Proceedings of DETC98:
1998 ASME Design Engineering Technical Conference
September 13-16, 1998, Atlanta, GA

DETC98/DFM-123

BUILDING BLOCK DESIGN FOR LAYERED SHAPE MANUFACTURING

Mike Binnard
Center for Design Research
Mechanical Engineering Design Division
Stanford University

Mark R. Cutkosky
Center for Design Research
Mechanical Engineering Design Division
Stanford University

ABSTRACT

We are interested in designing complex mechatronic systems which closely integrate electronics, actuators, and sensors with mechanical structures. Rapid prototyping techniques open new design possibilities for these systems, such as the ability to fabricate pre-assembled mechanisms. This paper presents a design approach that should encourage exploration of these new possibilities and thereby facilitate robot design.

In the described approach, engineers build designs using a library of three dimensional primitives and aggregations of primitives. Associated with each primitive is enough manufacturing process information to enable immediate manufacturability analysis. An algorithm is presented which automatically combines the process information from multiple primitives. A prototype system using this algorithm has been implemented with AutoCAD and the ObjectARX programming interface.

KEYWORDS: Layered manufacturing, design library, rapid prototyping, CAD, process planning.

1. INTRODUCTION

Three dimensional rapid prototyping processes (also called SFF for Solid Freeform Fabrication), such as layered shape deposition and laser sintering, allow designers to create complex structures and electromechanical assemblies with embedded sensors and electronics [Weiss, Merz, et al. 96] [Weiss, Prinz, et al. 96]. Although these processes overcome many limitations of traditional manufacturing, they also present new constraints that make it difficult for designers use them effectively. We argue that designers may be slow to realize the potential of SFF unless they are provided with design tools that encourage exploration and assure manufacturability. To this end, we have developed a "Design by Composition" approach that allows designers to construct integrated assemblies, including embedded parts, using a library of primitive elements that obey certain rules.

1.1 Two approaches to design

There are two approaches to designing parts for automated processing. In the first case, which we refer to as decomposition, the designer supplies a complete design, and the manufacturer is responsible for creating the manufacturing process plan. In the second approach, design by composition, the designer assembles the design from building blocks which include some process plan information.

1.1.1 Design Decomposition

The most common approach to design and process planning for 3D prototyping processes is to create the part geometry using a CAD system and then submit it to an algorithm that decomposes (e.g., slices) it into layers or other shapes that match manufacturing operations. This approach has several drawbacks. The algorithms available today do not always produce efficient or even correct decompositions of complex parts [Ramswami 97]. Moreover, because a purely geometric decomposition algorithm knows nothing about the intended function of a part, it cannot tailor the decomposition for individual features (e.g., avoiding seams or discontinuities on bearing surfaces). Adding such knowledge to the algorithm makes the decomposition process even more complex and error prone. The decomposition calculations are also time consuming and, because they are performed on the complete geometry of the part, they force the designer to endure an iterative design-submit-analyze-revise cycle.

1.1.2 Design by Composition

An alternative approach is to have the designer "build" a part from primitives, or building blocks, which already have partial process plans. A simple example part and the primitives used to create it are shown in Figure 1. In this case, the designer

sacrifices some design flexibility and assumes part of the computational burden of process planning. The primary advantage is that the process plan is available at design-time, allowing instant evaluation of part manufacturability.

Design composition distributes the manufacturability analysis and process planning tasks over a large number of designers, alleviating computational bottlenecks at the manufacturing site. This approach also facilitates development of design libraries; any design can be used as a primitive element in future designs.

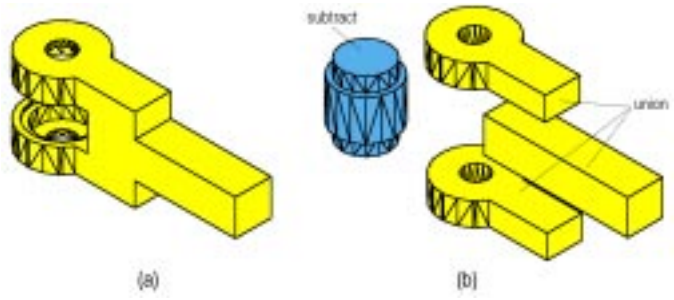

Figure 1. The design of the link in (a) can be created with the primitives in (b), which were created by combining cylinders and rectangular prisms and stored in a library.

2. SHAPE DEPOSITION MANUFACTURING

The manufacturing process we have examined most is Shape Deposition Manufacturing (SDM, see Figure 2), developed at Carnegie Mellon and Stanford University [Merz 94]. The basic cycle for this class of process is shown in Figure 2. Part material or support material is first deposited to near net shape using casting, micro-casting, laser welding, or other deposition processes. The material is then shaped using a CNC mill, or EDM machining, to obtain the desired accuracy and surface finish. Support material can be used as a mold to cast the next layer of part material. Prefabricated components can be embedded after any shaping step.

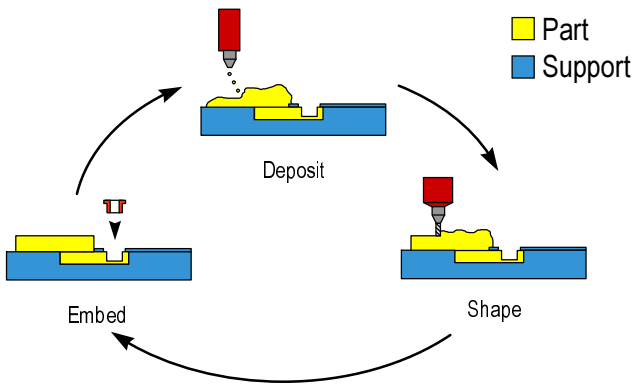

Figure 2. The Shape Deposition Manufacturing (SDM) process cycle. Material is alternately deposited and shaped by different processes. This methodology permits selecting a deposition process that gives good material quality and a shaping process that has good tolerances and surface finish.

Although the Design by Composition system presented here was developed for SDM, it is applicable to other automated manufacturing techniques, such as 3D Printing, Stereo-lithography, and UC Berkeley's CyberCut[1] machining process. A CyberCut interface is being developed in collaboration with UC Berkeley.

2.1.1 Pre-assembled mechanisms

One capability of SDM that we are especially interested in is the fabrication of pre-assembled mechanisms. Current SDM mechanisms are created with the same technique as monolithic parts. Moving parts are formed by depositing a thin layer of support material between two regions of part material. Because there is a minimum thickness for the support material, these mechanisms have a great deal of play in their joints. The performance of these devices is also limited by the materials which can be deposited and the achievable surface finish.

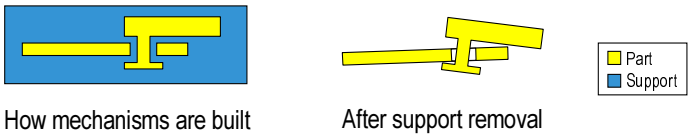

Figure 3. The current method for fabricating pre-assembled mechanisms with SDM. During fabrication the moving parts are separated by a thin layer of support material. The finite thickness of this material causes play in the mechanisms bearings.

The concept of composing a design suggests a better approach, in which some of the building blocks are prefabricated components. Because these components are embedded instead of deposited, a wider variety of materials is available. For example, Teflon bushings or precision steel shafts can be easily inserted into urethane or epoxy parts. A pre-assembled SDM mechanism built with embedded components can have the same precision and friction performance as a traditionally assembled mechanism, but without fasteners and other assembly constraints.

[1] http://CyberCut.berkeley.edu

3. DEFINITIONS

Before presenting the details of our algorithm for adding primitives and library components to an existing design, it is useful to define some terms associated with the SDM process. This section describes the concepts used in our Design by Composition system.

3.1 Growth direction and datum plane

Layered parts are normally built on a palette or elevator which can be accurately positioned in machine tools. The top surface of this palette defines a datum plane which represents the "bottom" of the part. The direction perpendicular to the datum plane is the growth axis, or growth direction. This is the direction in which subsequent layers will be deposited.

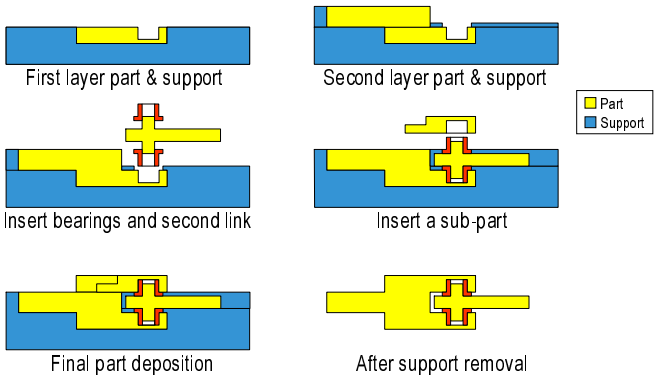

Figure 4. Building a pre-assembled mechanism with embedded components. Precision bearings and pre-fabricated parts can be inserted during the SDM fabrication process. This technique allows tighter fits and wider material choices.

3.2 Compacts

A compact is a three dimensional volume that represents one processing step [Merz 94]. Note that this definition of a "compact" is not related to the mathematical definition of a compact set. The information required to characterize the process step consists of material type and desired geometry.

3.2.1 Material

Currently, the compacts in our CAD system are either part material, support material, or an embedded component. An area for further research is the use of additional material types to support multi-material parts, or specialized manufacturing techniques for parts with specific functions (e.g., shafts or bearing surfaces).

3.2.2 Geometry

The most important information stored for each compact is the three dimensional geometry that must be deposited and shaped. Unlike the layers used in other rapid-prototyping processes, compacts are not 2D slices of a part – each compact can have complex three dimensional geometry. In fact, the compacts need not be contiguous; designs frequently have separate volumes of a single material that can be manufactured at the same time. In our system, these volumes would be grouped as a single compact.

3.2.3 Test for "compactness"

The geometry of legal compacts is determined by what is manufacturable in a single SDM processing cycle, which includes a deposition step and a shaping step. During the deposition step, new material is deposited in a liquid form on top of existing compacts. As a result, the underside of the new compact is shaped by a casting-like process in which the prior compacts act as a mold.

During shaping, the geometry of the new compact's upper surfaces is formed by CNC milling or EDM. These processes are limited by tool access constraints. This fact leads to the primary rule for compact geometry: all machined surfaces must face "upwards" (relative to the growth direction) [Ramaswami 97]. More precisely, the dot product of the surface normal and the growth direction must be positive for machined surfaces, and negative for cast surfaces.

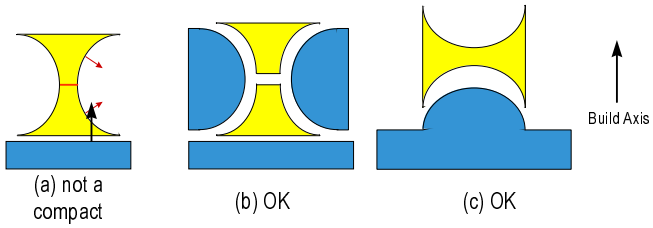

Figure 5. Testing for a legal compact. In (a), a ray in the build direction leaves and re-enters the shape, violating the "test for compactness." To make valid compacts, the shape can be split at its waist (b), or rotated by 90º (c)..

Figure 5 illustrates a simple test for compactness. In a valid compact, any ray cast in the build direction will enter and leave the solid no more than once. An hourglass shape, aligned with the build axis, violates this test, and must be split into two compacts along its "waist." Algorithms to split an arbitrary solid model into legal compacts are an area of current research [Ramaswami 97].

Equation 1 is equivalent to the graphical test shown in Figure 5. It states that a solid bounded by a surface, $\boldsymbol{S}$, is a compact only if there exists no point, p, on $\boldsymbol{S}$ which is an

inflection point with an undercut surface above an upwards-facing surface (with respect to the growth axis, *Z*).

$$\sim \exists p \left[\frac{\partial \mathbf{S}}{\partial Z} = 0 \quad \wedge \quad \frac{\partial^2 \mathbf{S}}{\partial Z^2} < 0 \right] \tag{1}$$

3.3 Applicability to other processes

Although the concept of a compact was invented in regard to the SDM process, we believe it has applicability to other layered manufacturing techniques. Nearly all layered techniques, including stereo-lithography and 3D Printing, use some form of temporary support structure for overhanging features. (For stereo-lithography, the support structure is a web-like framework that is manually removed after fabrication; in 3D Printing, the support structure is unfused powder.)

Determining were support is required, and ensuring that it is removable at the end of the fabrication process adds additional planning complexity for these processes. Our design composition technique, which automatically determines support compacts, could facilitate these computations.

3.4 Compact lists

In our design system, each primitive includes a compact list which captures the basic plan for its fabrication. A compact list is an ordered list of the compacts which would be used to manufacture the primitive. The designer is free to ignore the details of the compact lists. In our prototype implementation, the compacts are created on hidden layers in an AutoCAD model. Each compact in the list has specifications for:

- Build order
- Material: part, support, or embedded component
- 3D Geometry.

The compact lists for simple shapes (2½D extrusions) can be generated algorithmically. Compact lists for more complex shapes will be stored in the design library with the primitive. Any new design constructed from existing primitives can be added to the design library as a new primitive.

Because manufacturability rules are specified in terms of compact geometry and topology, including the compact lists in the design representation permits on-the-fly evaluation of part manufacturability. The CAD system can notify the designer immediately when a new feature produces a non-manufacturable design. This feedback eliminates the need for an iterative design-evaluate-revise approach.

Correct operation of the compact list merging algorithm described in Section 4 requires that the compacts in the compact list must completely fill the space defined by projecting the union of all part material in the growth direction to the top and bottom of the manufacturing workspace. The support material on top of the part is not required for fabrication, but is required for the algorithm to function properly.

3.5 Build order

Each compact in the compact list has an integer that represents its build order; i.e., the compact that will be manufactured first has build order 1, and so on. In general, there are many valid build orders; the algorithm described in this paper produces a single, arbitrary build order. We have begun to develop an extension to this system that uses a directed graph to represent all of the possible build orders to permit subsequent optimization.

3.6 Library elements - Primitives

The system we envision is based on a design library which contains three dimensional shapes called primitives that can be used as design building blocks. These primitives can be combined with standard Boolean operations: union, subtraction, and intersection. As stated above, to the designer, each primitive appears as a single 3D solid with arbitrary geometry. On hidden layers of the CAD drawing, however, is a compact list, that specifies the manufacturing plan for the primitive.

It is important to realize that the compact list of a primitive represents a “high-level” process plan for its fabrication; the compact list specifies the material type and geometry to be created in each manufacturing step. The library elements do not include detailed manufacturing data like CNC paths or deposition parameters. The purpose of the merging algorithm described below is to combine the compact lists from two primitives into a new list that represents a feasible process plan for the new part design.

Maintaining this level of manufacturing information in the design permits quick manufacturability feedback to the designer, and potentially gives him partial control over the fabrication process. In the short term, providing the SDM manufacturer with this level of process plan information permits more automation in the planning and manufacturing operations.

3.7 Simple shapes

We define a simple shape as a three dimensional solid formed by extruding a closed two dimensional curve in the growth direction. The resulting solid will have an upper and lower face which are perpendicular to the growth direction, and a number of side faces which are parallel to the growth direction. The compact list for a simple shape can be generated automatically.

4. COMPACT LIST MERGING ALGORITHM

The heart of the design-by-composition approach is a merging algorithm which allows simple shapes and library components to be combined to create new designs. When the designer combines two primitives to form a new design, the CAD system must combine the corresponding compact lists. We have developed and implemented an algorithm which can combine two arbitrary compact lists and will always produce a valid result.

In this section, we refer to the two source compact lists, *A* and *B*, corresponding to two primitives, either of which may be

a simple shape, a library element, or an incomplete design. These lists are combined to form a result list, *C*. Lower case letters refer to individual compacts, e.g., *a* is a compact in *A*.

4.1 Intersection compacts

When compact lists *A* and *B* are combined, the algorithm finds all the intersections between a compact in *A* and a compact in *B*. The algorithm creates an intersection compact, *c*, for every intersection between any *a* and *b*. If *A* contains *n* compacts, and *B* contains *m* compacts, there are at most *n·m* intersection compacts.

The material for an intersection compact is specified by a truth table (see Figure 6) that depends on the type of merging operation, addition, subtraction, or intersection. The truth table is the only part of the algorithm that depends on the merging operation.

The build order for an intersection compact is the sum of the build orders of the two source compacts:

$$order(c) = order(a) + order(b) \qquad (2)$$

Add

a	*b*	*c*
P	P	P
P	S	P
S	P	P
S	S	S

Subtract

a	*b*	*c*
P	P	S
P	S	P
S	P	S
S	S	S

Intersect

a	*b*	*c*
P	P	P
P	S	S
S	P	S
S	S	S

Figure 6. Truth tables for compact list merging operations. The tables shown specify the material type (part or support) for a compact, c, which is formed by the intersection of two source compacts, a and b.

4.2 Non-intersecting compacts

When two compact lists are combined, portions of some of the compacts in *A* or *B* may not intersect any compact in the other list. These compacts, after subtraction of any portions that do intersect, are appended to the result list, *C*. The build order of the non-intersected compacts is multiplied by 2. Doubling the build order produces an arbitrary, valid order for the result compact list. More complex build order techniques may produce valid process plans that are more optimal. In the future, we plan on moving to using a directed graph to represent all possible manufacturing sequences for a compact list. This modification will remove many of the current implementations arbitrary aspects, including this rule.

The material type for the non-intersecting compacts depends on the merging operation, as shown in the truth tables in Figure 7.

Add

a	*b*	*c*
P		P
S		S
	P	P
	S	S

Subtract

a	*b*	*c*
P		P
S		S
	P	S
	S	S

Intersect

a	*b*	*c*
P		S
S		S
	P	S
	S	S

Figure 7. Truth tables for non-intersecting compacts. The geometry of c is the portion of a or b which does not intersect with any compact in the other list (B or A, respectively).

For compact lists *A* and *B* with *n* and *m* compacts, respectively, the maximum number of non-intersecting compacts is *n+m*.

4.3 The algorithm

The rules presented above for intersecting and non-intersecting compacts are the basis of our compact list merging algorithm.

4.3.1 Description

Two compact lists can be combined by the algorithm in Figure 12. Each compact in list *A* is intersected with each compact in list *B*. The intersection solids (if they exist) are then subtracted from the two source compacts. A truth table (the function *f*) which depends on the operation (add, subtract, or intersection) specifies the material (part or support) of the intersection compact. This algorithm allows arbitrarily complex compact lists to be combined.

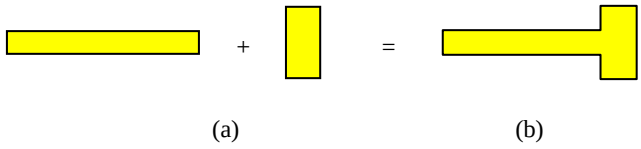

Figure 8. The designer wants to combine the two primitives in (a) to form the shape in (b).

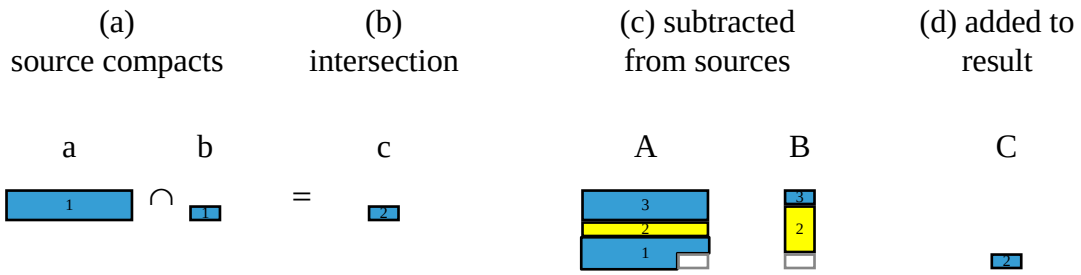

Figure 9. Step 1 of the merging operation. The two compacts shown in (a) are intersected, producing the new compact shown in (b). The intersection is subtracted from the two source compact lists (c), and is added to the result list, C, in (d).

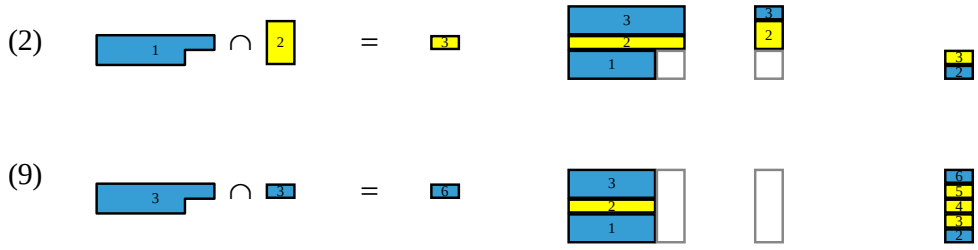

Figure 10. Steps 2 and 9 of the merging operation. In step 2, the remaining piece of compact A1 is intersected with compact B2. In step 9, the remainder of compact A3 is intersected with B3. This produces the final intersection compact.

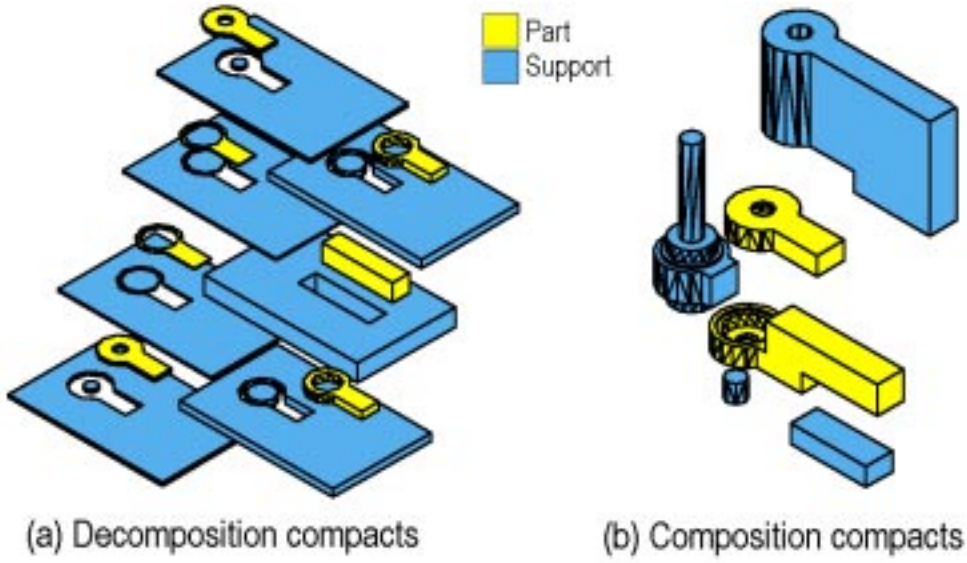

Figure 11. Different compact lists for the simple link part in Figure 1 . In (a) the compacts were created by automatic decomposition software [Ramaswami 97] (b) the complete compact list generated by the merging algorithm.

Each compact in the result compact list has a build order. These build order values, however, are not unique, and there may be gaps in the numbering sequence. (For example, there will never be a compact with build order 1). This order does represent a valid manufacturing plan, however. The sequence of compacts with equal build order numbers is arbitrary; during the simplification step described in Section 4.4, the order of these compacts is adjusted to minimize the number of compacts.

$\forall\, b \in B; \forall\, a \in A$
  new($c \in C$) = $a \cap b$
  $a = a - c$
  $b = b - c$
  order(c) = order(a) + order(b)
  material(c) = f(material(a), material(b), operation)
$\forall\, b \in B$
  order(b) = 2•order(b)
  material(b) = g(material(b), operation)
$\forall\, a \in A$
  order(a) = 2•order(a)
  material(a) = g(material(a), operation)
$C = C \cup A \cup B$

Figure 12. Compact-list merging algorithm. The function f is specified by the truth tables shown in Figure 6. The function g is specified by the truth tables in Figure 7.

4.3.2 Example

This section illustrates the compact list merging algorithm with an example. Figure 8 shows the two primitives the designer is combining, and the resulting shape. Figure 13 shows the compact lists for the two primitives. (In all illustrations in this section, the growth direction is vertical.)

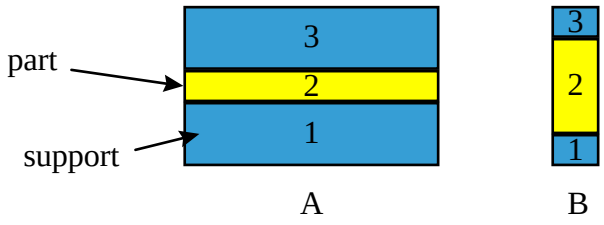

Figure 13. The two compact lists that will be combined.

Figure 9 shows the first step of the merging operation. Compacts A1 and B1 are intersected to form compact c. The intersection is subtracted from the two source compacts, removing a portion of A1 and all of B1. Finally, the intersection compact is added to the result compact list, C.

In step 2, shown in Figure 10, the remainder of compact A1 is intersected with B2. Again, the intersection is subtracted from the source compacts and added to the result list. In step 3 (not shown) the remainder of compact A1 will be intersected with compact B3. The result of this intersection is a null compact, so no changes are made to the source or result compact lists. In step 9, compact A3 (minus the part that was removed by step 8) is intersected with B3 to produce the final intersection compact.

After step 9, the compacts in list B have been completely "consumed" by the intersection and subtraction operations. There are still three compacts in list A, however. These non-intersecting compacts are added to the result list and their build orders are doubled. The complete result list is shown in Figure 14(a). A simplification operation, described in Section 4.4, converts the result list to the compact list shown in Figure 14(b).

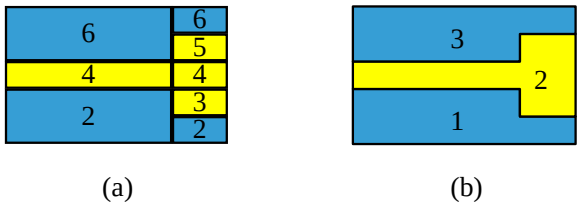

Figure 14. The non-intersecting pieces of the A compacts are added to the result list, with the build orders doubled. There are no non-intersecting B compacts in this example. (a) shows the complete result compact list. (b) shows the result compact list after simplification (Section 4.4).

4.4 Compact list simplification

The merging algorithm produces a result compact list with many compacts (Figure 14(a)). In the worst case, merging n primitives which each contain m compacts produces a compact list with $m^n + nm$ compacts.

The result compact list can be simplified considerably by combining (with a Boolean union) sequential compacts of the same material. Figure 14(b) shows the compact list for the part after simplification. Currently, we simplify the compact list after every merging operation to produce the minimum number of compacts. An area for future refinement is to investigate alternative build orders to optimize other figures of merit.

4.5 Modifications to the algorithm

The compact-list merging algorithm presented above can be modified to support several extensions including embedded components and special merging operations.

4.5.1 Supporting embedded components

To support embedded components, the primary change to the algorithm presented above is to create a new material type, “embedded.” Space limitations preclude a discussion here, but the extension is straightforward. The basic difference is that, unlike the existing part and support materials, embedded component compacts cannot be subdivided by primitive merging operations.

4.5.2 Build order graph

The compact list merging algorithm maintains a valid, but arbitrary, build order for the compacts in the process plan. We have begun development of a directed graph (called the Compact Adjacency Graph, or CAG) to represent the order dependencies between compacts. [Pinilla, Kao, et al. 97] This compact adjacency graph is capable of representing all possible build orders for a given set of compacts. This representation is a least-commitment approach which allows more opportunity to optimize the build sequence based on design requirements, or even to make scheduling decisions during fabrication based on machine availability and workload.

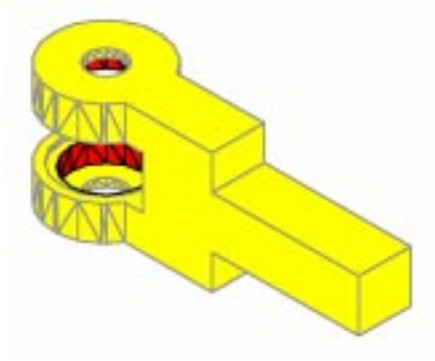

Figure 15. The shaded surfaces locate bearings in this link.

Figure 15 shows the surfaces of the part from Figure 1 which have critical tolerances. Using the CAG representation could allow the designer or manufacturer to select a process plan (a subset of the graph) that manufactures these surfaces at the same time, ensuring that they will be properly aligned.

4.5.3 An algorithm for fluid piping

A horizontal pipe can be used as a library primitive. The corresponding compact list consists of 5 compacts: support material above and below, the upper and lower halves of the pipe, and the pipe’s interior.

When two pipes intersect, the designer probably, but not necessarily, wants to form a fluid connection. With the standard algorithm presented in Section 4, the interior and exterior of the pipes would be support material, and the pipe walls would be part material. Using the Add truth table, two intersecting pipes would form a cross with no fluid connections between the four ports. Adding a third material type, fluid, for the interior of the pipe, and extending the truth table as shown in Figure 16, causes two intersecting pipes to form a fluid connection.

Add Pipes

a	*b*	*c*
P	P	P
P	S	P
P	F	F
F	P	F
F	S	F
F	F	F
S	P	P
S	S	S
S	F	F

Figure 16. Pipe merging truth table. Because the intersection of pipe and fluid is fluid, two intersecting pipes will form a fluid connection.

5. IMPLEMENTATION

The compact merging algorithm has been implemented as an ARX (AutoCAD Runtime Extension) application that works with AutoCAD Release 14. This implementation allows a designer to create cylindrical and rectangular prism primitives which can be manipulated by standard AutoCAD commands. Once the designer has created and positioned primitives, they can be merged together. The output of the merging operation is a new, more complex primitive which can be saved in a design library and used in future designs.

Part and support material compacts are automatically generated on hidden layers of the drawing. The compact lists of both primitives are then combined using the algorithm described above. Both addition and subtraction operations are supported (an intersection operation can be easily added). On the visible layer, the designer sees the primitives combine with the appropriate action (either union or subtraction).

The net result is that the designer can create parts using the familiar AutoCAD interface. With the part and support layers hidden, this system behaves the same as the standard AutoCAD 3D modeling interface. The ARX application can then output all of the files necessary for the SDM process planner to manufacture the part. (Currently, because of software limitations in the Stanford SDM facility, part fabrication involves several manual steps.)

The design-by-composition ARX implementation is part of a broader effort to provide a design/manufacturing interface for SDM and related prototyping processes. In this interface, designers can use a web-based broker to find suitable prototyping processes and facilities, as well as libraries of parts

and tools for design and manufacturing analysis. The ARX application described in this paper will be available as a downloadable "plug in" for AutoCAD.

ACKNOWLEDGMENTS

Thanks are due to members of the Stanford CDR and RPL project teams for their advice regarding this paper. This work has been supported by the National Science Foundation under grant MIP9617994.

REFERENCES

Merz, R., Prinz, F.B., Ramaswami, K., Terk, M., Weiss, L., "**Shape Deposition Manufacturing**", *Proceedings of the Solid Freeform Fabrication Symposium*, University of Texas at Austin, August 8-10, 1994.

Pinilla, J.M., Kao, J.H., Binnard, M., Prinz, F.B., "**The Compact Graph Format: an Interchange Standard for Solid Freeform Fabrication**", *NIST Measurement and Standards Issues in Rapid Prototyping Workshop*, Gaithersburg, MD, 1997

Ramaswami, K., "**Process Planning for Shape Deposition Manufacturing**," *PhD Dissertation*, Stanford University, Stanford, California, 1997.

Requicha, A., Chan, S., "**Representation of Geometric Features, Tolerances, and Attributes in Solid Modelers Based on Constructive Geometry**," *IEEE Journal of Robotics and Automation*, Vol. RA-2 No. 3, September, 1986.

Requicha, A., Voelcker, H., "**Boolean Operations in Solid Modeling: Boundary Evaluation and Merging Algorithms**," *Proceedings of the IEEE*, Vol. 3 No. 1, January, 1985.

Weiss, L.E., Merz, R., Prinz, F.B., Neplotnik, G., Padmanabhan, P., Schultz, L., Ramaswami, K., "**Layered Manufacturing of Heterogeneous Structures**", submitted to the *Journal of Manufacturing Systems*, 1996.

Weiss, L.E., Prinz, F.B., Neplotnik, G., Padmanabhan, P., Schultz, L., Merz, R., "**Shape Deposition Manufacturing of Wearable Computers**", *Proceedings of the Solid Freeform Fabrication Symposium*, The University of Texas at Austin, August 10-12, 1996.

Woodbury, R., Carlson, C., Heisserman, J., "**Geometric Design Spaces**," *EDRC Memo 48-13-89*, Engineering Design Research Center, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1989.