and the pair $(\mathcal{P},\subset)$ is a digraph whose arrows connect slices with subslices. The pair $(\mathcal{P},\subset)$ will be called a *boundary system* for K.

A sequence of slices $(X_1,...,X_m)$ is a *cross-sectional slicing (cs-slicing)* of K just in case for all $1\leq i<j\leq m$ the slices X_i and X_j are disjoint and X_j is a subset of $K_l(X_j)$, or equivalently X_i is a subset of $K_u(X_j)$. Keep in mind that cs-slicing is defined combinatorially and does not require the *3*-pseudomanifold to be embedded in any Euclidean space.

From the perspective of Morse theory (see **[2] [6]**,**[10]**,**[11]),** the most interesting cs-slicings are those which (1) begin and end with minimal patches and which (2) mark the *critical regions* of the *3*-pseudomanifold under study, that is, regions in which either the number of components or their topological properties change. An example of a change of the second kind is the first appearance of a new boundary patch (or void surrounded by boundary elements) in a patch whose earlier stages had none. Let cs-slicings of this kind be called *Morse slicings*. In general, *3*-pseudomanifolds admit of many Morse slicings.

Given a 3-complex K and a Morse slicing $(X_1,...,X_m)$, it is possible to build a combinatorial variant of a *Reeb graph* (compare **[9]**):

Under the assumption that the slicing has already been reduced by eliminating consecutive pairs of homeomorphic slices, then
(**RG1**) the vertices of the Reeb graph are components of slices; and
(**RG2**) the arrows of the Reeb graph link each component of slice X_i to all of the components of slice X_{i+1} which descend from it.

As in the smooth case, Reeb graphs concisely encode important topological properties of *3*-pseudomanifolds: the vertices encode the components of slices, and the arrows encode the *3*-pseudomanifolds (like cylindrical solids) which link a component of slice X_i with all of the components of slice X_{i+1} from which descend from it. The *3*-pseudomanifolds paired with Reeb graph arrows may have voids and tunnels. However, Reeb graphs are less successful in capturing properties affecting the geometric realizations of *3*-pseudomanifolds. For example, there are many ways to embed an encoded Reeb graph into *3*-space which are not geometrically equivalent (isotopic) to one another. This has to do with the mathematics of knots in *3*-space (see **[1]**).

4. Digraph Automata

In this section, I will propose a class of automata, called *digraph automata*, which I can provide a plausible computational framework for a parametric engineering model for SFF. In particular, I propose that this framework be based upon a *functional model of computation* in which lists are used to specify the application of a function (the first, or head, expression of the list) to its arguments (the rest of the list), and in which the computational processes are carried out by entities called *evaluators* (see **[3]**).

Though evaluators may differ in their implementational details, they are all founded on an interpretive mode of operation. The behavior of the evaluator is described as a potentially infinite loop:

(E1) *read* expressions from sources of input expressions, called *input streams*;
(E2) *evaluate* the expressions in the context of the *symbolic environment* **env** (a list of symbol/value pairs), modifying **env** as required; and then
(E3) *write* the expression resulting from the evaluation to an output stream, as required.

The set of expressions includes three main classes. First, there is a collection of *atoms*, which includes such familiar elements as characters, strings, integers, and floating-point numbers. When the evaluator is given any of these expressions to evaluate, it returns the expression itself. Second, *symbols* are expressions that have a more complicated role: when given a symbol, the evaluation procedure tries to match the symbol with the entries in the symbolic environment **env**. If a match is found, value corresponding to the symbol is returned; but if there is no match, then the interpreter returns a message to that effect. Third, when the evaluator is given a *list* to evaluate, it assumes that the expression is the application of a function, which must be the first component of the list, to the interpreted values of the arguments, which must be the remaining components of the list. The value obtained is then returned as the value of the original list.

The organization and control of the symbolic environment is perhaps the single most important task of the evaluation process. Each evaluator has access to an environment and can modify it in the course of operation. Moreover, during the evaluation of an expression a new local environment may be created and then destroyed after the evaluation is completed. The process is to create a list of symbol/value pairs and then append that list to the current environment; and when the appended material is no longer needed, remove it. *Symbolic environments may be nested within each other and access rights for different evaluators can be precisely controlled* (see **[3]**, 44-46 for details). *Thus, the value of a given expression is a context-sensitive function of the inputs and the currently active symbolic environment.*

It's now a short step to digraph automata. The general approach to be taken is to develop a computational theory based on an *agent-oriented model of computation.* Agents are computing elements that have persistent internal states, which are able to sense and act upon their external environment, and which, if suitably configured, are able to communicate with one another. Agents which share the same communication conventions and which are mutually accessible to one another, form a society of agents. The complexity of the behavior of such a society depends in part upon the complexity of the conventions governing the semantic interpretation of the messages that agents can send one another.

To handle the combinatorial structures of interest to SFF, it will be sufficient to establish that the SFF-family of digraphs can be correlated with societies of agents.

First, every vertex is mapped onto to a pair (**Λ**, **env**), where **Λ** is an evaluator and **env** is a mutable symbolic environment. That is, vertices behave as *iterative functions* that are presented a time-sequence of inputs, and then in the context of the current *state* of **env** produce within a bounded time a time-sequence of outputs and possibly a revision of **env**.

Second, every edge is associated with a *transfer function* which, given a time-sequence of values on the *start* vertex, yield a corresponding time-sequence of values on the *stop* vertex by applying the transfer function.

Digraph automata are distributed networks of evaluators that communicate with each other by transfer functions. Among the expression that they may communicate are those that cause changes in **env**.

It is possible to implement digraph automata as asynchronous networks but this interesting approach has been only partially implemented in **[3]**. There are also other execution styles. For example, since each node is connected to its neighbors in a precisely defined way, it is not unreasonable to adopt a computational approach in which the updates happen by a *Jacobi sweep.* That is, all nodes are evaluated at time *t* and all are determined values for time *t+1*, then the state is updated by swapping in the *t+1* values. This style is used by cellular automata (see **[12]** for details).

5. Concluding Remarks

Suppose that a digraph automaton has been wired up and that each vertex has been assigned a reasonable **env.** How would one use it to generate geometric realizations of *3*-pseudomanifolds?

First, if *K* is the *3*-pseudomanifold being worked on, then there are huge numbers of realizations in Euclidean *3*-space. But given the focus on a single component, this reduces to a specific set of requirements, specifications, and constraints which have been derived from the stratified product network that the component is being designed for. *It is a reasonable assumption that these requirements, specifications, and constraints can be coded in a symbolic environment*

Second, a boundary system $(\mathcal{P}, \subset)$ for *K* and the set of *active patches* in ∂K must be selected and symbol/value pairs for each of them must be added to the symbolic environment. The associated symbol will be the official way to reference a patch and the value will be the set of simplexes that are elements of the patch. Patches go proxy for geometric surfaces in *3*-space that are homeomorphic to disks. *These patches are the sources or sinks of edges which connect the part under study with other components of the product, thereby anchoring requirements, specifications, and constraints inherited from the product digraph, and thus marking the regions in which co-design will be required.*

Third, a reasonable sample of *Morse slicings* must be generated. This will be a difficult task, but once a sample has been provided, it is exercise to code all of them in the symbolic environment.

Fourth, produce a *Reeb digraph* for every Morse slicing in the sample. This can be coded directly into our fictional digraph automaton, but with a surprising twist. Every

vertex *and* every edge of the Reeb digraph is assigned its own separate evaluator. Let's call them *v*-evaluator and *e*-evaluators. The edges of the digraph automaton are connected in a manner prescribed by the Reeb digraph, that is, every edge in the Reeb digraph is mapped onto an edge path in the digraph automata which links up the source *v*-evaluator, the corresponding *e*-evaluator, and the sink *v*-evaluator. The local environment of each *v*-evaluator contains a symbol/value pair for the component of Morse slice paired with corresponding vertex in the Reeb digraph. The local environment of each *e*-evaluator has access to the local environment of its companions.

To obtain geometric realizations for K at this point is a matter of outputting polygons from v-evaluators, blended cylindrical solids from e-evaluators, and surface geometries for the active patches – and of course to do all that while satisfying the product-based constraints!

To make this agenda technically plausible will require a better understanding of role of algebraic topology in the optimal design of physical shapes and a working prototype of digraph automata.

References

[1]Adams, C.C. *The Knot Book: An Elementary Introduction to the Mathematical Theory of Knots*, W.H. Freeman, 2001.

[2] Bott, R, "Lectures on Morse Theory, Old and New," *Bulletin (new series) AMS,* vol 7 (1982); 331-358.

[3] Boudreaux, J.C. "Concurrency, device abstraction, and real-time processing in AMPLE," in W.A. Gruver and J.C. Boudreaux (eds.), *Intelligent Manufacturing: Programming Environments for CIM*, Springer-Verlag, 1993; 31-91.

[4] Boudreaux, J.C. "Solid Freeform Fabrication and Parametric Engineering," *Solid Freeform Fabrication Symposium, (SFF '02)*, University of Texas at Austin, August 5-8, 2002; 297-304.

[5] Dey, T.K., H. Edelsbrunner and S. Guha, "Computational Topology," in B. Chazzle, J.E. Goodman, and R. Pollak (eds.), *Advances in Discrete and Computational Geometry,* AMS, 1999.

[6] Forman, R. "A User's Guide to Discrete Morse Theory," *Seminaire Lotharingien de Combinatoire*, 48 (2002), B48c.

[7] Koenderink, I. *Solid Shape*, MIT Press, Cambridge, MA, 1990.

[8] Kunii, T.L. "Research Issues in Modeling Complex Object Shapes," *IEEE Computer Graphics and Applications*, vol 14(1994); 80-83.

[9] Shinagawa, Y. and T.L. Kunii, "Constructing a Reeb Graph Automatically from Cross Sections," *IEEE Computer Graphics and Applications*, vol 11(1991); 44-51.

[10] Shinagawa, Y., T.L. Kunii Y.L. Kergosien, "Surface Coding Based on Morse Theory," *IEEE Computer Graphics and Applications*, vol 11(1991); 66-78.

[11] Spanier, E.H., *Algebraic Topology*, Springer-Verlag, 1966.

[12] Toffoli, T. and N. Margolus, *Cellular Automata Machines: A new environment for modeling*, MIT Press, 1987.

Contact information:
J.C. Boudreaux, NIST/Advanced Technology Program
Admin A221
Gaithersburg MD 20899
tel (301)975-3560
jack.boudreaux@nist.gov