Design by Composition for Rapid Prototyping

A DISSERTATION

SUBMITTED TO THE DEPARTMENT OF

MECHANICAL ENGINEERING

AND THE COMMITTEE ON GRADUATE STUDIES

OF STANFORD UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

IN

MECHANICAL ENGINEERING

MICHAEL B. BINNARD

FEBRUARY, 1999

© COPYRIGHT BY MICHAEL BINNARD 1999

ALL RIGHTS RESERVED

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Mark Cutkosky, Principal Adviser

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Friedrich Prinz

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Thomas Kenny

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Kosuke Ishii

Approved for the University Committee on Graduate Studies:

ABSTRACT

Three dimensional Rapid Prototyping processes, also called Layered Manufacturing or Solid Freeform Fabrication (SFF), promise designers the ability to automatically fabricate complex shapes. SFF processes were invented with the assumption that designers would submit complete part models for automated planning and manufacturing. This planning process is normally based on some form of "decomposition," for example slicing into layers. Especially for newer, more complex SFF processes, there are several disadvantages to this approach, primarily that decomposition is difficult and doesn't reliably produce good process plans. Furthermore, it is hard for the designer to get feedback on the manufacturability of his design, and today's decomposition systems are not fully automated.

This dissertation presents an alternative approach, "design by composition," where users build designs from "primitives" that include high-level manufacturing plans. When the user combines two primitives with a Boolean operation, software will automatically generate a manufacturing plan for the new design from the plans for the source primitives. In contrast to the decomposition method, design by composition offers several benefits to designers, primarily access to manufacturability feedback during design-time, a greater degree of automation, the ability to create designs with embedded components (such as sensors, electronic circuits, bearings, and shafts), and enhanced control over manufacturing plans. These advantages make design by composition a more attractive approach to SFF processing, especially for designers who are new to these techniques. This dissertation presents scalable, robust algorithms for the implementation of design by composition, and describes a prototype implementation and sample parts which were designed with it.

ACKNOWLEDGMENTS

I wish to thank all the members of the Stanford Center for Design Research and Rapid Prototyping Laboratory for their advice and support for this research. This work has been supported by the National Science Foundation under grant MIP-9617994 and by the Office of Naval Research under grant N00014-98-1-0669.

TABLE OF CONTENTS

GLOSSARY xi

ILLUSTRATION AND NOTATION CONVENTIONS xv

1. INTRODUCTION - TWO APPROACHES TO DESIGN 16
- **1.1 Design Decomposition versus Design by Composition** 17
  - 1.1.1 Design Decomposition 17
  - 1.1.2 Design by Composition 18
- **1.2 Questions to Answer** 19
  - 1.2.1 Algorithms 20
  - 1.2.2 Designs that include manufacturing information 20
  - 1.2.3 Design flexibility tradeoffs 21
- **1.3 A Design-Manufacturing Interface** 22

2. DESIGN AND PLANNING FOR RAPID PROTOTYPING 23
- **2.1 Shape Deposition Manufacturing** 24
  - 2.1.1 Multi-material and graded material parts 25
  - 2.1.2 Embedded components 25
  - 2.1.3 Pre-Assembled mechanisms 26
- **2.2 Process planning for SDM** 28
  - 2.2.1 ACIS decomposition 30
  - 2.2.2 Unigraphics decomposition 32
  - 2.2.3 Design by Composition 32
- **2.3 Manufacturing** 33

3. DEFINITIONS FOR DESIGN BY COMPOSITION 36
- **3.1 Growth direction & Datum plane** 36
- **3.2 Projected volume** 37
- **3.3 Simple shapes** 38
- **3.4 Compact** 39
  - 3.4.1 Material 39
  - 3.4.2 Geometry 39
  - 3.4.3 Valid operations 40
- **3.5 Compact sets** 41
  - 3.5.1 Rules for valid compact sets 42
  - 3.5.2 Simple shapes 43
  - 3.5.3 Storage in a design library 43
- **3.6 Compact precedence graph - CPG** 44
  - 3.6.1 Rules for a valid CPG 44
  - 3.6.2 Simple shapes 45
- **3.7 Primitives (aka. Library elements)** 45
  - 3.7.1 The Compact graph 46
  - 3.7.2 Valid operations 46

4. THE COMPACT MERGING ALGORITHM 48
- **4.1 How it works** 48
  - 4.1.1 Intersection compacts 51
  - 4.1.2 Non-intersecting compacts 52
  - 4.1.3 Embedded components 53
  - 4.1.4 Compact set simplification 55
- **4.2 Example** 56
- **4.3 Completeness of the algorithm** 58
  - 4.3.1 Closure 59
  - 4.3.2 Scalability 67

5. COMPACT PRECEDENCE GRAPHS 69
- **5.1 Merging algorithm** 69
  - 5.1.1 Non-intersecting compacts 71

5.1.2 Intersection compacts ... 71
5.1.3 Embedded components ... 74
5.1.4 Extraneous precedence links ... 78
5.2 Simplification ... 82
5.2.1 Rules for legal simplifications ... 82
5.2.2 Choosing between simplification options ... 83
5.3 Algorithm Completeness ... 85
5.3.1 Closure proofs ... 86
5.3.2 Scalability ... 89

6. DESIGN BY COMPOSITION ... 91
6.1 Compact graphs ... 91
6.2 Restrictions on design flexibility ... 92
6.3 Benefits of this approach ... 92
6.3.1 Immediate manufacturability feedback ... 93
6.3.2 Level of automation ... 95
6.3.3 Mechanisms with embedded components ... 97
6.3.4 Design libraries ... 98
6.3.5 Enhanced control over manufacturing ... 99
6.3.6 Computational burden ... 99
6.4 Example scenario ... 100

7. IMPLEMENTATION ... 103
7.1 Design system - AutoCAD plug-in ... 103
7.2 Compact Precedence ... 104
7.3 Manufacturing interface ... 105
7.4 Lessons learned ... 106
7.4.1 Algorithm development ... 107
7.4.2 Use by novice designers ... 107

8. FABRICATION EXAMPLE ... 109
8.1 Comparison to decomposition approach ... 109
8.2 Fabrication sequence ... 112
8.2.1 Fabrication of the lower leg link ... 114
8.2.2 Fabrication of the leg mechanism ... 114
8.3 Observations ... 116
8.3.1 Manufacturing process constraints ... 117
8.3.2 Embedded components ... 118

9. CONCLUSIONS AND FUTURE WORK ... 120
9.1 Tradeoff between decomposition and composition ... 121
9.1.1 Design flexibility ... 121
9.1.2 Feedback to the designer ... 121
9.2 Possible extensions ... 123
9.2.1 Multi- and graded-material parts ... 123
9.2.2 Fluid piping and other specialized applications ... 123
9.2.3 Parametric primitives ... 125
9.3 Integration with a compatible Decomposition system ... 125

EPILOGUE ... 128

10. APPENDIX - SOFTWARE IMPLEMENTATION ... 129
10.1 Overview ... 129
10.1.1 New AutoCAD commands ... 130
10.1.2 Source code files ... 131
10.1.3 Interface with SDM path planning ... 131
10.2 Class heirarchy ... 132
10.2.1 Primitive ... 133
10.2.2 CylinderPrim - cylindrical primitives ... 134
10.2.3 BoxPrim - rectangular box primitives ... 135
10.2.4 Compact ... 135
10.2.5 Reactor ... 137

10.3 GLOBAL FUNCTIONS 137
10.4 KNOWN BUGS AND LIMITATIONS 140
10.4.1 slice height propagation 140
10.4.2 Manual compact set creation 140
10.4.3 Ordering of embedded components 141
10.4.4 Non-intersecting compacts 141
10.4.5 Reactors 141
10.5 CURRENT DEVELOPMENT EFFORTS 141
10.5.1 New primitive types 142
10.5.2 Compact Precedence Graph manipulations 142

REFERENCES 143

LIST OF FIGURES

Figure 1. Colors used in illustrations xv
Figure 2. Design by composition example 18
Figure 3. Primitives include high-level manufacturing plans 19
Figure 4. The Shape Deposition Manufacturing (SDM) process cycle. 24
Figure 5. SDM injection molding tools 25
Figure 6. "The Cranky" pre-assembled mechanism 26
Figure 7. Current SDM Mechanisms. 27
Figure 8. Building a pre-assembled mechanism with embedded components 27
Figure 9. Decomposition design-manufacturing interface 28
Figure 10. Levels of decomposition in SDM 29
Figure 11. Decomposition example 31
Figure 12. A problem with automated compact-splitting 31
Figure 13. Design by composition compacts for Figure 11(a). 32
Figure 14. Degree of automation in Process Planning 33
Figure 15. Polymer deposition 34
Figure 16. The SDM milling machine 34
Figure 17. The SDM milling machine in operation 35
Figure 18. The SFF workspace 37
Figure 19. Projected volume of a primitive. 38
Figure 20. Simple shapes 38
Figure 21. Testing for a legal compact 40
Figure 22. A primitive and its compact set 41
Figure 23. Compact sets for simple shapes 43
Figure 24. A primitive (a) and its compact precedence graph (b) 44
Figure 25. CPGs for simple shapes 45
Figure 26. Z-translation of a compact set 46
Figure 27. An example merging operation 49
Figure 28. The result compacts from Figure 27 50
Figure 29. Compact-set merging algorithm 50
Figure 30. The four merging operations 51
Figure 31. Truth tables for compact set merging operations 52
Figure 32. Result compact set projected volume 52
Figure 33. Embedded component merging algorithm 54
Figure 34. Embedded component truth tables 55
Figure 35. Compact set simplification 56
Figure 36. A simple design by composition example 56
Figure 37. The two compact sets that will be combined. 57
Figure 38. Step 1 of the merging operation 57
Figure 39. Steps 2 and 9 of the merging operation 58
Figure 40. Compact set simplification 58
Figure 41. The two compacts whose intersection is shown in Figure 42. 64
Figure 42. Intersection compact 65
Figure 43. The result compacts from Figure 27 66
Figure 44. Source CPGs for the example in Figure 27 70
Figure 45. Result CPG for the example 70
Figure 46. Precedence rule for intersection compacts 71
Figure 47. Intersecting compacts (case 1) 72
Figure 48. Intersecting compacts (case 3) 72
Figure 49. Algorithm for generating CPG links 73
Figure 50. Condition 1 from Figure 46. 73
Figure 51. Condition 3 from Figure 46. 74
Figure 52. Precedence rules for embedded components 75
Figure 53. Merging example with embedded component b_2 75
Figure 54. Result of merging operation with embedded component 76
Figure 55. A partially-intersecting embedded compact 76
Figure 56. Algorithm for embedded component CPG links 77
Figure 57. Extraneous CPG links 78

Figure 58. The four possibilities for a_2 and a_3 intersecting b_1 and b_2 80
Figure 59. Special cases of embedded component precedence 81
Figure 60. Compact graph simplification 81
Figure 61. Two options for simplification 83
Figure 62. CPG simplification rules 84
Figure 63. Simplification algorithm 85
Figure 64. A simplification that creates a cycle 89
Figure 65. Example design rules for SDM 94
Figure 66. A simple robot leg built with SDM 95
Figure 67. Closeup of cutter radius fillets 96
Figure 68. Design by composition design-manufacturing interface 96
Figure 69. A pin joint with embedded components 97
Figure 70. A robot leg with embedded DC motors 98
Figure 71. An example design 99
Figure 72. The design with an array of holes 101
Figure 73. The completed design 101
Figure 74. Manufacturing preview of the design 102
Figure 75. The effects of compact order and shrinkage 105
Figure 76. Single Step Geometries (SSGs) for Figure 75(a) 106
Figure 77. Jorge Cham's spacer invention 108
Figure 78. Robot leg with integrated sensors and actuator 110
Figure 79. Compacts and fabrication sequence of lower link 111
Figure 80. Photograph of lower link 112
Figure 81. The embedded gear and piston rod connection 113
Figure 82. The compacts for the leg mechanism 113
Figure 83. The first urethane compact 115
Figure 84. Insertion of the embedded components 115
Figure 85. The second support material compact 116
Figure 86. The final part material compact 116
Figure 87. Process plan for embedded components 118
Figure 88. Using a temporary material around embedded components 119
Figure 89. Compacts displayed in AutoCAD 122
Figure 90. Output from the SDM CNC path preview utility 122
Figure 91. Pipe merging truth table 124
Figure 92. A parametric primitive 124
Figure 93. Integrating design by composition and decomposition 126
Figure 94. Custom classes in the SDM plug-in 132

GLOSSARY

Child compact	For a given source compact in a merging operation, its children are the intersection compacts formed by its intersection with the other source compact set. For example, the children of compact a_3 are all the intersection compacts a_3b_y. (Section 5.1)
Compact	A volume of a single material (which may not be contiguous) that can be produced in a single SDM process cycle. (See Section 3.4)
Compact graph	A compact set and its precedence graph; the compact graph represents a high-level manufacturing plan for a primitive: material and geometric information is provided by the compact set, and ordering information is represented by the CPG. (Section 3.7.1)
Compact set	A set of compacts that specify a the geometry and material type for each step in a manufacturing plan for a primitive. (Section 3.5)
Compact splitting	The procedure of decomposing a complete design into the part and support material compacts that will be used for its fabrication. (Section 2.2)
CPG	A Compact Precedence Graph – see Precedence Graph below.
Datum plane	The XY plane which is the bottom of the SFF workspace. (Section 3.1)
Decomposition	The process of breaking a complete design into simpler geometric elements which are then used to generate machine tool paths. (Section 1.1)
Embedded component	Any prefabricated component which is inserted into an SDM part or assembly during fabrication. Examples include circuit boards, sensors, bearings, and shafts. (Section 2.1.2)
Global interactions	Interactions between different features (or compacts) on a part. For example, determining whether cutting tools can access a particular part feature depends on the "global" part geometry.

Growth direction	The Z-direction. This is the direction, perpendicular to the datum plane, in which subsequent layers are deposited. Also called growth axis. (Section 3.1)
Layer slicing	For SFF processes that use thin layers, this is the procedure of slicing a complete design into two dimensional cross sections. It is analogous to compact splitting for SDM. (Section 1.1)
Library element	A three dimensional shape, a feature, simple shape, part, or integrated assembly, that is stored in a design library. In design by composition, designers construct designs by combining library elements. Synonymous with "Primitive." (Section 3.7)
Local interactions	Interactions between a particular feature (or compact) that depend only on the geometry and material type of itself and its neighboring features (or compacts). For example, the surface quality on an SDM part depends primarily on the material type of the part and the adjacent support material.
Merging	The basic operation in design by composition; when the designer combines two primitives with a Boolean operation, the design by composition software merges the compact graphs of the two source primitives to produce a compact graph for the result primitive. (Chapter 4)
Parent compact	One of the two source compacts that was intersected to form a particular intersection compact. For example, the intersection compact $a_x b_y$ has two parent compacts: a_x and b_y. (Section 5.1)
Part material	Part material compacts specify the shape and material of finished SDM parts. (Section 2.1)
Path planning	The process of generating CNC (deposition or cutting) paths from geometric models. (Section 2.2)
Precedence graph	(also called Compact Precedence Graph, or CPG) A graph that represents the ordering constraints between compacts in a compact set. (Section 3.6)
Primitive	An element in a design library used as a building block for subsequent designs. (Section 3.7)

Process plan	The sequence of manufacturing steps that will be used to fabricate a part. Process planning is the task of generating a process plan. (Section 2.2)
Process	The word "process" is often used to refer to a manufacturing process.
Projected volume	The volume defined by projecting a primitive in the growth direction. (Section 3.2)
SDM process cycle	A single deposition step followed by a single shaping step. (Section 2.1)
SDM	The Shape Deposition Manufacturing process. SDM produces parts by sequential deposition and shaping processes. (Section 2.1)
SFF	Solid Freeform Fabrication. A class of manufacturing processes often referred to as layered manufacturing or rapid prototyping that create parts by sequential deposition (and optionally shaping) of layers or compacts.
Simple shape	a three dimensional shape formed by extruding a closed two dimensional shape in the growth direction. A simple shape has horizontal top and bottom surfaces, and vertical sides. (Section 3.3)
Simplification	The process of combining nodes in a compact graph to produce a simpler graph. (Section 5.2)
Single Step Geometry	(also called SSG), a three dimensional shape that represents the state of a SFF part at some point in the fabrication process. At each stage, the SSG is defined by the union of the current compact with all prior compacts. SSGs are guaranteed to have all their surfaces accessible for shaping; there are no undercut features.
Stereolithography	(also called SLA), an SFF process that builds parts by using a laser to selectively cure a liquid polymer resin. SLA is the oldest and most widely available SFF technique.

Support material	SDM parts are built embedded in a sacrificial support material. Support material compacts specify the shape of this support. Essentially, the union of the support material compacts is the inverse of the union of the part material compacts. (Section 2.1)
2½ dimensional (2½D)	Any three dimensional shape that has only horizontal and vertical surfaces. In other words, a shape that consists of a number of layers with a 2D cross-section and corresponding thickness.

ILLUSTRATION AND NOTATION CONVENTIONS

The illustrations in this dissertation use the color convention shown in Figure 1. In all of the figures, the growth direction is vertical. Many of the pictures show 2D cross sections of compacts and parts; it is important to remember that the actual compacts and results are 3D shapes.

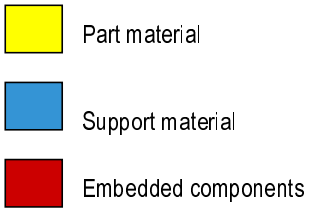


Figure 1. Colors used in illustrations

The notation used throughout the dissertation is:

- Boldface letters, **A** and **B**, refer to primitives.
- Upper case italics, A and B, are used for compact sets.
- Lower case italics refer to individual compacts from the corresponding compact set: a is a compact in A. A particular compact is identified with a subscript, e.g. a_1.
- In Compact Precedence Graphs, A link from node i to node j is written as $i \rightarrow j$ (read as "i precedes j"), and indicates that compact i must be fabricated before compact j.
- The intersection of compacts a_x and b_y is a compact, c, identified as $a_x b_y$.
- W refers to the set of all points in the SFF workspace.
- $p(\mathbf{A})$ is the projected volume (defined in Section 3.2) of a primitive, **A**.

1. Introduction - Two Approaches to Design

Three dimensional rapid prototyping processes (also called SFF for Solid Freeform Fabrication), such as Stereolithography (SLA) **[Jacobs 92]**, 3D Printing **[Sachs, et al. 90]**, and Shape Deposition Manufacturing (SDM) **[Merz, et al. 94]** offer two principal benefits to designers: the ability to fabricate complex structures that cannot be built with traditional techniques, and a greater degree of automation in process planning and manufacturing than traditional manufacturing processes.

SFF processes, particularly SDM allow designers to create complex structures and electromechanical assemblies with embedded sensors and electronics **[Weiss, et al. 96a] [Weiss, et al. 96b]**. Although these processes overcome many limitations of traditional manufacturing, they also present new constraints that make it difficult for designers to use them effectively. Without design tools that encourage exploration and assure manufacturability, designers will be slow to exploit the potential of SFF.

Automation of traditional manufacturing techniques is difficult because of issues like fixturing, tool access, and stiffness analysis. These problems are "global interactions" because they depend on multiple features, or even the complete part geometry. For example, fixturing design often requires analysis to determine the stiffness of a feature. A feature's stiffness, however, depends not only on its own geometry, but on the geometry and location of other features on the part; a thin rib will be flexible if unsupported, but not if it's connected to several bosses or stiffeners. Global feature interactions limit the ability to automate process planning because the calculations required quickly grow out of control **[Sarma & Wright 97]**.

SFF techniques, on the other hand, operate on simpler geometric elements, such as two dimensional layers. Additionally, SFF parts are built while encapsulated in a sacrificial support material. Usually the manufacturability of individual features depends only on local geometry and topology. As a result, the computational requirements for automated planning are reduced. While simpler and more mature SFF processes like stereolithography are mostly automated, planning software for newer techniques like SDM is still under development. If designers are forced to wait until all the automation issues are resolved, the widespread use of SFF techniques will be delayed by several years.

To date, most of the work on SFF processing uses a paradigm called "design decomposition," which assumes that designers will submit a geometric model of their part for automatic processing (e.g., layer slicing and machine tool path planning) by the manufacturer. The research presented in this dissertation is based on the hypothesis that an alternative approach, "design by composition," can make SFF more accessible to novice designers, and can increase the level of automation. Both of these effects should make it easier for designers to explore the new capabilities offered by SFF processes, and will therefore speed their adoption.

This chapter presents a comparison between the decomposition and composition paradigms as the motivation for the design by composition approach, and outlines the basic issues that must be addressed to build a practical design composition environment. These questions, and the contributions of this dissertation in answering them are presented in Section 1.2. Finally, the role of this work in a larger effort to develop a design-manufacturing interface for SFF is discussed.

1.1 Design Decomposition versus Design by Composition

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 automatically generates instructions for machine tools. An alternative approach is to have the designer "build" a part from primitives, or building blocks, which already have partial process plans.

1.1.1 Design Decomposition

The automated decomposition approach has two fundamental limitations:

- Because decomposition is based on geometry, the algorithm cannot tailor the decomposition for functional features (e.g., avoiding discontinuities on bearing surfaces). Adding functional requirements

to the algorithm would make the decomposition process even more complex and error prone.

- The decomposition calculations are 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. A better alternative would offer immediate manufacturability feedback during design.

In addition to these problems, the automated decomposition approach is hampered by limitations in the algorithms and software available today. Because many of the decisions which must be made during decomposition do not have unique solutions, the algorithms often produce decompositions that are inefficient or impossible to manufacture. These problems are especially severe in the compact-splitting algorithms for SDM, an SFF process which is described in Section 2.2.

1.1.2 Design by Composition

In this case, the designer sacrifices some design flexibility and assumes part of the computational burden of process planning. A simple example part and the primitives used to create it are shown in Figure 2. While this approach does restrict the designer's flexibility, it offers several benefits in return, as described in briefly in Section 1.2.3 and more fully in Section 6.3. In many ways, design by composition is similar to Constructive Solid Geometry (CSG) CAD systems **[Requicha & Voelcker 85]**.

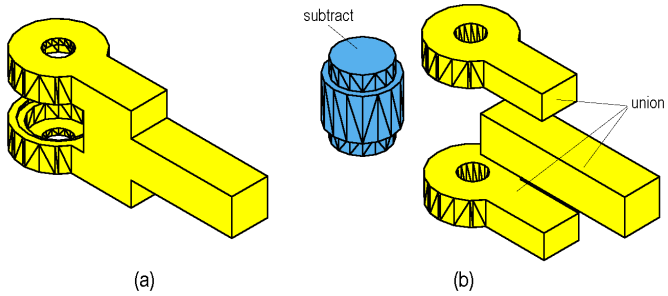


Figure 2. Design by composition example

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

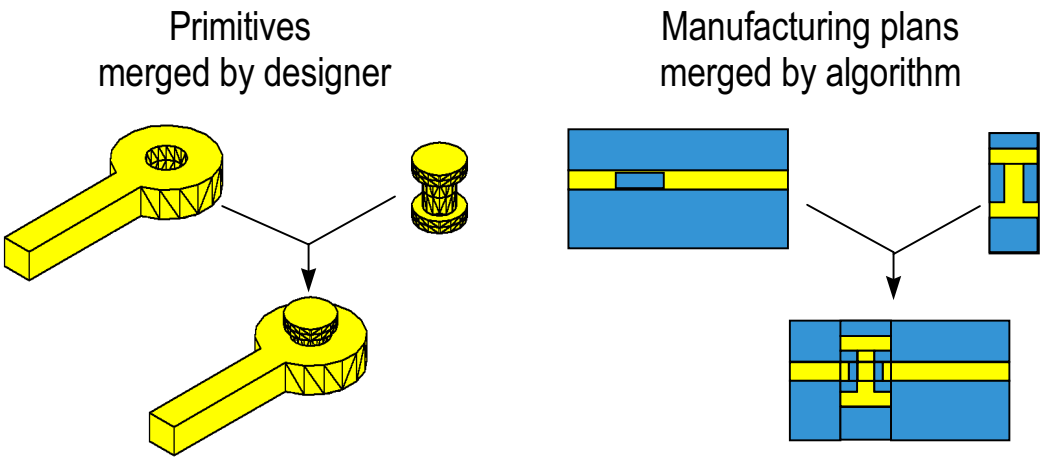


Figure 3. Primitives include high-level manufacturing plans

When the designer combines two primitives to form a new design, (left), the merging algorithms combine the two manufacturing plans to produce a manufacturing plan for the new design (right). The manufacturing plans are shown as a compact set, defined in Section 3.5.

When the designer combines two primitives to form a new design, the design by composition software merges the two process plans to produce a plan for the new design (Figure 3). In this way, the system maintains a manufacturing plan for the part throughout the design process. The benefits and disadvantages of including this information in the design are discussed in Chapter 6.

1.2 Questions to Answer

The research described here was based on the hypothesis that design by composition is an alternative to decomposition that is preferable in some circumstances. In particular, composition is a way for designers who are new to SFF processes to explore new possibilities and to design structures with embedded components.

The goals of this research were to determine the practicality of design by composition, to develop algorithms for its implementation, and to explore advantages and limitations of this approach. The three main questions I have addressed are:

- Are there robust, scalable, and practical algorithms which can be used to implement design by composition?
- How much manufacturing information should be included in the design?
- If the designer is willing to sacrifice some design flexibility, what tradeoffs can be achieved?

1.2.1 Algorithms

Implementing a design by composition system requires algorithms for the merging operation shown in Figure 3. To be practical, these algorithms must be robust – when provided with valid input, they must produce valid output – and they must scale acceptably as designs grow more complex. The algorithms I have developed meet these requirements, and are described in Chapters 4 and 5. A prototype design by composition system that partially implements the algorithms is presented in Chapter 6 and the Appendix.

1.2.2 Designs that include manufacturing information

The design by composition approach is fundamentally a decision to include high level representations of manufacturing plans in the design system. **[Voelcker 94]** points out that the trend in mechanical design standards is towards greater process-independence in design specifications. While the idea that designers should specify what they want, not how to achieve it, is intellectually elegant and attractive, it ignores the fact that unambiguously specifying every aspect of a design is prohibitive.

Including manufacturing information in a design tends to reduce the portability of the design between different manufacturing techniques. However, the level of process information included in the design by composition method is general enough to permit some process portability. For example, if a part was originally designed for the SDM process, it will still be manufacturable with SLA or other SFF techniques, assuming there is a common file format available. Moreover, because design by composition will be used primarily by designers who wish to explore or exploit the specific capabilities of SFF processes, limited process-portability is not a major disadvantage.

Although the cornerstone of the design by composition system, the concept of compacts, was invented for the Shape Deposition Manufacturing process, it does have some applicability to other SFF techniques. Nearly all layered techniques, including stereolithography and 3D Printing, use some form of temporary support structure for overhanging features. For stereolithography, 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. The support material compacts in design composition systems specify the location and geometry of support structures, and can be used in these computations. Although design by composition may not be as helpful for other SFF techniques as it is with SDM, it will offer some benefits. Other researchers in the Rapid Prototyping group at Stanford University have begun studying the design by

composition approach for the UC Berkeley CyberCut **[Smith & Wright 96]** machining process.

1.2.3 Design flexibility tradeoffs

Adopting the design by composition approach restricts the designer's flexibility in two major ways:

- the growth direction is fixed, and
- designs must be built out of primitives.

Design by composition requires choosing a growth direction[1] at the beginning of the design process. Therefore, users of design by composition forfeit the ability to choose an optimal growth direction when the complete part geometry is known. In practice, however, this sacrifice is not serious. Because the choice of growth axis affects feature quality and accuracy, it is logical to make the orientation decision based on functional requirements which are available at the beginning of the design process.

As mentioned above, in design composition, the user creates designs by combining pre-defined primitives with Boolean operations. Each primitive must have an associated manufacturing plan (in the form of a compact graph, defined in Sections 3.5 and 3.6), and therefore designers are restricted to designs they can build out of the available primitives. Furthermore, because there are restrictions on the valid transformations which can be applied to compact graphs, there are some geometric operations (such as rotation about the X or Y axes) which are prohibited.

For designers who accept these restrictions, design by composition offers several advantages:

- **manufacturability feedback during design-time:** The compact graph for a design can be checked against manufacturability rules, and can be used to generate a manufacturing preview of the design.
- **a greater degree of automation:** Currently, for SDM, design composition provides more highly automated process planning than the available decomposition systems. Automated process planning is vital to make SFF processes, SDM in particular, approachable by designers who are unfamiliar with them.
- **easy creation of designs with embedded components:** The composition approach is well suited to parts and mechanisms with embedded mechanical and electronic components. As these integrated designs mature, they may provide better performance and

[1] SFF processes normally build parts by depositing a series of layers on top of each other. The growth direction (or axis) is the direction in which part fabrication progresses.

reliability for mechatronic devices, by encapsulating delicate components and reducing the need for connectors and fasteners.

- **easy creation of design libraries:** Any design can be stored in a library and used as a primitive element in future designs.
- **enhanced control of the manufacturing plan, and thereby of part quality:** Including a high-level process plan in the design system will allow experienced designers to examine, edit, and optimize the manufacturing plan, giving them greater control over the accuracy, surface finish, and strength of part features.
- **fewer computational bottlenecks:** design composition reduces computational bottlenecks by distributing process planning tasks over a number of designers.

These advantages are described in greater detail in Chapter 6.

1.3 A Design-Manufacturing Interface

This project is part of a larger effort to develop a "clean" interface between design and manufacturing for rapid prototyping (SFF) processes **[Rajagopalan, et al. 98]**, with the ultimate goal of making novel fabrication processes accessible to a wide community of designers. 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 hope is that this interface specification – and the corresponding infrastructure – will encourage exploration and exploitation of SFF capabilities. To this end, it is imperative that the Interface allows both designers and manufacturers to use representations that match their requirements, and does not require them to have detailed knowledge of each others' capabilities and intent.

The solution is to use an intermediate representation that permits

- manufacturability analysis on the design side,
- some design portability between manufacturing processes, and
- automated translation between design, intermediate, and manufacturing representations.

The representation selected as the basis for the Interface, and the design by composition system, is to include a high-level process plan, in the form of a compact graph (defined in Section 3.7.1) in the design.

2. Design and Planning for Rapid Prototyping

Shape Deposition Manufacturing, or SDM, is an SFF process that combines several different deposition, shaping, and material processing steps **[Merz, et al. 94]**. During fabrication, parts are embedded in a sacrificial support material which eliminates the need for specialized clamps and fixtures. The process builds both the part and support structure in a sequence of layers.

SDM offers several benefits and opportunities to designers. However, before SDM is widely accepted, there are several major obstacles that must be overcome. First is the experimental nature of the processes – there are still significant material quality, automation, and tolerance issues being studied. Secondly, there is a lack of knowledge about SDM in the design community. Research on process quality and automation is ongoing both in academia and industry **[Marsan & Dutta 97]**. The challenge of educating designers about these techniques, however, is not being sufficiently addressed. Before SDM and other processes change the nature of electromechanical design, designers must learn about and experiment with the new capabilities.

This chapter describes Stanford University's Shape Deposition Manufacturing process, the available design and planning alternatives, and the current degree of automation.

2.1 Shape Deposition Manufacturing

The basic cycle for SDM is shown in Figure 4. First, part or support material is deposited using casting, micro-casting, laser welding, or other deposition processes. The existing support material is used as a mold to shape the underside of the deposited material. Next, CNC milling, EDM machining, or other techniques shape the material to obtain the desired geometry, accuracy, and surface finish. Prefabricated components can be embedded after any shaping step. Additional processing steps, such as washing, stress relief, or surface preparation may also be inserted at any point in the SDM cycle.

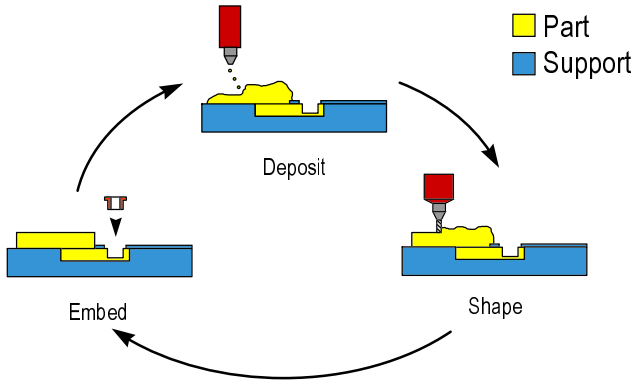


Figure 4. The Shape Deposition Manufacturing (SDM) process cycle.

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.

Most SFF techniques, including stereolithography, laser sintering, and 3D Printing use a single process to deposit and shape material. Because SDM employs separate processes for deposition and shaping, a deposition process can be selected based primarily on material quality issues, not geometric accuracy. Increased freedom in choosing a deposition process translates into a wider variety of materials choices – SDM has been used to make parts out of urethane, epoxy, stainless steel, Invar, copper, and gel-cast ceramics. Some parts have been fabricated with multiple materials **[Fessler, et al. 97]**.

With most traditional manufacturing processes, like machining and forging, the fabrication process only has access to the exterior surfaces of a part. SFF techniques, including SDM, on the other hand, provide access at some time in the fabrication process to every point in the part's interior. Without the tool access limitations of traditional processes, SFF can fabricate parts with complex internal geometry.

2.1.1 Multi-material and graded material parts

An interesting capability of some SFF processes is the fabrication of multi-material parts **[Kumar, et al. 98]**. With SDM, this capability is extended to producing parts with material composition and properties that vary with position. For example, SDM has been used to make injection molding tools (See Figure 5) where the bulk of the material is INVAR (to minimize thermal distortion) and the mold surface is stainless steel (for wear resistance). The material composition changes linearly from 100% INVAR in the mold's interior to 100% stainless steel on the outer surface **[Fessler, et al. 97]**.

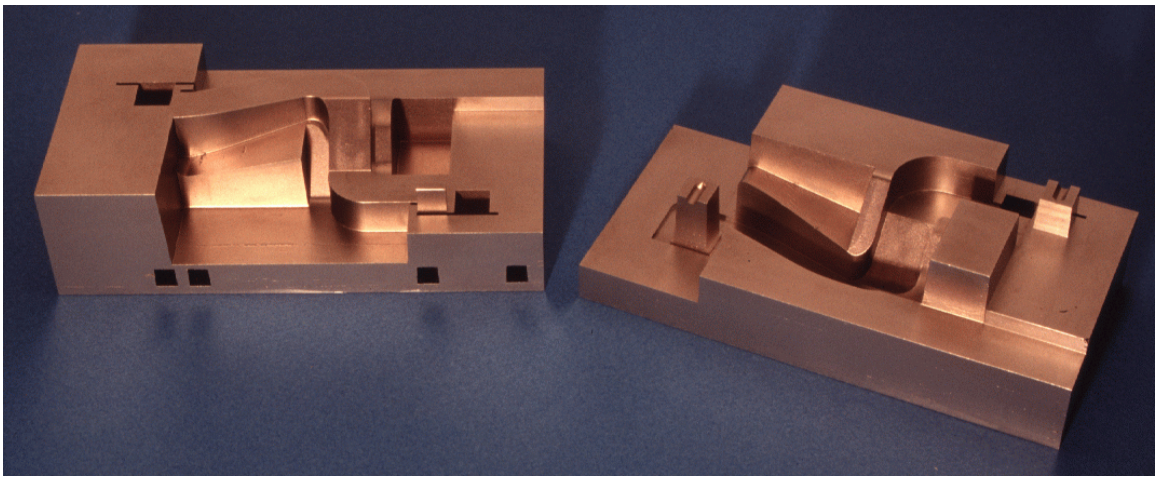


Figure 5. SDM injection molding tools

These parts have an INVAR core for low thermal distortion, a stainless steel shell for wear resistance, and internal copper deposits for heat conduction. (from **[Kao, et al. 98]**)

2.1.2 Embedded components

Designing robots and other state-of-the art electromechanical devices is difficult. In addition to dealing with complexity, designers are often challenged to keep size, weight, cost, and power consumption to a minimum. In many cases, the performance and reliability of a design is governed by assembly constraints and the connections between components. The most common failure mode in mobile robots is usually a loss of electrical contact at connectors. Especially with small machines, the size and weight of components is often determined by the need to provide sufficient material for screw holes and fasteners. Off-the-shelf sensors and actuators have excessively large housings and connections.

SDM has the capability to produce parts with embedded electronic and mechanical components **[Weiss, et al. 96a] [Weiss, et al. 96b]**. This ability may free robot designers from many of the reliability and performance limitations of traditional fabrication using off-the-shelf components.

2.1.3 Pre-Assembled mechanisms

An especially interesting capability of SDM is the fabrication of pre-assembled mechanisms, as shown in Figure 6. 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 (Figure 7). 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 (the surface finish on downward facing surfaces of metal SDM parts is quite poor.)

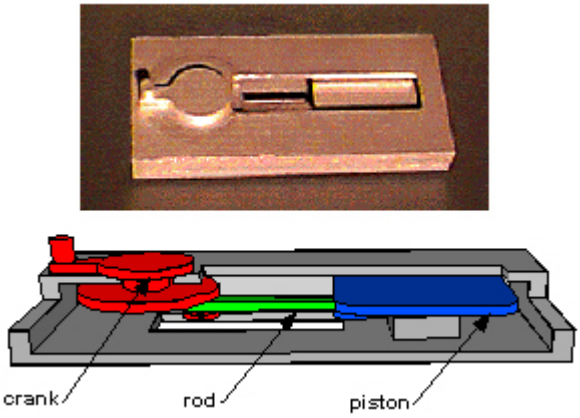


Figure 6. "The Cranky" pre-assembled mechanism

The Cranky is a simple slider-crank mechanism that was built at the Stanford SDM facility. The four separate parts were built in their assembled configuration, resulting in a functional mechanism that is free of assembly constraints (and cannot be disassembled).

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 easily be 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.

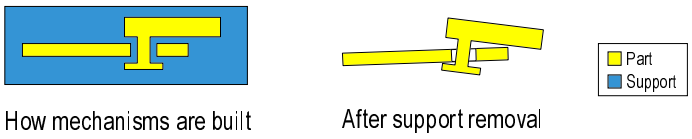


Figure 7. Current SDM Mechanisms.

In the current method for fabricating pre-assembled mechanisms with SDM, moving parts are separated by a thin layer of support material. The finite thickness of this material causes play in the mechanism's bearings.

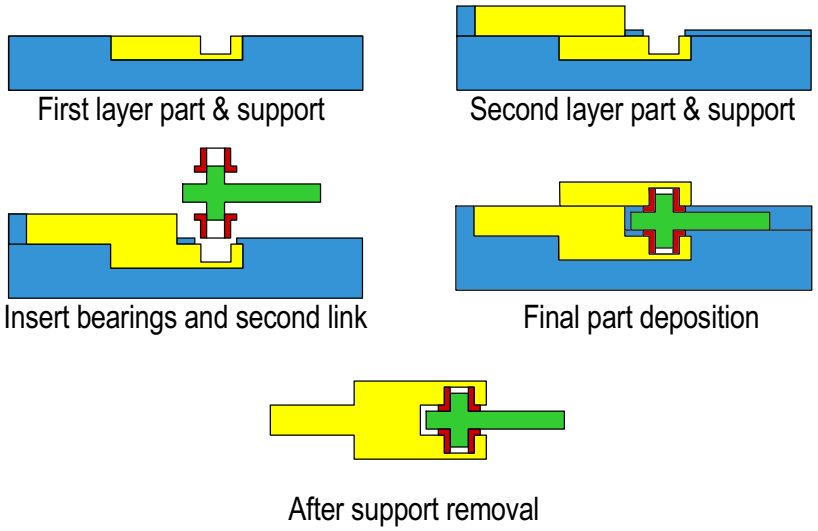


Figure 8. Building a pre-assembled mechanism with embedded components

Precision bearings and pre-fabricated parts can be inserted during the SDM fabrication process. This technique allows tighter fits and wider material choices.

2.2 Process planning for SDM

Process planning for SDM and other SFF processes consists of three steps: part orientation, compact-splitting, and path planning **[Kumar, et al. 98]**. (See Figure 9)

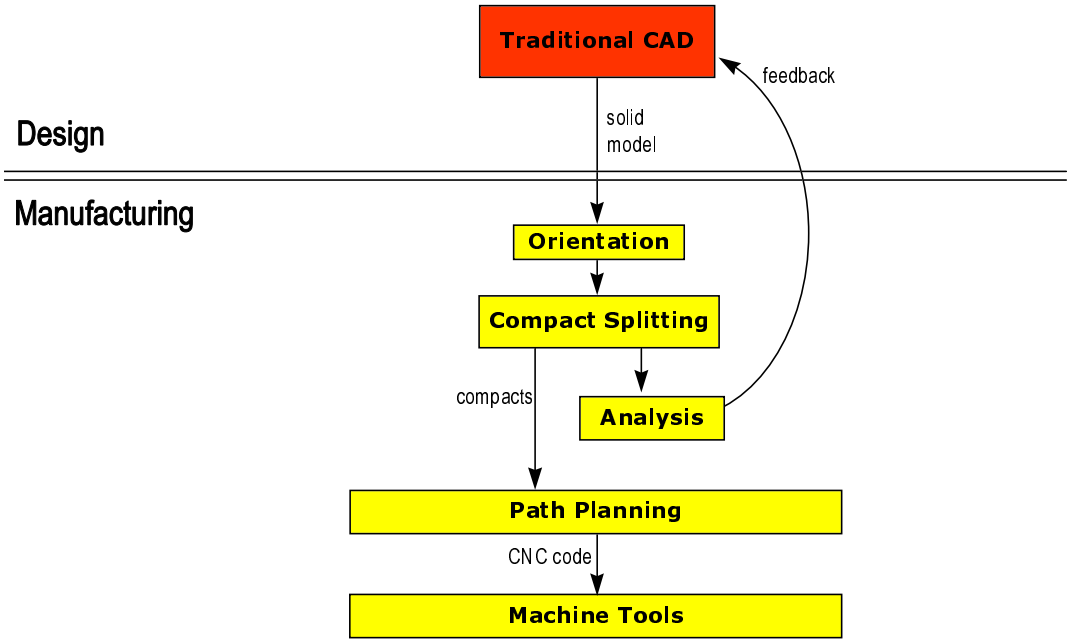


Figure 9. Decomposition design-manufacturing interface

There are two problems with this interface for SDM: the software for compact splitting is not complete, and the feedback process is too slow.

Orientation is the selection of the growth direction that will be used for fabrication. This decision can affect the amount of support structure required, manufacturing time and expense, feature accuracy, and surface quality. Researchers have investigated part orientation issues for stereolithography (SLA) and other layered manufacturing techniques **[Allen & Dutta 95] [Alexander, et al. 98]**. These systems are capable of choosing a fabrication orientation that minimizes the amount of support structure required, a reasonable criterion for processes like SLA where support structures must be removed manually. In general, automated orientation algorithms do not consider tolerances and critical features because this information is only available to designers.

The second step in SFF process planning is to decompose the design into shapes that correspond to individual manufacturing operations. For purely layered processes like SLA, this decomposition step consists of cutting the part design into two-dimensional slices. For SDM, the primary decomposition step is to break the part geometry into compacts (Figure 10), which are volumes of material that can be deposited and shaped in one process cycle (Compacts are described more fully in Section 3.4.) Because compacts are three dimensional shapes, compact splitting operations are more complex than layer slicing. The compact splitting algorithm used for automated SDM processing, developed by Krishnan Ramaswami, is presented in **[Ramswami 97]** and **[Ramaswami, et al. 97]**.

The final step in SFF process planning is generating machine tool deposition or cutting paths based on the compact or slice geometry. Although this procedure may seem straightforward for processes that use 2D layers, optimal path generation is difficult **[Kao & Prinz 98]**. For SDM, path planning requires breaking compacts into slices and producing CNC cutting paths for 3- and 5-axis machining (Figure 10). Although this problem is non-trivial, it is relatively well understood in industry.

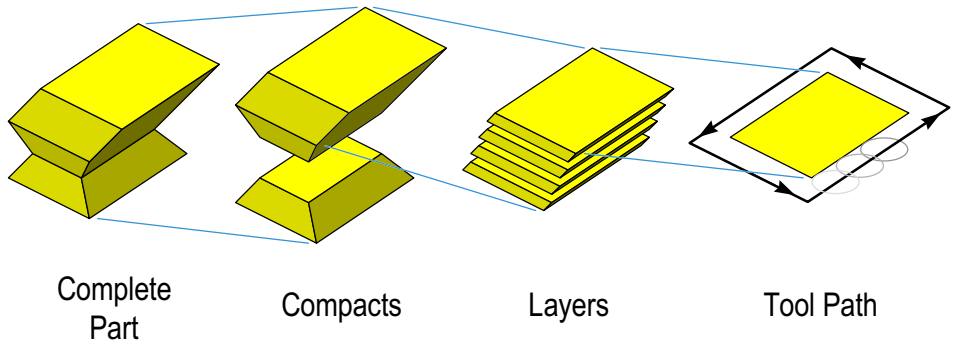


Figure 10. Levels of decomposition in SDM

First, the part is split into compacts, which are volumes of material that can be deposited and shaped in one manufacturing cycle. Then, compacts are sliced into layers to generate CNC tool paths.

The most interesting part of the decomposition process is compact splitting. The compacts represent a high-level plan for the fabrication of the part; they specify the output of each SDM cycle in the manufacturing process. Because manufacturability rules for SDM are specified as restrictions on compact geometry, it is possible to provide feedback and a "manufacturing preview" to the designer once the compacts are known. Furthermore, splitting a complete part into compacts is the most difficult part of process planning – compact splitting algorithms are the subject of ongoing research **[Ramaswami, et al. 97]**.

Currently, at Stanford, there are three design-manufacturing options available for SDM:

1. Design decomposition using AutoCAD and ACIS-based planning software written by graduate students
2. Design and process planning with the Unigraphics CAD package
3. Design by composition using the AutoCAD plug-in and ACIS-based planning software.

2.2.1 ACIS decomposition

The oldest software package for SDM planning uses the ACIS solid modeling kernel **[ACIS 95]**, and is described in **[Ramaswami & Prinz 95]** and **[Ramaswami, et al. 97]**. Geometry for the ACIS decomposition planner can be created with ACIS itself, with AutoCAD, or with other CAD programs that can output .SAT files. While this planner was used for several years to build parts with SDM, it has several limitations:

- The algorithm for compact-splitting does not always produce practical decompositions, as described below.
- Because of limitations in the geometry engine provided by ACIS, this planner only functions reliably with parts that have only horizontal and vertical surfaces (also called 2½D parts).
- The planner requires ASCII text files which contain parameters for the decomposition process, including minimum and maximum layer thickness, cutting tool parameters, and support structure size. These files are generated manually by the designers.
- Often, to maintain control of the manufacturing process, or to fix problems in the CNC files, designers are required to make manual modifications to the G-code files produced by the planner.
- The only feedback available to designers is a utility that shows the 2D path of the CNC cutting tools – software to view the 3D shape of compacts is not provided.

Because of these limitations, this software package is no longer used for SDM.

Figure 11 shows the compact decomposition produced by Ramaswami's algorithm, which works by identifying silhouette edges and sweeping them in the growth direction. This technique often produces manufacturing plans that require machining of sharp pockets, as shown in Figure 12. Because of this limitation, very few parts have ever been built using the automated decomposition system for SDM – instead, the compact-splitting operations are performed manually.

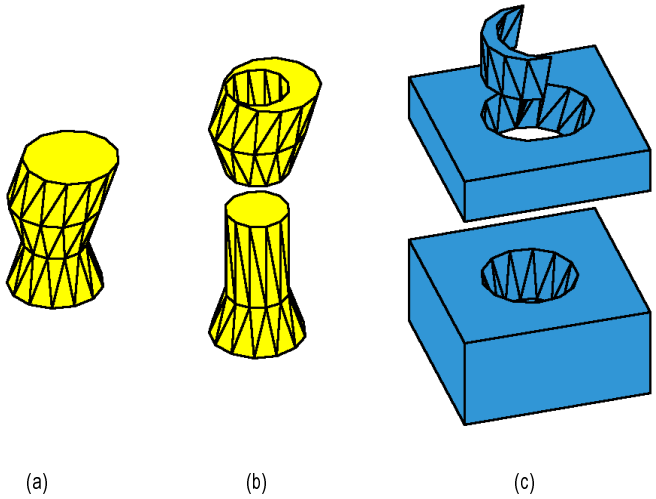


Figure 11. Decomposition example

The complete part geometry is shown in (a). The part and support material compacts created by the compact splitting algorithm in **[Ramswami 97]** are shown in (b) and (c), respectively. Note that some of the compacts have sharp features on the bottom, which are difficult to manufacture with CNC cutting.

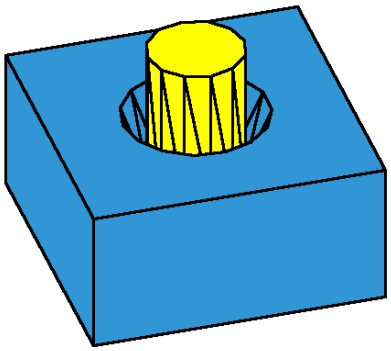


Figure 12. A problem with automated compact-splitting

This figure shows the first part and support material compacts from the example in Figure 11. The sharp, conical trench around the part material is difficult to create with CNC milling; as a result, this part will be expensive or impossible to produce without manual intervention in the decomposition process.

2.2.2 Unigraphics decomposition

A newer decomposition system is currently being developed at Stanford University's SDM facility. This package is based on the Unigraphics CAD environment. The goal of the Stanford researchers is to offer automated decomposition of 3D parts into compacts, and automatic generation of 3D CNC cutting and deposition files.

Today, the Unigraphics design and planning system is the only option that allows true 3D parts. The process planning operations are still mostly manual, however. Automation of the planning for 2½D cutting is nearly complete, and work is progressing on 3D cutting. Software for automated compact-splitting has been implemented for simple geometry.

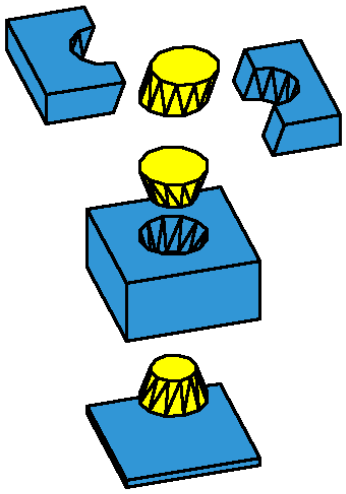


Figure 13. Design by composition compacts for Figure 11(a)

These are the part and support material compacts created by the design by composition system for the part in Figure 11(a). The manufacturing plan shown here does not contain any problematic features like those in Figure 12.

2.2.3 Design by Composition

The current design by composition plug-in (described in Chapter 6 and the Appendix) lets designers work in the standard AutoCAD environment. Primitives are manipulated by familiar AutoCAD commands. The plug-in provides new commands for creating cylindrical and rectangular (box) primitives and combining them with Boolean operations. The user interface for these commands is similar to that for normal

AutoCAD solids. Figure 13 shows the compacts produced by the design by composition system for the part in Figure 11(a).

After a design is complete, the design composition plug-in will produce all the geometry and planner instruction files required for fabrication. These files must be transmitted (via FTP) to the manufacturing site and run through the CNC path planning software. The resulting CNC files can then be used to manufacture the part. As shown in Figure 14, the prototype Design by Composition system described in Chapter 7 and the Appendix offers a greater degree of automation than the other process-planning options.

Planning Operation	ACIS Decomposition	Unigraphics Decomposition	Design by Composition prototype
Part orientation	manual	manual	chosen at start of design
Generating compact geometry	automated algorithm exists, but usually performed manually	manual	automatic
Creating planner instruction files	manual	N/A	automatic
CNC path planning	automatic	manual with Unigraphics interface	automatic

Figure 14. Degree of automation in Process Planning

This table shows the degree of automation currently available for each of the three SDM process-planning options.

2.3 Manufacturing

Although one of the main motivations for developing SFF techniques, including SDM, is the potential for complete automation, today SDM is still a combination of manual and automated processing.

Deposition processes

At Stanford, metals are deposited by an automated system that uses a high power laser to melt metallic powders. Process planning for metal deposition is mostly automated, although some work is still continuing **[Kao & Prinz 98]**. Polymers are deposited manually, as shown in Figure 15. Wax support material is usually deposited manually, but can be deposited automatically by the modified CNC milling machine in

Figure 16. Ceramic parts are made by a process called "Mold SDM" which uses soldermask as a temporary part material to build a wax mold **[Cooper, et al. 1998]**.

Figure 15. Polymer deposition

The deposition of polymers is still often a manual process for SDM. This picture shows urethane being poured onto four parts-in-progress. A duct-tape dam prevents the urethane from spilling.

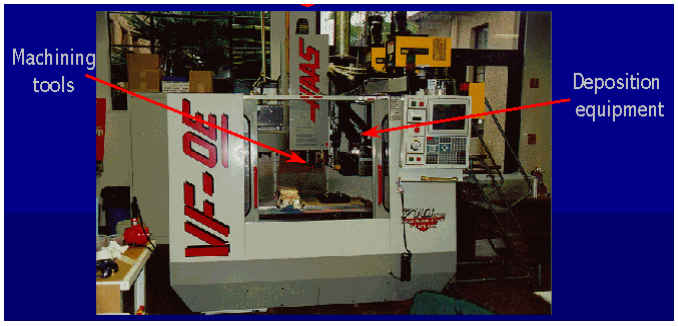


Figure 16. The SDM milling machine

SDM parts are fabricated on this milling machine, which has been retrofitted with equipment for depositing wax and soldermask.

Shaping

Shaping operations are performed by either CNC milling or Electric Discharge Machining (EDM). The Stanford SDM facility has two CNC mills (one is shown in Figure

16) and one EDM machine. All shaping operations are controlled automatically. Process planning for these processes is primarily a matter of generating CNC code based on compact geometry. For 2½D parts, this procedure is relatively automated. For 3D parts, CNC code is generated under manual control using the Unigraphics CAD environment. Automated path planning software for 3D cutting is a current research topic.

Figure 17. The SDM milling machine in operation

This photograph shows two parts being built in a support structure of blue wax.

3. Definitions for Design by Composition

The design by composition approach is based on two geometric elements: primitives and compacts. Primitives are the three dimensional shapes that designers use to create part and mechanism designs. Compacts represent the volumes of material that will be deposited in each step of the SFF fabrication process. This chapter contains detailed definitions of these and other terms used in this dissertation.

3.1 Growth direction & Datum plane

SFF processes generally build parts in layers 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.

The SFF workspace is a Cartesian coordinate space, W, in R^3 with the X-Y plane on the datum surface, and the Z axis pointing in the growth direction. (see Figure 18) Points in W are referred to as (x,y,z). Several of the proofs in the following chapters will use (x,y,z_i) and (x,y,z_j), or (x,y,z) and (x,y,z'), to refer to points with the same xy, but different z coordinates.

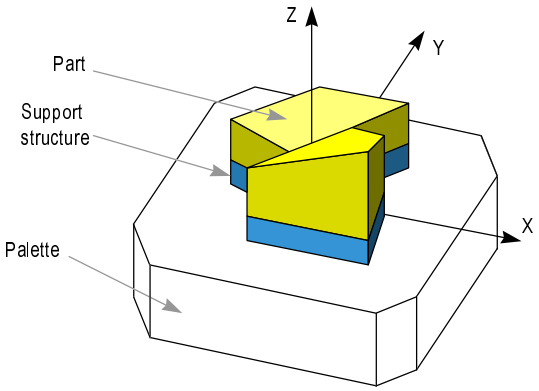


Figure 18. The SFF workspace

The SFF workspace is represented by a Cartesian coordinate frame with the origin located on the top surface of the palette, and the Z axis pointing in the growth direction.

3.2 Projected volume

The term "projected volume" refers to the space defined by projecting a primitive in the growth direction to the top and bottom of the manufacturing workspace, as shown in Figure 19. Mathematically, the projected volume of a primitive is defined as

$$p(\mathbf{A}) = \left\{ (x,y,z) \middle| \exists (x,y,z') \in \mathbf{A} \right\} \cap W .$$

The projected volume is the intersection of the SDM workspace with the set of points (x,y,z) that share the same x and y coordinates with any point (x,y,z') in the primitive. Equivalently,

$$\forall x \forall y \forall z \forall z' \left[(x,y,z) \in \mathbf{A} \wedge (x,y,z') \in W \rightarrow (x,y,z') \in p(\mathbf{A}) \right].$$

In other words, for any point (x,y,z) in the primitive $\mathbf{A}$, all points in W with the same xy coordinates are in $p(\mathbf{A})$.

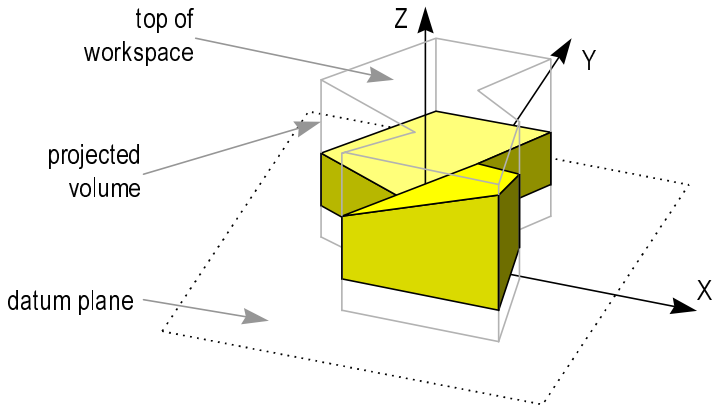


Figure 19. Projected volume of a primitive

The dotted lines are the limits of the SDM workspace. The gray lines define the projected volume of the primitive.

3.3 Simple shapes

A "simple shape" is a solid that can be formed by extruding a closed two-dimensional curve in the growth direction. These shapes have top and bottom surfaces that are parallel to the datum plane, and a number of side faces parallel to the growth direction. Rectangular prisms and cylinders (when aligned with the growth axis) are two examples of simple shapes (see Figure 20).

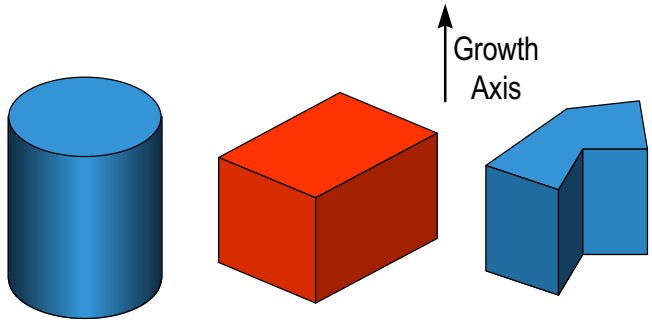


Figure 20. Simple shapes

A simple shape is solid formed by extruding a two dimensional, closed curve in the growth direction.

3.4 Compact

A compact is a three dimensional volume that represents one deposition and shaping cycle **[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 compact consists of material type and desired geometry.

3.4.1 Material

Currently, the compacts in the design by composition CAD system are either part material, support material, or embedded components. 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) **[Kumar, et al. 98]**.

3.4.2 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; these volumes can be grouped as a single compact.

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 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. Mathematically, a given shape c is a compact if

$$\forall x \forall y \left[\sim \exists z_1 \exists z_2 \exists z_3 \left(\begin{array}{l} \left((x,y,z_1) \in c \right) \wedge \left((x,y,z_2) \notin c \right) \wedge \\ \left((x,y,z_3) \in c \right) \wedge \left(z_1 < z_2 < z_3 \right) \end{array} \right) \right]$$

or, equivalently, if

$$\forall x \forall y \forall z \exists z_1 \exists z_2 \left[(x,y,z) \in c \leftrightarrow z_1 \le z \le z_2 \right]. \tag{1}$$

In other words, if two points in the compact have the same xy position, but different z coordinates, all points on the line between them are also in the compact. Equation 1 states that for any (x,y) position, all the points (x,y,z) that are within the compact have z coordinates in the range $z_1 \le z \le z_2$.

The mathematical test shown in Equation 2 can also be used to determine if a three dimensional solid is a valid compact. 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 upward-facing surface (with respect to the growth axis, Z):

$$\sim \exists p \left\{ p \in \mathbf{S} \wedge \left. \frac{\partial S}{\partial z} \right|_p = 0 \wedge \left. \frac{\partial^2 S}{\partial z^2} \right|_p < 0 \right\}. \tag{2}$$

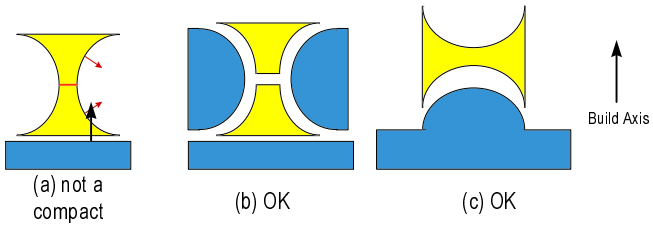


Figure 21. Testing for a legal compact

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 21 illustrates a simple test for compactness which is equivalent to Equation 1. 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]**.

3.4.3 Valid operations

There are several geometric operations which are closed with respect to the set of valid compacts. These operations, when applied to a valid compact will always result in a shape that is also a compact, and include

- uniform and non-uniform (different X, Y, and Z scale factors) scaling applied to the complete compact geometry,
- rotation about the Z-axis,
- mirroring through a vertical plane, and
- translation.

In general, any operation which does not change the sign of the Z-component of any surface normal vectors will be closed on the set of valid compacts.

3.5 Compact sets

In the design by composition paradigm, each primitive includes a compact set (Figure 22) which captures the basic plan for its fabrication. Each compact in the set captures the geometry and material type of a single step in the manufacturing process. The set itself has no order information. In the prototype implementation described in Chapter 6 and the Appendix, the compacts are created on hidden layers in an AutoCAD model, so the designer is free to ignore the details of the compacts.

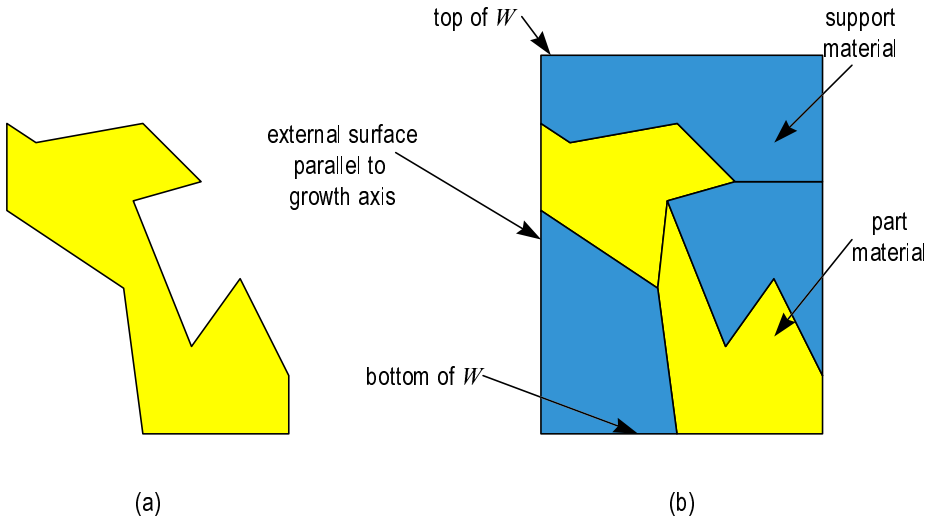


Figure 22. A primitive and its compact set

The compact set (b) for the primitive (a) contains two part material compacts and three support material compacts. The external surfaces of the compact set are either the top or bottom of the workspace, W, or parallel to the growth axis.

3.5.1 Rules for valid compact sets

Correct operation of the compact merging algorithm described in Section 4 requires that the compact sets obey these rules:

1. Each member of the compact set must be a valid compact.
2. Compacts in the set must completely fill the projected volume of the primitive.
3. There must be no intersections between any compacts in the set.

Rule 1. This rule can be checked by ensuring that all the compacts satisfy the tests described above.

Rule 2. Every point in the projected area of **A**, $p(\mathbf{A})$, must belong to a compact in A:

$$\forall p \exists a \left[p \in p(\mathbf{A}) \rightarrow p \in a \wedge a \in A \right].$$

Note that the volume defined by a compact set may be a superset of the projected volume for the primitive; there is nothing wrong with a compact set that includes extra support material, as long as the volume of the compact set exactly fills the projected volume of some shape. For example, the compact set shown in Figure 13 extends beyond the projected volume of the primitive to fill the projected volume of a square. In other words, the external surfaces of a compact set must be

- a subset of the top of the workspace, W,
- a subset of the bottom of W, or
- parallel to the growth direction.

Rule 3. For a given compact set, A, there must be no two compacts in A that intersect with each other:

$$\forall p \forall a_1 \forall a_2 \left[a_1 \in A \wedge a_2 \in A \wedge p \in a_1 \rightarrow p \notin a_2 \right].$$

Rules 2 and 3 can be combined in this statement:

$$\left(x, y, z\right) \in \mathbf{A} \rightarrow \forall z' \exists a \left[(x, y, z') \in W \leftrightarrow \left(a \in A \wedge (x, y, z') \in a \right) \right]. \tag{3}$$

In other words, for any point (x,y,z) in the primitive **A**, all points (x,y,z') that are within the SDM workspace belong to one of the compacts, $a \in A$.

3.5.2 Simple shapes

The compact sets for simple shapes can be generated algorithmically (see Figure 23) by extruding the closed curve along the growth axis to the top and bottom of the SDM workspace. The resulting solid is then subdivided on the horizontal planes that contain the top and bottom surfaces of the primitive. The top and bottom compacts, which will not exist if the primitive reaches the top or bottom of the workspace, represent the support material which surrounds the shape. The middle compact is the part material. Although the support material on top of the part is not required for fabrication, it is required for the merging algorithm (Chapter 4) to function properly.

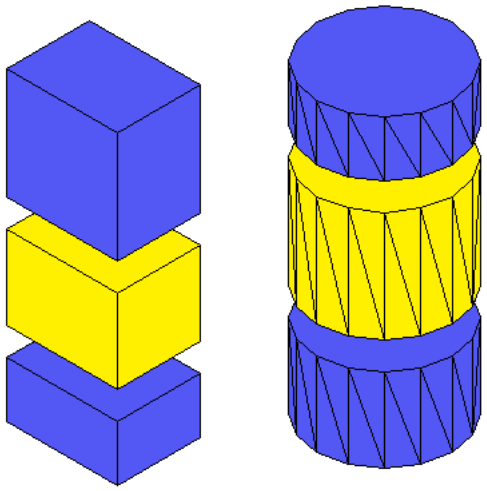


Figure 23. Compact sets for simple shapes

The compact sets for simple shapes consist of three compacts: the part material with support above and below. In this picture the compacts have been separated slightly for clarity. The support material compacts extend to the top and bottom of the SDM workspace. Either, or both, of the support material compacts may not exist if the part material reaches the extents of the workspace.

3.5.3 Storage in a design library

Compact sets for more complex primitives 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.

3.6 Compact precedence graph - CPG

The compact set for a primitive represents the material type and geometry of each step in the fabrication process. The possible orderings of these steps are represented by a directed graph, called a Compact Precedence Graph or CPG **[Pinilla, et al. 97]**. A link in the CPG from compact a to compact b, written $a \rightarrow b$, indicates that a must be fabricated before b.

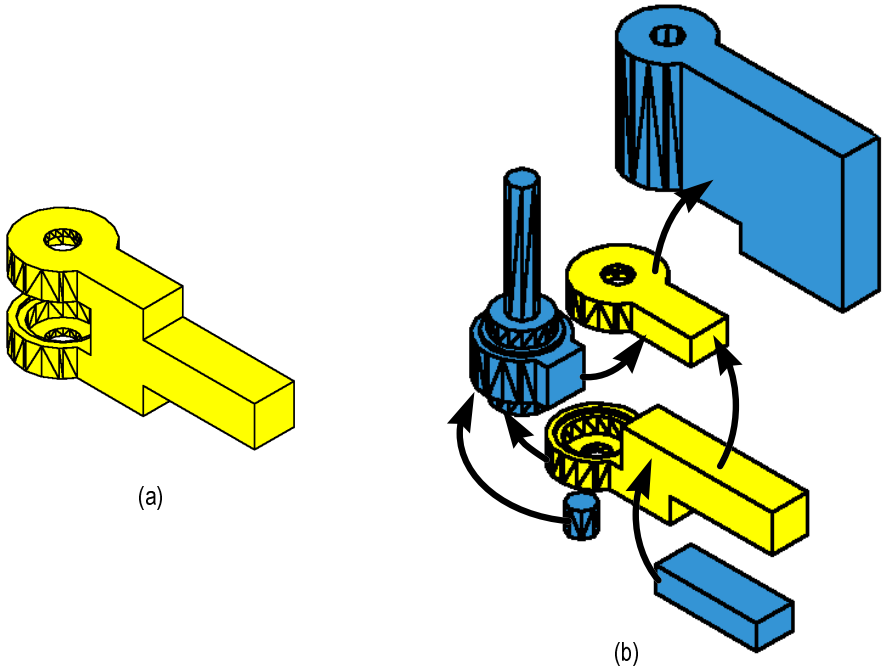


Figure 24. A primitive (a) and its compact precedence graph (b)

3.6.1 Rules for a valid CPG

The requirements for a valid CPG are:

- The nodes of the CPG must be the compacts of a valid compact set (see Section 3.5.1).
- There must be a link in the CPG for every shared surface that is not parallel to the growth axis.
- Links must be directed from the lower compact to the upper compact.
- The CPG must be acyclic.

The second and third conditions ensure that a compact is preceded by every compact which touches one of its downward facing surfaces. Because these surfaces

are formed by casting, the adjacent compacts must be built first so they can act as a mold for the new compact.

The fourth condition ensures that there is a valid "path" through the graph which corresponds to a manufacturing plan. If the CPG were cyclic, then it would be impossible to build any of the compacts in the cycle.

3.6.2 Simple shapes

The CPG for a simple shape can be generated automatically along with the compact set. As mentioned above, the compact set for a simple shape consists of three compacts. The CPG simply expresses that the compacts must be built in order from the bottom up. Figure 25 shows two examples of the CPGs for simple shapes.

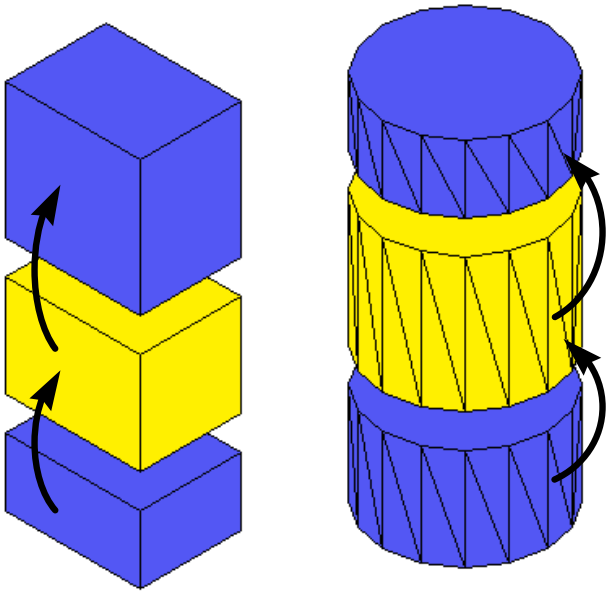


Figure 25. CPGs for simple shapes

The arrows indicate the precedence links between the compacts. For simple shapes, the CPG indicates that the compacts must be built from bottom to top.

3.7 Primitives (aka. Library elements)

Design by composition 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. Each primitive is a 3D solid, which can have arbitrary geometry, and has

an associated compact set and precedence graph that specify the manufacturing plan for the primitive. (In the implementation described in Chapter 6 and the Appendix, the compact graph is stored on hidden layers of the CAD drawing.)

3.7.1 The Compact graph

"Compact graph" refers to the compact set for a primitive and its associated CPG. It is important to realize that the compact graph of a primitive represents a high-level process plan for its fabrication; the compact set specifies the material type and geometry to be created in each manufacturing step, and the CPG specifies the possible sequences of manufacturing operations. The library elements do not include detailed manufacturing data like CNC paths or deposition parameters. The purpose of the merging algorithms described in Chapters 4 and 5 is to combine the compact graphs from two primitives into a new graph that represents a feasible process plan for the new part design

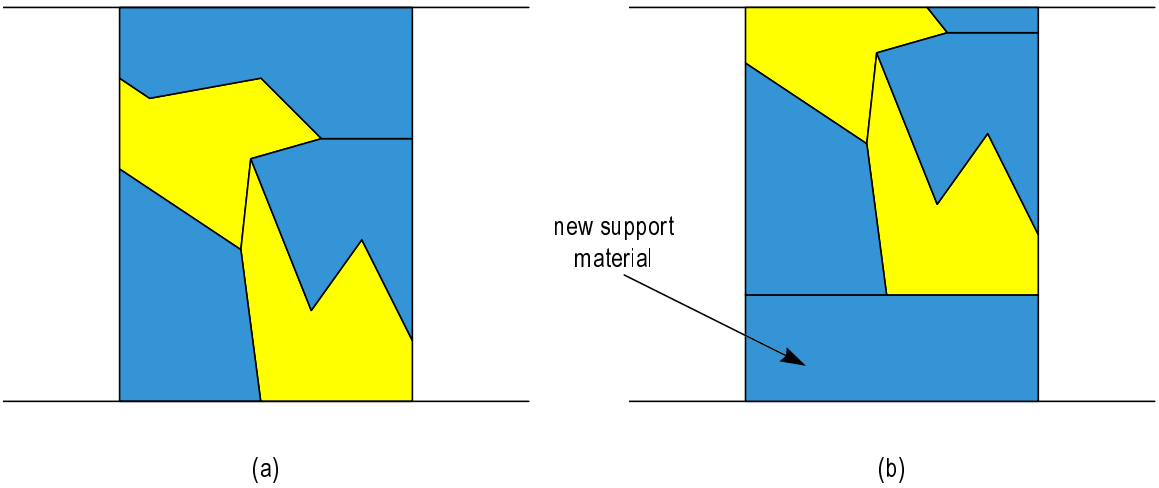


Figure 26. Z-translation of a compact set

(a) shows the original compact set. In (b), the compacts have been translated in the growth direction. The compact set is truncated at the top of the SFF workspace, and new support material is added underneath the compacts.

3.7.2 Valid operations

Any geometric operation applied to a primitive must also be applied to its compact set. Therefore, the permitted operations for primitives are the same as for compacts. Translation perpendicular to the growth direction, rotation about an axis parallel to the growth direction, mirroring through a vertical plane, and horizontal scaling are

straightforward. Translation and scaling in the vertical direction require additional operations to maintain the compact set's validity. Scaling operations that increase the size of the compacts in the Z direction and vertical translations will produce compacts that extend past the top or bottom boundaries of the SFF workspace. These compacts must be truncated at the workspace boundary. Scaling operations that reduce the vertical size and vertical translations will also leave a portion of the primitive's projected area unfilled. New support material compacts must be created to fill this volume. The geometry of these compacts is defined by extruding the projection of the compact set in the XY plane. Figure 26 shows an example of the modifications to a compact set caused by a vertical translation.

4. The Compact Merging Algorithm

As discussed in Chapter 3, each primitive in the design system has an associated compact graph that represents a high-level plan for its manufacture. In the design by composition paradigm, the user designs parts (and pre-assembled mechanisms) by combining these primitives with Boolean operations: addition (union), subtraction, intersection, and exclusive-or (XOR). When the user combines two primitives, The CAD system calculates a compact graph for the new design by combining the compact graphs of the two source primitives. In other words, the software automatically creates a manufacturing plan for the new design by combining the plans for the source primitives.

There are two central innovations that make the design by composition system possible. First, the algorithm presented in this chapter, which determines the geometry and material type for the compacts in the result compact graph. This algorithm produces a relatively large number of compacts which are the nodes of the new compact graph. The second innovation is the set of algorithms described in Chapter 5 that determines the precedence links between the graph's nodes, and performs some graph simplification.

4.1 How it works

Note: In the following discussion, the two source compact sets are always A and B, and the result set is C. The number of compacts in A is n, and the number of compacts in B is m. This section assumes that subtraction operations are always performing **A-B** (and not **B-A**).

In the example shown in Figure 27, the user is performing a Boolean addition operation between two primitives, **A** and **B**. The compact sets A and B are shown in the lower half of Figure 27 with the result compact set produced by the merging algorithm. Compacts in the result set are classified in two types: intersection compacts and non-intersecting compacts (see Figure 28). The merging algorithm calculates material type and geometry for each compact, using different rules for the two compact types. (The amount of overlap between the two primitives is determined by their relative position in the CAD model.)

A mathematical description of the algorithm is shown in Figure 29. The first part of the algorithm creates all of the intersection compacts, and sets their material type. Then, the volume of the intersecting compacts is subtracted from the two source compact sets. When this part of the algorithm is complete, the compact set C contains all of the intersection compacts, while A and B contain the non-intersecting compacts. The second part of the computation copies the non-intersecting compacts into the result set as needed.

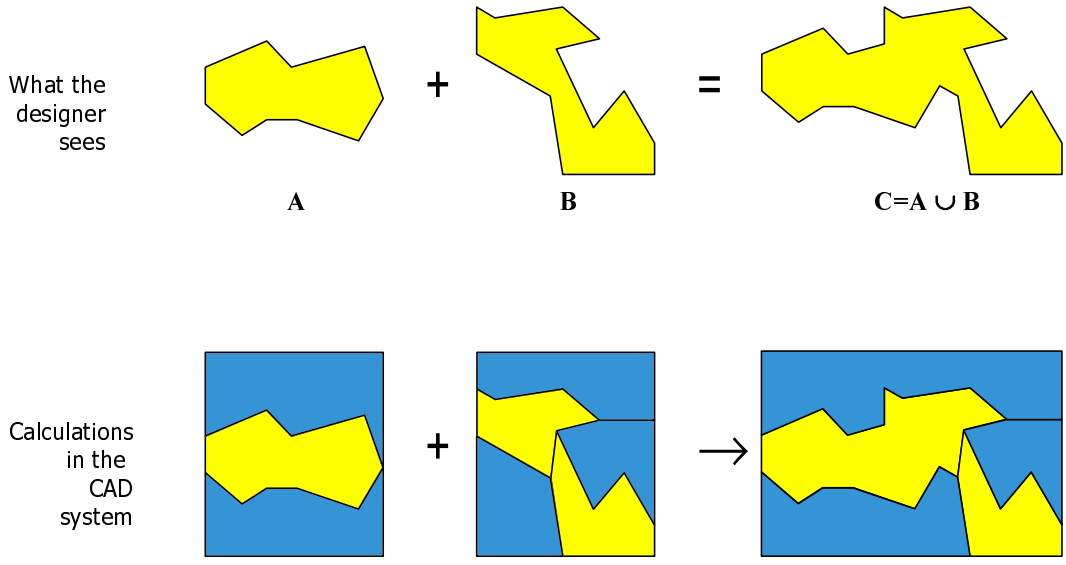


Figure 27. An example merging operation

The top half shows the two source primitives and the result of a union operation. The lower half shows the two source compact sets on the left and the result compact set produced by the merging algorithm on the right. An exploded drawing of the result compacts is shown in Figure 28.

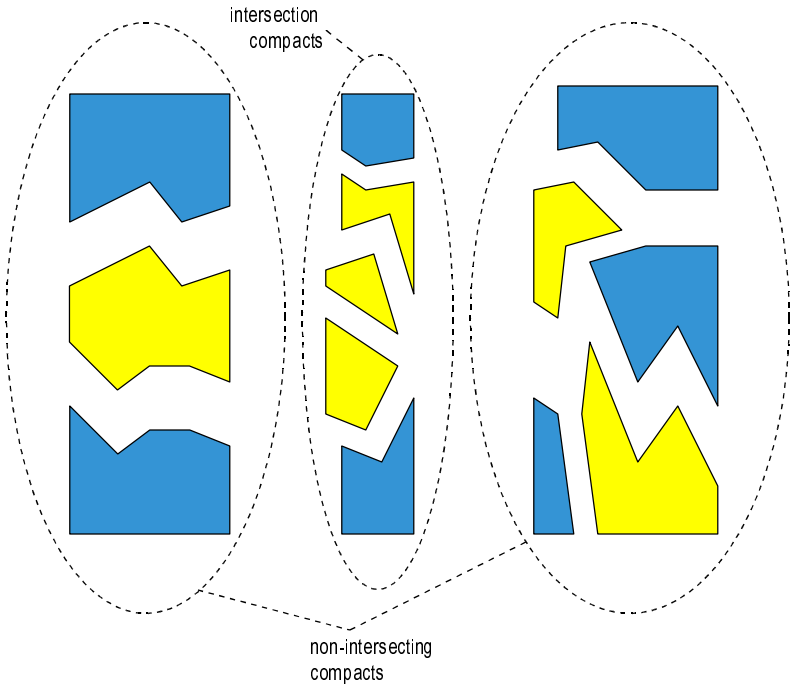


Figure 28. The result compacts from Figure 27

The result compact set contains both intersection and non-intersecting compacts. Intersection compacts are formed in the volume of $A \cap B$. The portions of A and B that do not intersect form the non-intersecting compacts.

```
∀ b ∈ B                                                    // PART 1: loop through all
    ∀ a ∈ A                                                // A and B compacts to create
        new(c ∈ C) = a ∩ b                                 // intersections, and remove
        material(c) = f(material(a), material(b), operation) // them from both source
        a = a - c                                          // compact sets.
        b = b - c

if (operation = ADD) ∨ (operation = XOR)                   // PART 2: copy non-intersecting
    C = C ∪ A ∪ B                                          // compacts to result set as
if operation = SUBTRACT                                    // needed.
    C = C ∪ A
```

Figure 29. Compact-set merging algorithm

The first part of the algorithm creates the intersection compacts. The function f is specified by the truth tables shown in Figure 31. The second part puts non-intersecting compacts into the result set.

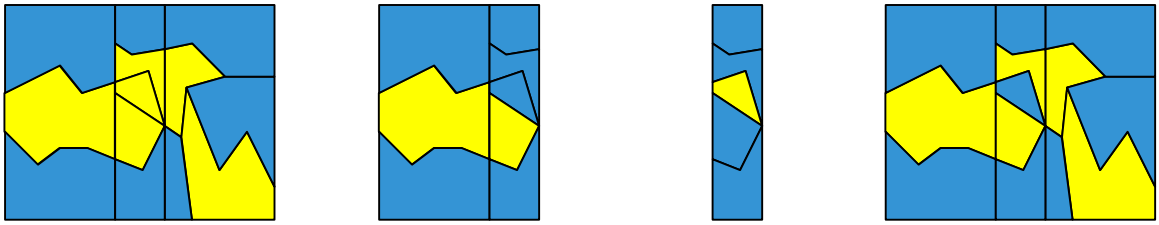


Figure 30. The four merging operations

This picture shows the result compact sets created by the four Boolean operations. The geometry of individual compacts is the same for all three operations; what changes are the material types for intersection compacts, and which non-intersecting compacts are included in the result set.

4.1.1 Intersection compacts

When compact sets 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. In this way, the volume defined by the intersection of the two primitives and their support structure is divided into a number of intersection compacts.

Geometry: The geometry of intersection compacts is determined by taking each compact, $a_i \in A$, and performing a Boolean intersection with every compact $b_j \in B$. Since A contains n compacts, and B contains m compacts, there are at most $n \cdot m$ intersection compacts. The shape of the intersection compact $a_i b_j$ is simply the intersection between the volumes defined by a_i and b_j (see Figure 28).

Material Type: The material for an intersection compact is specified by a truth table (see Figure 31) that depends on the type of merging operation: addition, subtraction, intersection, or exclusive-or. For the addition (union) operation, result compacts are part material if either of the two "parent" compacts is part material. For subtraction, the truth table shows that the intersection compact, c, has the same material type as a, unless a and b are both part material. Intersection compacts produced by the intersect operation are only part material if a and b are both part material. Finally, for the exclusive-or operation, intersection compacts are part material if a and b have different material types, and support material if a and b are the same material.

Adding support for multi-material parts to the design by composition system is primarily a matter of expanding these truth tables. This concept is discussed further in Section 9.2.1.

Add $A \cup B$

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

Subtract $A - B$

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

Intersect $A \cap B$

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

Exclusive or $(A+B)-(A \cap B)$

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

Figure 31. Truth tables for compact set merging operations

The tables shown specify the material type (part, P, or support, S) for a compact, c, which is formed by the intersection of two source compacts, a and b.

4.1.2 Non-intersecting compacts

When two compact sets are combined, portions of some of the compacts in A or B may not intersect any compact in the other set. For each of the four Boolean operations, the Venn diagrams in Figure 32 show which portions of the projected volume contain the new primitive. As shown, for addition and exclusive-or, non-intersecting compacts from both A and B are required in the result compact set. For subtraction, only the non-intersecting compacts from A are needed. For intersection, only the intersection compacts are needed in the result set.

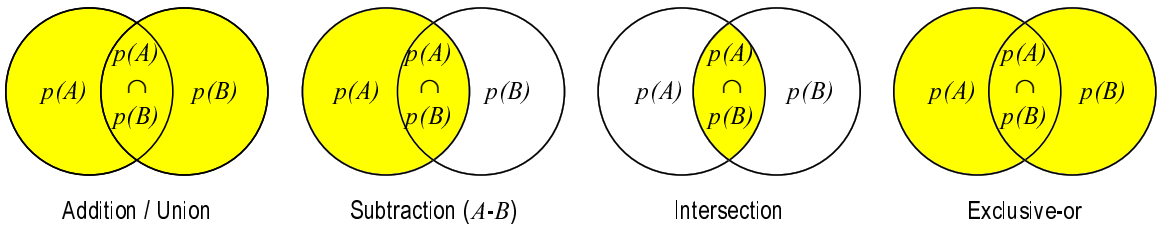


Figure 32. Result compact set projected volume

These Venn diagrams represent the **projected volume** (not the actual volumes) for the four Boolean operations. As shown, for addition and exclusive-or, non-intersecting compacts from both A and B contain part material. For subtraction, only A contains part material, and for intersection, neither A nor B has part material.

The second part of the algorithm in Figure 29 copies the needed non-intersecting compacts to the result set. For compact sets A and B with n and m compacts, respectively, the maximum number of non-intersecting compacts in the result set is $n+m$ for addition or exclusive-or, n for subtraction, and zero for intersection.

Geometry: The geometry of non-intersecting compacts is determined by subtracting from each compact, $a_i \in A$, all the intersection compacts $a_i b_Y$, where $Y \in \{1,2,...m\}$, and similarly subtracting all the intersections $a_X b_j$, where $X \in \{1,2,...n\}$ from every $b_j \in B$. (See Figure 28)

Material Type: The material type for non-intersecting compacts does not change from the original compact sets.

4.1.3 Embedded components

As mentioned earlier, an interesting capability of SDM is the fabrication of parts with embedded components. Some examples of embedded components are bearings, shafts, electronic circuit boards, sensors, and actuators. To support embedded components, the primary change to the algorithm presented above is to create a new material type, "embedded." Unlike the existing part and support materials, embedded component compacts cannot be subdivided by primitive merging operations.

There are three modifications required, as shown in Figure 33. First, Part 1 of the algorithm must be extended to distinguish between the four possible cases of compacts a and b being embedded or not:

1. **both** a **and** b **embedded**: this case is illegal because it represents an interference between two components which are to be inserted in the SDM part.
2. **only** a **embedded**: in this case, the a compact is left unchanged, and the portion of the b compact that intersects with it is removed from b to ensure that there will not be an intersection between compacts in the result compact set.
3. **only** b **embedded**: this case is the opposite of 2, except for the subtraction operation. For subtraction, the embedded component b will not be part of the result compact set. Therefore, a support material intersection compact must be created to "fill in" the volume removed from a.
4. **neither** a **or** b **embedded**: this case is unchanged from the algorithm in Figure 29.

```
∀ b ∈ B                                                    // PART 1: loop through all
    ∀ a ∈ A                                                // A and B compacts.
        if matl(a)=EMBED ∧ matl(b)=EMBED
            error: EMBED components intersecting.          // EMBED's can't intersect
        if (matl(a)≠EMBED ∧ matl(b)≠EMBED)
            ∨ (matl(a)≠EMBED ∧ matl(b)=EMBED ∧ oper=SUBTRACT)
            new(c ∈ C) = a ∩ b
            matl(c) = f(matl(a), matl(b), oper)
            a = a - c
            b = b - c
        if matl(a)=EMBED ∧ matl(b)≠EMBED                   // embedded compacts in A
            b = b - (a ∩ b)
        if matl(a)≠EMBED ∧ matl(b)=EMBED                   // embedded compacts in B
            a = a - (a ∩ b)

if (oper = ADD) ∨ (oper = XOR)                             // PART 2: copy non-intersecting
    C = C ∪ A ∪ B                                          // compacts
if oper = SUBTRACT
    C = C ∪ A
if oper = INTERSECT                                        // make sure embedded compacts
    if ∃a[a∈A ∧ matl(a)=EMBED]                             // and support get copied.
        ∀a[a∈A ∧ matl(a)≠EMBED]
            matl(a)=SUPPORT
        C = C ∪ A
    if ∃b[b∈B ∧ matl(b)=EMBED]
        ∀b[b ∈ B ∧ matl(b)≠EMBED]
            matl(b)=SUPPORT
        C = C ∪ B
```

Figure 33. Embedded component merging algorithm

Because embedded components cannot be subdivided, they must be handled differently by the merging algorithm.

The second modification is to Part 2 of the algorithm. Part 1 of the algorithm leaves all the embedded components in A and B. Therefore, in the case of an intersection operation, the embedded components would be left out of the result primitive. As shown, in the case of intersection, the algorithm is modified to copy all embedded compacts from both A and B into the result compact set. However, there may be portions of these embedded components "sticking out" of the geometric intersection of **A** and **B**. To provide support material for these compacts, the other compacts in A or B are changed to support material and copied into the result compact set.

The third modification to the algorithm is to extend the truth table for subtraction to handle the special case of b being an embedded component (see Figure 34).

Add $A \cup B$

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

Subtract $A - B$

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

Intersect $A \cap B$

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

Exclusive or $(A+B)-(A \cap B)$

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

Figure 34. Embedded component truth tables

To handle the special case of subtracting embedded components, the subtraction truth table must be extended.

4.1.4 Compact set simplification

The merging algorithm produces a result compact set with many compacts (see the right side of Figure 27). The result compact set can be simplified considerably by combining (with a Boolean union) sequential compacts of the same material. Figure 35 shows the compact set for the example part after simplification. Details of compact set simplification are described in the next chapter.

Merging two compact sets with n and m compacts, respectively will produce at most nm intersection compacts and $n+m$ non-intersecting compacts, which leads to this equation for the maximum number of compacts:

$$nm + n + m = (n+1)(m+1) - 1 .$$

Applying this rule to two compact sets with n compacts gives a maximum number of compacts of

$$(n+1)^2 - 1 .$$

Using this value as m in the previous equation shows that merging 3 compact sets with n compacts produces at most $(n+1)^3-1$ compacts:

$$\begin{aligned}(n+1)\left((n+1)^2 - 1 + 1\right) - 1 &= (n+1)(n+1)^2 - 1 \\ &= (n+1)^3 - 1\end{aligned}$$

Continuing this process leads to the general result that in the worst case, merging p primitives which each contain n compacts produces a compact set with $(n+1)^p$-1 compacts.

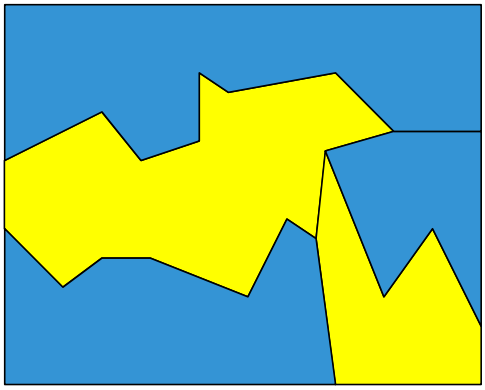


Figure 35. Compact set simplification

This picture shows the compact set from Figure 27 after simplification. Because the simplification step depends on precedence information, discussion is presented in Chapter 5.

4.2 Example

This section illustrates the compact set merging algorithm with a simple example. Figure 36 shows the two primitives the designer is combining, and the resulting shape. Figure 37 shows the compact sets for the two primitives. (In all illustrations the growth direction is vertical.)

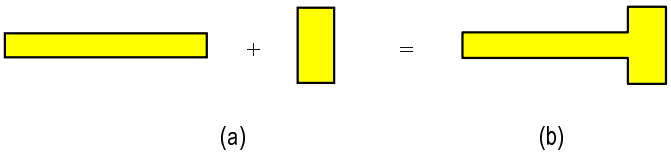


Figure 36. A simple design by composition example

The designer combines the two primitives in (a) to form the shape in (b).

Figure 38 shows the first step of the merging operation. Compacts a_1 and b_1 are intersected to form compact c_1. The intersection is subtracted from the two source

compacts, removing a portion of a_1 and all of b_1. Finally, the intersection compact is added to the result compact set, C.

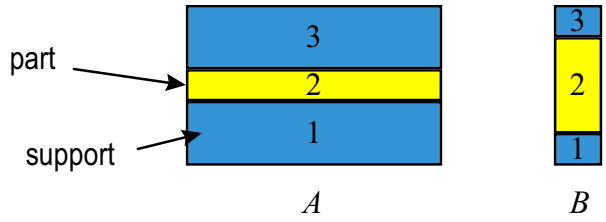


Figure 37. The two compact sets that will be combined.

In step 2, shown in Figure 39, the remainder of compact a_1 is intersected with b_2. Again, the intersection is subtracted from the source compacts and added to the result set. In step 3 (not shown) the remainder of compact a_1 will be intersected with compact b_3. The result of this intersection is a null compact, so no changes are made to the source or result compact sets. The process continues through step 9, where compact a_3 (minus the part that was removed by step 8) is intersected with b_3 to produce the final intersection compact.

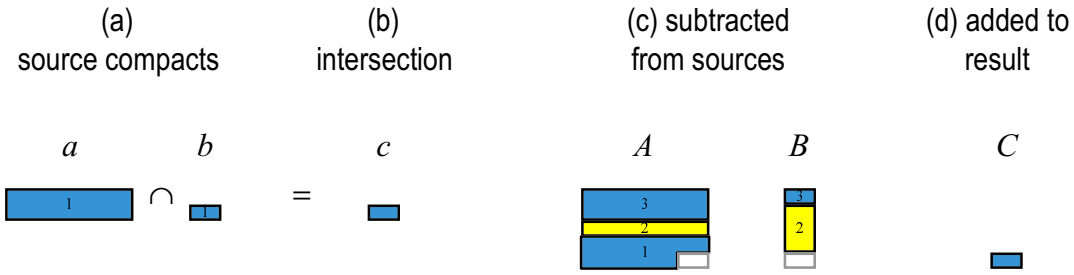


Figure 38. Step 1 of the merging operation

The two compacts shown in (a) are intersected, producing the new compact shown in (b). The intersection is subtracted from the two source compact sets (c), and is added to the result set, C, in (d).

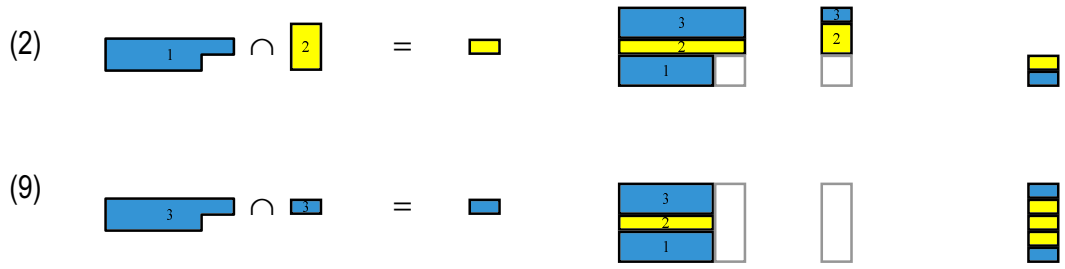


Figure 39. Steps 2 and 9 of the merging operation

In step 2, the remaining piece of compact a_1 is intersected with compact b_2. In step 9, the remainder of compact a_3 is intersected with b_3. This produces the final intersection compact.

After step 9, the compacts in set B have been completely consumed by the intersection and subtraction operations. There are still three compacts in set A, however. These non-intersecting compacts are added to the result set. The complete result set is shown in Figure 40(a). The simplification operation, described in Chapter 5, converts the result set to the compact set shown in Figure 40(b).

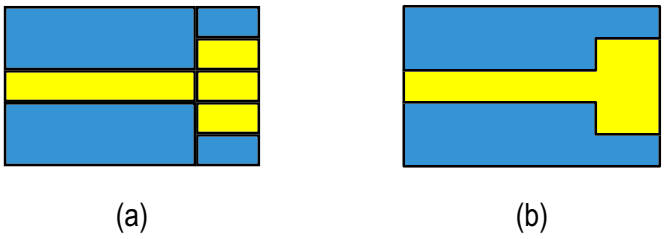


Figure 40. Compact set simplification

The non-intersecting pieces of the A compacts are added to the result set. There are no non-intersecting B compacts in this example. (a) shows the complete result compact set. (b) shows the result compact set after simplification

4.3 Completeness of the algorithm

To produce a useful design system, the compact merging algorithm must be a closed operation, and must scale efficiently for complex compact sets.

4.3.1 Closure

Closure requires that the compact merging algorithm always produces a valid compact set when both input compact sets are valid. To satisfy the definition of a valid compact set, the output of the merging algorithm must meet these four requirements:

1. the result compacts will completely fill the projected volume of the new primitive
2. there will be no intersections between compacts in the result set
3. the intersection of two compacts will be a compact
4. subtracting the intersection pieces from non-intersecting compacts will not produce an illegal compact

Proof of Condition 1

Synopsis: For Subtraction, the volume of the result compact set, C, is the volume of the source set A. Since A completely fills $p(\mathbf{A})$, and $\mathbf{C} \subseteq \mathbf{A}$, we know that $p(\mathbf{C}) \subseteq p(\mathbf{A})$, and therefore C at least fills $p(\mathbf{C})$. For Union and XOR, the volume of C is the union of the volumes of A and B, which is $p(\mathbf{A} \cup \mathbf{B})$. We know that $\mathbf{C} \subseteq \mathbf{A} \cup \mathbf{B}$ and thus $p(\mathbf{C}) \subseteq p(\mathbf{A} \cup \mathbf{B})$, so we can say that C fills $p(\mathbf{A} \cup \mathbf{B})$. Finally, for Intersection, the compacts in C contain all the points in $p(\mathbf{A} \cap \mathbf{B})$, and $\mathbf{C} = \mathbf{A} \cap \mathbf{B}$. Therefore, C contains all points in $p(\mathbf{C})$.

Proof for Subtraction: Given a point q_1 in $\mathbf{C}$, we must show that all points $q_2 \in W$ with the same xy coordinates as q_1 are in a compact in C.

Take any point q_1 from the primitive $\mathbf{C}$:

$$q_1 = (x, y, z_1)$$
$$q_1 \in \mathbf{C}$$

Let q_2 be any point in W with the same x and y coordinates:

$$q_2 = (x, y, z_2)$$
$$q_2 \in W$$

We know, from the compact merging algorithm, that every point in $\mathbf{C}$ is also in $\mathbf{A}$:

$$q_1 \in \mathbf{C} \rightarrow q_1 \in \mathbf{A}$$
$$\therefore q_1 \in \mathbf{A}.$$

We know, from Equation 3, and because A is a valid compact set for $\mathbf{A}$, that

$$(x,y,z) \in \mathbf{A} \rightarrow \forall z' \exists a\left[(x,y,z') \in W \leftrightarrow \left(a \in A \wedge (x,y,z') \in a\right)\right].$$

Substituting q_1 and q_2 into this statement, and using the fact that $q_2 \in W$ gives:

$$q_1 \in \mathbf{A} \rightarrow \exists a\left(q_2 \in W \leftrightarrow a \in A \wedge q_2 \in a\right)$$
$$\therefore \exists a\left(a \in A \wedge q_2 \in a\right).$$

Therefore, we know that there is a compact a in A that contains q_2 and that q_2 is in $p(\mathbf{A})$:

$$q_2 \in p(\mathbf{A}).$$

The compact merging algorithm ensures that every point in the compacts of A is included in C (because C includes the points in A that intersect with B and the points that do not intersect with B)

$$\forall q \forall a \exists c\left[\left(a \in A \wedge q \in a\right) \rightarrow \left(q \in c \wedge c \in C\right)\right]$$
$$\therefore \exists c\left[q_2 \in c \wedge c \in C\right]$$

This final statement shows that there is a compact in C that includes every q_2, so the proof is complete.

Proof for Union and Exclusive-Or: Again, given a point q_1 in **C**, we must show that all points $q_2 \in W$ with the same xy coordinates are in a compact in C.

Start with the same q_1 and q_2:

$$q_1 = (x,y,z_1)$$
$$q_1 \in \mathbf{C}$$
$$q_2 = (x,y,z_2)$$
$$q_2 \in W$$

For union and exclusive-or operations, we know that $\mathbf{C} \subseteq \mathbf{A} \cup \mathbf{B}$, so q_1 must be in $\mathbf{A} \cup \mathbf{B}$:

$$q_1 \in \mathbf{C} \rightarrow q_1 \in \mathbf{A} \cup \mathbf{B}$$
$$\therefore q_1 \in \mathbf{A} \vee q_1 \in \mathbf{B}$$

Using Equation 3 and the assumption that A and B are valid compact sets for **A** and **B**, respectively, we can say that q_2 must belong to a compact $a \in A$ or $b \in B$:

$$\exists a \exists b \left[(a \in A \wedge b \in B) \wedge \left(q_2 \in a \vee q_2 \in b \right) \right]$$

Examination of the compact merging algorithm shows that the compacts in C contain every point from the compacts of A and B:

$$\forall q \forall a \exists c \left[\left(a \in A \wedge q \in a \right) \rightarrow \left(q \in c \wedge c \in C \right) \right]$$
$$\forall q \forall b \exists c \left[\left(b \in B \wedge q \in b \right) \rightarrow \left(q \in c \wedge c \in C \right) \right]$$

Combining these two statements tells us that for any two compacts a and b, every point q in $a \cup b$ belongs to a compact $c \in C$, and therefore q_2 belongs to a compact in C:

$$\therefore \forall a \forall b \exists c \left[\left(a \in A \wedge b \in B \wedge \left(q \in a \cup b \right) \right) \rightarrow \left(q \in c \wedge c \in C \right) \right]$$
$$\therefore \exists c \left[q_2 \in c \wedge c \in C \right]$$

Proof for Intersection: This proof is similar to the proofs for the other Boolean operations.

$$q_1 = \left(x, y, z_1 \right)$$
$$q_1 \in \mathbf{C}$$
$$q_2 = \left(x, y, z_2 \right)$$
$$q_2 \in W$$

For intersection operations, we know that $\mathbf{C} \subseteq \mathbf{A} \cap \mathbf{B}$, so q_1 must be in $\mathbf{A} \cap \mathbf{B}$:

$$q_1 \in \mathbf{C} \rightarrow q_1 \in \mathbf{A} \cap \mathbf{B}$$
$$\therefore q_1 \in \mathbf{A} \wedge q_1 \in \mathbf{B}$$

Equation 3 implies that q_2 must belong to both $a \in A$ and $b \in B$:

$$\exists a \exists b \left[(a \in A \wedge b \in B) \wedge \left(q_2 \in a \wedge q_2 \in b \right) \right]$$

Examination of the compact merging algorithm shows that the compacts in C contain every point from all the intersections of compacts from A and B:

$$\forall q \forall a \forall b \exists c \left[\left(a \in A \wedge b \in B \wedge q \in a \cap b \right) \rightarrow \left(q \in c \wedge c \in C \right) \right]$$

Since we know that $q_2 \in a \cap b$, the previous statement shows that q_2 must also belong to a compact in C:

$$\therefore \exists c \left[q_2 \in c \wedge c \in C \right]$$

Proof of Condition 2

There are three cases to consider:

1. two non-intersecting compacts
2. two intersection compacts
3. an intersection compact and a non-intersecting compact

Synopsis:

1. Assuming that the two input compact sets are valid implies that there are no intersections between any two compacts in A. Likewise, there are no intersections between any compacts in B. There are therefore no intersections between any of the non-intersecting compacts. By definition, there is no intersection between a non-intersecting compact in A and a non-intersecting compact in B.
2. In the volume where A intersects B, each point belongs to only one compact $a \in A$ and one compact $b \in B$. The points in $a \cap b$ defines one intersection compact, c. Because there are no self-intersections in the two source compact sets, c does not intersect any other compacts from A or B (and therefore it also doesn't intersect any other compacts from C).
3. A non-intersecting compact a_i cannot intersect with an intersection compact $a_j b_k$ because there is no intersection between a_i and a_j. For the case where $i=j$, the volume of $a_i b_k$ is removed from a_i, so there is again no intersection between the result compacts.

Proof for Case 1: If there are two non-intersecting compacts that intersect with each other, than one of these three statements must be true:

$$\exists a_1 \exists a_2 \left[a_1 \in A \wedge a_2 \in A \wedge \left(a_1 \cap a_2 \neq \varnothing \right) \right]$$
$$\exists b_1 \exists b_2 \left[b_1 \in B \wedge b_2 \in B \wedge \left(b_1 \cap b_2 \neq \varnothing \right) \right]$$
$$\exists a \exists b \left[a \in A \wedge b \in B \wedge (a \cap b \neq \varnothing) \right]$$

The first two statements contradict the assumption that A and B are valid compact sets. The third statement cannot be true because any points that belong to $a \cap b$ would be in one of the intersection compacts, and would therefore have been subtracted from both a and b.

Proof for Case 2: If two intersection compacts intersect, than it must be true that

$$\exists p \exists a_1 \exists a_2 \exists b_1 \exists b_2 \begin{bmatrix} a_1 \in A \wedge a_2 \in A \wedge \\ b_1 \in B \wedge b_2 \in B \wedge \\ p \in (a_1 \cap b_1) \wedge p \in (a_2 \cap b_2) \end{bmatrix}.$$

However, this statement would require that

$$p \in (a_1 \cap a_2) \wedge p \in (b_1 \cap b_2).$$

Because A and B are valid compact sets, we know that this statement is impossible, and therefore that there are no intersection compacts that intersect each other.

Proof for Case 3: If an intersection compact intersects a non-intersecting compact, we have two possibilities

$$\exists p \exists a_1 \exists a_2 \exists b_1 \begin{bmatrix} a_1 \in A \wedge a_2 \in A \wedge b_1 \in B \wedge \\ p \in (a_1 \cap b_1) \wedge p \in a_2 \end{bmatrix}$$

$$\exists p \exists a_1 \exists b_1 \exists b_2 \begin{bmatrix} a_1 \in A \wedge b_1 \in B \wedge b_2 \in B \wedge \\ p \in (a_1 \cap b_1) \wedge p \in b_2 \end{bmatrix}$$

Again, these two statements contradict the assumption that A and B are valid compact sets, and that therefore

$$\sim \exists a_1 \exists a_2 [a_1 \cap a_2 \neq \varnothing]$$

$$\sim \exists b_1 \exists b_2 [b_1 \cap b_2 \neq \varnothing]$$

Proof of Condition 3

Synopsis: We can prove this statement by induction. The definition of a compact requires that every ray cast in the growth direction passes through the compact only once. If we consider the intersection of two compacts, a and b (Figure 41), as shown in Figure 42, rays in the growth direction will fall into these four categories:

1. those that pass through compact a only
2. those that pass through compact b only
3. those that pass through both compacts a and b (and therefore pass through the intersection)
4. those that pass through neither compact.

If the intersection solid were not a compact, there must be at least one ray in Category 3 that passes through the intersection more than once. We can define Z_{a1} and Z_{a2} as the heights where one of these rays enters and exits compact a. Likewise, we can define Z_{b1} and Z_{b2} as the heights where the ray enters and exits compact b. We know that $Z_{a1}<Z_{a2}$ and $Z_{b1}< Z_{b2}$ because the ray must enter a compact before it exits. The ray enters the intersection solid at the greater of Z_{a1} and Z_{b1}, and exits the intersection solid at the lesser of Z_{a2} and Z_{b2}. Because max(Z_{a1}, Z_{b1}) and min(Z_{a2}, Z_{b2}) define a single contiguous range, every ray in Category 3 enters and exits the intersection only once. Since no other rays pass through the intersection solid, it satisfies the definition of a compact. We can therefore state:

Theorem 1: The intersection of two compacts is also a compact.

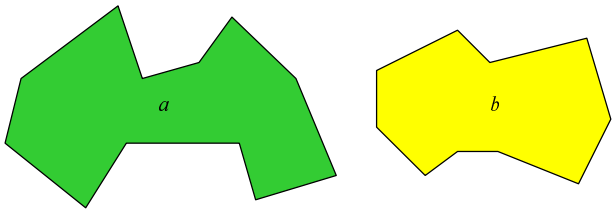


Figure 41. The two compacts whose intersection is shown in Figure 42.

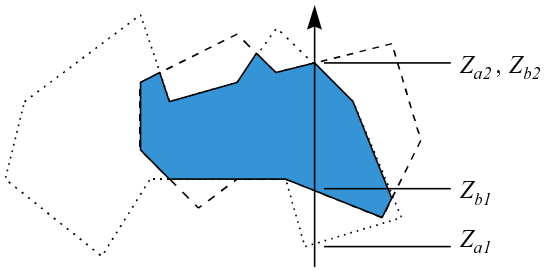


Figure 42. Intersection compact

Any ray in the growth direction will pass through the intersection of two compacts only once. The intersection is therefore also a compact.

Proof: Beginning with two valid compacts, a and b, and their intersection $c=a\cap b$, we need to show that c satisfies the mathematical definition of a compact, Equation 1:

$$\forall x\forall y\forall z\exists z_1\exists z_2\left[(x,y,z)\in c\leftrightarrow z_1\le z\le z_2\right]$$

Using the definition of a compact for a and b, we can say that for any point, $p=(x,y,z)$:

$$\exists z_{a1}\exists z_{a2}\left[p\in a\leftrightarrow z_{a1}\le z\le z_{a2}\right]$$
$$\exists z_{b1}\exists z_{b2}\left[p\in b\leftrightarrow z_{b1}\le z\le z_{b2}\right]$$

If $p\in a\cap b$, then both of these statements must be true, and we know

$$\exists z_{a1}\exists z_{a2}\exists z_{b1}\exists z_{b2}\left[p\in a\cap b\leftrightarrow\left(z_{a1}\le z\le z_{a2}\right)\wedge\left(z_{b1}\le z\le z_{b2}\right)\right]$$

which simplifies to

$$\exists z_{a1}\exists z_{a2}\exists z_{b1}\exists z_{b2}\left[p\in a\cap b\leftrightarrow\max\left(z_{a1},z_{b1}\right)\le z\le\min\left(z_{a2},z_{b2}\right)\right]$$

Therefore, we can say:

$$\therefore\forall x\forall y\forall z\exists z_{c1}\exists z_{c2}\left[\left(x,y,z\right)\in a\cap b\leftrightarrow z_{c1}\le z\le z_{c2}\right]$$

by letting $z_{c1}=\max(z_{a1},z_{b1})$ and $z_{c2}=\min(z_{a2},z_{b2})$. This statement is what we set out to prove, so we know that if a and b are both compacts, $a\cap b$ is also a compact.

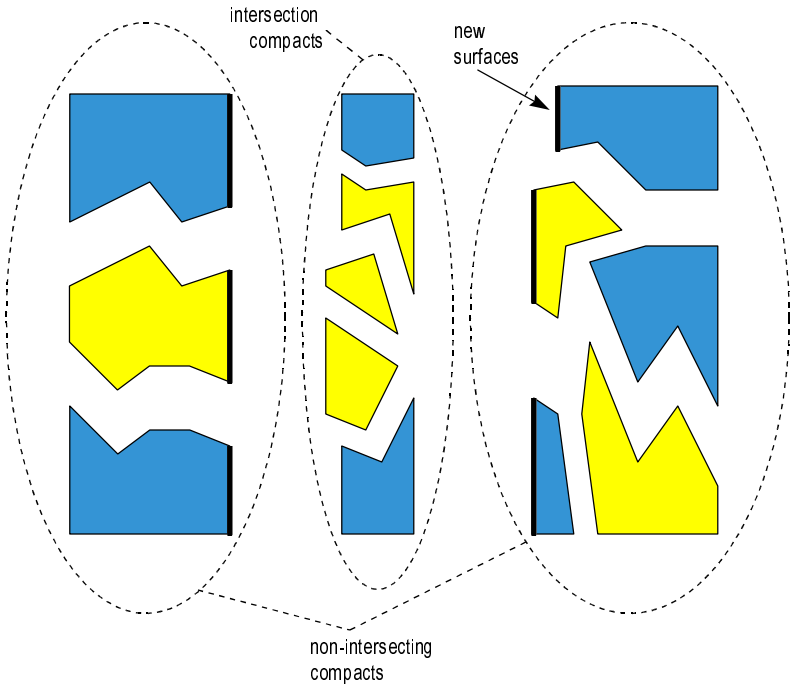


Figure 43. The result compacts from Figure 27

The result compact set contains both intersection and non-intersecting compacts. Theorem 1 states that the intersection compacts are all valid. The highlighted surfaces show where the non-intersecting compacts were cut by the edge of the other compact set. These surfaces are all parallel to the growth axis.

Proof of Condition 4

Synopsis: Cutting a compact with a surface parallel to the growth axis will not create any position where a ray in the growth direction will intersect the compact more than once. This fact leads to:

> **Theorem 2:** Cutting a compact with a surface parallel to the growth axis produces two solids that are both compacts.

The exterior surfaces of a valid compact set (excluding the top and bottom of the workspace) are all parallel to the growth axis. The merging operation subtracts from every compact a its intersection with the compact set B. The new surfaces (see Figure 27) created in a are all external surfaces of the compact set B, which must be parallel to the growth axis. From Theorem 2, we can say that non-intersection compacts created by the merging algorithm will be legal compacts.

Proof: If the merging algorithm is to violate Condition #4, there must be at least one point $p=(x,y,z)$ that satisfies this statement:

$$\exists z_1 \exists z_2 \exists p_1 \exists p_2 \left[\begin{array}{c} z_1 < z < z_2 \\ p_1 = (x,y,z_1) \wedge p_1 \in W \\ p_2 = (x,y,z_2) \wedge p_2 \in W \\ \exists b \left(\begin{array}{c} b \in B \wedge p \in b \\ \sim \exists b_1 \exists b_2 \left[b_1 \in B \wedge p_1 \in b_1 \wedge b_2 \in B \wedge p_2 \in b_2 \right] \end{array} \right) \end{array} \right] \quad (4)$$

In other words, there must be at least one point in a compact b which shares xy coordinates with one point above it and one point below it that are not contained by any compacts in B.

However, we know from Equation 3 that

$$(x,y,z) \in \mathbf{B} \rightarrow \forall z' \left((x,y,z') \in W \leftrightarrow \exists b \in B \left((x,y,z') \in b \right) \right)$$

because it is assumed that B is a valid compact set for **B**, and completely fills p(**B**).

Therefore, if the point p is in b, there are no points p_1 and p_2 that satisfy Equation 4, and Condition 4 is therefore true.

4.3.2 Scalability

In this section, n refers to the number of compacts in a compact set, and p is the number of merging operations.

The time required for the algorithm scales as $O(n^2)$ because to find every intersection requires looping through all the compacts in A and all the compacts in B. The algorithm is therefore efficient (i.e., it runs in polynomial time), provided n does not grow exponentially.

As stated above, in the worst-case, the number of compacts produced by p merging operations is $(n+1)^p$-1, which is an exponential function. In practice, though, there are normally plentiful opportunities to simplify the result compact set. Experiments with the prototype design by composition CAD system rarely produce compact sets with more than a dozen compacts. Provided the user does not create pathological designs that have little opportunity for simplification, the number of compacts, n, grows slowly, and the time required for the compact merging algorithm scales acceptably. For example,

the mechanism shown in Figure 66 was created with approximately forty merging operations. The final compact set contains only twelve compacts.

5. Compact precedence graphs

In design by composition, compact precedence graphs (CPGs) are used to represent the possible fabrication sequences for primitives. The CPG is based on a least-commitment approach to process planning; a given compact set may have many valid build orders. Early selection of one of these possibilities may produce inefficient process plans or inferior part quality. Preserving a range of possible plans in the CPG allows the designer, the manufacturer, or an automated algorithm to make sequencing decisions as needed.

After the algorithm in Chapter 4 creates the compact set (the nodes of the CPG) for a merged primitive, it is necessary to create the CPG links. First, the algorithms in Section 5.1 create the links in the CPG. Next, the simplification rules described in Section 5.2 are used to combine compacts that can be built simultaneously, reducing the number of compacts and simplifying the CPG.

5.1 Merging algorithm

This section describes the construction of the CPG for the result of a merging operation, based on the CPGs from the two source primitives. Of course, the result CPG could be constructed by examining the geometry of each compact, determining the direction of the surface normals, and identifying all the adjacent compacts. The algorithms presented in this Section, however, construct the CPG without any geometric computations. This approach makes the CPG calculations more efficient and more portable between CAD and solid modeling environments. The tradeoff that must be

made for these advantages is that the resulting CPGs will sometimes contain unnecessary links. These links are discussed in Section 5.1.3.

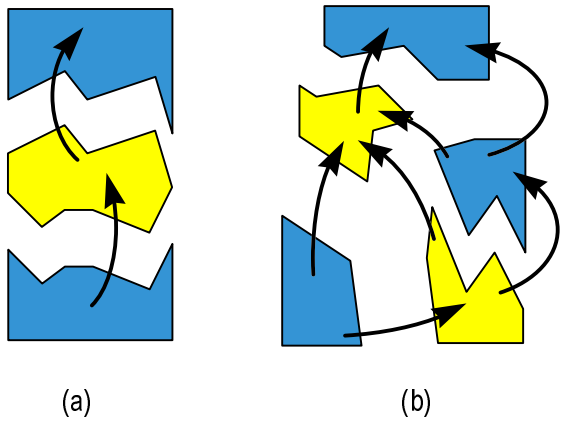


Figure 44. Source CPGs for the example in Figure 27

This figure shows the compact precedence graphs for the two source compact sets.

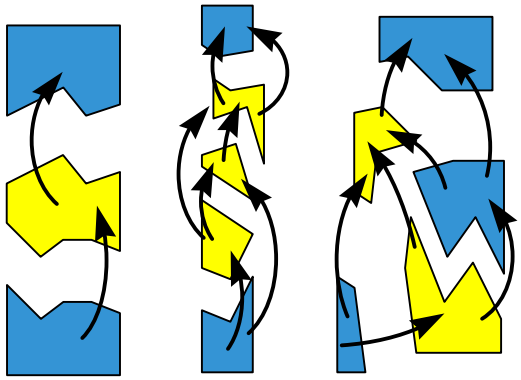


Figure 45. Result CPG for the example

This figure shows the precedence links for non-intersecting and intersection compacts in the result compact set. Although some of the links between intersection compacts are unnecessary, they do not cause any problems, as discussed in Section 5.1.3.

Figure 44 shows the two precedence graphs for the source compact sets in Figure 27 on page 49. After the compact merging algorithm of Chapter 4 is completed, the

algorithms presented below are used to construct the result precedence graph shown in Figure 45.

5.1.1 Non-intersecting compacts

No changes are needed to the CPG links for non-intersecting compacts. The remaining a and b compacts are copied to the result compact set with their corresponding CPG links intact. Because the surfaces which separate intersection compacts from non-intersecting compacts are always parallel to the growth direction, there will be no precedence links that connect intersection compacts with non-intersecting compacts.

5.1.2 Intersection compacts

The rule in Figure 46 can be used to generate the CPG for intersection compacts. This rule states that one intersection compact, $a_x b_y$, precedes another, $a_i b_j$, if one of three conditions are met:

1. $a_x = a_i$ and $b_y \rightarrow b_j$
2. $b_y = b_j$ and $a_x \rightarrow a_i$
3. $a_x \rightarrow a_i$ and $b_y \rightarrow b_j$

if $(a_x = a_i \wedge b_y \rightarrow b_j) \vee (b_y = b_j \wedge a_x \rightarrow a_i) \vee (a_x \rightarrow a_i \wedge b_y \rightarrow b_j)$
$a_x b_y \rightarrow a_i b_j$

Figure 46. Precedence rule for intersection compacts

This rule specifies the precedence relationships between intersecting compacts in the result compact set, C.

Condition 1 applies to intersection compacts that are formed when a surface that separates two compacts in B splits the compact a_i into two pieces. This condition is illustrated in Figure 47 where the surface separating b_1 and b_2 subdivides a_2. Because the surface which separates $a_2 b_2$ from $a_2 b_1$ is a portion of the surface which separated b_2 from b_1, the direction of the precedence link across the surface does not change. Therefore, the result CPG must include the link $a_2 b_1 \rightarrow a_2 b_2$.

Condition 2 is the same as Condition 1, but for compacts from B being split by surfaces that separate compacts in A. Condition 3 accounts for the case where the surfaces that separate compacts in A and B are coincident, as shown in Figure 48.

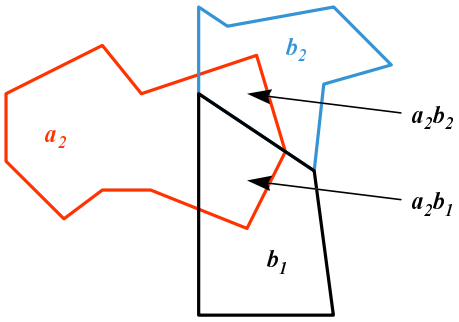


Figure 47. Intersecting compacts (case 1)

This picture shows the intersection of one compact from A with two compacts from B. Condition 1 in Figure 46 states that $a_2b_1 \rightarrow a_2b_2$.

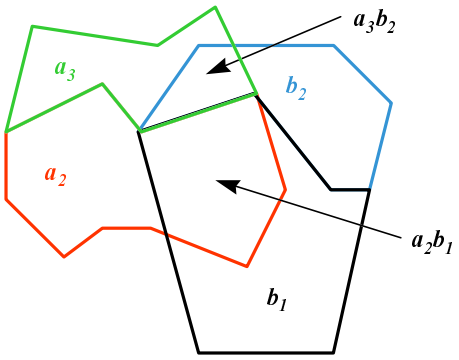


Figure 48. Intersecting compacts (case 3)

Here, the surface that separates a_3 from a_2 is coincident with the surface separating b_2 from b_1. Condition 3 from provides the precedence link $a_2b_1 \rightarrow a_3b_2$.

The algorithm shown in Figure 49 can be used after a merging operation to generate all precedence links for a CPG that does not contain embedded components. It loops through all the intersection compacts in the result list, C. Two inner loops then evaluate the rule in Figure 46; the first loop evaluates Conditions 1 and 3, and the second loop evaluates Condition 2. In this algorithm and the explanation below, the terms parent and child refer to source and intersection compacts. For example, the intersection compact a_xb_y is a child of the two parent compacts, a_x and b_y.

$\forall a_x b_y \{a_x b_y \in C \wedge \text{matl}(a_x b_y) \neq \text{EMBED}\}$	// loop through intersection compacts in C
a_x=parenta($a_x b_y$)	// a_x is the first parent of $a_x b_y$
b_y=parentb($a_x b_y$)	// b_y is second parent of $a_x b_y$
$\forall b_j \{b_j \in B \wedge b_y \rightarrow b_j\}$	// loop for all b_j preceded by b_y
$\forall a_i b_j \in$children(b_j)	// check all children, $a_i b_j$, of b_j
a_i=parenta($a_i b_j$)	
if $a_i = a_x$ then $a_x b_y \rightarrow a_i b_j$	// Check condition 1
else if $a_x \rightarrow a_i$ then $a_x b_y \rightarrow a_i b_j$	// Check condition 3
$\forall a_i \{a_i \in A \wedge a_x \rightarrow a_i\}$	// loop for all a_i preceded by a_x
$\forall a_i b_j \in$children(a_i)	// check all children, $a_i b_j$, of a_i
b_j=parentb($a_i b_j$)	
if $b_j = b_y$ then $a_x b_y \rightarrow a_i b_j$	// Check condition 2

Figure 49. Algorithm for generating CPG links

This algorithm implements the rules in Figure 46. The parent functions return pointers to the a and b compacts that were used to create the intersection compact. The complementary children function returns the set of all intersection compacts that were formed from a single source compact.

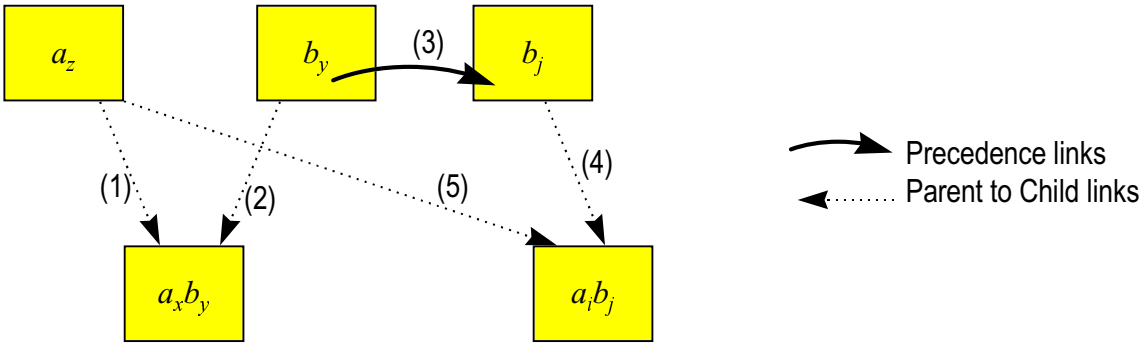


Figure 50. Condition 1 from Figure 46

This picture shows the compacts, precedence links, and parent-child relationships used by the algorithm in Figure 49 to evaluate Condition 1.

The operation of the algorithm is illustrated in Figure 50 and Figure 51. The compact being examined is $a_x b_y$. First, the parent links (links 1 and 2) of $a_x b_y$ are used to find the two parent compacts, a_x and b_y. Next, the algorithm loops through all compacts b_j which are preceded by b_y (link 3). The children (link 4) of b_j are the intersection compacts $a_i b_j$, which are the candidates for compacts which may be

preceded by $a_x b_y$. To evaluate Condition 1 from Figure 46, the algorithm checks (with link 5) if the other parent of $a_i b_j$, a_i is the same compact as a_x. If $a_i \neq a_x$, to evaluate Condition 3, it checks for a precedence link (link 6) from a_x to a_i.

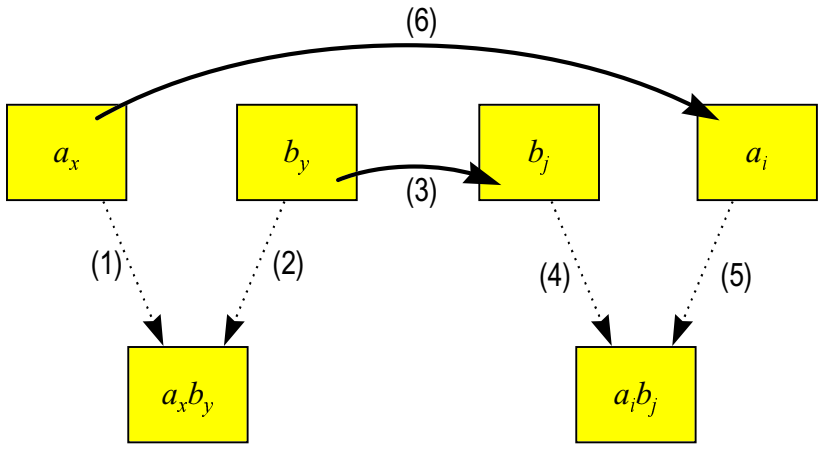


Figure 51. Condition 3 from Figure 46

This picture illustrates the part of the algorithm in Figure 49 that takes care of Condition 3.

5.1.3 Embedded components

Because embedded components are not subdivided by the merging operation, they need special treatment in the CPG, as described in this Section.

Embedded components and intersection compacts

The precedence rules for embedded compacts are shown in Figure 52. The first rule takes care of embedded compacts in A, and consists of four cases. The first case states that an embedded compact a_x precedes an intersection compact $a_i b_j$ if $a_x \rightarrow a_i$ and a_x intersects b_j. This statement corresponds to the case where the surface separating a_x and a_i intersects the compact b_j. The condition $a_x \rightarrow a_i$ ensures that this surface is a upward-facing surface of a_x. Therefore, there must be a precedence link out of a_x across this surface in the result compact set. The compact a_i must intersect b_j because $a_i b_j$ exists. Since a_x also intersects b_j, the surface separating a_x from a_i probably passes through b_j. Hence, the precedence link $a_x \rightarrow a_i b_j$ is needed in the result CPG.

The second case accounts for the situations where the surface separating a_x from a_i is coincident with the surface separating b_j from b_k. In this case, there must be a

precedence link from a_x to $a_i b_k$, but there is no intersection between a_x and b_k. This part of the rule is similar to Condition 3 of the rule in Figure 46.

The third and fourth cases create precedence links that point into the embedded components in A. They operate the same as the first two cases, but with the direction of the precedence links reversed. The second rule in Figure 52 is the same as the first, but for embedded components in B, instead of A.

if $(a_x \cap b_j) \neq \varnothing \wedge$ material(a_x)=EMBED
  if $(a_x \rightarrow a_i)$ then $a_x \rightarrow a_i b_j$
  if $(a_x \rightarrow a_i) \wedge (b_j \rightarrow b_k)$ then $a_x \rightarrow a_i b_k$
  if $(a_i \rightarrow a_x)$ then $a_i b_j \rightarrow a_x$
  if $(a_i \rightarrow a_x) \wedge (b_k \rightarrow b_j)$ then $a_i b_k \rightarrow a_x$

if $(b_y \cap a_i) \neq \varnothing \wedge$ (material(b_y)=EMBED)
  if $(b_y \rightarrow b_j)$ then $b_y \rightarrow a_i b_j$
  if $(b_y \rightarrow b_j) \wedge (a_i \rightarrow a_k)$ then $b_y \rightarrow a_k b_j$
  if $(b_j \rightarrow b_y)$ then $a_i b_j \rightarrow b_y$
  if $(b_j \rightarrow b_y) \wedge (a_k \rightarrow a_i)$ then $a_k b_j \rightarrow b_y$

Figure 52. Precedence rules for embedded components

These rules identify precedence links between embedded component compacts and intersection compacts.

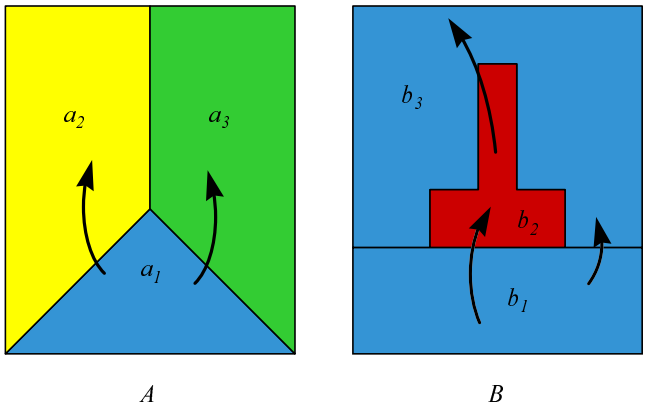


Figure 53. Merging example with embedded component b_2

An example of these rules is shown in Figure 53 and Figure 54. In this example, the compact b_2 is an embedded component, so the second rule from Figure 52 will provide

the precedence links. The links $b_2 \rightarrow a_2 b_3$, and $b_2 \rightarrow a_3 b_3$ are specified by the first case of the rule, while the links $a_1 b_1 \rightarrow b_2$, $a_2 b_1 \rightarrow b_2$, and $a_3 b_1 \rightarrow b_2$ are from the third case.

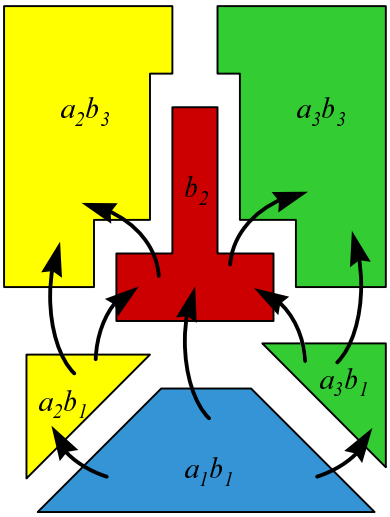


Figure 54. Result of merging operation with embedded component

The precedence links $b_2 \rightarrow a_2 b_3$, and $b_2 \rightarrow a_3 b_3$ are specified by the first case of the second rule in Figure 52. The link $a_1 b_1 \rightarrow b_2$, $a_2 b_1 \rightarrow b_2$, and $a_3 b_1 \rightarrow b_2$ are from the third case from that rule.

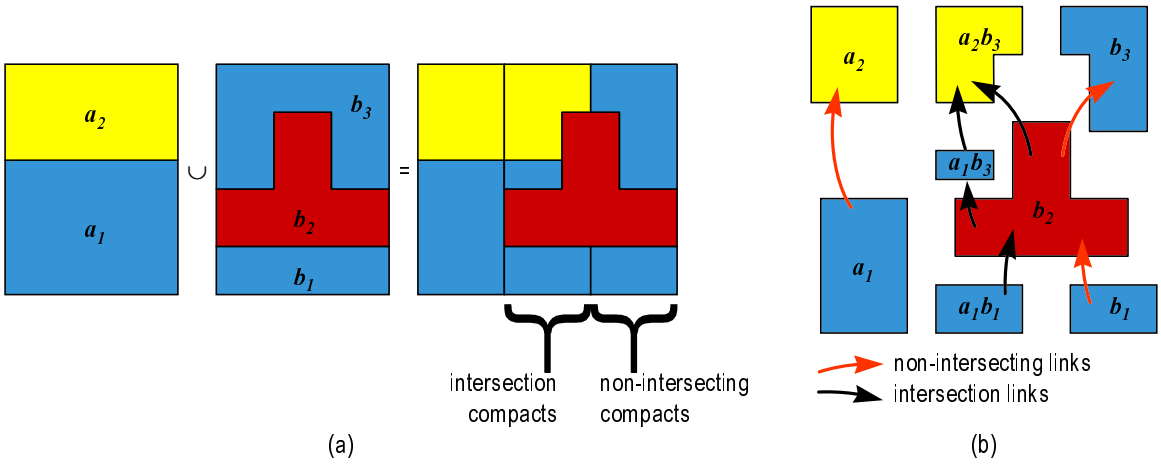


Figure 55. A partially-intersecting embedded compact

In the merging operation shown in (a), the embedded component compact, b_2, only partly intersects the compact set A. As indicated in (b), b_2 has CPG links produced by the algorithms in Sections 5.1.2 and 5.1.3.

Embedded components and non-intersecting compacts

Unlike normal compacts, which are either intersection or non-intersecting compacts, an embedded component compact may belong to both groups, as shown in Figure 55(a). Despite this "dual-citizenship," no special treatment is necessary for the compact's precedence links. Figure 55 shows that the precedence links between the embedded component and the non-intersecting compacts are maintained from the source compact set, as described above in Section 5.1.1. The precedence links between the embedded component and the intersection compacts will be created by the rules in Figure 52.

$\forall a_x\{a_x \in A \wedge \text{matl}(a_x)=\text{EMBED}\}$	// loop through embedded compacts in A
$\forall b_j\{b_j \in B \wedge \text{intersect}(a_x, b_j)\}$	
$\forall a_i\{a_i \in A \wedge a_x \neq a_i\}$	// intersect function computed during the
$\forall b_k\{b_k \in B \wedge b_k \neq b_j\}$	// algorithm in Chapter 4
if $a_x \rightarrow a_i$ then	
$a_x \rightarrow a_i b_j$	// case 1
if $b_j \rightarrow b_k$ then $a_x \rightarrow a_i b_k$	// case 2
if $a_i \rightarrow a_x$ then	
$a_i b_j \rightarrow a_x$	// case 3
if $b_k \rightarrow b_j$ then $a_i b_k \rightarrow a_x$	// case 4
$\forall b_y\{b_y \in B \wedge \text{matl}(b_y)=\text{EMBED}\}$	// loop through embedded compacts in B
$\forall a_i\{a_i \in A \wedge \text{intersect}(a_i, b_y)\}$	// intersect computed in Chapter 4
$\forall b_j\{b_j \in B \wedge b_y \neq b_j\}$	
$\forall a_k\{a_k \in A \wedge a_k \neq a_i\}$	
if $b_y \rightarrow b_j$ then	
$b_y \rightarrow a_i b_j$	// case 1
if $a_i \rightarrow a_k$ then $b_y \rightarrow a_k b_j$	// case 2
if $b_j \rightarrow b_y$ then	
$a_i b_j \rightarrow b_y$	// case 3
if $a_k \rightarrow a_i$ then $a_k b_j \rightarrow b_y$	// case 4

Figure 56. Algorithm for embedded component CPG links

This algorithm implements the rules in Figure 52. It is assumed that the intersect function (which returns a Boolean value indicating whether two compacts intersect) is computed during the merging algorithm in Chapter 4, which must evaluate all possible intersections.

Algorithm for embedded compacts

The algorithm in Figure 56 implements the precedence rules from Figure 52. The only difficulty in constructing this algorithm is the intersect function, which indicates whether two source compacts intersect. It is assumed that the merging algorithm in Chapter 4 will build a matrix (or another data structure) that contains the values for this function. Because Boolean operations on solids are computationally difficult, and the merging algorithm in Chapter 4 must check all intersections, it would be inefficient to re-check all the possible intersections here.

5.1.4 Extraneous precedence links

As indicated above, and shown in Figure 57, the precedence graphs produced by these algorithms will contain some unnecessary links. While these extraneous links are unsightly and inelegant, they do not have a major impact on the result compact graph. This section shows that these links are always redundant with other links in the CPG, and therefore their presence or absence does not change the sequencing information in the graph. Their only adverse effect is to slightly degrade the efficiency of the CPG merging algorithms. This performance handicap, however, is tiny compared to the geometric computations that would be required to remove the extra links.

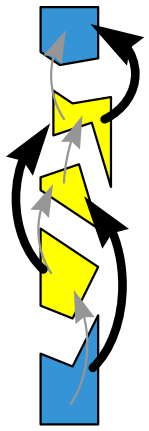


Figure 57. Extraneous CPG links

The links shown in bold add no additional information to the CPG, and are therefore not necessary. However, because they are redundant, they do not invalidate the CPG, or provide any additional constraints.

Intersection compacts (non-embedded)

Extra links between intersection compacts are produced by Condition 3 from Figure 46. This rule is needed to provide precedence links across surfaces that are shared by both A and B compacts. In cases where the surfaces are not coincident, however, this rule will produce an extraneous link.

Consider the intersection of two A compacts, a_2 and a_3, with two B compacts, b_1 and b_2. Let S_a be the surface separating a_2 and a_3 S_a, and S_b the surface separating b_1 and b_2. Assuming, without loss of generality, that $a_2 \rightarrow a_3$ and $b_1 \rightarrow b_2$, Condition 3 from Figure 46 will create the precedence link $a_2b_1 \rightarrow a_3b_2$. As shown in Figure 58, there are four possible configurations of S_a and S_b:

1. they can be coincident,
2. S_b is above S_a,
3. S_a is above S_b,
4. or they intersect.

Case 1 is the situation the rule is designed to accommodate, and the link $a_2b_1 \rightarrow a_3b_2$ is necessary.

In case 2, the intersection compact a_3b_1 lies between a_2b_1 and a_3b_2. In this case, the link $a_2b_1 \rightarrow a_3b_2$ is redundant with the precedence path $a_2b_1 \rightarrow a_3b_1 \rightarrow a_3b_2$. Since we know that a_2b_1 must be built before a_3b_1 and that a_3b_1 precedes a_3b_2, the statement $a_2b_1 \rightarrow a_3b_2$ adds no additional information. Similarly, in case 3, the link $a_2b_1 \rightarrow a_3b_2$ will be redundant with the precedence path $a_2b_1 \rightarrow a_2b_2 \rightarrow a_3b_2$.

Finally, in case 4, the compacts a_2b_1 and a_3b_2 will be tangent along a line or at a point. Again, the link $a_2b_1 \rightarrow a_3b_2$ is unnecessary, but it is redundant with the precedence paths $a_2b_1 \rightarrow a_3b_1 \rightarrow a_3b_2$ and $a_2b_1 \rightarrow a_2b_2 \rightarrow a_3b_2$.

Proof: As mentioned, in case 1 there are no unnecessary links. In case 2, S_b lies above S_a. Therefore, the intersection compact a_3b_1 must exist. Conditions 2 and 1 from Figure 46 create the links $a_2b_1 \rightarrow a_3b_1$ and $a_3b_1 \rightarrow a_3b_2$, respectively. The transitive property of precedence then gives $a_2b_1 \rightarrow a_3b_2$. Likewise, in case 3, there must be an intersection compact a_2b_2, and Figure 46 says that $a_2b_1 \rightarrow a_2b_2$ and $a_2b_2 \rightarrow a_3b_2$. Again, the transitive property yields $a_2b_1 \rightarrow a_3b_2$. Finally, in case 4, both a_2b_1 and a_2b_2 exist, and $a_3b_1 \rightarrow a_3b_2$ is redundant again. This result is unchanged if S_a and S_b cross more than once. Because there are no other possible configurations of S_a and S_b, this proof shows that $a_2b_1 \rightarrow a_3b_2$ is either needed in the CPG, or is redundant (and therefore harmless).

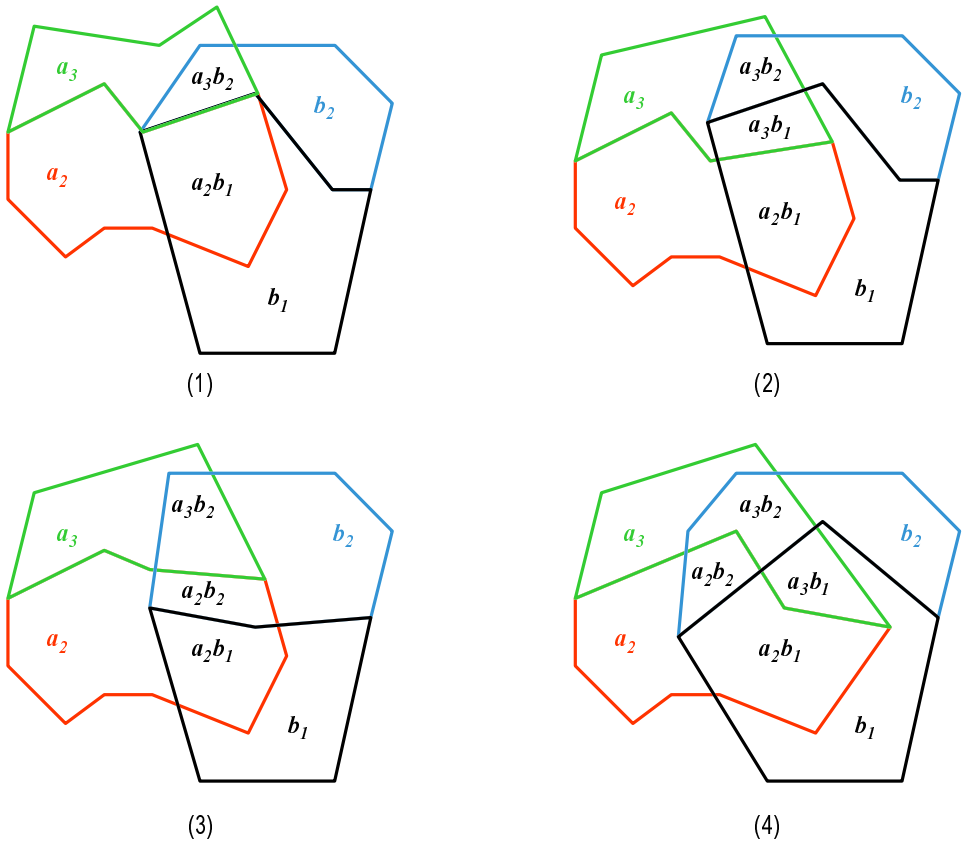


Figure 58. The four possibilities for a_2 and a_3 intersecting b_1 and b_2

In case 1, the precedence link $a_2b_1 \rightarrow a_3b_2$ is required in the result CPG. In the other three cases, the link is unnecessary, but will be redundant with the precedence paths $a_2b_1 \rightarrow a_3b_1 \rightarrow a_3b_2$ and $a_2b_1 \rightarrow a_2b_2 \rightarrow a_3b_2$.

Embedded components

Similarly, the rules in Figure 52 can produce unnecessary precedence links, but these links will always be redundant. As implied above, it is possible that a compact b_j will intersect both an embedded compact a_x and a regular compact a_i without the surface separating a_x from a_i passing through b_j. An example is shown in Figure 59, where a_x is an embedded component. Rule 1 in Figure 52 will create the precedence links $a_x \rightarrow a_ib_j$ and $a_x \rightarrow a_1b_j$. Because the surface S_a which separates a_x from a_i does not intersect b_j, the link $a_x \rightarrow a_ib_j$ should not be included in the result CPG. However, this link is redundant with the precedence path $a_x \rightarrow a_1b_j \rightarrow a_ib_j$.

Proof: It is given that $a_x \rightarrow a_i$. Since CPGs are acyclic, every point in a_i must lie above all points in a_x that have the same xy coordinates. Therefore, a ray in the growth direction from a point in $a_x \cap b_j$, must pass through a series of one or more compacts, a_1,

a_2, … a_i. (If the surface S_a intersects b_j, the series will have just one compact, a_i.) In other words, it will always be true that if S_a does not intersect b_j, there will be one or more compacts, a_1, a_2, … that are between a_x and a_i. Rule 1 in Figure 52 and the rules in Figure 46 will create the precedence path $a_x \rightarrow a_1 b_j \rightarrow a_2 b_j \rightarrow \ldots \rightarrow a_i b_j$. The link $a_x \rightarrow a_i b_j$ will be redundant with this precedence path, because the transitive property applies to precedence – if a_x precedes a_1 and $a_1 \ldots a_{i-1}$ precede a_i, then it is redundant to say that a_x precedes a_i.

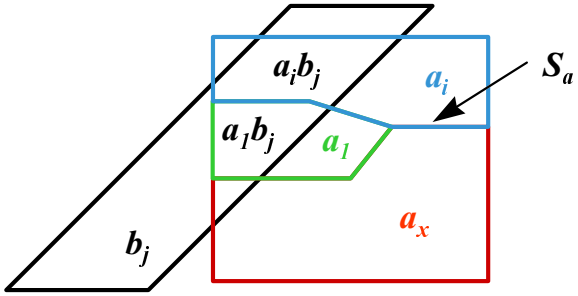


Figure 59. Special cases of embedded component precedence

The compact a_x is an embedded component. The surface S_a which separates a_x from a_i does not intersect b_j, and the link $a_x \rightarrow a_i b_j$ is not needed in the result CPG. However, it is redundant with $a_x \rightarrow a_1 b_j \rightarrow a_i b_j$.

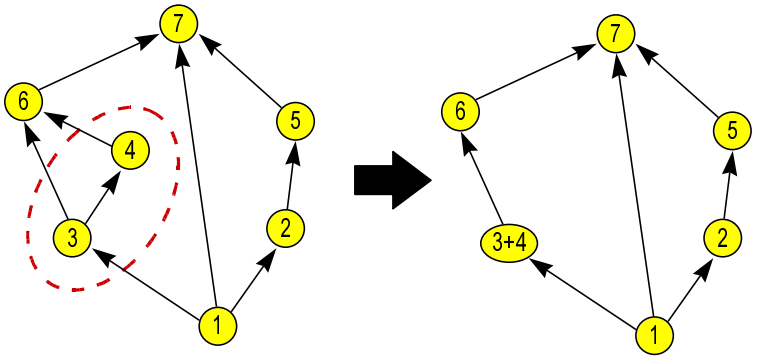


Figure 60. Compact graph simplification

In this example simplification operation, the two compacts 3 and 4 are combined together. All precedence links pointing into and out of the two compacts are maintained.

5.2 Simplification

Simplification refers to the process of combining two compacts in the precedence graph into a single compact, as shown in Figure 60. Because the merging algorithm in Chapter 4 produces a large number of intersection compacts, it is important to simplify the compact graph to keep the graph's complexity manageable. This section presents algorithms and rules to identify compacts that can legally be combined, and to suggest which of those possibilities are desirable.

5.2.1 Rules for legal simplifications

The basic rules for determining whether two compacts can be combined are that:

- the two compacts must have the same material type
- a compact a_1 can be combined with another compact a_2 if there is no precedence path from a_1 to a_2 with length greater than one.

In other words, a_1 and a_2 can be combined if they are the same material, and there is no compact, a_3, which must be built before a_1 and after a_2 (or vice-versa). The path-finding subroutine can be implemented with any of the standard graph-searching algorithms (for an example, see page 416 in **[Standish 95]**) which scale as $O(n)$, where n is the number of nodes in the graph. When compacts are combined together, all incoming and outgoing links in the CPG are maintained.

Application of these rules to identify simplification opportunities is straightforward. Most CPGs, however, can be simplified in more than one way. For example, Figure 61 shows how a CPG can be simplified in two different ways. Determining the best option often requires consideration of functional or aesthetic issues. For example, the simplification in Figure 61(c) may be undesirable for a structural part because the center compact is so small. In another application, Figure 61(b) may be unacceptable because it will leave unsightly seams on the top and bottom surfaces of the part. I believe it is impossible to state an *a priori* simplification algorithm which will always produce satisfactory results.

The next section presents an algorithm that allows the design by composition system to generate a "best guess" simplification after each merging operation. Designers will have the option of accepting or modifying this suggestion.

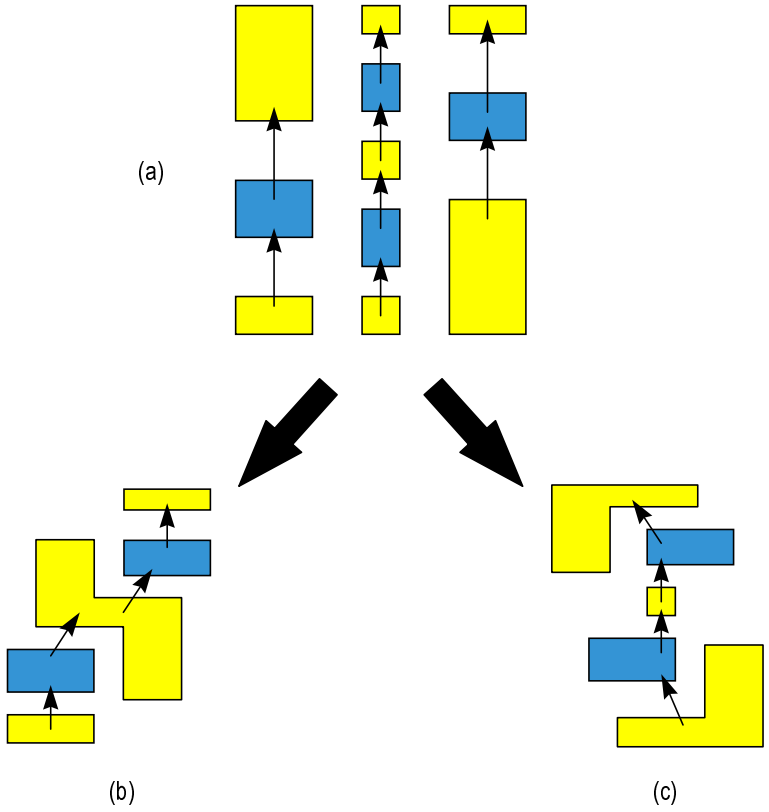


Figure 61. Two options for simplification

The compact graph in (a) can be simplified in different ways; two of the possibilities are shown in (b) and (c).

5.2.2 Choosing between simplification options

There are three rules that identify likely candidates for combination:

1. At the top and bottom of the CPG, all nodes of the same material
2. Parallel branches with the same material sequences
3. Horizontally adjacent compacts of the same material type

When these three rules are applied to the CPG shown in Figure 45, it produces the simplified CPG in Figure 62(c). However, as shown above, applying these rules will not always produce an optimal result. For example, when applied to the CPG in Figure 61, Rule 1 leads to the result in Figure 61(c).

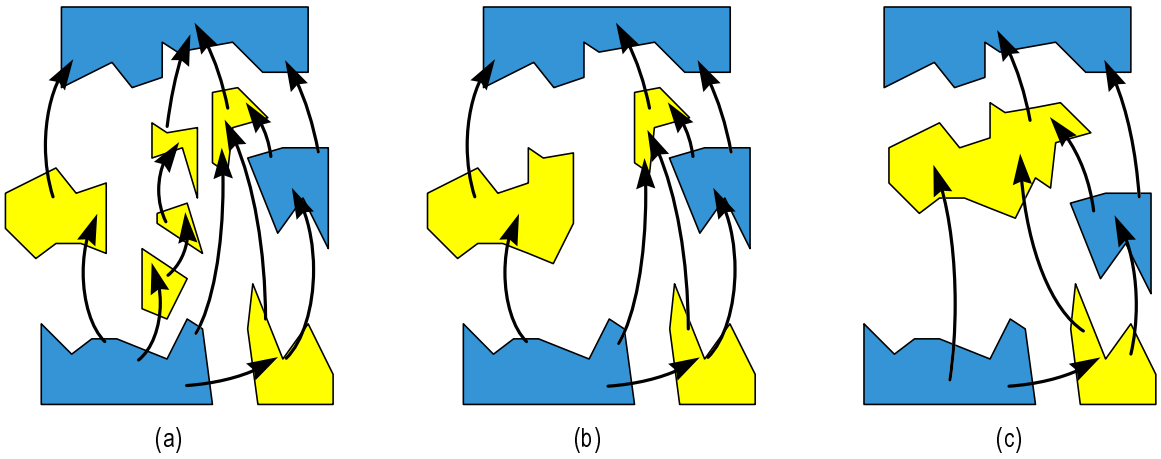


Figure 62. CPG simplification rules

The simplification rules have been applied to the CPG from Figure 45 on page 70. In (a), using Rule 1, the support compacts at the top and bottom have been combined. In (b), according to Rule 2, the two left branches of the CPG were combined; both had the sequence support-part-support. Finally, in (c), the top two part material compacts were combined because they are horizontally adjacent (separated by a vertical surface).

The solution I propose is to provide the design composition system with criteria for ranking simplification choices. By calculating a weighted "utility function" for each legal simplification option, the software can calculate a suggested simplification method for a given CPG. To provide maximum flexibility, the weights used in the utility function calculation should be designer-adjustable. Designers should also have the option of allowing automatic simplification, or reviewing and editing the software generated simplification suggestions. These options will let novice designers work without paying attention to simplification details, and will let experienced users control the CPGs for their parts.

An algorithm for generating the suggested simplification is shown in Figure 63. It works by checking each pair of compacts and evaluating the utility function to find the best simplification option. It is assumed that if the two compacts cannot legally be combined, the utility function will be zero. After evaluating all possibilities, the best simplification is performed. This process repeats until there are no legal simplification possibilities. At this point, the simplified CPG can be presented to the designer for approval or modification.

```
i=0                                          // initialize counter
j=0                                          // j will store number of best option
m=0                                          // m will store utility function of best option
repeat
    ∀ a1∈A                                   // loop through all pairs of compacts
        ∀ a2{a2∈A ∧ a2≠a1}
            i = i+1                          // increment counter
            ui = U(a1, a2)                   // calculate utility function
            if ui > m                        // is it a new maximum?
                j = i                        // store number of best option
                m = ui                       // store value of best utility function
    if j > 0                                 // is a simplification possible?
        perform simplification number j
until m=0                                    // repeat until no simplifications are legal
```

Figure 63. Simplification algorithm

This algorithm uses a utility function calculation to rank all possible simplification alternatives.

Three criteria for ranking which compacts should be combined are:

- **direction of shared surface normal**: Because of shrinkage, more reliable inter-compact bonds are formed across horizontal surfaces than vertical surfaces.
- **shared surface area**: Better adhesion occurs if compacts share a greater surface area.
- **compact volume**: Ensuring that all the deposition steps have relatively similar amounts of material will lead to better part quality because shrinkage effects are more evenly distributed.

These criteria are examples of the quantities that can be used to evaluate the utility function in Figure 63.

5.3 Algorithm Completeness

This section shows that the CPG algorithms produce valid output, and scale acceptably as the compact graphs grow more complex.

5.3.1 Closure proofs

The goal of the algorithms in this Chapter is to produce a valid CPG for the compact set which is created by the merging algorithm in Chapter 4. As stated in Section 3.6, the requirements of a valid CPG are:

1. The nodes of the CPG must be the compacts of a valid compact set
2. There must be a link in the CPG for every shared surface that is not parallel to the growth axis.
3. Links must be directed from the lower compact to the upper compact.
4. The CPG must be acyclic.

It has already been shown that the compact merging algorithm will produce a valid compact set, so the first requirement is satisfied. The proofs in this section show that the CPGs generated by the algorithms above meet the remaining requirements.

Proof of Condition 2

To satisfy this condition, we must show that there are no non-vertical surfaces created by any of the CPG algorithms which will not have corresponding precedence links.

Non-intersecting compacts: Non-intersecting compacts are copied to the result compact list with their CPG links intact. These compacts comprise a portion of the source compact graphs which has been cut by a vertical surface (the exterior of the other compact set). This cutting operation obviously does not introduce any new non-vertical surfaces in the non-intersecting compacts. Because all CPG links are maintained from the source list, and the source CPG contains a link for every non-vertical surface in the compact set, there will be no non-vertical surfaces in the non-intersecting compacts without a CPG. There may be "unnecessary" CPG links, precedence links which do not have a corresponding surface, but these links do not violate the conditions of a valid CPG.

Intersection compacts: As stated in Chapter 4, the surfaces of intersection compacts will always be surfaces from the compacts in A and B. Every non-vertical surface that separates two compacts, a_x and a_i, has a precedence link because we assume the source CPGs are valid. We can assume, without loss of generality, that $a_x \rightarrow a_i$. The surface which separates a_x and a_i will either divide a compact, b_y, into two parts, or will be coincident with a surface that separates two B compacts, b_y and b_j (assume $b_y \rightarrow b_j$). In the first case, the first condition in Figure 46 will create the link $a_x b_y \rightarrow a_i b_y$. In the second case, the third condition in Figure 46 will create the link $a_x b_y \rightarrow a_i b_j$. These two cases ensure that for every surface in A that has a precedence

link, the corresponding intersection compacts will have the correct precedence relationship. A parallel argument can be made for surfaces from the compacts in B.

Embedded components: The non-intersecting portion of an embedded component maintains its precedence links to the adjacent non-intersecting compacts. As with normal non-intersecting compacts, the presence of these CPG links, and the lack of any new surfaces on the embedded component ensure that there will be no non-vertical surfaces without a CPG link.

For the portions of an embedded component, a_x, that do intersect compacts from the other source set, there are again two cases to consider for each non vertical surface: either the surface lies within a compact, b_j, or it is coincident with the surface separating two compacts, b_j and b_k. In the first case, the other side of the surface will be the intersection compact $a_i b_j$. Depending on the direction of the surface normal, either $a_i \rightarrow a_x$ or $a_x \rightarrow a_i$, and the first part of the algorithm in Figure 56 will create the precedence link $a_i b_j \rightarrow a_x$ or $a_x \rightarrow a_i b_j$ (with case 1 or 3), respectively. Therefore, the result compact list will include the required precedence link for the surface.

In the second case, the given surface is coincident with the surface separating b_j from b_k. Assuming that b_j is on the a_x side of the surface, we know that the other side of the surface will be the intersection compact $a_i b_k$. If the given surface of a_x is downward facing, then the CPG for A will include the link $a_i \rightarrow a_x$, and the CPG for B will include the link $b_k \rightarrow b_j$; if it is upward facing, $a_x \rightarrow a_i$ and $b_j \rightarrow b_k$. These two cases correspond to cases 4 and 2 of the first part of the algorithm in Figure 56, which will create the link $a_i b_k \rightarrow a_x$ or $a_x \rightarrow a_i b_k$, respectively. Again, the correct precedence link for the given surface will be included in the result CPG.

Simplification: When two compacts are combined by a simplification operation, the surfaces they share are eliminated. All other surfaces from the two compacts are maintained in the result compact. No new surfaces are created. As stated above, the simplification operation maintains all precedence links, in both directions, from the two original compacts. Therefore, there will be no non-vertical surfaces in the simplified compact without appropriate precedence links.

Proof of Condition 3

As shown above, the result CPG contains links for all the non-vertical surfaces "inherited" from the source CPGs. None of the calculations in the compact merging algorithm will change the direction of the surface normal for these surfaces. Therefore, since the CPG algorithms maintain the sense of CPG links across surfaces, the links in the result CPG will point in the correct direction.

Proof of Condition 4

First, we must show that the CPG algorithms will not introduce cycles in the result CPG. To prove this statement is true for non-embedded intersection compacts, it is helpful to re-state the rule of Figure 46 as:

$$a_x b_y \rightarrow a_i b_j \rightarrow (a_x = a_i \wedge b_y \rightarrow b_j) \vee (b_y = b_j \wedge a_x \rightarrow a_i) \vee (a_x \rightarrow a_i \wedge b_y \rightarrow b_j).$$

(note that the bold arrow means "implies," not "precedes.") This statement can be simplified to

$$a_x b_y \rightarrow a_i b_j \rightarrow b_y \rightarrow b_j \vee a_x \rightarrow a_i. \tag{5}$$

Assume that the result CPG includes a cycle:

$$a_1 b_1 \rightarrow \ldots \rightarrow a_n b_m \rightarrow a_1 b_1.$$

The sequence $a_1 b_1 \rightarrow \ldots \rightarrow a_n b_m$ is a series of compacts where each precedence link obeys Equation 5. Application of the transitive property leads to the statement

$$(a_1 \rightarrow a_n \vee a_1 = a_n) \wedge (b_1 \rightarrow b_m \vee b_1 = b_m).$$

Applying Equation 5 to the last link in the cycle gives

$$a_n b_m \rightarrow a_1 b_1 \rightarrow b_m \rightarrow b_1 \vee a_n \rightarrow a_1$$
$$\therefore b_m \rightarrow b_1 \vee a_n \rightarrow a_1$$

This statement, however, when taken with the previous equation leads to eight possibilities,

$$\begin{aligned}
&(a_1 \rightarrow a_n \wedge b_1 \rightarrow b_m \wedge b_m \rightarrow b_1) \\
&\vee (a_1 = a_n \wedge b_1 \rightarrow b_m \wedge b_m \rightarrow b_1) \\
&\vee (a_1 \rightarrow a_n \wedge b_1 \rightarrow b_m \wedge a_n \rightarrow a_1) \\
&\vee (a_1 \rightarrow a_n \wedge b_1 = b_m \wedge a_n \rightarrow a_1) \\
&\vee (a_1 \rightarrow a_n \wedge b_1 = b_m \wedge b_m \rightarrow b_1) \\
&\vee (a_1 = a_n \wedge b_1 \rightarrow b_m \wedge a_n \rightarrow a_1) \\
&\vee (a_1 = a_n \wedge b_1 = b_m \wedge b_m \rightarrow b_1) \\
&\vee (a_1 = a_n \wedge b_1 = b_m \wedge a_n \rightarrow a_1),
\end{aligned}$$

which all violate the assumption that the source compact graphs are acyclic; in each possibility, either a cycle exists between two compacts, or a compact precedes itself.

Both of these conditions are illegal. Therefore, a cycle cannot exist in the result CPG if the source CPGs are valid.

A similar argument applies to compact graphs that include embedded components.

Next, we must prove that simplification does not introduce cycles. A simplification operation that produces a cycle is shown in Figure 64. Considering this example shows that for a simplification operation between two compacts to produce a cycle, there must be a precedence link that points out of one of them (in this case, compact 3), leads through some precedence path (through compacts 4 and 7), and then points into the other compact (compact 6). This statement, however, violates the condition for which compacts can legally be combined. Therefore, it is impossible for a legal simplification operation to produce a cycle in the CPG.

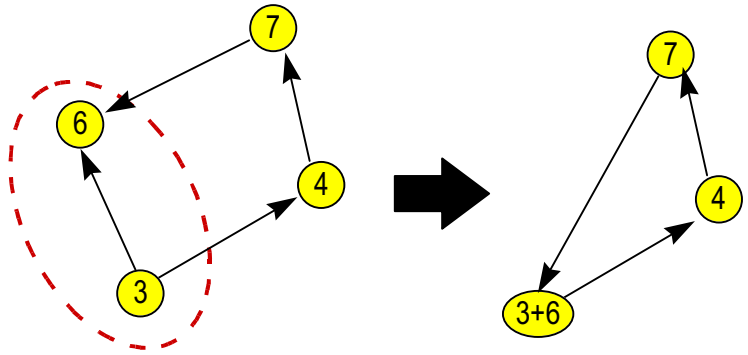


Figure 64. A simplification that creates a cycle

Compacts 3 and 6 are combined, producing a cycle. This simplification is illegal, however, because compacts 4 and 7 must be built after compact 3 and before compact 6.

5.3.2 Scalability

Like the compact set merging algorithm in the previous Chapter, the CPG algorithms in this Chapter all scale as polynomials of the number of compacts, and are therefore efficient.

Generating CPG links

For compact sets A and B with n and m compacts, respectively, there are at most nm intersection compacts in the result list, C. Therefore, the outermost loop of the algorithm in Figure 49 scales as $O(nm)$. The loop $\forall b_j\{b_j \in B \wedge b_y \rightarrow b_j\}$ will, in the worst

case, loop through all m compacts in B. In the worst case, b_j will intersect every compact in A, and will therefore have n children compacts. The inner loop will then run in time $O(n)$. As a result, the nested loops which evaluate Conditions 1 and 3 scale as $O(nm)$. Likewise, the loops that evaluate Condition 2 also scale as $O(nm)$. Because these two loops are nested within the outermost loop ($\forall a_x b_y \in C$), the total time required for the algorithm scales as $O(n^2m^2)$.

The CPG algorithm for embedded components (Figure 56) also runs in polynomial time. Each half of the algorithm contains four nested loops, two that loop through the A compacts, and two for the B compacts. As a result, the time required for this algorithm scales as $O(n^2m^2)$. Combining this result with the $O(n^2m^2)$ scaling law for non-embedded compacts shows that the total time required for generating CPG links is $O(n^2m^2)$ (because $O(n^2m^2)=O(2n^2m^2)$).

Simplification

The simplification algorithm is shown in Figure 63. Assuming the time required for calculating the utility function is constant, generating the ranking of simplification alternatives scales as $O(n^2)$, where n is the number of compacts in the CPG. In the worst case, the algorithm will perform $n-1$ simplifications – after that many, the CPG will only have one node. Therefore, the total time required for simplification scales as $O(n^3)$.

6. Design by Composition

The definitions and algorithms from the previous chapters can form the basis for a CAD environment. In a design by composition CAD system, designers build parts from a library of design "primitives" using Boolean operations – add (union), intersect, subtract, and exclusive-or. Each primitive has an associated compact graph which is a high-level representation of the manufacturing plan to produce it. When the designer combines two primitives, the design by composition software will merge the primitives' compact graphs to create a manufacturing plan for the new design.

This chapter describes how a design by composition CAD system operates, and what it is like to use.

6.1 Compact graphs

While the design by composition approach will work with arbitrary 3D geometry, it does require a valid compact graph for each primitive element. The geometry of the compacts can be used to determine machine tool paths, while the links in the CPG represent the possible fabrication sequences. Because the CPG is a least-commitment process plan, it allows both the designer and the manufacturer to adjust the fabrication process to minimize cost, control part quality, or to accommodate machine availability restrictions.

Compact graphs for primitives can come from four sources:

- algorithmic generation by the design by composition software,
- retrieval from a design library,

- merging of existing primitives, or
- manual creation by the designer.

Design by composition software will automatically create the compact graphs for extrusions (simple shapes), allowing designers to create a primitive by extruding any closed two dimensional curve (see Section 3.5.2). Additional primitives, many of which will represent embedded components, such as sensors and bearings, will be provided in design libraries. More complex primitives may be created by combining simple shapes and library elements. The fourth source for compact graphs is manual creation by the designer. This option, which requires knowledge of the manufacturing process, will probably only be used by experienced designers.

When the designer combines two primitives, the design composition system combines the corresponding compact graphs, using the algorithms presented in Chapters 4 and 5.

6.2 Restrictions on design flexibility

In a standard solid-modeling environment, designers are free to create complex shapes by sweep, extrusion, rotation, and Boolean operations. Users of design by composition are limited to a subset of these operations; primitives can be created only by extrusion and Boolean operations, and can be manipulated with the operations described in Section 3.7.2. This list includes most common geometric transformations, with the exception of rotations about the X and Y axes. Because each primitive has a predefined growth direction, it is illegal to perform any operation that changes the direction its Z-axis. For example, a primitive that represents an embedded sensor may be stored in the design library twice: once in a horizontal orientation and once vertically. A designer who wants to use this primitive can choose between these two orientations, but cannot use the sensor at a 45° angle. In the future, when design by composition is integrated with a newer generation of decomposition software, this limitation will evaporate, as described in Section 9.3.

6.3 Benefits of this approach

As outlined in Section 1.2.3, the benefits offered by the design composition approach include:

- manufacturability feedback during design-time
- a greater degree of automation for new processes like SDM
- easy creation of designs with embedded components
- easy creation of design libraries

- enhanced control of the manufacturing plan, and thereby over part quality
- fewer computational bottlenecks

These advantages will be especially attractive for designers who are unfamiliar with a new manufacturing process and wish to explore its capabilities.

6.3.1 Immediate manufacturability feedback

A cursory evaluation of SDM and other layered manufacturing processes suggests they can build parts without any geometric restrictions. A more detailed understanding of the process, usually based on fabrication experience, reveals that layered techniques do have manufacturing constraints, some of which are non-intuitive and difficult to evaluate based on the complete part geometry. For example, one limitation of many layered techniques is the limited strength of inter-layer bonds, which leads to a "minimum area of overlap" constraint on part geometry: adjacent layers must have a certain amount of shared area to form a good bond.

A primary advantage of the composition approach is that it permits the CAD system to automatically assess manufacturability at design-time, and to alert the designer immediately when a new feature produces an illegal or expensive design. This feedback allows a designer who is unfamiliar with process capabilities to create manufacturable parts. The compact graph can also be used to provide a manufacturing preview which shows what the final part will look like. (Note, these features have not been implemented in the prototype design by composition system.)

Manufacturability rules for compacts

Manufacturability rules for SDM and other SFF processes can be expressed as constraints on compacts. Unlike traditional manufacturing techniques, constraints in SFF processes are functions of local geometry and topology; the manufacturability, accuracy, and surface finish of a feature depend only on the geometry and material type of its own and adjacent compacts. Examples of compact manufacturability rules are shown in Figure 65.

The current design by composition implementation (see Chapter 7 and the Appendix) does not contain manufacturability feedback. Once design rules for SDM and other processes have been codified, however, it is a "mere matter of programming" **[Brooks 90]** to verify the validity of the compacts in a design by composition primitive. After each primitive merging operation, the system could automatically check the new compacts against the manufacturability rules, and notify the designer of any problems.

I expect that completing this work will be a research project for future graduate students at the Center for Design Research.

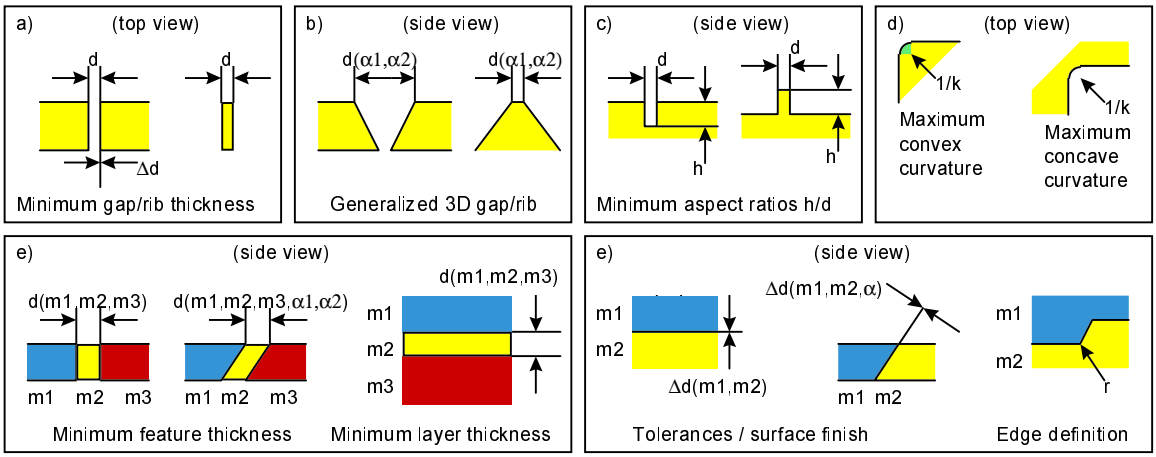


Figure 65. Example design rules for SDM

(a) and (b) Minimum gap and rib thickness. (c) minimum aspect ratio. (d) maximum curvature. (e) minimum feature and layer thickness. (f) surface finish and tolerances. (from **[Cutkosky, et al. 96]**)

Manufacturing preview

Because Shape Deposition Manufacturing uses CNC milling for most shaping operations, part corners are often filleted by the cutter radius. This problem is familiar to designers of machined components, who know that internal corners will be filleted, while external corners can be sharp. Mold designers use opposite constraints because internal corners of the mold create external corners of the molded part. SDM, however, creates part features by both machining and molding operations, so it is often difficult to predict which corners will be filleted. For example, Figure 66 shows the CAD model and manufactured mechanism for a simple robot leg. As shown in Figure 67, small protuberances are left on the part because of the size of the CNC cutting tool. For this part, it is possible to manufacture it without these defects, but only by (1) making substantial enhancements to the planning software, or (2) relying heavily on manual intervention by the manufacturer.

A design by composition CAD system could use the geometry of the compacts to produce a "manufacturing preview" of the current design. A sequence of Boolean operations and offsetting functions can simulate the manufacturing process to show the

designer where there will be cutter radius fillets and other defects on the finished part[2]. Again, this feature is not yet implemented in the prototype system.

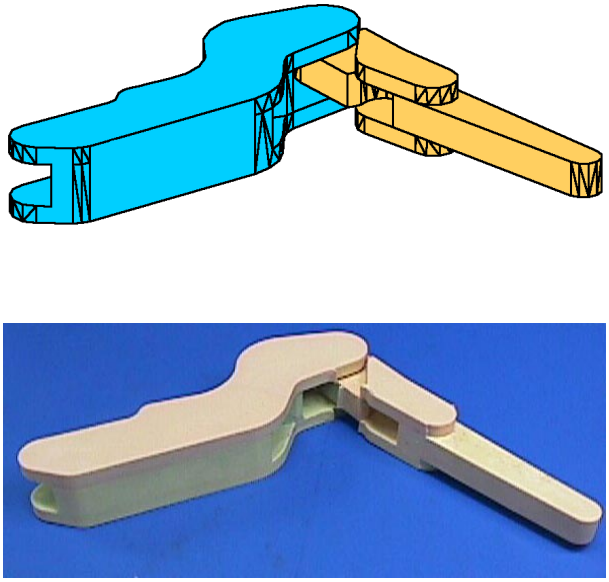


Figure 66. A simple robot leg built with SDM

This robot leg, which was built as a pre-assembled mechanism, contains fillets from the CNC machining operation that interfere with the device's function. The top picture shows the CAD model for the mechanism, the bottom photograph shows the result of the SDM process.

6.3.2 Level of automation

An additional benefit of design composition is that currently it is the most automated way to produce SDM process plans. As shown in Figure 14 on page 33, for the decomposition approach automated algorithms for part orientation and compact splitting are still under development. Today, these operations still require manual intervention by the designer or SFF operator. Users of design by composition, on the other hand, avoid these problems as shown in Figure 68. Design by composition provides design data directly to the path planning algorithms, where the level of automation is more developed.

[2] To simulate CNC cutting, the 2D contour of each layer can be offset away from the part by one cutter radius (this step produces the cutter path), then offsetting this curve towards the part, rounding internal corners, to produce the profile of the machined surface. Finally, these curves can be extruded in the Z-direction and unioned together to produce the manufacturing preview.

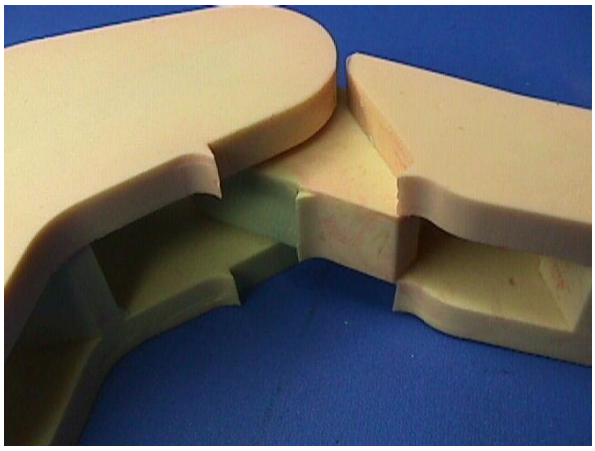


Figure 67. Closeup of cutter radius fillets

The small triangular protuberances on the parts are formed by the radius of the CNC cutting tool. In this case, they will restrict the range of motion of the mechanism.

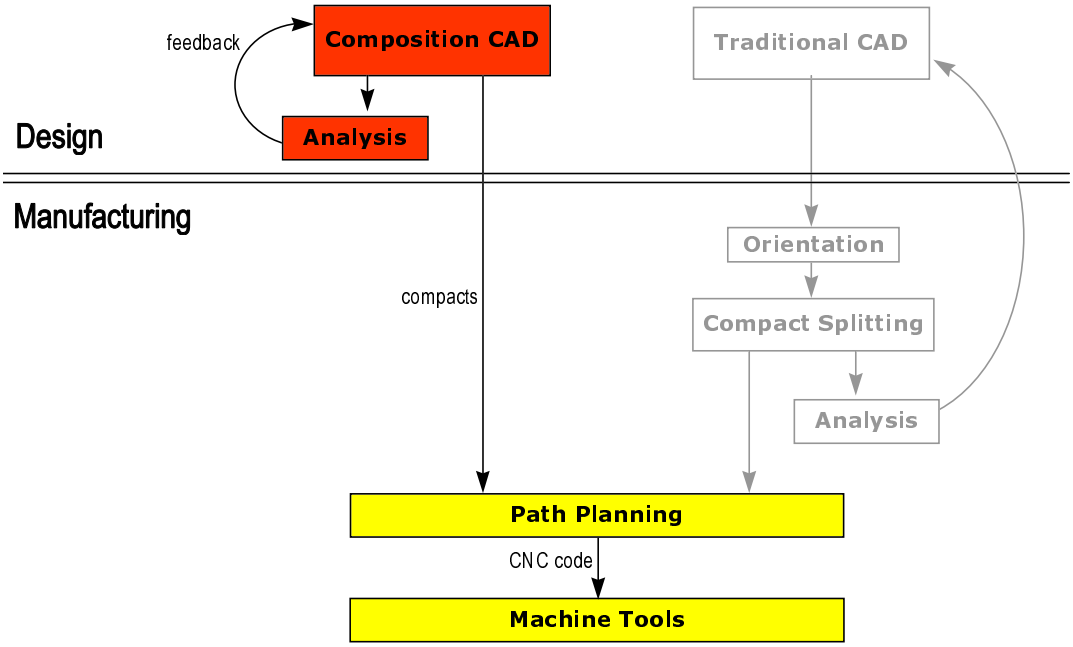


Figure 68. Design by composition design-manufacturing interface

Contrasting this picture with Figure 9, on page 28, illustrates how the design by composition approach, by maintaining a manufacturing plan in the design system, allows faster feedback to the designer and avoids the computationally difficult (and not-yet fully automated) compact-splitting operation.

6.3.3 Mechanisms with embedded components

Designing a pre-assembled mechanism with embedded components (both mechanical and electronic) should permit smaller, more complex, and more robust devices. The design composition method presented here is especially suited for designing parts and mechanisms with embedded components which are stored as library elements (In addition to the examples below, see Chapter 8).

Figure 69 shows a pin-joint mechanism that was built with SDM. The manufacturing process for this mechanism is:

1. The yellow urethane part is built with the SDM process.
2. The bottom half of the black epoxy part is built, embedded in a wax support material.
3. Teflon bushings, a steel shaft, and the yellow part are positioned in the mechanism.
4. Additional support material is deposited around the embedded components.
5. The top half of the black part is deposited and shaped.

In this way, the complete mechanism is built so that it cannot be disassembled. (Actually, it can be disassembled, but only once.) There is no need for fasteners, retaining rings, or other assembly-related features. When this approach is applied to more complex structures, it will allow more compact and more reliable mechanisms.

Figure 69. A pin joint with embedded components

This pre-assembled mechanism was built in the Stanford University SDM facility. The mechanism consists of two plastic parts (one epoxy and one urethane) Teflon bushings, and a steel shaft. It was fabricated with all the components in place, with no assembly operations.

Figure 70 shows a robot leg built by Jorge Cham at Stanford University. This leg was designed with the prototype design by composition system, and built with SDM at Stanford. The leg contains steel shafts and Teflon bushings like the pin joint in Figure 69. It also contains two small DC motors and an electrical connector. All wires and electrical connections are encapsulated in the structural part. The motors twist two Spectra (linearized polyethelene) "tendons" to actuate the leg. As the tendons twist up, they shorten. The two motors act as opposing extensor-flexor actuators. Designers using the decomposition approach have not produced designs with embedded components that are this complex. The experience of Cham and other users of the prototype composition system suggests that designers who are new to SDM can begin using embedded-components sooner, and more effectively, than decomposition users.

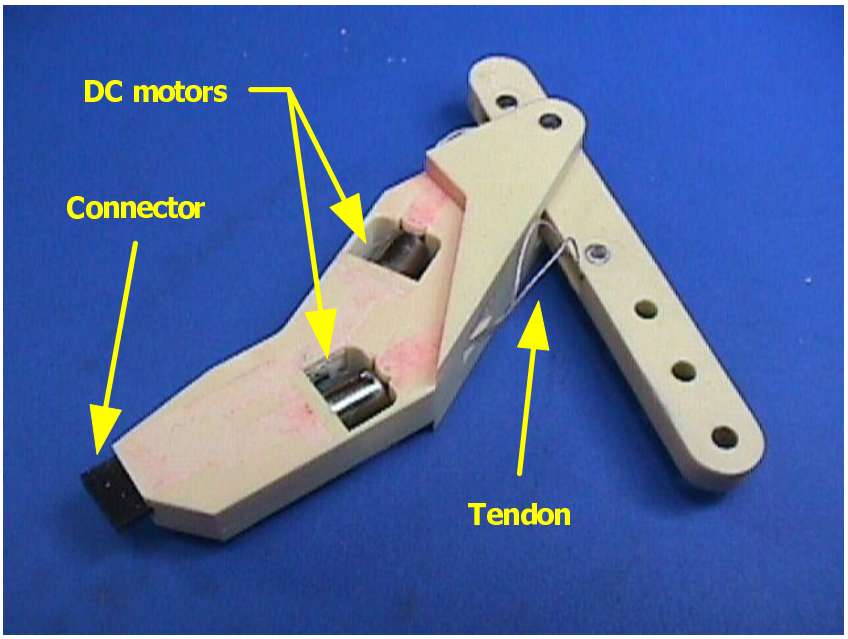


Figure 70. A robot leg with embedded DC motors

6.3.4 Design libraries

Primitive based design also facilitates the generation of design libraries which can disseminate part and feature designs, and facilitate design element re-use. The composition approach allows design libraries to contain three dimensional shapes of arbitrary complexity. Library elements can be individual part features (holes or bosses), complete parts (pistons or mechanism links), integrated assemblies, or any combination. Any design composed from library primitives will have a CAD

representation for the compact graph required for its manufacture, and can therefore be stored in the design library as a primitive for future work.

Although the prototype composition system (see Chapter 7 and the Appendix) does not implement library functions beyond what is provided by AutoCAD, the re-use of primitives for shafts and bushings in the parts shown in Figure 69 (page 97) and Figure 78 (page 110) illustrates this capability.

6.3.5 Enhanced control over manufacturing

For experienced designers, design by composition will offer greater control over process planning decisions than the decomposition approach. As mentioned earlier, the quality of SDM parts is greatly affected by the manufacturing plan selected. Designers with stringent requirements will be able to modify the compact graph for their designs to produce optimal process plans.

6.3.6 Computational burden

This approach removes some of the computational burden of process planning from the manufacturer. As manufacturing processes grow more complex, process planning increasingly becomes a bottleneck. Distributing the planning tasks over a wide variety of designers will avoid computational problems.

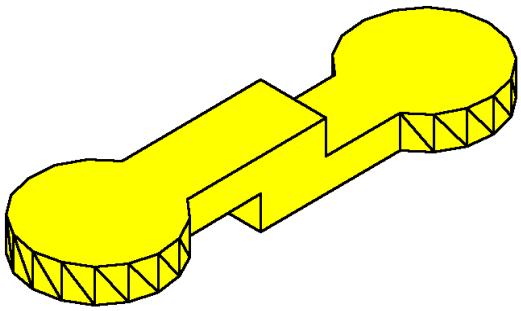


Figure 71. An example design

The designer can build this shape using cylinder and rectangular prism primitives.

6.4 Example scenario

This section provides an example of the design by composition approach in operation. (Note that the manufacturability feedback and manufacturing preview features are not yet implemented.)

The designer begins by loading the design by composition plug-in into his CAD environment. Because he knows the part will be fabricated at the Stanford SDM facility, he downloads the design rules for that shop over the Internet, and selects the urethane part and wax support material combination.

Using the cylindrical and rectangular box primitives provided by the plug-in, he creates the shape in Figure 71. The user interface for creating, moving, and copying these primitives is the same as the regular solid modeling interface in the CAD environment, so it is easy to start using the system.

Next, the designer creates holes in the part by creating a circular array of cylindrical primitives and subtracting them from the current design, as shown in Figure 72. During this operation, the design by composition software determines that the holes are too small to be manufactured with Stanford's SDM process because they will be formed by machining support material posts and casting the part material around them. The posts required for these holes are small enough that the cutting forces during machining will most likely break them off. A warning appears in the CAD window:

Unable to manufacture this design:
there are features smaller than 1/32"

The designer is presented with the options of continuing the design, or reversing the last operation. He chooses the latter option, removes the cylindrical primitives, and makes a new array of larger cylinders. This time, when he subtracts the cylinders from the design, there are no undersized features. The designer continues adding and subtracting primitives until he has created the design shown in Figure 73.

Next, the designer selects the Manufacturing Preview option from the design by composition menu. This produces the output shown in Figure 74. It indicates that all of the internal corners in the design will be filleted with the CNC cutter radius. At first, the designer is surprised by this defect, "I thought SDM could make any geometry." After reviewing the description of SDM on the World-Wide Web, he realizes the source of the fillets, and understands their presence in his design. In this case, they do not affect the functionality of the part, so he selects "Send to Manufacturer" from the design by composition menu to submit the design to the manufacturer for a quotation.

Later that day, the designer receives an email from the manufacturer with a price and delivery date for his part. He replies that the quotation is acceptable. A few days later, FedEx delivers the completed part to his office.

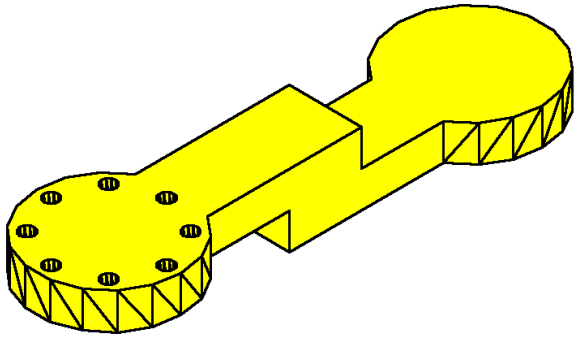


Figure 72. The design with an array of holes

The holes are created by subtracting a circular array of cylindrical primitives from the design.

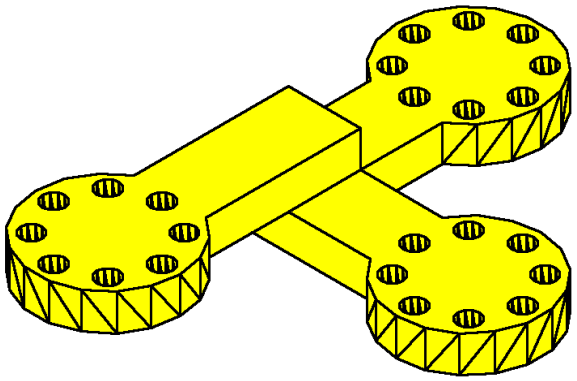


Figure 73. The completed design

The designer uses additional primitive merging operations to create this finished part design

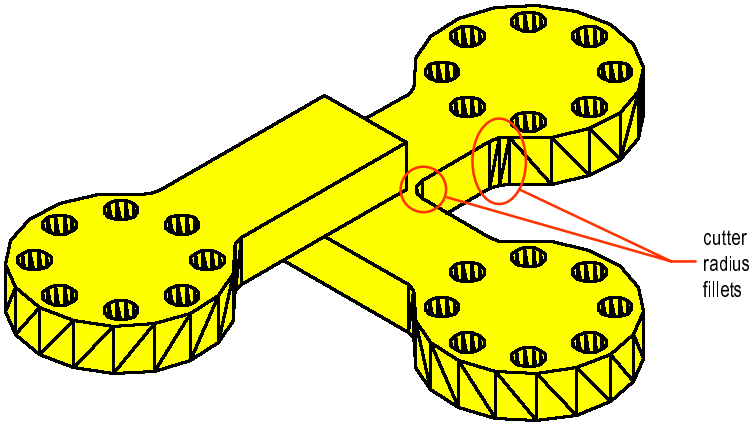


Figure 74. Manufacturing preview of the design

The preview feature shows the designer where there will be cutter radius fillets on the design – in this case, on all the internal corners.

7. Implementation

With other members of the Rapid Prototyping Laboratory and Center for Design Research at Stanford University, I have implemented a prototype design by composition system for Stanford's SDM facility. While this system does not include all the functionality described in this dissertation, it has been sufficient to test theories and algorithms, demonstrate the design by composition paradigm, and enable designers who are new to SFF to build parts with SDM.

The design composition implementation uses different layers of the CAD drawing to hold the current design and the compact graphs. At the designer's discretion, the process plan layers can be hidden or examined. With these layers invisible, the design system functions as a slightly restricted version of the standard solid modeling interface. By making the compact layers visible, a curious or expert designer can view and modify the compacts for precise control over the manufacturing plan.

7.1 Design system - AutoCAD plug-in

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 sets of both primitives are then combined using the algorithms described in Chapters 4 and 5. Both addition and subtraction operations are supported (intersection and exclusive-or operations 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.

7.2 Compact Precedence

The current version of the design by composition plug-in does not use precedence graphs to represent the possible orderings of compacts. Instead, it uses an algorithm which produces a single, arbitrary order for the compacts created by a merging operation. Because this algorithm produces a single order, the simplification operation is much easier. Instead of evaluating several possible alternatives, the simplification step in the current implementation combines all sequential compacts of the same material type.

The disadvantages of the single-order algorithm are that it often produces an order which is not very good. In particular, the way it combines compacts makes most of the cutting surfaces between compacts vertical, as shown in Figure 75(a). This process plan is not desirable because shrinkage effects cause horizontal positioning errors in SDM parts, especially for urethane SDM parts built with wax support structures. With the materials used in SDM today, the melting temperature of the wax support material is high enough to soften the urethane part material. When the wax shrinks, it deforms the existing part material, usually by moving the urethane horizontally. Because subsequent compacts are built in the correct absolute positions, there are alignment errors between compacts. As shown in Figure 75(c), these errors are often fatal for parts with vertical cutting surfaces. In contrast, the compact set in Figure 75(b) with a horizontal cutting surface, will lead to the fabricated part shown in Figure 75(d). In this case, although the part is inaccurate, it is still connected.

A work-around for this problem is included in the design by composition plug-in; after any merging operation, the designer can manually re-order the compacts. Future versions of the software will use precedence graphs as described in Chapter 5 with optional manual intervention.

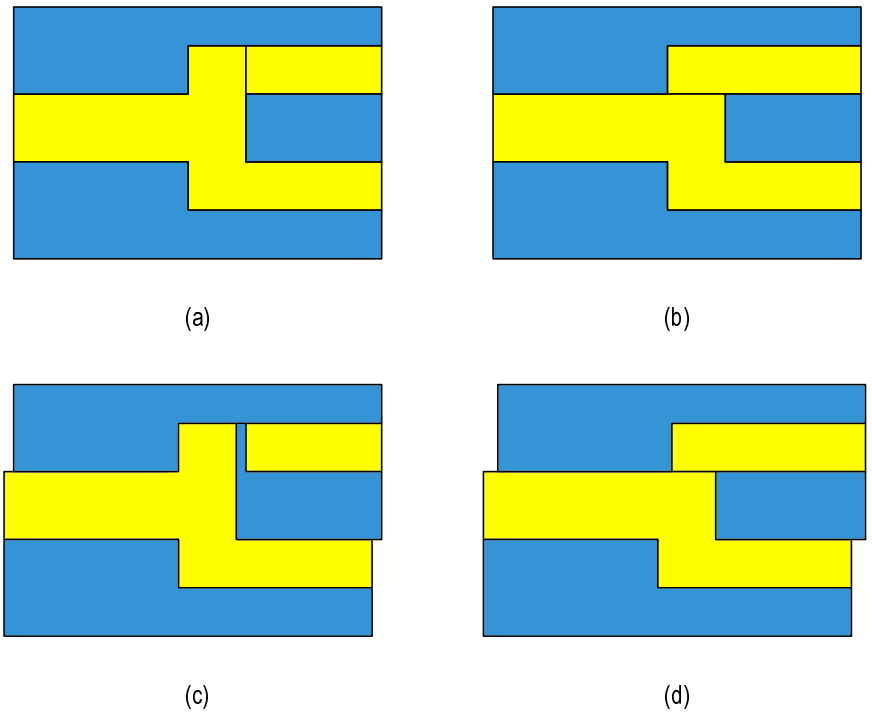


Figure 75. The effects of compact order and shrinkage

The compact order produced by the design by composition implementation will normally produce a simplified compact set like (a) where the surfaces that separate part material compacts are vertical. Compacts separated by vertical surfaces are susceptible to being "disconnected" by shrinkage effects as shown in (c). A better alternative is the compact set in (b) where the cutting surface is horizontal, and shrinkage effects are less severe (d).

7.3 Manufacturing interface

When the user has completed a design, the design by composition plug-in will automatically produce the .SAT geometry and planner instruction files necessary for CNC planning. These files are then transferred by FTP to computers at the manufacturing site. The current implementation works with the ACIS path-planning software that was developed as part of the original decomposition system.

Geometry is transmitted to the path planning software in the form of Single Step Geometries, or SSGs. At each stage in the fabrication sequence, the SSG is the union of the current compact with all prior compacts. The SSGs for the compacts in Figure 75(a) are shown in Figure 76. Each SSG contains the complete geometry of the SDM artifact at the end of the current shaping operation, and is therefore used to generate the CNC cutting code. The manufacturer uses the SDM path planning software to generate a

series of CNC cutting files for the shaping operations, and checks the CNC files with a simple path preview program (an example of the output from this program is shown in Figure 90 on page 122). Usually, only about half of the CNC files generated are necessary for fabrication; the rest are redundant and can be skipped.

Currently, for parts designed with the design by composition prototype, deposition is performed manually, as shown in Figure 15 on page 34. Because the polymer and wax materials are deposited in liquid form, the deposition process requires building a "dam" to contain the liquid, and then pouring a sufficient quantity of material to reach the proper deposition depth. The dams are usually constructed out of plasticine, or as is shown in Figure 15, out of duct tape.

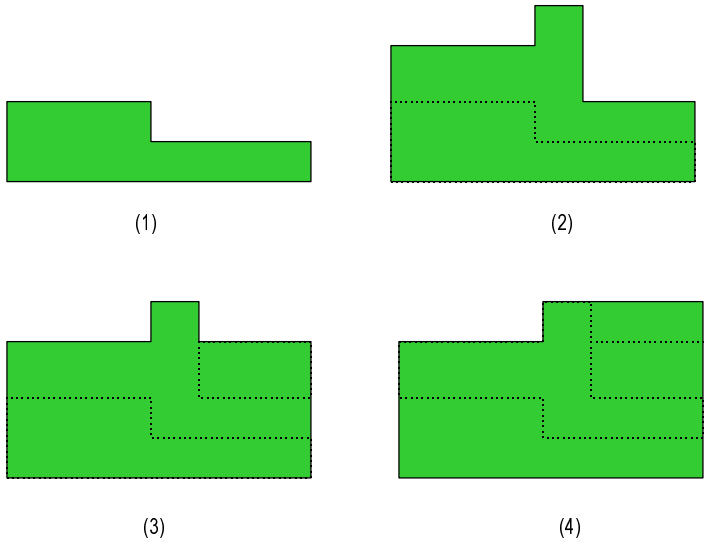


Figure 76. Single Step Geometries (SSGs) for Figure 75(a)

At each stage in the fabrication sequence, the SSG is the union of the current compact with all prior compacts. Dotted lines show the original compact boundaries.

7.4 Lessons learned

Although the prototype design by composition system does not implement all of the material presented in this dissertation, it has been sufficient to test the compact merging algorithms. More importantly, graduate students at the Center for Design Research have begun using the design by composition system to design and build parts with SDM.

7.4.1 Algorithm development

The prototype design by composition CAD system has been invaluable through the course of this research. The primary advantage of this software has been the ability to quickly test different compact merging algorithms on a wide range of example geometry. Working with pen and paper, it is difficult to consider enough scenarios to provide a thorough analysis, and even harder to capture 3D geometric effects.

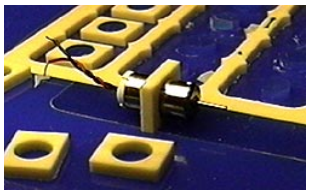
7.4.2 Use by novice designers

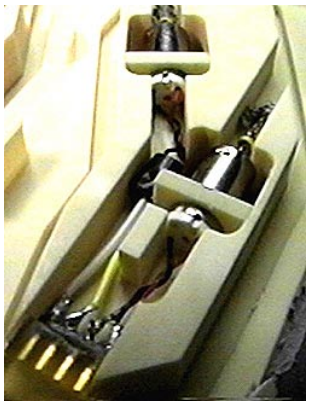
The design by composition software is now being used by several graduate students to design and fabricate SDM parts. These students have been able to jump in and make parts without a thorough understanding of the subtleties of SDM processing. Anecdotal evidence shows that these students have been able to design and fabricate complex SDM parts much sooner than students who began using SDM with the older ACIS decomposition system.

Furthermore, the experiments performed by these students have explored a wider design space than previous SDM users. As was seen with VLSI design, making the fabrication process available to a larger community of designers rapidly increases the pace of design innovation. **[Losleben 97]**. One example of this type of innovation is the use of spacers shown in Figure 77. These pictures are from the fabrication of the leg with embedded motors shown in Figure 70 on page 98. In this design, the motors are encased half in support material and half in part material. The front end of the motor, including the shaft and cable tendon are encased in a removable support (in this case, soap) which will leave them free to move when fabrication is complete. The stationary end of the motor and the electrical connections are encased in urethane to protect them from damage. The spacers are used to separate these two regions. Figure 70(a) shows a motor with the spacers; (b) shows the motors and spacers positioned in the leg mechanism; (c) shows the front half of the motors in the support material.

This use of spacers to control deposition was invented by Jorge Cham at Stanford's Center for Design Research while he was still new to the SDM process. The high degree of automation in planning and fabrication with design by composition was essential for him to tackle an ambitious embedding project like this with little SDM experience.

(a)





(b)

(c)

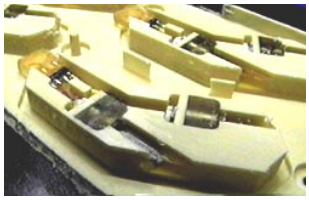


Figure 77. Jorge Cham's spacer invention

Often, embedded components, like these electric motors, must be embedded half in part material and half in support material. The spaces shown in (a) are assembled with the motors before they are encapsulated (b). The photograph in (c) shows the motors with the front end encased in soap support material.

8. Fabrication Example

To test and demonstrate the operation of the prototype design by composition implementation, I designed and fabricated an example mechanism, the simple one-joint robot leg shown in Figure 78. The goal for this mechanism was to produce a leg mechanism with an integrated actuator and sensors. Although a similar artifact could be built with conventional techniques, this leg demonstrates the ability of design by composition and SDM to create pre-assembled mechanisms with embedded components. As designers explore the possibilities created by this capability, their artifacts will look less and less like traditionally manufactured parts.

8.1 Comparison to decomposition approach

As shown below in Figure 82, the fabrication sequences produced by the design by composition system contain 7 compacts for the complete leg assembly. (The process plan produced automatically by the merging algorithm contained 8 compacts; manually changing the order of the compacts reduced the total number to 7.) The corresponding process plan created by the automatic ACIS-based decomposer requires 43 compacts, and would require extensive manual modifications, including hand-editing of G-code, to permit the use of embedded components. The greater number of compacts required by the decomposition process plan would have drastically increased manufacturing time and cost. Furthermore, the parts produced would be much weaker because they would contain many more layers.

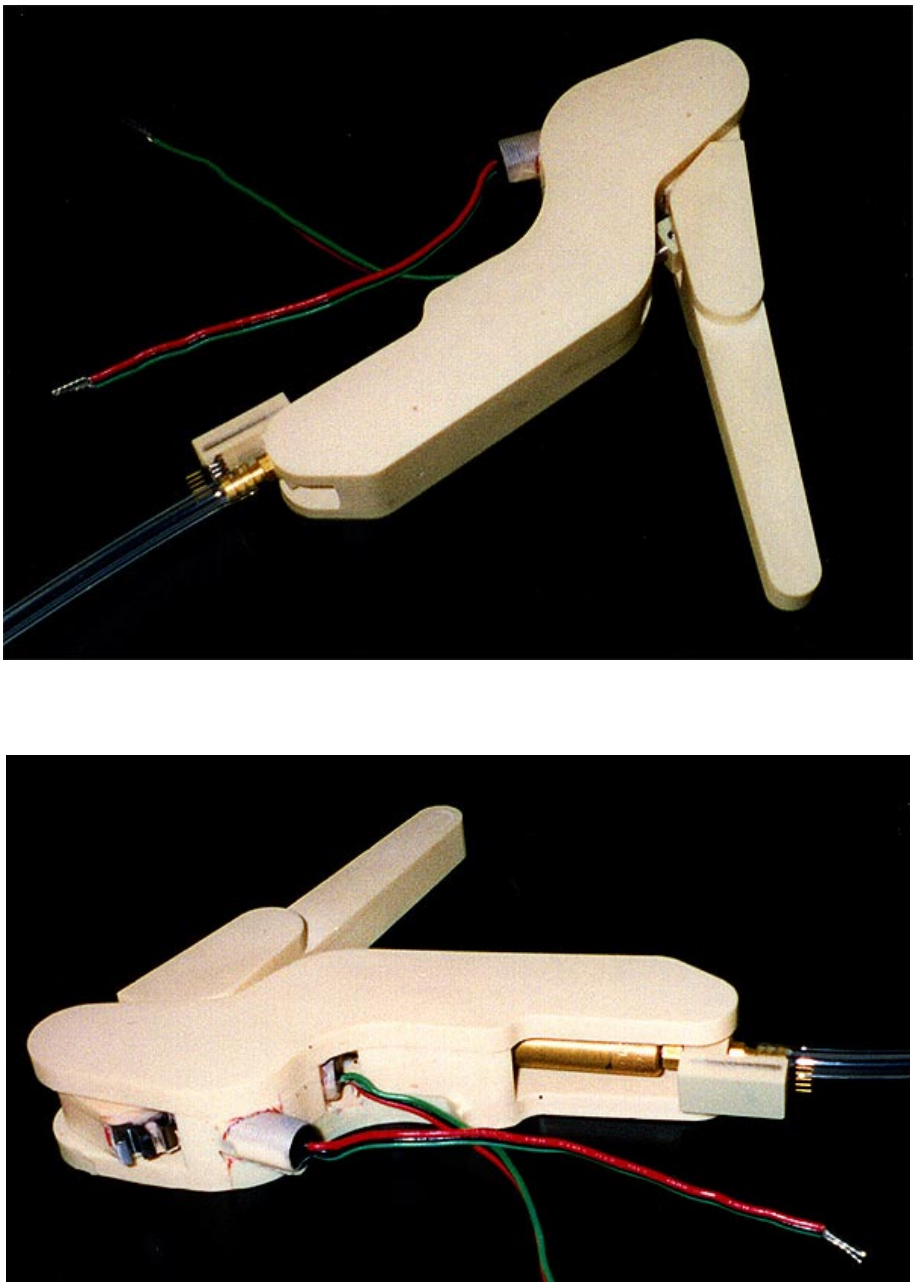


Figure 78. Robot leg with integrated sensors and actuator

This leg contains two Hall Effect gear tooth sensors, a pneumatic cylinder, and an integrated pressure sensor and amplifier circuit. It was designed with the prototype design by composition system and built with SDM at Stanford University.

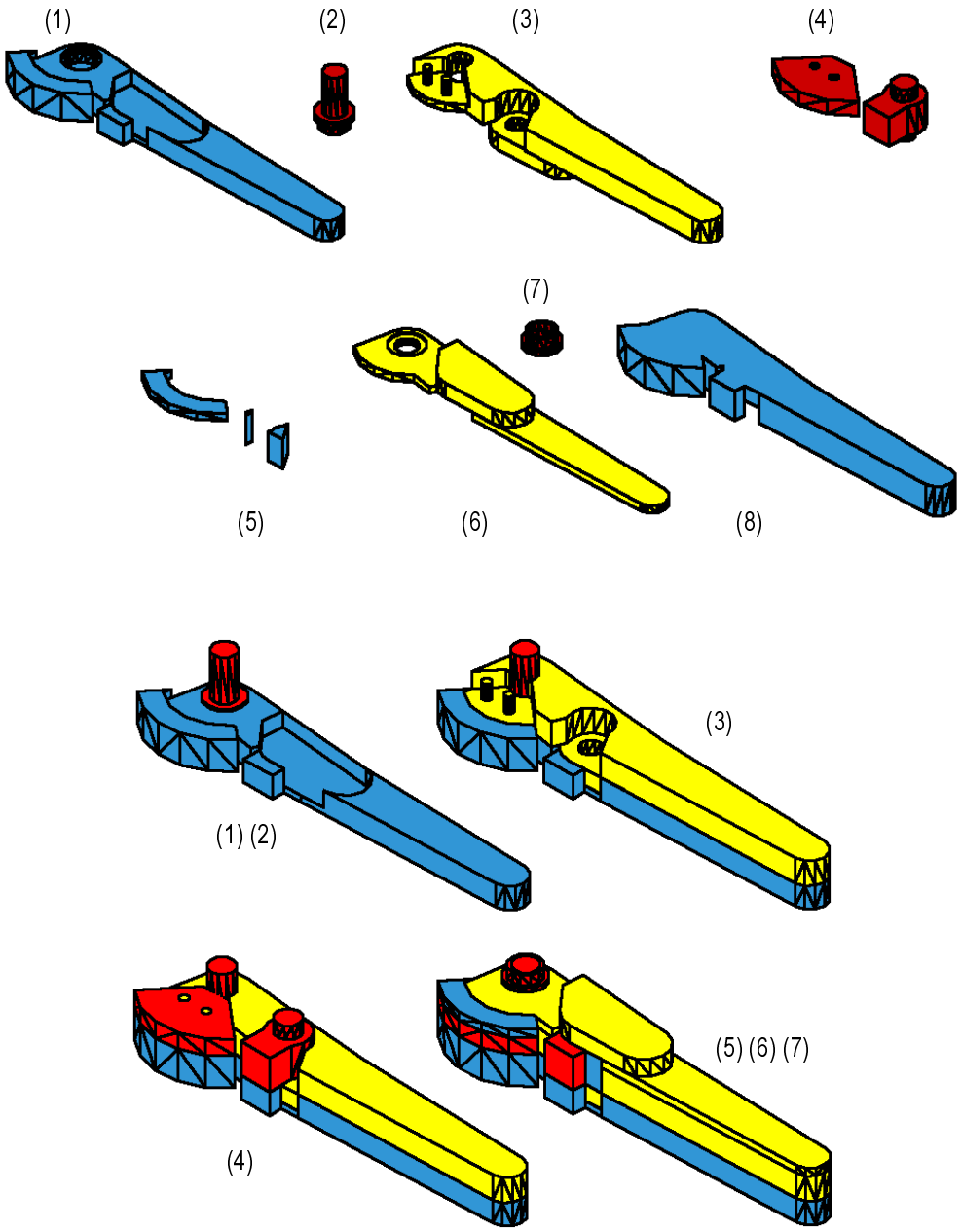


Figure 79. Compacts and fabrication sequence of lower link

Compacts 1, 5, and 8 are support material, compacts 3 and 6 are the part material, and numbers 2, 4, and 7 are embedded components.

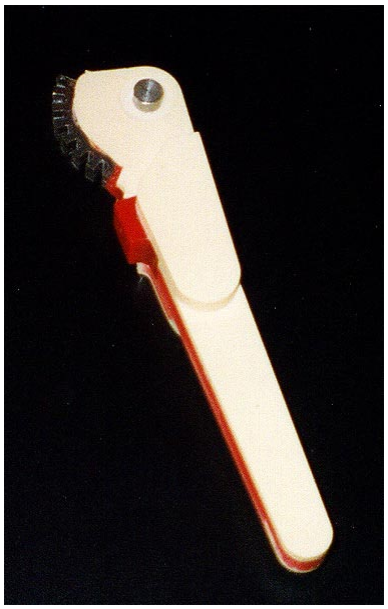


Figure 80. Photograph of lower link

This is the completed link with the embedded gear tooth and shaft. Support material still encloses the piston rod connector.

8.2 Fabrication sequence

This mechanism was built in two stages. First, the lower leg link (Figure 80) was fabricated with several embedded components:

- a steel gear target for the position sensors,
- Teflon bushings and a steel shaft for the leg's revolute joint,
- a pre-fabricated urethane part that connects to the actuator piston rod, and
- a pin-joint for the piston rod connector.

The link, these embedded components, and their support material were then used as a single embedded component during the construction of the upper leg link. The upper link was formed with two part material compacts which encapsulated the lower link, another pre-fabricated part and pin joint for the actuator mount, and two gear tooth position sensors.

Figure 81. The embedded gear and piston rod connection

The embedded steel gear is the target for the gear-tooth position sensors embedded in the upper leg link. The small rectangular piece is a pre-fabricated part that connects to the end of the actuator piston rod. The Teflon bushings and steel shaft permit it to rotate in the final assembly.

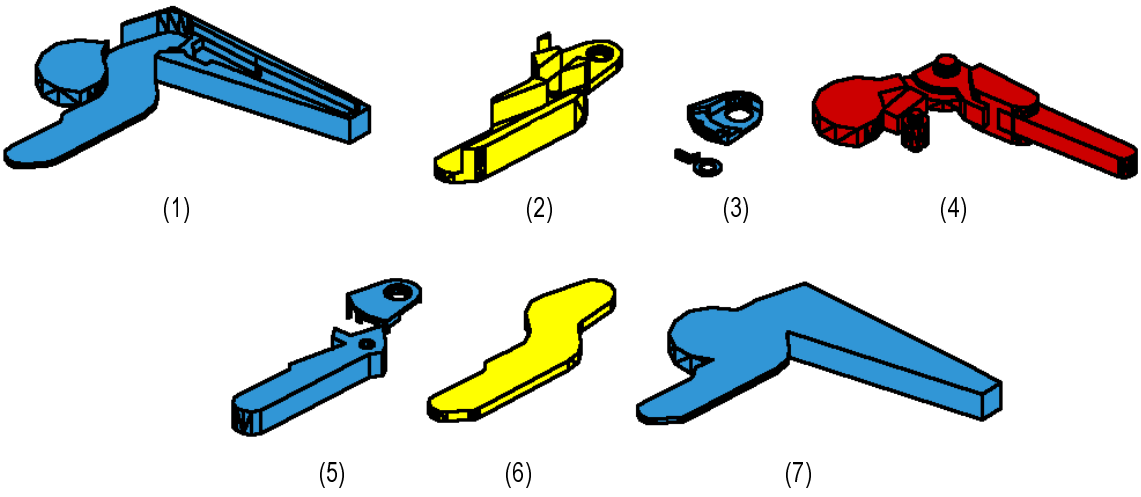


Figure 82. The compacts for the leg mechanism

Compacts 1, 3, 5, and 7 are the wax support material. 2 and 6 are the urethane part material for the upper leg link. Compact 4 is the union of all the embedded components: the position sensors, actuator pivot, and the pre-assembled lower link.

8.2.1 Fabrication of the lower leg link

The fabrication sequence for the lower leg link is shown in Figure 80. First, support material (1) is shaped to mold the bottom surface of the leg link, and a Teflon bushing and steel shaft (2) are inserted. Next, the urethane part material (3), for the bottom section of the link is deposited and shaped. The steel gear tooth and piston rod connector (4) are then inserted as embedded components. Figure 81 shows the the embedded components in position. A small amount of additional support material (5) is deposited to encapsulate the embedded components and provide a mold for the underside of the top layer of urethane (6). Then the second Teflon bushing for the leg joint is inserted (7). The final support material compact (8) is not required for fabrication, but is necessary for the merging algorithm.

8.2.2 Fabrication of the leg mechanism

The fabrication sequence for the complete leg mechanism is shown in Figure 82, and is similar to the plan for the lower link described above:

1. Support material is shaped to mold the bottom surface of the upper link, and to support the pre-fabricated lower link
2. Urethane part material for the bottom section of the upper link is deposited and shaped. (Figure 83)
3. A small amount of support material is needed under the embedded components.
4. The embedded components are inserted. (Figure 84)
5. Support material is deposited to encapsulate the embedded components. (Figure 85)
6. The top layer of urethane is deposited, using compact 5 as a mold for its underside. (Figure 86)
7. Again, the top support material is not required for fabrication.

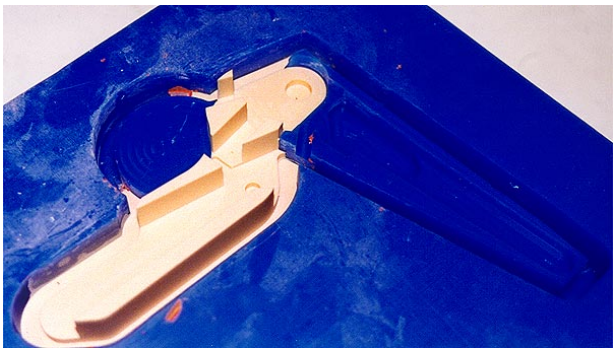


Figure 83. The first urethane compact

Compact 1, the first support compact was machined in a pocket in a wax substrate. Urethane for compact 2 was poured over this cavity, and then machined to this shape.

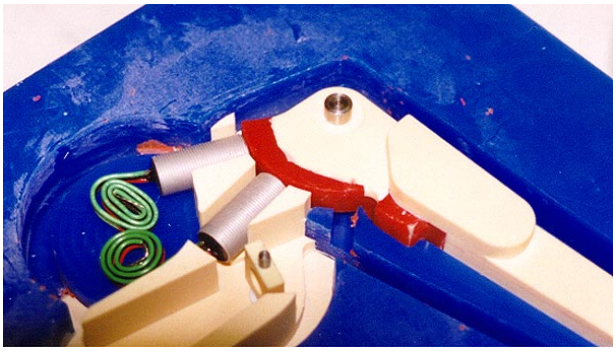


Figure 84. Insertion of the embedded components

The cylindrical position sensors, the actuator pivot, and the pre-fabricated lower link have been inserted. The wax support material still encapsulates the embedded components in the lower link assembly. These embedded components are represented by compact 4.

When the mechanism was built, compact 3 was omitted; there were already sufficient features to hold the embedded components in the proper position. When the support material for compact 5 was deposited, it flowed around the embedded components, producing both compacts 3 and 5 simultaneously. This departure from the automatically generated process plan is non-essential, but does reduce fabrication time. This type of optimization is often performed by experienced SDM users to reduce fabrication time and expense. Adoption of the design by composition approach does not

affect the manufacturer's ability to make these optimizations, but it does offer greater opportunity to the designer to request them.

After the final machining operation, the wax support material is removed by placing the entire artifact in a heated solvent bath.

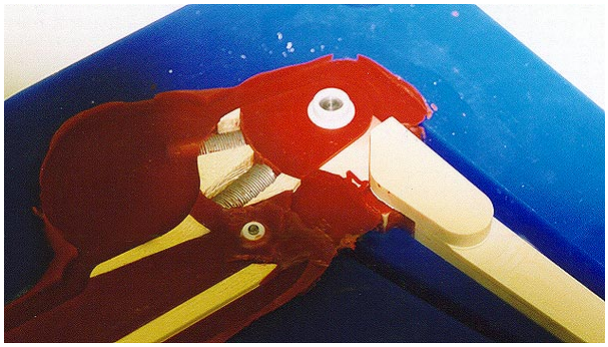


Figure 85. The second support material compact

Wax is poured around the embedded components, and is shaped to form the underside of the final part material compact.

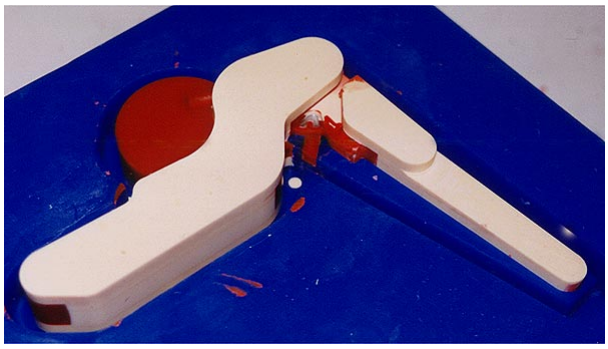


Figure 86. The final part material compact

After depositing the top layer of urethane, it is shaped and the remaining support material is removed from around the mechanism.

8.3 Observations

Although the Shape Deposition Manufacturing technique offers new design possibilities, its practicality is limited by effects like shrinkage and interlayer bonding.

Giving designers a way to live with these limitations will be a major task in the development of design and planning software in the future. Building the robot leg described above and other parts has shown that embedded components impose additional processing requirements, such as the need for temporary support material and special treatment by the CNC path planner.

These issues make it unlikely that SDM will ever be a 100% automated process, or that a complete de-coupling of design and fabrication will be achieved.

8.3.1 Manufacturing process constraints

SDM is based on processes that deposit material in a liquid form. The liquid to solid phase change always creates some shrinkage. Because SDM structures are built with multiple materials (at least one part and one support material is required), and deposition occurs at different times (one deposition per compact), shrinkage will invariably introduce distortion and geometric errors in SDM parts. The consequences of these errors be purely aesthetic, performance limiting, or destructive (as shown in Figure 75 on page 105). Because shrinkage is inherent in deposition processes, and the whole idea behind SDM is incremental deposition and shaping, this problem will not go away. Reducing the impact of shrinkage on functionality will require proper designs and process plans. The development and use of design libraries that include process plan information may help designers contend with shrinkage effects.

Another serious limitation of SDM is the strength of bonding between adjacent part material compacts **[Kietzman & Prinz 98]**. Especially for plastic parts, the bonds between compacts can be quite weak. As a result, the structural properties of SDM parts often depend more on the process plan produced by the compact splitting operation than on geometric decisions made by the designer. Design by composition, by giving designers an opportunity to examine and modify the compact graph, may provide some relief for this issue; at the least, designers will be able to see where their parts will be weak.

Inter-compact bonding between part and support material compacts can also be a problem during fabrication. Several parts, including the robot leg described above, have been pulled off the support material palette by cutting forces on the CNC mill. The solution we have adopted at Stanford is to manually drill holes into the support structure that will provide anchors for the part material. If the anchor holes are positioned correctly, they will be removed automatically by the final machining operations at the end of the fabrication process. Hopefully future generations of SDM process planning software will create anchors and other temporary structures automatically.

8.3.2 Embedded components

It is often a problem that after embedded components are placed in the SDM artifact and covered by a deposition step, the deposited material must be removed to expose the component for a later deposition step. For example, the fabrication sequence in Figure 87 shows bearings that are inserted as embedded components and then covered with support material. Some of this support material must then be removed so the bearing can be encapsulated by part material. However, shrinkage of the support material will move the embedded components away from their nominal position. The subsequent machining operation may therefore cut into the bearing.

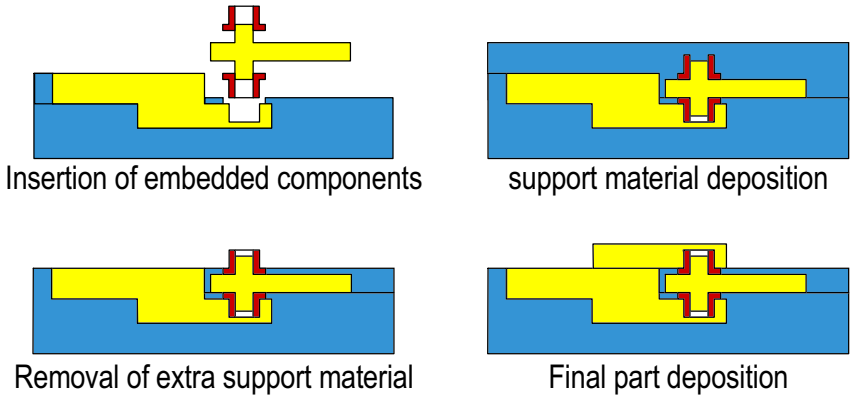


Figure 87. Process plan for embedded components

After inserting the embedded components, they are covered with support material which must be partially removed. Because of the positioning errors caused by shrinkage, it is difficult to remove all the support material without damaging the embedded component.

There are two solutions to this dilemma. First, to prevent damage to the embedded component, the machining step can be planned to leave a thin "tolerance zone" of support material around the embedded component. The material remaining in the tolerance zone can then be removed manually. I used this approach to build the robot leg shown in Figure 78 on page 110. An alternative would be to cover the bearing with a temporary material (such as a water-soluble material) that can be removed without affecting the support or part materials. This temporary material can be used to provide a buffer zone around the embedded component for the shaping operations, as shown in Figure 88.

In either case, the shaping operation must be modified to leave a tolerance zone around the embedded component. This modification to the normal process planning

behavior can be accomplished by making the embedded component compact in the design system larger than the actual component, or by making the path planning software treat embedded component compacts differently. The first option is much easier to implement, and is what has been used in embedded component designs.

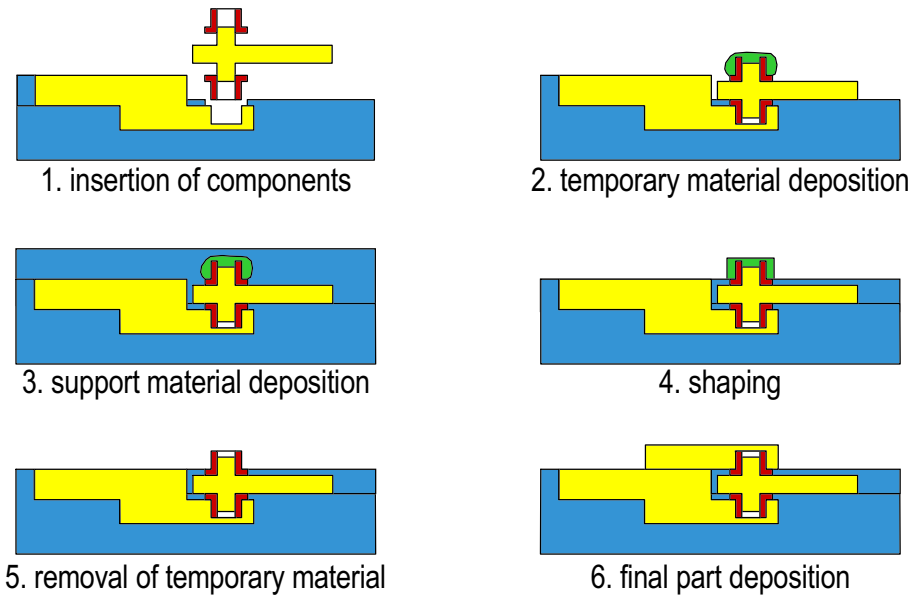


Figure 88. Using a temporary material around embedded components

The temporary material is deposited on the embedded component to provide a buffer zone for the shaping operation.

9. Conclusions and future work

The work in this dissertation is based on the hypothesis that automated decomposition is not always the best approach to SFF fabrication. Especially for designers who are new to the manufacturing process, the lack of quick feedback and the limited degree of automation available today make decomposition unattractive. Addressing these two shortcomings led to the development of the design by composition approach. My research has focused on understanding the theory, algorithms, and architecture required for construction of a practical design by composition environment for automated processing of 3D parts and assemblies with SDM.

The compact merging and CPG manipulation algorithms presented in this dissertation show that design composition is a practical approach for SFF processing. My preliminary results, and the success of other graduate students, using the prototype design by composition implementation to build parts and integrated assemblies are encouraging. However, the viability of design by composition depends on whether the algorithms are truly robust and scalable. Although Chapters 4 and 5 addressed this issue theoretically, they still must be tested empirically.

This chapter begins by summarizing the main advantages of design by composition. Section 9.2 then describes possible extensions to the approach. Finally, Section 9.3 argues that design by composition will become even more useful when it is integrated with a CPG-based decomposition system and is extended with a user interface that lets designers manipulate CPGs directly; in the future both decomposition and design by composition will be important parts of the design-manufacturing interface for SFF manufacturing.

9.1 Tradeoff between decomposition and composition

The basic tradeoff a designer must make in choosing between a decomposition or composition approach is between design flexibility and ease of use.

9.1.1 Design flexibility

The design by composition system does not impose any fundamental geometric limitations; primitives in the system can have any 3D geometry. In practice, though, designers are limited to:

- geometry they can construct from existing primitives, and
- designs for which they manually create the compact graph.

An ideal decomposition system, on the other hand, will work with any manufacturable geometry. Progress on automatic decomposition software at Stanford suggests that while such a system may be possible, it is unlikely to produce efficient decompositions for all designs. Furthermore, complete implementation of such a system is still several years in the future.

A somewhat similar concept is being advocated for design of MEMS (Micro Electro Mechanical Systems) by Gary Fedder at Carnegie Mellon University **[Fedder 99]**. Like design by composition, Fedder's approach removes process planning and design complexity from the designer, at the price of restricted design freedom.

9.1.2 Feedback to the designer

A significant advantage demonstrated by the prototype design by composition system is the ability of the designer to examine the compact set – and therefore the manufacturing plan – of the part at any stage in the design process. With the decomposition system, viewing this level of manufacturing feedback requires:

1. transmitting (with FTP) the CAD file to the manufacturing site,
2. creating (or modifying an existing) instruction file for the planner,
3. executing the planning software, and
4. examining the CNC files produced.

Using the design by composition system, all that is required is turning on some hidden layers in the drawing. The designer is then able to use the familiar AutoCAD commands to manipulate, view, and render the compacts in 3D (Figure 89).

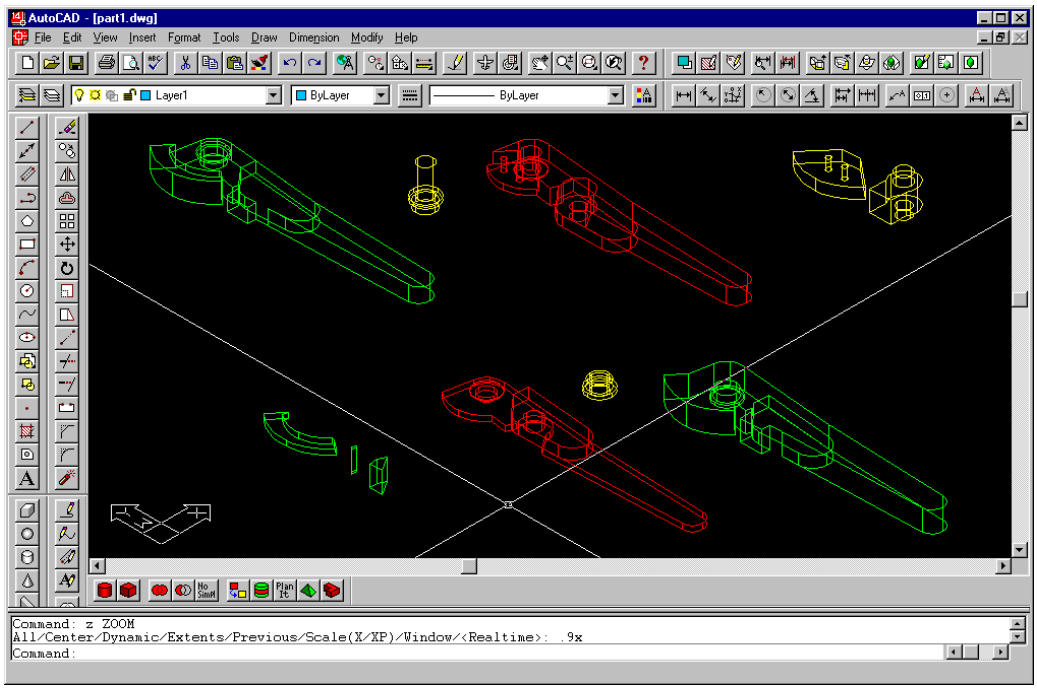


Figure 89. Compacts displayed in AutoCAD

With the design by composition system, the designer can use the standard AutoCAD interface to view the compacts and see how the artifact will be manufactured.

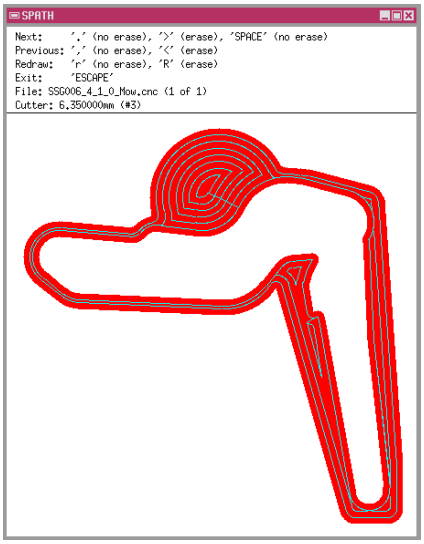


Figure 90. Output from the SDM CNC path preview utility

This program displays the 2D cutting paths produced by the SDM planner.

9.2 Possible extensions

There are several ways the design by composition system can be enhanced with additional capabilities.

9.2.1 Multi- and graded-material parts

The design by composition approach can support multi-material parts, such as the molding tools shown in Figure 5 on page 25. As presented here, the merging algorithms operate with only 3 material types: part, support, and embedded component. The only change to the algorithm required for additional material types is changing the truth tables shown in Figure 31 on page 52. In some cases, it may be appropriate to let the designer adjust the truth tables, or the material type for individual intersection compacts, manually.

Graded material support requires functionality to specify material properties or composition as a function of position. Most likely, the implementation of this feature would involve specifying a different material number for each functional composition. For example, the designer could specify that material 5 has an INVAR to stainless ratio of

$$f(x) = \begin{Bmatrix} 0 & if\ x < 1 \\ 2(x-1) & if\ 1 < x < 2 \\ 1 & if\ x > 2 \end{Bmatrix}$$

and that material 6 specifies an INVAR to stainless ratio as a function of y. Then the issue becomes what is the material function for a compact formed by the intersection of materials 5 and 6? In most cases, this decision will be made by the user, or by an algorithm which is controlled by user-specified parameters. For example, the designer could specify that material 5 "dominates" material 6, or that the intersection should be the minimum, maximum, or average of the two functions.

9.2.2 Fluid piping and other specialized applications

The ability of design by composition to work with multiple materials can be exploited to tailor the design system to specialized applications. For example, in fluid piping problems, a horizontal pipe can be used as a library primitive. The corresponding compact set 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 91, 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 91. Pipe merging truth table

Because the intersection of pipe and fluid is fluid, two intersecting pipes will form a fluid connection.

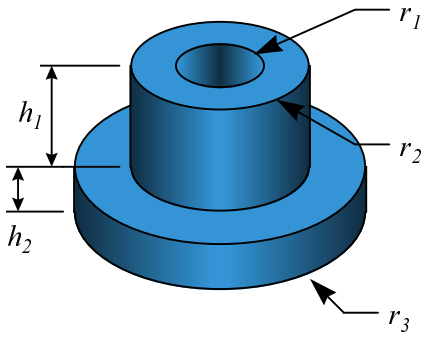


Figure 92. A parametric primitive

The design by composition system could be much more useful if primitives allowed parametric modifications – for example, changing the ratio between r_1 and r_2.

9.2.3 Parametric primitives

One extension to the design by composition system that would make it much more useful would be enabling parametric modifications to primitives. For example, the flanged bushing shown in Figure 92 is a primitive used in many designs. In the current implementation, the only modifications permitted are translation, rotation about Z, and scaling. With these operations, it is possible to change the horizontal to vertical aspect ratio with a non-uniform scaling operation. The designer cannot, however, change the ratios between the radii, r_1, r_2, and r_3.

Enabling parametric primitives requires two major functions:

- establishing parametric relations between all the compacts in the primitive's compact graph, and
- evaluating constraints that prevent topological changes.

For example, with the top-hat bushing in Figure 92, when the radius r_2 is changed, the design system must modify the geometry of the primitive, and the part and support material compacts. Additionally, it must ensure that $r_1<r_2<r_3$ to maintain the primitive's topology. Implementing these capabilities should be possible with parametric CAD environments like Pro-Engineer and AutoCAD's Mechanical Desktop.

9.3 Integration with a compatible Decomposition system

As mentioned earlier, researchers at the Stanford Rapid Prototyping Laboratory are currently developing a new decomposition system. This software package is based on the same compact graph format as the design by composition system. Eventually, the design by composition and design decomposition software can be integrated, using compact graphs as the common interchange format, as shown in Figure 93. This integration will permit several new design and manufacturing possibilities which will make SDM more useful to designers with a wide range of experience.

Creating new primitives

A decomposer that is compatible with the design by composition system can be used to create the compact graphs for new primitives. This feature will open more possibilities to designers – they will be able to generate new primitives with any desired shape.

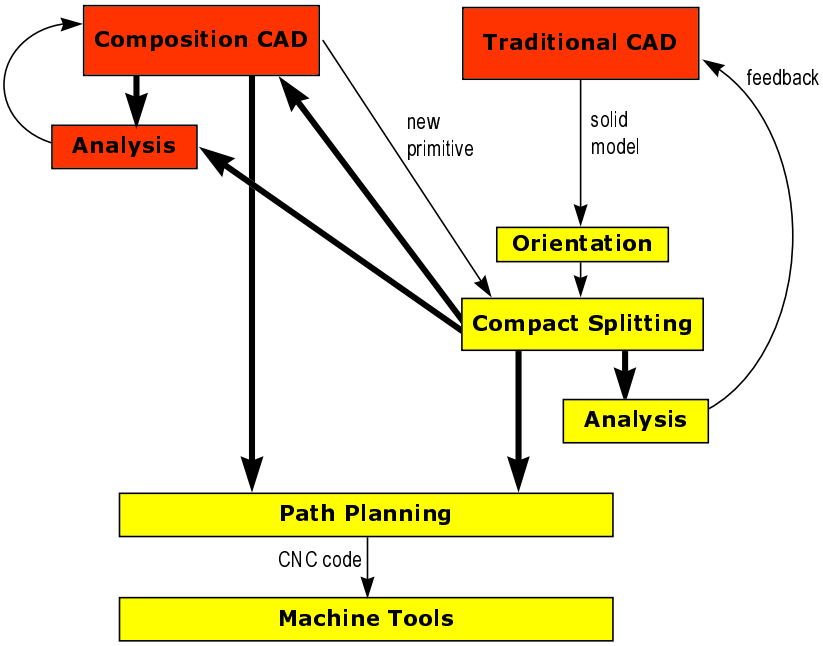


Figure 93. Integrating design by composition and decomposition

The bold arrows represent transmission of compact graphs. Compatible design by composition and decomposition systems permit many new capabilities: the decomposer can be used to provide plans for new primitives or primitives with a different growth axis, and the design by composition CAD system and feedback tools can be used to analyze and optimize decomposer output.

Changing the build-axis for primitives

As stated above, several geometric transformations can be applied to primitives and their compacts without destroying the validity of the manufacturing plan. Rotation about the X and Y axes are not included in this list. Quite often, however, designers may want to use a primitive from a design library, but in a different orientation. The solution to this problem is to take the geometry of the primitive from the design library, apply the desired transformation, and then submit this model to the decomposer. The decomposition output will be a compact graph for the primitive in its new orientation.

Designer access to decomposer output

If the manufacturing plan produced by the decomposer is transferred into the design by composition CAD and analysis environment, the designer can obtain manufacturability feedback and examine the fabrication plan for his part. Because the design by composition environment will include compact graph editing functions, the designer will now have the ability to modify the decomposer output. These

modifications could be used to optimize the manufacturing plan, or to ensure proper fabrication of critical features.

EPILOGUE

The algorithms and theory presented in this dissertation demonstrate that design composition can be a practical alternative to automated decomposition. The experience of designers like Jorge Cham and Beth Pruitt at Stanford's Center for Design Research with the prototype system described in Chapter 7 and the Appendix have shown that design by composition can really help designers who are new to SFF build successful products.

There is still important work remaining. The prototype design by composition plug-in is not complete. In particular, it does not implement the compact precedence graph operations described in Chapter 5. Completing this program will require developing a CPG-manipulating user interface and completing algorithms for the simplification operation described in Section 5.2. Ongoing work at the Stanford Rapid Prototyping Laboratory on a CPG-based decomposition system offers many synergies with a design composition CAD system, as mentioned in Section 9.3. When design composition software has been completed and integrated with CPG-based decomposition, it will become an important implement in the SFF toolbox.

10. Appendix - Software Implementation

This appendix provides brief documentation of the design by composition plug-in for AutoCAD (called the SDM plug-in below). This program was written with the ObjectARX 2.0 programming interface. This documentation assumes the reader is familiar with ObjectARX **[ARX 97]**.

The complete source code for the plug-in is available on the World-Wide Web at `http://cdr.stanford.edu/~binnard/sdm-arx.zip`.

10.1 Overview

The SDM plug-in allows AutoCAD users to use the design by composition approach to design parts for the SDM process at Stanford University. It adds new commands to the AutoCAD environment that allow creation and merging of primitives. Like standard AutoCAD commands, the user can either type in the command names, or select a button on a toolbar.

The program defines several custom entity types as new C++ classes. (An entity is any thing that appears in the AutoCAD drawing, e.g., lines, circles, and solids.) The most important of the custom classes are `Primitive` and `Compact`, which provide the basic creation, manipulation, and merging functions. Two subclasses of `Primitive`, `CylinderPrim` and `BoxPrim`, allow the user to create cylindrical and box-shaped primitives. The final custom class, `Reactor`, is used to ensure that a primitive and its compact set behave properly.

An interesting fact about the current implementation for the plug-in is that it performs essentially no computations based on the geometry of compacts and primitives. All geometric operations are handled by AutoCAD, or the ACIS solid modeler which is part of AutoCAD. By avoiding direct access the geometry of individual compacts, the implementation of the plug-in is kept simple and efficient. In some situations, the output produced by the plug-in may be non-optimal (see the discussion of slice heights in Section 10.4.1), but this sacrifice is well worth the advantages of keeping the code simple.

When the plug-in is loaded into AutoCAD, it creates four new layers: PART, SUPPORT, EMBED, and OOPS. Each layer corresponds to a different material type for compacts. OOPS is for the intersections between embedded components, which are illegal. Using a separate layer for each material type allows each material to be shown as a different color.

The major difference between the implementation and the algorithms and theory presented in this dissertation is the handling of sequence information for compact sets. The dissertation describes the use of a precedence graph (CPG) to represent the possible fabrication sequences for compacts. As implemented, however, the plug-in maintains a single build order from the (usually) many which are possible. Each compact in a compact set is assigned a build order number. This implementation again trades some optimality for simplicity and efficiency. In this case, however, experience has shown that the tradeoff is not worthwhile, and the software should be modified to include the complete CPG.

10.1.1 New AutoCAD commands

The SDM plug-in adds several new commands to the AutoCAD interface that allow creation and manipulation of primitives and compacts.

Primitive creation

`cyl`	Creates a new cylindrical primitive
`prism`	Creates a new box primitive
`embed`	Converts a primitive to an embedded component

Merging operations

`merge`	Merges two primitives with the ADD operation
`sub`	Merges two primitives with the SUBTRACT operation
`nosimple`	Merges two primitives with ADD, but doesn't simplify the result compact set

Compact set manipulation

makelist	Initializes the compact set for a primitive
primitive	Converts an AcDb3dSolid to a Primitive; requires manual creation of the compact set
reorder	Lets the user manually re-order a compact set

Output to manufacturer

out	Outputs the files needed to build a primitive

Debugging

reactors	Lists all the Reactor objects that are defined
compact	Converts an AutoCAD solid (class AcDb3dSolid) to a Compact

10.1.2 Source code files

These are the source code files, and a brief description of what's contained in each one.

arx part.cpp	Primitive and compact creation routines
compact.cpp	Implementation of the Compact class
layers.cpp	Procedures for manipulating layers in the AutoCAD drawing
merge.cpp	Compact set merging operations and associated procedures
MFCstuff.cpp	Microsoft Foundation Class stuff that's required for the Toolbar
output.cpp	Procedures for outputing geometry and RUN files to the path planner
prim.cpp	Implementation of the Primitive, CylinderPrim, and BoxPrim classes
reactor.cpp	Implementation of the Reactor class
sdm_main.cpp	Program initialization and clean-up routines
test.cpp	Test and debugging procedures
toolbar.cpp	Toolbar event-handling code
util.cpp	General utility functions

10.1.3 Interface with SDM path planning

As described in Chapter 6, the interface between the design by composition plug-in and the SDM path planning software is based on geometric entities called Single Step Geometries (SSGs). Each compact in the compact set for a primitive represents one SDM processing cycle. The corresponding SSG is defined by the union of the current

compact with all prior compacts. When the user invokes the `out` command in the design by composition interface, the plug-in creates ACIS geometry files (.SAT files) for each SSG. The SSG files are how the geometry of the design is transmitted to the path planning software.

Each SSG file has a corresponding .RUN file, which is an ASCII text file that contains parameters for the path planning software. When using the design by composition plug-in, the only parameters that vary in the .RUN files are the material type, substrate thickness, and the "slice heights."

The SDM path planning software requires a list of "slice heights" for each SSG. These slice heights are the Z coordinates of each place in the design where the XY cross-section changes. Each compact created by the plug-in is initialized with the correct slice heights. During merging operations, the software adds new slice heights to each compact as the compact shapes are changed by the merging algorithm. The current implementation assures that all necessary slice heights are maintained, and sometimes includes extra slice heights which are unnecessary. During the output operation, the slice heights for each compact are written into the appropriate .RUN file.

10.2 Class heirarchy

The SDM plug-in implements several new classes of AutoCAD entities. These custom classes allow the addition of primitives and compacts to the AutoCAD drawing. The Primitive and Compact classes are derived from the AutoCAD class `AcDb3dSolid`, which is a generic three dimensional solid. The Primitive and Compact classes add additional methods and data members to these classes.

Class	Derived From
`Primitive`	`AcDb3dSolid`
`CylinderPrim`	`Primitive`
`BoxPrim`	`Primitive`
`Compact`	`AcDb3dSolid`
`Reactor`	`AcDbReactor`

Figure 94. Custom classes in the SDM plug-in

Each custom class defines methods, including construction, destruction, and file I/O that are required by AutoCAD. Note that AutoCAD uses the `dwgInFields` and `dwgOutFields` for many operations besides file I/O, including copy, block, explode, array, and undo commands. This fact makes these two methods extremely important.

Additional methods modify behaviors inherited from the base class; for example, the `list` method is modified to display the primitive or compact's additional data members.

10.2.1 Primitive

This is the base class for primitives in the design by composition system. The class is derived from the built-in AutoCAD class for 3D solids, `AcDb3dSolid`. The compact set for the primitive is stored in an AutoCAD block, which is pointed to by `BTRid`. To make the compacts visible on the appropriate layers in the drawing, there must be a Block Reference in Model space. `blkRef` points to that Block Reference. When a cylindrical or box-shape primitive is created, the compact set for that primitive is not initialized until it is needed by a merging operation. The `haslist` flag keeps track of the state of the primitive's compact set.

Data members

`shape`	`Integer flag for the type of compact; CYLINDER, BOX, UNKNOWN, etc.`
`coordFrame`	`Transformation to local coordinates`
`blkRef`	`ID of Block Reference for compact set`
`BTRid`	`ID of compact set Block Table Record`
`haslist`	`Boolean flag for whether or not compact set is initialized`

Methods

Methods that are required by AutoCAD:

`Primitive`	`Class constructor`
`~Primitive`	`Destructor`
`dwgInFields`	`Read in data from a DWG filer`
`dwgOutFields`	`Write data to a DWG filer`
`dxfInFields`	`Read from DXF file`
`dxfOutFields`	`Write to DXF file`

Methods that modify behaviors inherited from `AcDbEntity` and `AcDb3dSolid`:

`list`	`Modified to display the extra data members in the Primitive class`
`transformBy`	`Modified to prohibit rotations about the X and Y axes, and to prohibit Z translations for primitives that have initialized compact sets`
`getSlice`	`Modified to prevent slicing a compact into two pieces`
`subErase`	`When a primitive is deleted, this method takes care of erasing its compact set`
`deepClone`	`Modified so when a primitive is copied, the compacts are also copied`

Methods that are specific to the `Primitive` class:

`setShape`	`Sets the shape flag to something`
`makeList`	`Initializes a new compact set; creates the Block Table Record and Block Reference`
`addCompact`	`Adds a new compact to the Block Table Record`
`origin`	`Returns the origin of the local coordinate frame, in world coordinates`
`getFrame`	`Returns the coordFrame transformation matrix`
`getHaslist`	`Returns the value of the haslist flag`
`getBTRid`	`Returns BTRid`
`getBlkRef`	`Opens the Block Reference and returns a pointer to it`
`getBTR`	`Opens the Block Table Record and returns a pointer to it`
`getCompactIter`	`Creates a Block Table Record Iterator for stepping through the compact list`

10.2.2 CylinderPrim - cylindrical primitives

This subclass allows creation of cylindrical primitives. The only differences from the base `Primitive` class are that there are additional data members for the radius and height of the cylinder, and the `makeList` method is implemented to create the part and support compacts needed to fabricate the cylinder.

Data members

`radius`	`The radius of the cylinder`
`height`	`Height of the cylinder`

Methods

Required methods:

`CylinderPrim`	`Class constructor`
`~CylinderPrim`	`Destructor`
`dwgInFields`	`Read in data from a DWG filer`
`dwgOutFields`	`Write data to a DWG filer`
`dxfInFields`	`Read from DXF file`
`dxfOutFields`	`Write to DXF file`

`AcDbEntity` overrides:

`list`	`Modified to display radius and height in addition to what's displayed by the Primitive class' list method.`

Cylinder specific methods:

makeList	Calls Primitive::makeList to initialize the compact set, and then creates the part and support compacts
setData	Sets the cylinder's radius and height to specified values
getRadius	Returns the cylinder's radius
getHeight	Returns the cylinder's height

10.2.3 BoxPrim - rectangular box primitives

This is the class for a rectangular prism primitive, which is also derived from `Primitive`. Like `CylinderPrim`, the only new stuff here is the data members to track the size of the primitive and the implementation of `makeList` which creates the compact set.

Data members

dX,dY,dZ	The X, Y, and Z measurements of the box

Methods

Required ARX stuff:

BoxPrim	Constructor
~BoxPrim	Destructor
dwgInFields	Read in data from a DWG filer
dwgOutFields	Write data to a DWG filer
dxfInFields	Read from DXF file
dxfOutFields	Write to DXF file

`AcDbEntity` overrides:

list	Modified to display X, Y, and Z size of the box

Box specific methods:

makeList	Calls Primitive::makeList to initialize compact set and then creates the part and support compacts
setData	Sets dX, dY, and dZ
getDx	Returns dX
getDy	Returns dY
getDz	Returns dZ

10.2.4 Compact

The `Compact` class extends the AutoCAD `AcDb3dSolid` class to maintain the additional information required by compacts, primarily build order, material type, and slice heights. The build order is stored as an integer, `order`. The data member `newOrder` was introduced as an attempt to solve the problems of maintaining the

correct order for embedded components, and could be removed. The material type for a compact is determined by which layer in the drawing it is on, PART, SUPPORT, or EMBED. Slice heights are maintained in the dynamically-sized array `slices`.

Data members

`order`	`Build order for this compact`
`newOrder`	`Integer to temporarily store a new build order for embeded compacts`
`sorted`	`Flag for compact-list simplification procedure.`
`slices`	`Array to keep all the slice heights`

Methods

Required methods:

`Compact`	`Constructor`
`~Compact`	`Destructor`
`dwgInFields`	`Read in data from a DWG filer`
`dwgOutFields`	`Write data to a DWG filer`
`dxfInFields`	`Read from DXF file`
`dxfOutFields`	`Write to DXF file`

AcDbEntity overrides:

`list`	`Modified to display the compact's build order and slice heights`
`transformBy`	`Modified to prohibit X and Y axis rotations`

Compact-specific methods:

`material`	`Returns the material type of the compact (based on its layer)`
`setMaterial`	`Changes the layer of a compact to reflect the specified material type`
`initSlices`	`Initializes the slices array to an empty list`
`addSlice`	`Adds a slice height to the slices array, unless its already there`
`numSlices`	`Returns the number of slice heights stored in slices`
`getSlice`	`Returns a slice height`
`getOrder`	`Returns the build order of the compact`
`setOrder`	`Sets the compact's build order to a specified value`
`getSorted`	`Returns the value of the sorted flag`
`setSorted`	`Sets the sorted flag`
`getNewOrder`	`Returns the value of newOrder`
`setNewOrder`	`Sets the newOrder data member`

10.2.5 Reactor

In ObjectARX programs, reactors provide the means to let one object in the database respond to changes to another database object. The SDM plug-in uses reactors to keep a primitive and its compact set together. Each primitive in the drawing has an associated reactor that receives notification from AutoCAD whenever the primitive is modified by the user. If the primitive was moved, the reactor then opens the block reference for the primitive's compact set and applies the same transformation to it. In this way, the reactor ensures that the primitive and its compact set always move together.

Data members

mId	The ID of the compact set this reactor points to.

Methods

Required methods:

Reactor	Constructor
~Reactor	Destructor
dwgInFields	Read in data from a DWG filer
dwgOutFields	Write data to a DWG filer
dxfInFields	Read from DXF file
dxfOutFields	Write to DXF file

Reactor specific methods:

linkTo	Establishes the link to the primitive's compact set
modified	Called when the primitive is modified; moves the compact set's block reference if needed.
linkId	Returns the ID of the block reference the reactor points to

10.3 Global functions

This section describes the functions and procedures in the plug-in which do not belong to a specific class. (Because my programming background is primarily in non-object-oriented Pascal, there are a lot of these.)

Main program:

initApp	Called when the program is loaded into AutoCAD; initializes custom classes, adds commands to AutoCAD, and displays the toolbar
unloadApp	Called when the plug-in is being unloaded; removes custom classes, commands, and toolbar

acrxEntryPoint	Called by AutoCAD when a plug-in or drawing load or unload event occurs

User-invoked commands:

makecylinder	Creates a cylindrical primitive
makebox	Creates a box primitive
makelist	Lets the user select a primitive and intializes its compact set
outputSSGs	Lets the user select a primitive and outputs the files needed for its manufacture
solidToPrimitive	Converts a normal Autocad solid (AcDb3dSolid) to a Primitive; also lets user pick compacts for the compact set
solidToCompact	Converts an AcDb3dSolid to a Compact
add	Merges two Primitives with the ADD operation
subtract	Merges two Primitives with the SUBTRACT operation
nosimple	Merges two Primitives with the ADD operation, but doesn't perform the simplification step
reorder	Allows the user to manually re-order the compacts in a compact set
makeEmbed	Converts a primitive into an embedded component

Output:

makeSliceString	Creates an ASCII string that has all the slice heights for a compact
makeRunFile	Creates the RUN file that has all the parameters needed by the SDM path planner

Compact merging:

clearSort	Clears the sorted flags of all the compacts in a compact set
nextCompact	Finds the next sequential compact in the compact set
simplify	Performs a compact set simplification operation; combines sequential compacts of the same material type
intersect	Performs the intersection between two compacts, and returns the intersection compact if it exists
merge	Implements the compact merging algorithm; accepts a flag that indicates the type of operation (ADD, SUBTRACT, or INTERSECT); lets the user select two primitives, performs initialization, and then calls mergePrims
mergePrims	Called by merge; steps through all the compacts in each compact set and calls mergeCompacts for each pair; copies non-

	intersecting compacts to the result compact set
mergeCompacts	Called by mergePrims; performs the actual merging operations
combineHeights	Called by mergeCompacts; combines the slice heights from two compacts
subCompact	Called by mergeCompacts when a new intersection compact is created; subtracts the intersection piece from the two source compacts, and deletes them if they are completely consumed

Utility functions:

newcylinder	Creates a new cylindrical primitive
newbox	Creates a new box primitive
newCylCompact	Creates a new cylindrical compact
newBoxCompact	Creates a new box compact
cloneCompacts	Copies the compacts from one primitive to another
manualCompact	Called by solidToPrimitive; lets the user pick the next compact and specify its slice heights
createReactor	Creates a new Reactor object
killReactor	Deletes a Reactor, called when the primitive is deleted
listReactors	Lists all the Reactor objects (for debugging)
createToolBar	Initializes MFC and creates the toolbar
toolbarProc	Called when the user selects a toolbar button; uses sendCommandToAutoCAD to issue the appropriate command
sendCommandToAutoCAD	Sends a specified command to the AutoCAD command line; this is how the tooblar buttons invoke AutoCAD commands
modelSpace	Returns a pointer to Model Space, where all visible entities live
addToDbase	Adds a new entity to the AutoCAD drawing
intMaterial	Performs the material-type truth table lookup; returns the material type of an intersection compact, based on the material types of the two source compacts and the merging operation
asPoint	Converts an vector (class AcDbVector3d) to a point (class AcGePoint3d)
selectEntity	Prompts the user to select an entity, and returns a pointer to it
Intersect1d	Called by boundingIntersect, checks if two ranges of numbers intersect
boundingIntersect	Checks if the bounding boxes of two solids intersect
cloneObject	Performs a deepClone on a database object
getPoint	Prompts the user to select a point

`sa_u2w`	Converts a point from user coordinates to world coordinates
`sa_w2u`	Converts a point from world to user coordinates

10.4 Known bugs and limitations

There are several known problems with the SDM plug-in that should be fixed in future versions.

10.4.1 slice height propagation

As described above, the SDM path planning software requires a list of "slice heights" for each Single Step Geometry. During merging operations, the `mergeCompacts` procedure copies all the slice heights from the *a* and *b* "parent" compacts to the *c* intersection compact. The slice heights from *a* and *b* are also added to the non-intersecting portion of *b* and *a*, respectively. This implementation ensures that all the necessary slice heights are maintained in the new compacts, but it also often creates additional, unnecessary slice heights.

A better implementation would be to use the bounding boxes of the compacts to remove some of the unnecessary heights. (Removing all the unnecessary heights would require analysis of the 3D geometry of the compacts, which is far more difficult.) For intersection compacts, only the slice heights from *a* and *b* that are within the bounding box of *c* should be used. When the volume of *c* is subtracted from the non-intersecting portion of *a* and *b*, only the slice heights that fall within the bounding box of the appropriate parent compact (*a* or *b*) and the bounding box of *c* should be added to the non-intersecting compact.

10.4.2 Manual compact set creation

The code for manually creating a compact set for a new primitive, in the `solidToPrimitive` and `manualCompact` functions, mostly works. There are some bugs, however, which can lead to corrupt data, or an AutoCAD crash. The user interface for this procedure is also not well polished; because the edges of compacts normally overlap, it is often difficult to select the right compact.

10.4.3 Ordering of embedded components

As mentioned above, the plug-in does not use a CPG to represent the fabrication sequence of compacts. Instead, each compact a compact set is assigned a unique integer (the `order` data member) that identifies its build order. This implementation effectively gives each compact set a single, arbitrary, sequence which is a subset of the complete CPG for that compact set.

For designs which contain only part and support material compacts, the build orders produced by the plug-in may not always be ideal, but they are guaranteed to be valid. For designs with embedded components, however, the order generated by the software may be illegal. Determining the correct build order for embedded compacts requires evaluation of the geometry of the embedded components and the adjacent compacts. As mentioned above, geometric calculations are beyond the scope of the current implementation. As a result, designers who use embedded components must occasionally re-order their compact sets with the `reorder` command.

10.4.4 Non-intersecting compacts

The current implementation of `mergePrims` always copies all non-intersecting compacts from both the A and B compact sets. As described in Chapter 4, not all of the non-intersecting compacts are necessary. Modifying the `mergePrims` procedure to copy non-intersecting compacts only when necessary is straightforward.

10.4.5 Reactors

There are two known problems with the implementation of the `Reactor` class and the associated global functions. First, the `Reactor` ensures that when the user moves a primitive, the compact set will follow it. However, the reverse behavior is not implemented. If the user moves the compact set, the primitive will not move to follow it, and the primitive and its compact set will no longer be valid for future merging or output operations. The solution to this problem is to create another `Reactor` object which watches for modifications to the compact set and makes corresponding changes to the primitive.

Second, there is a problem which is not completely diagnosed with the creation of reactors with the `solidToPrimitive` command. The reactors associated with manually created primitives do not always operate correctly. Most likely this problem lies in the initialization of the `coordFrame` data member for the primitive.

10.5 Current development efforts

At the time of this writing, there are students at Stanford's Center for design Research working on enhancing the capabilities of the SDM plug-in.

10.5.1 New primitive types

Adaptations to the code currently being developed provide automatic generation of compact lists for any simple shape, not just cylinders and boxes. This improvement will let designers create freeform shapes without having to use the awkward (and less than robust) `solidToPrimitive` procedure.

Work is also beginning on primitives will true 3D geometry, such as spheres and horizontal cylinders. By interfacing the plug-in to 3D path planning software from the CyberCut group at the University of California, Berkeley, users of the design by composition plug-in will be able to create true 3D designs. No changes to the actual merging code should be necessary – only routines for creating 3D primitives and producing the correct output needed by the path planner.

10.5.2 Compact Precedence Graph manipulations

An important addition to the program will be the addition of the compact graph manipulating algorithms described in Chapter 5. This work will involve developing a user interface to allow designer-control over CPG simplification operations, and formalization of automated simplification algorithms. Most likely, this effort will allow the software to suggest a simplification option for each merging operation. Designers will then be able to accept the recommendation or make modifications as required.

Completion of the CPG implementation will make the creation of designs with embedded components more robust – often the current single-order implementation does not correctly handle embedded components. It will also make the program easier to use for novice designers, and will make it practical to design much more complex geometric shapes.

REFERENCES

ACIS Geometric Modeler: Application Guide, Spatial Technology Inc., 1995.

Alexander, P., Allen, S., Dutta, D., "**Part Orientation and Build Cost Determination in Layered Manufacturing**," *Computer Aided Design*, v 30 n 5, p 343-356, April, 1998.

Allen, S., Dutta, D., "**Determination and Evaluation of Support Structures in Layered Manufacturing**," *Journal of Desgin and Manufacturing*, v5, p 153-162, 1995.

ARX Reference Manual, Autodesk Inc., 1997

Brooks, R.A., Massachusetts Institute of Technology, personal communication, 1990.

Cutkosky, M.R., Prinz, F.B., et al., "**A Design Interface for 3D Manufacturing**", Stanford University Center for Design Research, proposal to NSF and DARPA, `http://cdr.stanford.edu/interface/publications.html`, August 1996.

Cooper, A.G, Kang, S., Kietzman, J.W., Prinz, F.B, Lombardi, J.L, Weiss, L, "**Automated Fabrication of Complex Molded Parts Using Mold SDM**," *Proceedings of the Solid Freeform Fabrication Symposium*, University of Texas at Austin, August, 1998.

Fedder, G.K., "**Structured Design of Integrated MEMS**," *MEMS 99: Twelfth IEEE International Conference on Micro Electro Mechanical Systems*, IEEE Robotics and Automation Society, Orlando, Florida, January 17-21, 1999.

Fessler, J., Nickel, A., Link, G., Prinz, F., Fussell, P., "**Functional Gradient Metallic Prototypes through Shape Deposition Manufacturing**," *Proceedings of the Solid Freeform Fabrication Symposium*, University of Texas at Austin, August, 1997.

Jacobs, P.F., "**Rapid Prototyping & Manufacturing: Fundamentals of Stereolilthography**," Society of Manufacturing Engineers, Dearborn, MI, 1992.

Harary, F., "**Graph Theory**," Addison-Wesley Publishing Company, Reading, MA, 1969.

Kao, J.H., Prinz, F.B., "**Optimal Motion Planning for Deposition in Layered Manufacturing**," *Proceedings of the 1998 ASME Design Engineering Technical Conferences*, Atlanta, Georgia, September 13-16, 1998.

Kao, J.H., Pinilla, J.M., Pinz, F.B., "**Evolution of Process Planning for Solid Freeform Fabrication Processes**," *Computer Technology Solutions (formerly Autofact'98) Conference*, Detroit, MI, September, 1998.

Kietzman, J.W., Prinz, F.B., "**Material Strength in Polymer Shape Deposition Manufacturing**," *Proceedings of the Solid Freeform Fabrication Symposium*, University of Texas at Austin, August, 1998.

Kumar, V., Dutta, D., "**An Assessment of Data Formats for Layered Manufacturing**," *Advances in Engineering Software*, vol 28, n. 3, p 151-164, April, 1997.

Kumar, V., Rajagopalan, S., Cutkosky, M., Dutta, D., "**Representation and Processing of Heterogeneous Objects for Solid Freeform Fabrication**," *IFIP WG5.2 Geometric Modeling Workshop*, Tokyo, Japan, Dec 7-9, 1998.

Losleben, P., Stanford University, personal communication, 1997.

Marsan, A., Dutta, D., "**A Survey of Process Planning Techniques for Layered Manufacturing**," *Proceedings of the 1997 ASME Design Engineering Technical Conferences*, Sacramento, CA, 1997.

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

Rajagopalan, S., Pinilla, J.M., Losleben, P., Tian, Q., Gupta, S.K., "**Integrated Design and Rapid Manufacturing over the Interenet**," *Proceedings of the 1998 ASME Design Engineering Technical Conferences*, Atlanta, Georgia, September 13-16, 1998.

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

Ramaswami, K., Yamaguchi, Y., Prinz, F.B., "**Spatial partitioning of solids for solid freeform fabrication**," *Proceedings of the Symposium on Solid Modeling and Applications 1997*, ACM, New York, NY, p 346-353.

Ramaswami, K., Prinz, F.B., "**CNC cutter path generation in shape deposition manufacturing,**" *Concurrent Product and Process Engineering*, American Society of Mechanical Engineers, Manufacturing Engineering Division, MED v 1 1995. ASME, New York, NY, p 213-220.

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.

Sachs, E., Williams, P., Brancazio, D., Cima, M., Kremmin, K., "**Three Dimensional Printing: Rapid Tooling and Prorotypes Directly from a CAD Model**," *Proceedings of Manufacturing International '90*, p 131-136, Atlanta, GA, March, 1990.

Sarma, S.E., Wright, P.K., "**Reference Free Part Encapsulation: A New Universal Fixturing Concept**," *Journal of Manufacturing Systems*, vol 16 No. 1, p 35-42, 1997.

Smith, C.S., Wright, P.K., "**Cybercut: A World Wide Web Based Design to Fabrication Tool**," Integrated Manufacturing Laboratory, University of California Berkeley, `http://kingkong.me.berkeley.edu/cybercut/cybercut.htm`, 1996.

Standish, T.A., "**Data Structures, Algorithms, and Software Principles in C**," Addison-Wesley Publishing Company, Reading, MA, 1995.

Voelcker, H., "**New Paradigms for Mechanical Design & Manufacturing?**" NSF Workshop: *New Paradigms for Manufacturing*, Arlington, VA, May, 1994.

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.

Can someone stop the ride, please? I'm ready to get off.