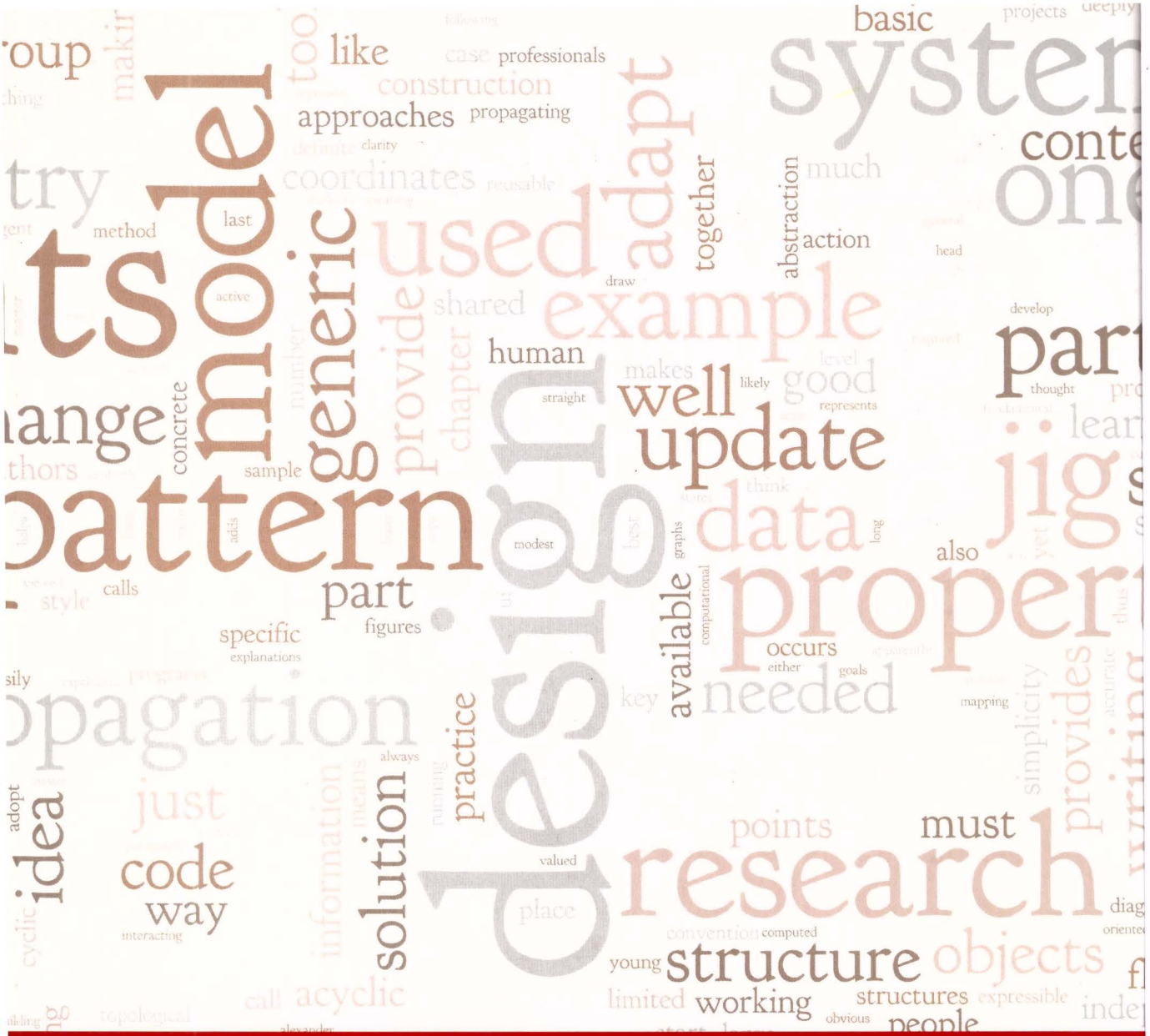


ELEMENTS OF PARAMETRIC DESIGN

ROBERT WOODBURY

WITH CONTRIBUTIONS FROM ONUR YÜCE GÜN,
BRADY PETERS AND MEHDI (ROHAM) SHEIKHOLESLAMI

ROUTLEDGE

Elements of Parametric Design

Elements of Parametric Design

Robert Woodbury

with contributions by

Onur Yüce Gün, Brady Peters and Mehdi (Roham) Sheikholeslami

Routledge
Taylor & Francis Group
LONDON AND NEW YORK

First published 2010
by Routledge
2 Park Square, Milton Park, Abingdon, Oxon, OX14 4RN

Simultaneously published in the USA and Canada
by Routledge
270 Madison Avenue, New York, NY 10016

Routledge is an imprint of the Taylor & Francis Group, an informa business

Typeset in URW Garamond and Bitstream Vera Sans by Robert Woodbury
Printed and bound in India by Replika Press Pvt. Ltd.

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

Library of Congress Cataloging-in-Publication Data
A catalog record has been requested for this book

ISBN10: 0-415-77986-3 (hbk) ISBN10: 0-415-77987-1 (pbk)

ISBN13: 978-0-415-77986-9 (hbk) ISBN13: 978-0-415-77987-6 (pbk)

To Gwenda, Ian and Cailean

Contents

Foreword

Parametrics is more about an attitude of mind than any particular software application. It has its roots in mechanical design, as such, for architects it is borrowed thought and technology. It is a way of thinking that some designers may find alien, but the first requirement is an attitude of mind that seeks to express and explore relationships.

Embedded in this method of exploration is the idea of capturing *design history* and returning it in an editable form – that can be varied and then re-played. The power of the concept is the belief that design history can be extrapolated to produce *design futures.* Sometimes it can – but this requires much practice to achieve a level of fluency which still allows intuition to play its part.

As a concept parametrics is far more likely to be understood by a musician than by an artist. This is because the musician is dedicated to rehearsing for performance – which is an essential characteristic of a virtuoso in parametrics. To the artist on the other hand the accumulation of technique is incidental to the production of an artefact, which is the result of direct interaction with a medium. For this activity there is no written score that can be fine tuned and re-played. However, at the highest level of fluency we may yet see a generation emerge who can "sketch with code".

Parametrics should perhaps be clearly labelled with a warning along the lines of "drink deep – or taste not". So the best advice might be to make your choice before reading further – or just allow your curiosity to guide you!

— Hugh Whitehead

The beginning of the third millennium brings growing recognition that the practice of building design will change more rapidly than in preceding decades. With increasing economic pressure, established practice gives way in favour of tight integration of design and delivery as well as innovation in sharing risks and rewards. In parallel, climate change reinvigorates deep concern about our excessive use of resources, rebalancing values of capital costs and long term design performance. Integrated design teams make simultaneous, interrelated design decisions across disciplines and project phases. Such decisions concern interconnected subsystems with interfaces that propagate change through the overall system and allow the design team to create many design alternatives. In addition, investment in validation of design assumptions through analysis or simulation cycles can further reduce risks.

With parametric modeling, early design models become conceptually stronger than conventional CAD models and less constrained than building information models. Parameters express the concepts contained in these new models and give interactive behaviour to building components and systems. This means a change in how tools need to support design activities. For example tools like Bentley Systems' GenerativeComponents offer a fluid transition between a CAD-like modeling-based design approach on one side and a scripting-based design approach on the other side. These new parametric systems support a shift from one-off CAD-modeling to thinking in and working with geometric concepts and behaviour. Instead of building a single solution, designers explore an entire parametrically described solution space.

The new parametric tools challenge CAD work practices – practitioners and students alike must learn how to use such tools well. We know that quality of learning depends on quality of teaching. The author of this book, Dr. Robert Woodbury, has been teaching GenerativeComponents workshops for several years, while intellectually penetrating the mere instrumental layer of tool use and elevating his teaching to a new layer of concepts. Dr. Woodbury and his students chose the motif of patterns to explain this conceptual layer, to unravel its components' behaviours and to provide new functions useful for parametric design. Initial results appeared online at `www.designpatterns.ca` and now are revised for this book. Dr. Woodbury also reviews geometric foundations in a quick but thoroughly understandable way because they are instrumental to designing with parameters. Interspersed are practice case studies illustrating types of design this new generation of tools can help designers achieve.

I have enjoyed witnessing Dr. Woodbury's teaching of GenerativeComponents over the past years and refer to his design patterns frequently. I hope that this book will inspire instructors in their teaching of parametric design and invoke practitioners' and students' imaginations about new approaches to design.

— Volker Mueller

Acknowledgements

It takes a community to write a book. Though the conception, writing and any errors are mine, I am deeply indebted to many people for ideas, technical help and personal support.

First of all, Onur Gün, Brady Peters and Roham Sheikholeslami bring practice perspectives and fresh voices to their respective sections. Thank you guys!

Throughout my career, I have been blessed with great teachers and mentors. Ron Brand, Gulzar Haider, Jim Strutt, Livius Sherwood, Steve Tupper, Chuck Eastman, Irving Oppenheim, Steve Fenves, Art Westerberg, Mark Allstrom, John Dill and Tom Calvert each taught me enduring and important lessons. I am largely a self-taught writer (and it probably shows). Chris Carlson, Mikako Harada and Antony Radford each helped me sharpen whatever craft I have.

The basis for the book is the ongoing *Patterns for Parametric Design* project in my research group at Simon Fraser University. Without Yingjie (Victor) Chen, Maryam Maleki, Zhenyu (Cheryl) Qian and Roham Sheikholeslami, there would be no patterns and no book. Victor especially has tolerated and met my incessant demands for amendments to the pattern website's programs.

Through many conversations, writing sessions and too much email, Robert Aish and Axel Kilian helped me discern the main themes and structure of the book. I treasure both the intellectual context they provide and, especially, our many differences of opinion about parametric design. In reviewing the book Lars Hesselgren, Axel Kilian, Ramesh Krishnamurti, Volker Mueller, Makai Smith, Rudi Stouffs and Bige Tunçer pointed out its numerous errors and flaws (which I hope I have corrected). Diane Gromala gave indispensable advice on design and typography. Maureen Stone sharpened several graphical tools. The book has hundreds of figures. Roham Sheikholeslami helped immensely in the horrible task of wresting a semblance of visual coherence from unruly symbolic beginnings. Makai Smith, Volker Mueller and the rest of the Bentley team put up with many questions and much nagging about their systems.

Ideas need tempering. SmartGeometry provides the crucible. There is nothing like a room of 200 practical people to tell you when your ideas and explanations

don't work. Over the years Maria Flodin at Bentley brilliantly organized many events – I couldn't do your job Maria, and all of us in SmartGeometry are in debt to you.

In 2004, Caroline Mallinder, then at Taylor & Francis, first approached me with the idea for a book. Her gentle persistence kept the idea in my field of view. My editors, Francesca Ford, Georgina Johnson and Jodie Tierney have given me far too much latitude in time and graphical control. I hope I have not given them too many headaches in return.

I spent much of a sabbatical writing and thank the School of Interactive Arts and Technology at Simon Fraser University (SFU) for this precious time. The Interdisciplinary Research in the Mathematical and Computational Sciences Centre at SFU loaned me a quiet office in which I could avoid my usual tasks at SFU. My departmental colleagues put up with me being distracted for many months and I thank them for their patience. In the summer of 2009, Don and Donna Woodbury lent me their boathouse to edit the book. The sound of wind and waves motivates both work and afternoon naps. In the fall of 2009, I visited Osaka University as the recipient of the Tee Sasada Award. My gracious host Professor Kaga Atsuko gave me less work to do than she might have. I was able to accomplish much writing in a beautiful place.

I've stretched my family's patience. I'm a hermit when I write, ignoring far too much of the rich life around me. Being the wonderful positive people they are, my children still talk to me. I am sure that Gwenda looks forward to having her husband back from his literary affair. Minnie the dog wants more walks.

This work was partially supported through the Canadian Natural Science and Engineering Research Council Discovery Grants Program; Bentley Systems, Incorporated; the MITACS Accelerate program; the Networks of Centres of Excellence program through the Canadian Design Research Network and the Graphics, Animation and New Media Network; and the BCcampus Online Program Development Fund.

Oh, yes. LaTeX, which invokes joy and despair in equal measure. I could not have written this book without it.

I am deeply grateful for all of this support.

Author's note

Neither fish nor fowl.

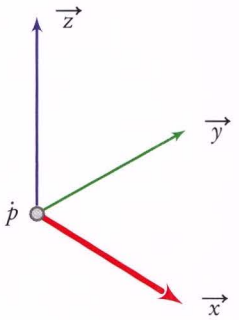
Any *Elements of...* book had better be about practice. And it must be useful and accurate. I write in two worlds. Computer-aided design depends utterly on mathematics and computing. Designers use it to expressive ends. Bringing the two together forces compromise and I've leaned towards design throughout.

I use unusual mathematical notation throughout the book. My mathematician friends may cringe when they see it, but I made a deliberate choice. Designers generally don't do mathematics – they see it, use it and move on. The notation visually describes the objects it denotes: points $\dot{p}$ have dots, vectors $\overrightarrow{v}$ have arrows and frames ${}^{A}_{B}\mathsf{T}$ express both their name and where they sit. Between established convention and clarity for non-mathematicians, I choose the latter every time.

Many references that should appear do not. Again, this is by choice. To get into the bibliography, most books have passed the same test I would hope to pass with this one. Clarity over completeness. Explanation over erudition. Utility over intellectual virtuosity. I aim to explain. This means that I have often left out much detail that would be essential in an academic tome. In doing so, I have selected the material offering the best explanation. For example, the chapter on curves uses the simplest equations and remains with them throughout. It is far from complete; hopefully its omissions let its few key ideas shine through.

The book's longest chapter explains its elements as *Patterns for parametric design*. It contains little computer code – a book is the wrong medium. Programs should be online and executable. The website `www.elementsofparametricdesign.com` provides working code for each of the patterns (and more).

Many people in this world cannot distinguish among red, green and blue. Yet computer-aided design systems freely use these colours to signal direction and type. To address the most common color vision deficiencies (deuteranomaly and deuteranopia), I have avoided using red and green together. The principal exceptions – coordinate systems (aka frames) – appear with their x-axes drawn slightly thicker and with a gap between the end of line and arrow head.



Chapter 1

Introduction

Design is change. Parametric modeling represents change. It is an old idea, indeed one of the very first ideas in computer-aided design. In his 1963 PhD thesis, Ivan Sutherland was right in putting parametric change at the centre of the Sketchpad system. His invention of a representation that could adapt to changing context both created and foresaw one of the chief features of the computer-aided design (CAD) systems to come. The devices of the day kept Sutherland from fully expressing what he might well have seen, that parametric representations could deeply change design work itself. I believe that, today, the key to both using and making these systems lies in another, older idea.

People do design. Planning and implementing change in the world around us is one of the key things that make us human. Language is what we say; design and making is what we do. Computers are simply a new medium for this ancient enterprise. True, they are the first truly active medium. As general symbol processors, computers can present almost limitless kinds of tools. With craft and care, we can program them to do much of what we call design. But not all. Designers continue to amaze us with new function and form. Sometimes new work embodies wisdom, a precious commodity in a finite world. To the human enterprise of design, parametric systems bring fresh and needed new capabilities in adapting to context and contingency and exploring the possibilities inherent in an idea.

What new knowledge and skill do designers need to master the parametric? How can we learn and use it? That is what this book is about. It aims to help designers realize the potential of the parameter in their work. It combines the basic ideas of parametric systems with equally basic ideas from both geometry and computer programming.

It turns out that these ideas are not easy, at least for those with typical design backgrounds. Mastering them requires us to be part designer, part computer

scientist and part mathematician. It is hard enough to be an expert in one of these areas, yet alone all. Yet, some of the best and brightest (and mostly young) designers are doing just that – they are developing stunning skill in evoking the new and surprising. Mostly, the book is about the idea that *patterns* are a good tool for thinking about and using parametric modeling. Patterns are themselves another old idea. A *pattern* is a generic solution to a shared problem. Readers with architecture backgrounds will find this definition more modest and limited than is common in the field. It is perhaps more familiar to those with a software background. Using patterns to think and work may help designers master the new complexity imposed on them by parametric modeling.

Patterns work when grounded in practice. I'm not an architectural practitioner. So I asked three young and thoughtful practitioner/researchers to demonstrate how they and their firms resolved novel and complex design situations using parametric modeling. Onur Yüce Gün, Brady Peters and Roham Skeikholeslami responded with well-considered and crafted chapters.

My hope is that the book's ideas and explanations foster both understanding and meaningful action in the human enterprise we call design.

The idea of the book originated in 2003, when Robert Aish suggested to me that both new and expert designers needed better explanations of parametric design. In 2005 and 2006, three of us, Robert, Axel Kilian and I met several times to draft ideas for a possible book. Our aim was both broad and high. Events transpired for each of us so that authorship took different directions. For me, the result narrowed our original aspirations into this book. Perhaps its focus and voice will be useful in what is surely a growing body of research, writing and computer code about parametric design.

Who should read this book?

If you are a practitioner using parametric design, you will find many manuals and tutorials, both in print and online. Mostly, these provide lists of commands or detailed, keystroke-by-keystroke instructions to achieve specific tasks. These may help you see what the tool can do, but are unlikely to teach much about how you can adapt it to new situations or how to extend your skills. They show you how to do small things, but leave the next steps to your imagination and skill. For you, the book provides foundational geometry for expressing your own models and models of computing particular to parametric systems. Mostly though, it provides patterns, which you can adopt and adapt to the problems at hand. The trick to using the book is to see patterns in your problem, that is, to learn to divide your work into parts that can be cleanly and clearly resolved and then combined into a whole. The case studies may help you see how others have woven parametric thinking and design into entire projects.

If you are a student learning parametric design, your aim is the practitioner's craft. Everything relevant to the practitioner applies to you too. Design can

only be learned by doing. "Talkitecture" is a derogatory term, reserved for those who discuss but do not draw. Don't draw it onto yourself. You need more; particularly, you need to understand how parametric systems work; how their structure makes them perform and how people have used and are using them to do design. The middle chapters of the book may have special meaning for you.

If you are a teacher, you will find strategies here. I believe that we teachers do our best work when we attend to all aspects of design, from vocational skill, through technique and strategy, and all the way to helping our charges discover their muse. The book aims mostly at the middle ground, at linking underlying skills with the higher order understanding needed for good design. Patterns do yeoman's work in this enterprise, at least for the hundreds I have taught and the dozens of tutors who have worked with me in pattern-oriented courses.

If you are a CAD system developer, I believe that you will find some of what is missing in contemporary systems. Without exception, the market provides systems with wonderful capability, cleverly constructed and often with nifty human-computer interfaces. Largely because the needed knowledge is not yet available, current systems are of little help with strategizing, reflecting and developing individual and group practice. As software design patterns have done for software engineering, perhaps the patterns here can suggest new ways of solving the compositional problems that are at the heart of making systems scale in complexity, both of model size and human use.

Almost all who use parametric modeling are amateur programmers . I use the word "amateur" in its literal and complimentary sense, describing one who has interest and skill in an area, but who lacks formal education in it. Amateur and professional programmers differ in more than expertise. Amateurs tend to work on programs that relate to current work tasks, write short programs, use simple data structures and create sparse documentation. Amateurs prefer a copy-and-modify style in their programming work, in which they find, skim, test and modify code until it works for the task at hand. Amateurs satisfice – they leave abstraction, generality and reuse mostly for "real programmers". Professionals might decry such practices, but they cannot change them. Amateurs program because they have a task to complete for which programming is a good tool. The task is foremost, the tool need only be adequate to it. Amateurs write most programs used in our world. Yet almost all programming tools are designed for the professional and are overly complex for the tasks amateurs attempt. If, like almost all designers, you are an amateur programmer, you will find in the book's patterns ideas and techniques for achieving your programming tasks.

It is ironic that this book for amateurs is itself a work of amateur programming. Almost all of its figures were created using GenerativeComponents®, itself a parametric modeling system. Rather than rely on the tedious and limited image export capabilities available in the host system at the time, I wrote a system that output code from the parametric modeler to the Ti*k*Z/PGF graphics macro

package in the LaTeX package in which I typeset the book. This system has three parts. The first is DR, a graphics package much simpler than TikZ/PGF, that provides just the functions I needed. DR translates function calls to TikZ/PGF calls. The second is code in the parametric modeler's scripting language that uses the DR package to describe figures for the book. The third is an Excel® spreadsheet interface to the parametric modeler, so that all of the figures can be described in a single Excel® worksheet. Outside of this sytem, I programmed many macros in LaTeX, largely to gain control over page layout. This sometimes messy code suffices for producing the book. Though I know how to make it general (and know how much work that would take), I focused on the book. The code will need work if it is ever used for another purpose. So be it.

Lastly in this introduction, I must explain the title. In 1919, William Strunk first published *The Elements of Style*,[1] a brilliant, and brilliantly short, book giving strategies for effective writing. Much in that book pertains to writers today. Its clear imperative voice is remarkably similar to much writing about design patterns. Strunk himself borrowed the title; in 1857, John Ruskin (1857) had published the wordy and minimally graphical *Elements of Drawing*. Seeing an obvious good idea, many other authors have undertaken *The Elements of...* books on topics ranging across colour (Itten, 1970), cooking (Ruhlman, 2007), ecology (Smith and Smith, 2008), graphic design (Williams, 2008)[2], interaction design (Garrett, 2002), mentoring (Johnson and Ridley, 2008), programming (Gamma et al., 1995), rhetoric (Maxwell and Dickman, 2007; Rottenberg and Winchell, 2008), typography (Williams, 1995, 2003; Bringhurst, 2004) and, of course, writing (Flaherty, 2009). Strunk and all subsequent authors had a strong precedent in *Euclid's Elements* written *circa* 300BC. A work and writing style could not be more deeply embedded in our culture. Absent an original idea, go with one that works. So I took Euclid's and Strunk's leads, with a twist. I make two points in omitting the "The". First, the field is young, and I would commit a ludicrous error in implying that I cover anything like a complete set of ideas. Second, my premise for design patterns is that they are important only if useful, and useful only if used. The way people use patterns is to try them out, reflect on them and change them. For me, the set will never be complete. The definite article "The" might well be replaced by the indefinite "Some". But no article is shorter still.

[1] My personal copy is the 1959 edition, (Strunk and White, 1959)

[2] I have included Robin Williams' books *Non-Designer's Design Book*, *The PC is Not a Typewriter* and *The Mac is Not a Typewriter* in this list. Though not one uses *Elements* in its title, each is completely within the genre and each is a very good book, too!

Chapter 2

What is parametric modeling?

The archetypal design medium is pencil and paper. More precisely: pencil, eraser and paper. The pencil adds and the eraser subtracts. Add a few tools, like a T-square, triangle, compass and scale, and drawings can become accurate and precise models of a design idea. Designers are used to working in this mode; add marks and take them away, with conventions for relating marks together.

Conventional design systems are straightforward emulations of this centuries-old means of work. *Parametric modeling* (also known as *constraint modeling*) introduces a fundamental change: "marks", that is, *parts of a design,* relate and change together in a coordinated way. No longer must designers simply add and erase. They now *add, erase, relate* and *repair.* The act of *relating* requires explicit thinking about the kind of relation: is this point *on* the line, or *near* to it? *Repairing* occurs after an erasure, when the parts that depend on an erased part are related again to the parts that remain. Relating and repairing impose fundamental changes on systems and the work that is done with them.

Many parametric systems have been built both in research laboratories and by companies. An increasing number are present in the marketplace. Certainly the most mature parametric system is the spreadsheet, which operates over a usually rectangular table of cells rather than a design. In some design disciplines, like mechanical engineering, they are now the normal medium for work. In others, such as architecture, their substantial effects started only about the year 2000.

The first computer-aided design system was parametric. Ivan Sutherland's PhD thesis on Sketchpad (1963) provided both a propagation-based mechanism and a simultaneous solver based on relaxation. It was the first report of a feature that became central to many constraint languages – the *merge operator* that combines two similar structures into a single structure governed by the union of all the constraints on its arguments.

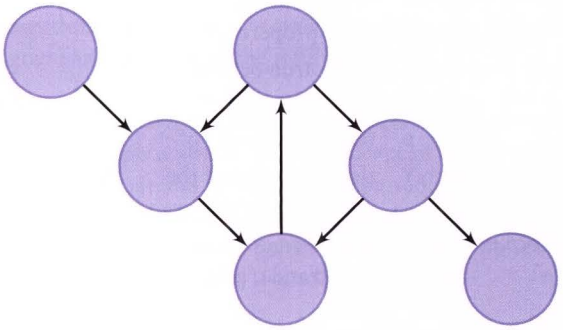
Hoffmann and Joan-Arinyo (2005) provide an overview of different kinds of parametric systems. Each is defined by its approach to constraint solving, and each has its own characteristics and implications for design work. *Graph-based* approaches represent objects as nodes in a graph and constraints as links. The solver attempts to condition a graph so that it divides into easily solvable sub-problems, solves these problems and composes their answers into to complete solution. *Logic-based* approaches describe problems as axioms, over which search for a solution occurs by applying logical inference rules. *Algebraic* approaches translate a set of constraints into a non-linear system of equations, which is then solved by one or a variety of techniques. Constraints must be expressed before they can be solved. Large designs can embody thousands of constraints, which must be clearly expressed, checked and debugged as design proceeds. In addition to their contributions to solving constraints, several research projects have focused on devising clear languages for expressing constraints. Borning's ThingLab (1981) had both graphical and programming constructs for constraints. At the same time, Steele and Sussman (1980) reported a LISP-based language for constraints. *Constraint languages* such as ASCEND (Piela et al., 1993) use a declarative object-oriented language design to build very large constraint models for engineering design. *Constraint management systems*, for example, Delta Blue (Sannella et al., 1993) provides primitives and constraints that are not bundled together and with which the user can overconstrain the system, but must give some value (or utility) for the resolution of different constraints. In this system, a constraint manager does not need access to the structure of the primitives or the constraints. Rather its algorithm aims to find a particular directed acyclic graph that resolves the most highly valued constraints.

Propagation-based systems (Aish and Woodbury, 2005) derive from one aspect of Hoffmann and Joan-Arinyo's graph-based approach. They presume that the user organizes a graph so that it can be directly solved. They are the most simple type of parametric system. In fact, they are so simple that the literature hardly mentions them, focusing rather on more complex systems that address problems beyond those directly solvable with propagation. Discussed in more detail later in this chapter, propagation arranges objects in a directed graph such that known information is upstream of unknown information. The system propagates from knowns to compute the unknowns.

Of all types of parametric modeling, propagation has the relative advantages of reliability, speed and clarity. It is used in spreadsheets, dataflow programming and computer-aided design due to the efficiency of its algorithms and simplicity of the decision-making required of the user. Propagation systems also support a simple form of end user extensibility through programming. This simplicity exacts a price. Some systems are not directly expressible, for instance, tensegrity structures. Also, the designer must explicitly decide what is known and order information from known to unknown. Propagation's simplicity makes it is a good place from which to start building an acount of parametric modeling. The rest of this chapter explains the basic structure and operation of a propagation-based parametric modeling system.

It is useful to be precise with language. The following section defines terms needed for accurate dicsussion of parametric modeling systems. These terms are generic. Any particular propagation-based system has a similar description, though some details will vary.

Graphs are *nodes* connected by *links*. In a *directed graph*, the links are arrows; they explicitly link *arrow tail* or *predecessor* to *arrow head* or *successor* nodes. *Paths* or *chains* are sequences of nodes, each except the last linked to the next node in the path. A graph is *cyclic* if it has paths in which nodes recur.



2.1: A graph has a collection of nodes joined by links. In a directed graph, links join tail (predecessor) to head (successor) nodes. This graph is both directed and cyclic.

In parametric modeling, nodes have *names*. Further, the nodes are *schemata*, that is, they are objects containing *properties*. Each property has an associated *value*, accessed by *dot notation*, that is, by appending to a schema name a period followed by the property name. For example, p.X accesses the X property of point p and has the value stored in that property.

```
Point p
 {
   CoordSystem: cs;
   X:           3.0;
   Y:           4.0;
   Z:           1.0;
 }
```

2.2: A schema for a point named p, with properties for its coordinate system and x, y and z coordinates. The value of the CoordSystem property is the name of a coordinate system node elsewhere in the model that contains the point. Using dot notation, p.X identifies the X property of p and equals 3.0.

The algorithms needed are most simply described by considering only nodes with a single property. Dot notation accesses the single property of such a node, for example, n.Value gives the data held in the Value property of node n. By convention, for single-property nodes, the name of the node itself returns the data in its single property.

A *constraint expression* is a well-formed formula comprising objects, function calls and operators. The objects comprise numbers and property values given with dot notation. When evaluated, constraint expressions result in values.

Constraint expression	Result
3.0	3.0
Sin(30.0)	0.5
false	false
true	true
3.0+Sqrt(5.0)	approximately 5.236067977
p.X + 3.0	the X property of p + 3.0
p.X > 1.0 ? true : false	either true or false depending on the X property of p.
p	the node named "p"
p.CoordSystem.Y	the Y property of the CoordSystem property of p
distance(p,q)+1.618	the distance between p and q plus 1.618

2.3: Examples of constraint expressions.

Property values can be constraint expressions, which in turn can use, that is, *contain* properties from other nodes. Such properties are said to be *contained* by both the property and expression in which they occur. They define the links in the graph. The system ensures that properties and their expressions are evaluated whenever their contained properties change value. Informally, we say that data *flows into* a node when its constraint expressions are evaluated. Nodes (properties) used in a constraint expression are predecessors of the node (property) holding the expression. Links in the graph record that a successor node has a constraint expression that uses a property value from a predecessor node. In single-property nodes, links directly encode property predecessors and successors.

A property can *have* (or be *assigned*) an explicit value or an expression using no property values; such properties are called *graph-independent*. Alternatively it can have a constraint expression using one or more property values from other nodes; such are called *graph-dependent* properties.

A *source node* has no graph-dependent properties and thus no predecessor nodes. A *sink node* is used in no constraint expressions; it has no successor nodes. An *internal node* is neither source nor sink. A node can be both source and sink. A subgraph has its own source and sink nodes.

The system maintains graph consistency by evaluating the expressions in each property. We say that it *evaluates a node* by evaluating all node's properties and thus all its contained expressions. It must choose an order of evaluation so that a

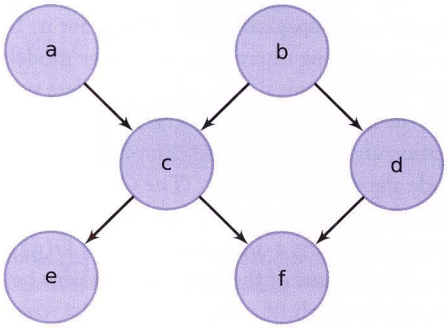
property is evaluated *after* all of its predecessor properties have been evaluated. The graph thus cannot have any cycles, else a node would have to be evaluated in order for it to be evaluated, leading to a contradiction of the algorithm's need for prior evaluation of predecessor nodes.

A *parametric design* (or just *design*) is an *directed graph* of the nodes and links above. A *well-formed* design has no cycles – it is a *directed acyclic graph*.

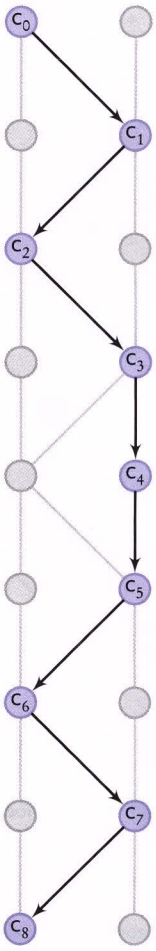
A *chain* is an ordered set C of nodes, with each node $c_i, 0 < i < |C|$ in the chain being an immediate successor of c_{i-1}.

The system uses three algorithms: one *orders* the graph, one *propagates values* through the graph and one *displays* the results.

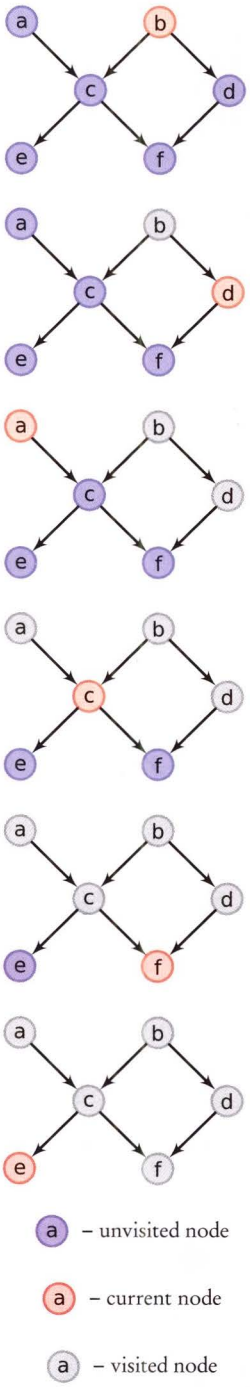
The first algorithm requires a well-formed parametric model and produces a total ordering of the graph nodes. It finds a sequence of nodes such that a node occurs in the order only after all of its predecessor nodes. Such a sequence is called a *topological order*, many of which may exist for a given graph. It does not matter which of the possible total orders is chosen. For a given graph, this algorithm need only be run once.



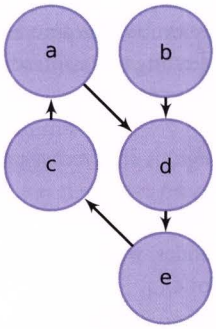
2.5: The tuple ⟨ b,d,a,c,f,e ⟩ is a topological sort – a sequence such that all predecessor nodes of any node occur before the node in the sequence. A graph can have many such, for example, ⟨ a,b,c,d,e,f ⟩, ⟨ a,b,c,d,f,e ⟩ and ⟨ b,a,c,e,d,f ⟩ are among the possible sorts for this graph. Any one suffices as a result of the sorting algorithm, though there may be advantages in the user interface for choosing one over another.



2.4: Nodes c_0 ... c_8 form a chain.



2.7: Using the topological sort ⟨ b, d, a, c, f, e ⟩, the propagation algorithm visits each node in sequence. It evaluates the graph-dependent properties in each node.



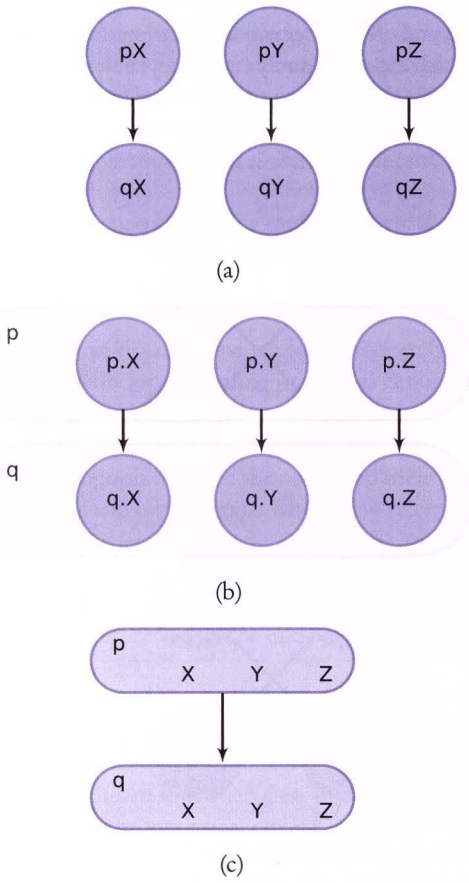
2.6: In a graph with cycles a node can be predecessor of itself. Such a graph cannot be sorted topologically. A naïve algorithm that tried to evaluate it would loop infinitely.

The second algorithm is propagation. In its most simple form, it evaluates each node in a sequence by evaluating its contained constraint expressions. More sophisticated and faster versions of this algorithm only evaluate those nodes that are successors of the nodes that have changed.

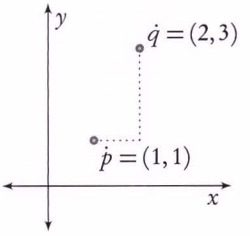
A graph models a usually infinite collection of *parametric design instances* (or just *instances* when clear in context), each of which is defined by assigning values to the graph-independent properties of the graph. To compute an instance order its graph and propagate values throughout it. Both algorithms are simple and efficient, enabling interaction with large models. For instance, the ordering algorithm is topological sort, with worst case time complexity of $O(n + e)$ where n is the number of nodes and e is the number of links in the graph. The propagation algorithm has time complexity $O(n + e)$, presuming that the internal node algorithms are $O(1)$. (The O function is called big-Oh and describes the running time or memory requirement of an algorithm as the size n of inputs grow. Informally, a time complexity of $O(n)$ means that, as n grows, a plot of the running time of the algorithm remains below some non-vertical straight line on the graph. If an algorithm is $O(1)$ the running time is independent of the size of the input n.)

The third algorithm displays the graph symbolically (that is, as nodes and links) and as a model in 3D. A useful, though not universal, convention for symbolic views is to arrange the nodes such that the links flow in a consistent direction (up, down, right or left). Such arrangements reveal the inherent flow of data through a propagation graph. The system invokes the propagation and display algorithms continuously. When the model is sufficiently small that each cycle of these algorithms takes less than approximately 1/30 of a second, designers feel like they are directly interacting with the parametric model.

For systems of one-property nodes the ordering algorithm can be viewed equally as ordering properties and nodes. Multi-property nodes are more complex. Figure 2.8 shows that such a node can be viewed as containing (or *condensing*) a collection of single property nodes and as replacing those nodes in a graph.



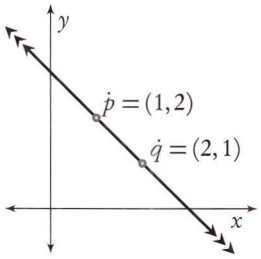
2.8: Multi-property nodes can be viewed as collections of single-property nodes. (a) A parametric model comprising six nodes, representing the coordinates of two points, with one point being a simple translation of the other. (b) Nodes p and q collect the respective single-property nodes. (c) The prior single-property nodes become properties in p and q. One link joining p and q replaces the three prior single-node links.



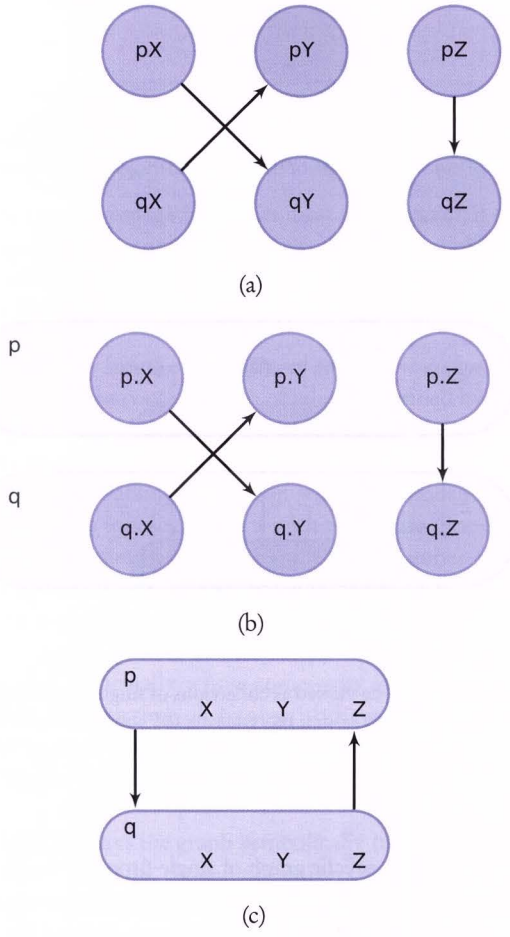
2.9: Point $\dot{q}$ is a simple translation of point $\dot{p}$, in this case $\dot{q}_x = \dot{p}_x + 1$ and $\dot{q}_y = \dot{p}_y + 2$. When $\dot{p}$ moves, so does $\dot{q}$.

A problem arises in that an acyclic graph of single-property nodes can become cyclic when it is condensed into multi-property nodes. For example, consider two points $\dot{p}$ and $\dot{q}$, with the y coordinate of $\dot{q}$ being assigned the x coordinate of $\dot{p}$ and the y coordinate of $\dot{p}$ being assigned the x coordinate of $\dot{q}$. Points $\dot{p}$ and $\dot{q}$ will always define a line segment at 135° to the origin, with its endpoints equidistant from their nearest axis. This graph is acyclic when it comprises single-property nodes, but becomes cyclic when points $\dot{p}$ and $\dot{q}$ are condensed into nodes. Further, if multi-property nodes are used from the outset, then some models that "should" be expressible are not due to the acyclic constraint of the topological ordering algorithm (Algorithm #1). Some practical solutions to this problem are to define the sorting and propagation algorithms to work over properties; or to accept the inexpressibility of some apparently sensible models. There are advantages and disadvantages to both approaches.

Propagating over properties yields a larger and more "complete" range of expressible models and often faster model updating. It can create problems in user interfaces, which most often rely on the acyclic constraint to make model visualizations readable. Propagating over multi-property nodes can simplify the user interface, but also can result in slower updates and confusion when a modeling step that "should" work fails for no apparently sensible reason.



2.11: The y property of point $\dot{p}$ depends on the x property of point $\dot{q}$ and vice versa. When $\dot{p}_x = \dot{q}_x$, $\dot{p}$ and $\dot{q}$ coincide, otherwise they define a line at 135° to the x-axis.



2.10: Condensing single-property nodes into multi-property nodes can produce cyclical graphs. (a) A parametric model comprising six nodes, representing the coordinates of two points, with the y coordinates of each point being the x coordinate of the other. (b) Points p and q collect the respective single-property nodes. (c) The prior single-property nodes become properties in p and q. The three links joining the single-property nodes are replaced by two links joining p and q to form a cycle.

Multi-property nodes present major advantages in what computer scientists call *encapsulation* and *data abstraction*. With them, data describing a distinct conceptual object can be stored in one logical place; multiple operations can

be defined over this data; logical relations among the data can be automatically maintained; and data access can be made uniform. Figure 2.2 on page 13 shows a simple multi-property node comprising a coordinate system and the three coordinates needed to define a point. It demonstrates two key ideas: first, that properties can contain (refer to) other multi-property nodes (the cs held in the CoordSystem property); and second, that nodes can be *typed.*

Dot notation extends to properties holding multi-property nodes. For example, the *path* notation p.CoordSystem.X accesses the x coordinate of the coordinate system held in p.CoordSystem.

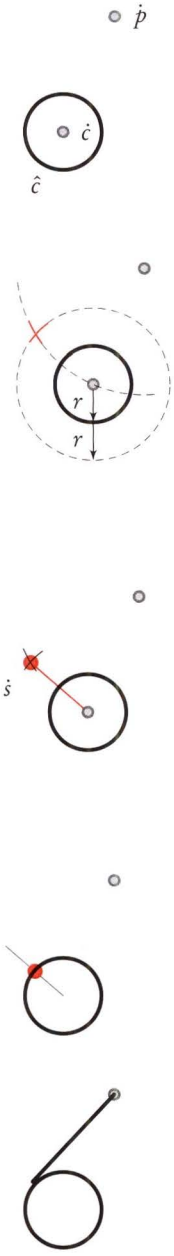
Typed nodes are *instances* of *types.* A type is a template specifying a property set and a set of *update algorithms* for computing properties defined within the type.

We take a type and its set of properties to represent a concept or object in the world. It is useful to distinguish between the nodes and their properties and the corresponding objects and properties to which they refer. By convention, we render nodes and properties in a sans-serif font and their corresponding objects in *italicized* (mathematical) notation. For example, a node named p of type Point might have properties X, Y and Z. The corresponding point $\dot{p}$ has coordinates $\dot{p}_x$, $\dot{p}_y$ and $\dot{p}_z$ respectively. The node p *represents* the concept or object $\dot{p}$.

An update algorithm uses some node properties to compute others. In the scope of a node, those properties whose values are computed by an update algorithm are *node-dependent* (or just *dependent* when clear in context). Node-dependent properties are successors to the input properties to the update algorithm. The node-dependent properties are determined by the update algorithm and cannot have a user-defined value or expression attached to them. All other properties are *node-independent* (or just *independent*). An instance of a type selects one of the available update algorithms as an *active update algorithm.*

```
Point p
 update byCartesianCoordinates
 {
   CoordSystem: cs;
   X:           q.X + 2.0;
   Y:           3.0 * 2.0;
   Z:           1.0;
   Azimuth:     dep Atan2(X,Y)      = 51.13;
   Radius:      dep Sqrt(X*X + Y*Y) = 5.0;
   Height:      dep Z               = 1.0;
 }
```

2.12: A node of type Point with a ByCartesianCoordinates update algorithm may have a user-defined constraint expression in each of its X, Y or Z properties. In contrast, the constraint expressions in its Azimuth, Radius or Height properties are given by its update algorithm. In this figure the CoordSystem and X property are graph-dependent and the Y and Z properties are graph-independent.



2.14: Steps in constructing a line through a point $\dot{p}$ and tangent to a circle $\hat{c}$ with centre $\dot{c}$ and radius r. Find the intersection $\dot{s}$ of two circles, with centres at $\dot{p}$ (radius $|\overrightarrow{\dot{p}\dot{c}}|$) and $\dot{c}$ (radius $2r$). Intersect the circle $\hat{c}$ and $\overline{\dot{s}\dot{c}}$ to find the tangent point.

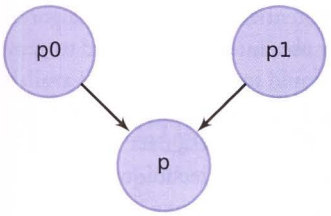
Even objects as simple as points have multiple update algorithms. For instance, a point node might include its containing coordinate system CoordSystem, its X, Y and Z coordinates and, in cylindrical coordinates, its Azimuth, Radius and Height. A ByCartesianCoordinates update algorithm would require that the CoordSystem, X, Y and Z properties be defined and use these to compute the point node's Azimuth, Radius and Height properties. Conversely, using a ByCylindricalCoordinates update algorithm would use the CoordSystem, Azimuth, Radius and Height properties to compute values for its X, Y and Z properties.

```
Point p
 update byCartesianCoordinates
 {
   CoordSystem: cs;
   X:           3.0;
   Y:           4.0;
   Z:           1.0;
   Azimuth:     dep Atan2(X,Y)       = 51.13;
   Radius:      dep Sqrt(X*X + Y*Y) = 5.0;
   Height:      dep Z                = 1.0;
 }

Point q
 update byCylindricalCoordinates
 {
   CoordSystem: cs;
   X:           dep Radius*cos(Azimuth) = 3.0;
   Y:           dep Radius*sin(Azimuth) = 4.0;
   Z:           dep Height              = 1.0;
   Azimuth:     51.13;
   Radius:      5.0;
   Height:      1.0;
 }
```

2.13: Different update algorithms imply different sets of node-dependent and node-independent properties. The two points p and q are at the same location. Using the ByCartesianCoordinates update algorithm, the properties CoordSystem, X, Y and Z are independent. The node-dependent properties are marked with the keyword dep. Using the ByCylindricalCoordinates update algorithm, CoordSystem, Azimuth, Radius and Height are independent.

A node may use several nodes in its contained expressions, that is, it can be a successor of several nodes. It can also use several properties from each used node; a link thus indicates that one or more properties from a predecessor node are used in the expressions within a node.

```
Point p
 update byCartesianCoordinates
 {
   CoordSystem: cs;
   X:           (p0.X+p1.X)/2.0;
   Y:           (p0.Y+p1.Y)/2.0;
   Z:           (p0.Z+p1.Z)/2.0;
 }
```



2.15: A node may depend on several nodes and several properties within itself. For instance, the constraint expressions in point p use the properties in points p0 and p1, which, by this fact, are predecessors of p. Evaluating these expressions places the point $\dot{p}$ equidistant from and collinear with points $\dot{p}_0$ and $\dot{p}_1$, that is, the midpoint of a line segment between $\dot{p}_0$ and $\dot{p}_1$.

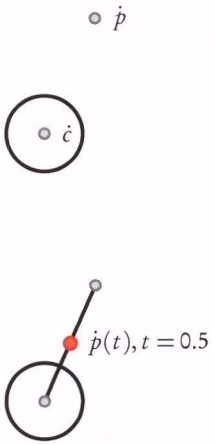
Constraint expressions can be written in a way that expresses the flow of data. In the example above, $\dot{p}$ depends on $\dot{p}_0$ and $\dot{p}_1$, by taking the average, that is, the expression, $\dot{p} = \frac{\dot{p}_0 + \dot{p}_1}{2}$. Reversing the order of the expression and using an arrow to indicate data flow gives the following expression.

$$\frac{\dot{p}_0 + \dot{p}_1}{2} \rightarrow \dot{p}$$

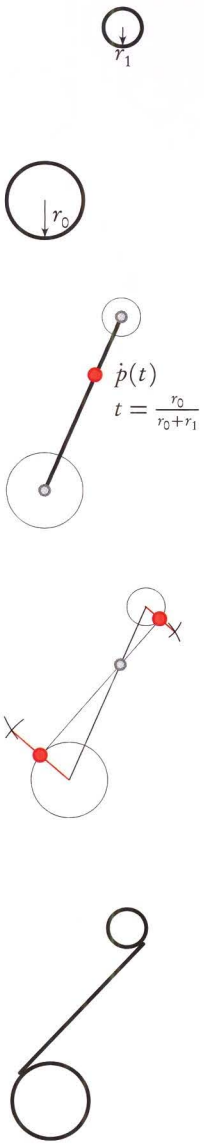
This vector equation expands to an equation for of its coordinates. These are shown below as parametric modeling constraint expressions.

```
(p0.X + p1.X)/2 → p.X

(p0.Y + p1.Y)/2 → p.Y

(p0.Z + p1.Z)/2 → p.Z
```

Dataflow visualization of constraint expressions provides a more accurate view of property values; one that reverses their usual reading. In programming, the common view of a property value is that it *holds* or *contains* the object it names. In parametric modeling, a more insightful view is that a property value *uses* the nodes named within it. Such nodes are predecessors of the property in the model. Dot notation therefore gives access to only those parts of a model that precede (are upstream from) the property. An expression in dot notation records a chain of nodes, starting at the bottom of the chain. The notation provides no direct way of discovering the nodes (properties) that use a particular node (property). If provided, such *back links* must be computed by the modeler itself.



2.16: An alternative construction sequence for a line through a point $\dot{p}$ and tangent to a circle $\hat{c}$ with centre $\dot{c}$. Draw a circle centred on the midpoint of $\dot{p}$ and $\dot{c}$. Intersect this circle and the circle $\hat{c}$ to find the tangent point.



2.17: To construct a line tangent to two circles find the parametric point $\dot{p}(t)$ between the circle centres that divides the centre-to-centre distance proportional to the circle radii. Using $\dot{p}(t)$ replaces several steps of Euclidean construction. Use Figure 2.14 or 2.16 to construct the tangents from $\dot{p}(t)$ to each circle.

Propagation is by far the most simple form of parametric modeling. Over other bases for modeling, it has the major advantage of generality. Update algorithms can compute anything (or at least, anything computable), whereas other schemes restrict their domain, for example, to real-valued expressions. With it, many constructs that designers would like to use become difficult to express. For example, computing the two lines tangent to two circles requires a multi-step geometric construction unless a specific update algorithm for such lines exists. Some constructions require cyclical networks and these can only be solved with global graph techniques. The number of potentially useful update algorithms boggles the mind and would devastate any user interface that tried to provide them all. Even if a huge set of algorithms could somehow be made available and accessible in a system, geometry is too big a topic and design too adventurous an enterprise to be fully covered. The reality is that designers will work at the boundaries of any system and need a combination of techniques to do so. Two key techniques are *geometric construction* and *programming*.

Geometric construction involves making sequences of simple operations to solve problems. It is the child of the compass and straight-edge constructions of Euclidean geometry, but adds the entire set of parametric update methods to the primitive operations of this ancient system. Almost all constructions can be done in different ways. For example, Figures 2.14 and 2.16 show distinct ways to construct a tangent from a point to a circle. While brevity is important (see Figure 2.16), sometimes longer solutions are easier to find and may give new and perhaps valuable insight (see Figure 2.14). Once a good geometric construction has been discovered, it can be used in other, more complex constructions, for example, in finding the tangent line between two circles (see Figure 2.17).

Programming is writing algorithms that either build models or work as update algorithms in their own right. Both construction and programming are foreign to most designers. The last half of the 20th Century saw a dramatic decline in teaching the closest geometric topic, descriptive geometry; and a modest and erratic introduction of programming. Parametric modeling and contemporary design conspire to demand both of these skills. Before addressing either of the technical skills of programming or geometry, the next chapter outlines how designers are using parametric modeling as they work.

Chapter 3

How designers use parameters

The generic description of parametric modeling in the previous chapter defines important technical terms and structures, but does not speak to the effects of such a system on design work. This chapter sketches how parametric design work changes what designers do and what they must think about while they are doing it. The treatment is mainly descriptive. It derives from the properties of parametric systems themselves; from my own knowledge of computation and design; but mostly from working, over several years, with designers using and learning parametric systems.

3.1 Conventional and parametric design tools

In conventional design tools it is "easy" to create an initial model – you just add parts, relating them to each other by such things as *snaps* as you go. Making changes to a model can be difficult. Even changing one dimension can require adjusting many other parts and all of this rework is manual. The more complex the model, the more work can be entailed. From a design perspective, decisions that should be changed can take too much work to change. Tools like these can limit exploration and effectively restrict design.

On the other hand, erasing conventional work is easy. You select and delete. Since parts are *independent*, that is, they have no lasting relationship to other parts, there is no more work to do to fix the representation. You might well have to fix the design, by adding parts to take the place of the thing erased or adjusting existing parts to fit the changed design.

Since the 1980's, conventional tools have used the ubiquitous generic concepts of *copy, cut* and *paste.* These combine erasure and addition of parts to support rapid change by copying and repositioning like elements. Copy, cut and paste work in conventional design precisely because of part independence.

Parametric modeling aims to address these limitations. Rather than the designer creating the design solution (by direct manipulation) as in conventional design tools, the idea is that the designer establishes the *relationships* by which parts connect, builds up a design using these relationships and edits the relationships by observing and selecting from the results produced. The system takes care of keeping the design consistent with the relationships and thus increases designer ability to explore ideas by reducing the tedium of rework.

Of course, there is a cost. Parametric design depends on defining relationships and the willingness (and ability) of the designer to consider the relationship-definition phase as an integral part of the broader design process. It initially requires the designer to take one step back from the direct activity of design and focus on the logic that binds the design together. This process of relationship creation requires a formal notation and introduces additional concepts that have not previously been considered as part of "design thinking".

The cost may have a benefit. Parametric design and its requisite modes of thought may well extend the intellectual scope of design by explicitly representing ideas that are usually treated intuitively. Being able to explain concepts explicitly is a part of at least some real understanding.

Defining relationships is a complex act of thinking. It involves strategies and skills, some new to designers and some familiar. The following sections outline some of these strategies and connect them with what designers already have in their repertoire. The first section, entitled *New Skills,* outlines the small-scale, technical knowledge and craft in evident use by effective parametric modeling practitioners. The second section, entitled *New Strategies,* steps slightly closer to design to sketch the new tasks that designers can and do undertake with the new tools. Both sections are descriptive, not normative. By this I mean they are based on observing and working with designers using parametric modelers, not on surmising what designers might, in some sense, need to know.

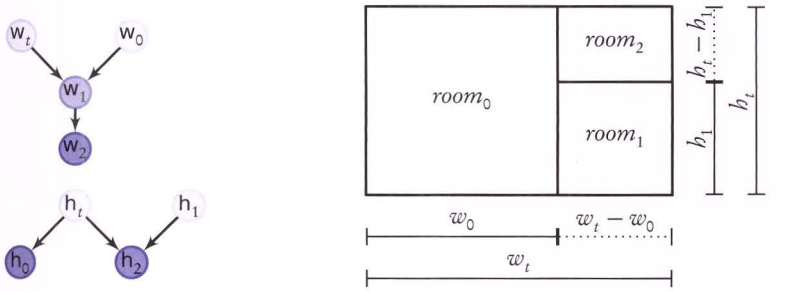
3.2 New skills

Drawing is a skill. Combining multiple orthographic and perspective sketches to reveal the implications of a design idea is strategy. Here are six skills held by those who know and use parametric tools. Some have analogues to historical design skills. Others are new to design. Parametric mastery requires them all.

3.2.1 Conceiving data flow

Caveat: The examples in Sections 3.2.1 to 3.2.4 are very simple, almost trivial. This is deliberate. Through simplicity, I hope to explain crucial principles that are easily obscured. **Reader, please bear with me.**

The detailed explanation of propagation-based systems in Chapter 2 reflects a real need to understand propagation in use. Data *flows* through a parametric model, from independent to dependent nodes. The way in which data flows deeply affects the designs possible and how a designer interacts with them. This can be illustrated with a very simple example: a three-room rectangular plan drawn with lines representing walls. In the figures that follow, the propagation graph represents only the dimensions of the rooms; the lines would, in turn, depend on the nodes in this graph. In Figure 3.1, room dimensions are related by open dimension chains. Figure 3.2 shows the same set of rooms, with the additional relationship that $room_1$ is always square. Here the graph has an new link (between w_1 and h_1) and one fewer source node (h_1). In Figure 3.3, $room_1$ remains square and is a constant proportion of the overall width. This makes w_1 an internal node and introduces the proportional constant a as a new source node.

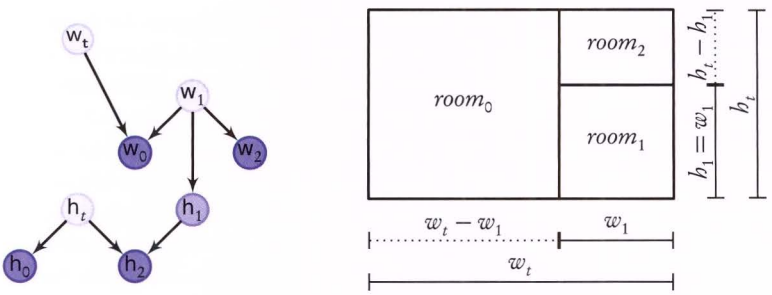


3.1: The plan has $room_{0...2}$. Each $room_i$ has a width w_i and a height h_i. The total width is w_t; the total height is h_t. Dimensions w_t and w_0 are independent. Dimensions w_1 and w_2 are dependent: $w_t - w_0 \rightarrow w_1$ and $w_1 \rightarrow w_2$. Dimensions h_t and h_1 are independent, whereas h_0 and h_2 are dependent: $h_t \rightarrow h_0$ and $h_t - h_1 \rightarrow h_2$. An increase in h_t results in $room_1$ remaining the same height: $room_0$ and $room_2$ expand to take up all of the new space. For graphical clarity, the floor plans omit the simple relations of equality between dimensions where such are implied by the drawing, for instance, $w_2 = w_1$.

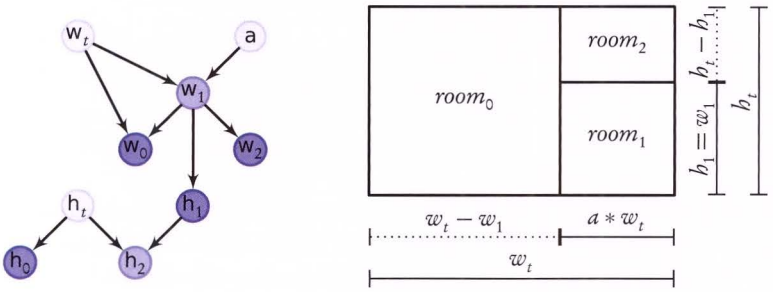
a – source node a – internal node a – sink node

Conceiving, arranging and editing dependencies is the key parametric task.

To make things more complex, dependency chains – several nodes in sequential dependency – tend to grow. Figure 3.4 expands the examples above it to include the points and lines representing the floor plan. It shows that long dependency chains are the norm, and that visualizing the graph can become difficult.

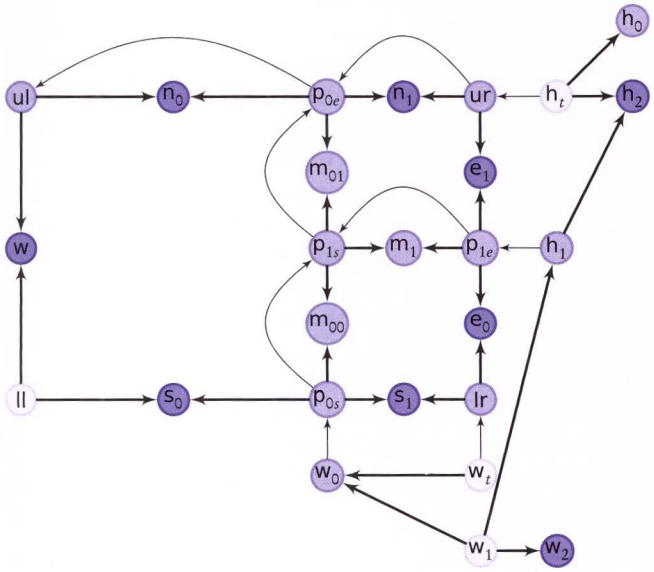


3.2: The main interactive difference between this design and that of Figure 3.1 is that $room_1$ is always square and its size is explicitly controlled. The propagation graph is distinctly different.



3.3: In addition to the constraints in the design in Figure 3.2, the ratio between w_t and w_1 remains constant: $a * w_t \rightarrow w_1$.

Designers use dependencies in combination to exhibit some desired aggregate form or behaviour. Dependencies may correspond to geometric relationships (for example, between a surface and its defining curves), but are not restricted to this and may in fact represent higher order (or more abstract) design decisions. Parametric approaches to design aim to provide designers with tools to capture design decisions in an explicit, auditable, editable and re-executable form.



3.4: Adding the points and lines defining the floor plan increases both the length of the dependency chains, the overall graph complexity, the difficulty of inventing short, clear, descriptive names, and, especially, the challenge of achieving a readable graph layout. This diagram is based on that of Figure 3.2. It changes the layout of the nodes in the prior graph and breaks the convention of uniform direction of dataflow in favour of a layout that mimics the location of points and lines in the floor plan. Some arcs are curved to avoid intervening nodes. Arcs whose sole purpose is to carry coordinate information are thin. External walls are labeled with n, e, s and w (for *north, east, south* and *west*); and internal walls are prefixed with the character m (largely because this character was not used elsewhere). Finally the node ll (for *l*ower *l*eft) is presumed to be at (0,0); if it were locatable anywhere, arcs would have to go from it to other nodes in the graph.

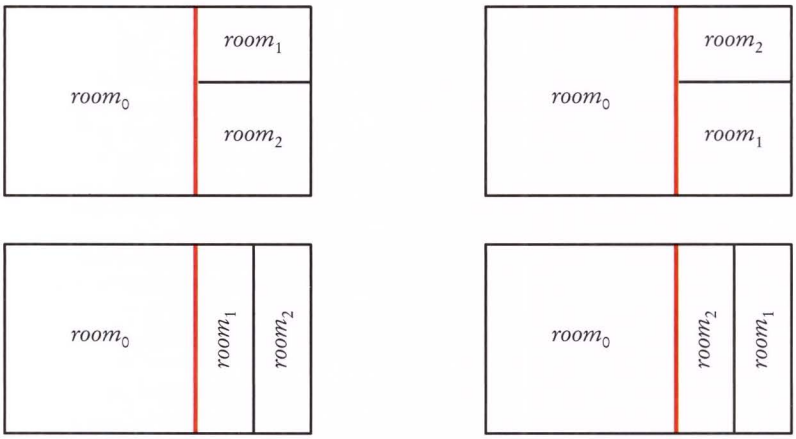
3.2.2 Dividing to conquer

For very good reasons, designers organize their work as *near-hierarchies*, that is, recursive systems of parts with limited interactions between parts. This is a near-universal claim, and it is easy to test. Think of a designed object that is so organized, for example, an automobile organized into body, drive train and electrical systems. Now think of a designed object that is a non-near-hierarchy in some way, either by having only one part or by having extremely complex interactions among its parts. Compare the relative difficulty of imagining each. **See?**

One of the many reasons for near-hierarchies is that the limited interactions among system parts enables a divide-and-conquer design strategy – divide the design into parts, design the parts and combine the parts into an entire design, all the while managing the interactions among the parts. The strategy works best when the interactions are simple.

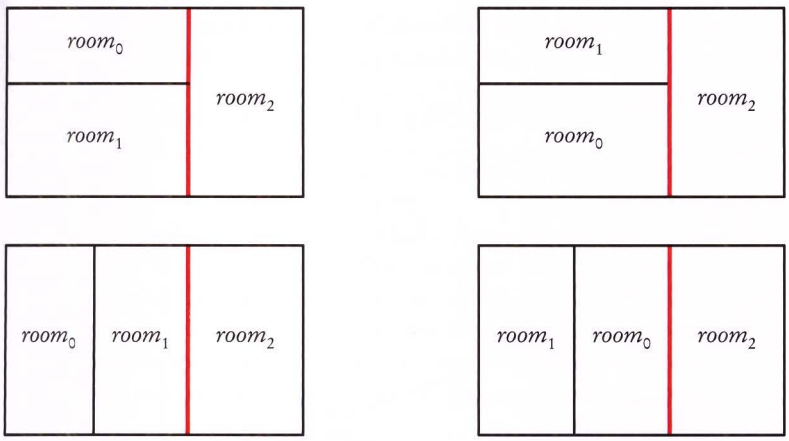
Parametric modeling enables, indeed almost requires, a divide-and-conquer strategy. In building a parametric design, it is easy to keep adding nodes to the graph. A moment comes though, when the graph is too complex to fully grasp. At a much earlier moment, it becomes difficult to explain the graph to another, or to resume work on it after an inevitable interruption (in this situation, there really is "another" – it is you, after you've taken a break and come back with a different memory state). Using a divide-and-conquer strategy is to organize a parametric design into parts so that there are limited and understandable links from part to part. Directional of data flow assures a hierarchical model, with parts higher in the flow typically being assemblies – organizing concepts. Parts at the bottom of the flow usually correspond to physical parts of the design.

Returning to the three-room floor plan, even this seemingly simple design could be given a hierarchical structure. Figures 3.5 and 3.6 show that a decision to model the three rooms in two *wings*, assigning each room to one of the wings, has profound effects on the plans obtainable, particularly when the number of rooms in each wing increases.



3.5: All possible arrangements of an organization into two wings with $room_0$ in the west wing and $room_1$ and $room_2$ in the east wing.

Skilled designers spend much time on developing and refining the near-hierarchical structure of their models. They arrive at parametric design as able practitioners of divide-and-conquer – architects usually organize designs (especially at the construction documentation phase) into technical subsystems. In conceptual design, common design schema separately play on space and tectonics. But skills transfer poorly across domains. The divide-and-conquer of parametric modeling requires knowledge from both the design domain and about how to structure parametric designs so that data flows from part to part in a clear and explainable manner.

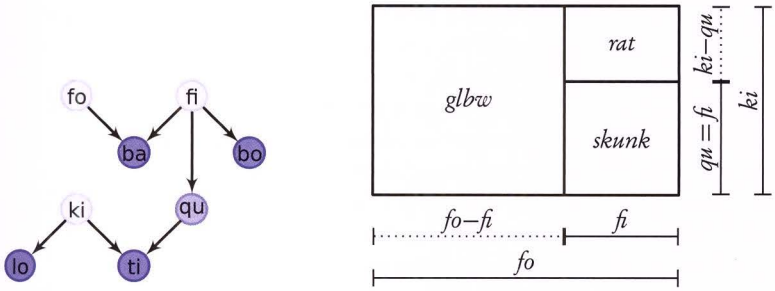


3.6: All arrangements of an organization into two wings with $room_0$ and $room_1$ in the west wing and $room_2$ in the east wing.

3.2.3 Naming

Parts have names. This is designerly practice, not physical law. But there is a good reason for this – names facilitate communication. "The column at grid location E2:S4" is a more reliable way of identifying a particular column than "that square mark a third or the way across and halfway up the sheet".

Parametric modelers spend much time in devising and refining the names of their parts. Simply renaming the rooms and dimensions of the three-room floor plan shows why they do this. Figure 3.7 is identical to Figure 3.2 in all respects, except that the nodes and rooms have been given arbitrary names.



3.7: Confusion reigns with arbitrary names. Even though this figure and Figure 3.2 are identical except for names, it takes much more effort to understand this one.

3.2.4 Thinking with abstraction

The word "abstraction" is laden, that is, its meaning depends on context.

Designers and computer scientists use the term differently.

An abstraction describes a general concept rather than a specific example. In common usage, abstraction is associate with vagueness; it may be hard to infer much from an abstract idea. In design, abstract ideas are often protean, that is, they are used as a base from which to generate many alternatives. In this role, both connotations of the word apply: a general concept can be realized in many ways, and a vague concept can be given many interpretations, each of which may have multiple realizations.

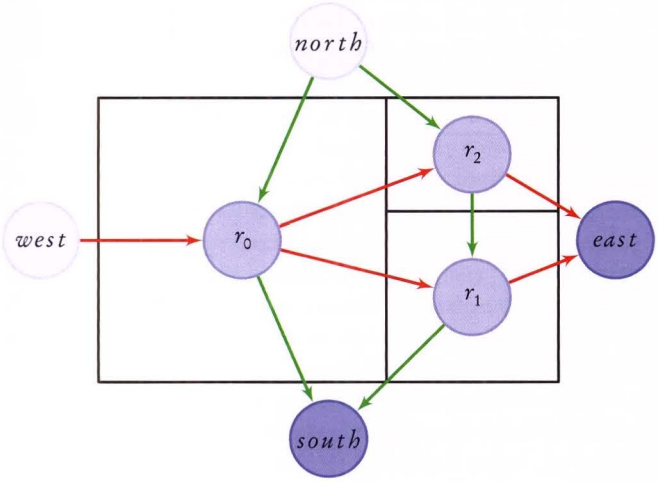
In computer science, abstraction has the first meaning: an abstraction describes of a class of instances, leaving out inessential detail. Computer scientists (and their craftful cousins, programmers) are constantly seeking formalisms and code that apply in many situations. In fact, the utility of a computational idea is deeply linked to its generality – the more often it applies, the more useful it becomes). Designers too know and practice such abstraction. Dimensional modules, structural centrelines and standard details all are media for abstract design ideas.

To abstract a parametric model is to make it applicable in new situations, to make it depend only on essential inputs and to remove reference to and use of overly specific terms. It is particularly important because much modeling work is similar, and time is always in short supply. If part (remember divide-and-conquer?) of one model can be used in another, it displays some abstraction by the very fact of reuse. Well-crafted abstractions are a key part of efficient modeling. For example, in floor plans comprising rectangular rooms, two good abstractions are to consider the rooms and the walls respectively as as nodes. Using rooms as nodes (Figure 3.8) creates two independent subgraphs in the design, one for west-to-east relations and one for north-to-south relations.

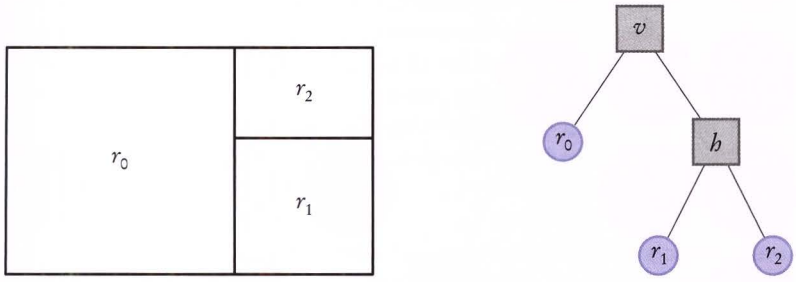
When walls are nodes, as in Figure 3.9, the graph becomes a very simple tree structure of successive subdivisions, either vertical or horizontal, dividing an overall rectangular plan. Each of the four abstractions in this section, based on dimensions (Figures 3.1, 3.2 and 3.3); points and line segments (Figure 3.4); rectangles (Figure 3.8); and walls (Figures 3.9 and 3.10) represents layouts of two-dimensional rectangles; each offers advantages and disadvantages; and, sadly, each requires work to understand, develop and use. Computer scientists use the term *representation* to describe abstractions with mathematical proofs relating properties of the abstraction to properties of a class of objects.

An important form of abstraction for parametric modeling is *condensing* and *expanding* graph nodes. In any graph, a collection of nodes can be condensed into a single node; and graphs with condensed nodes are called *compound graphs.*

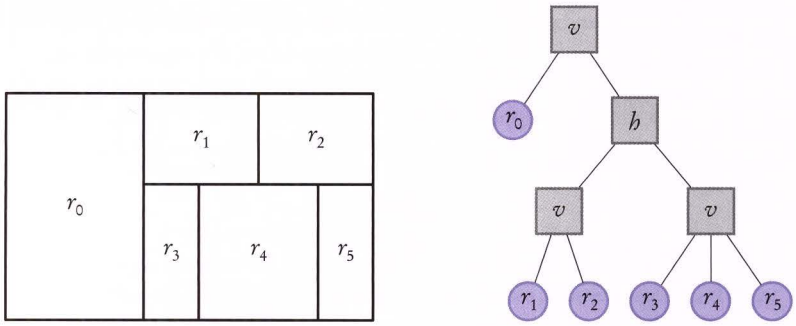
A condensed node can be expanded to restore the graph to its original state. Condensing and expanding implement hierarchy and aid divide-and-conquer strategies. Parametric modelers implement this strategy to create new kinds of multi-property nodes, to support copying and reuse of parts of a graph and thus to build user-defined libraries of parametric models. See Section 3.3.7 on p. 45.



3.8: The *loosely-packed arrangement of rectangles (LOOS) representation* (Flemming, 1986, 1989). Treating each rectangle as a graph node creates two separate subgraphs in the design (west-to-east and north-to-south). There are four distinguished nodes (called *north, east, south* and *west*) that bound the actual rectangles in the design. Each internal node carries two minimum dimensions: one for the west–east direction and one for the north–south direction. The node computes a location of the wall consistent with these constraints. One such algorithm computes every vertical (horizontal) wall location as being as far west (north) as it can possibly be. In addition to its simple graph, the LOOS representation has the benefit of being able to represent every possible layout of rectangles, and provides relatively simple operations for inserting and deleting rectangles.



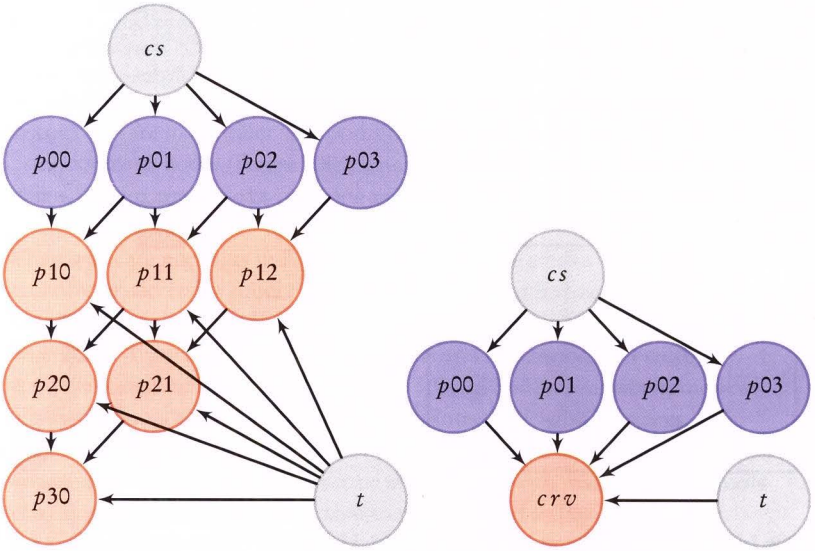
3.9: In the *subregion representation* (Kundu, 1988; Harada, 1997), the representation is a simple tree, with each square node representing both a rectangular region and either a vertical (v) or horizontal (h) wall dividing the region and each round node representing a specific rectangle. This representation is extremely simple and has an easily understood dimensioning scheme; each v and h node contains an independent parameter.



(a)

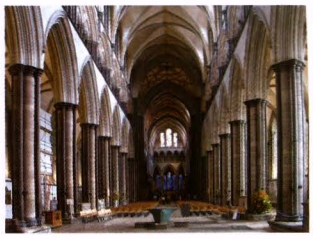
(b)

3.10: The subregion representation can (a) simply represent complex arrangements of rectangles, and provides a natural hierarchy of regions. Every v and h node of the tree contains a single parameter that specifies the proportion at which the larger rectangle is subdivided. However, walls must completely divide a region; (b) shows that some arrangements cannot be represented at all.



3.11: The graph on the left condenses to the compound graph on the right. The selected collection of nodes (in red) on the left becomes the single selected node on the right.

3.2.5 Thinking mathematically

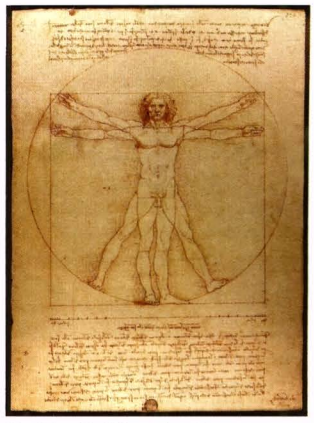


Salisbury Cathedral.
Source: Bernard Gagnon.

Whether conventional or parametric, a CAD model is a set of mathematical propositions. A line object is a statement that the segment between its end points is a part of the model. Mathematical calculations are routinely made by the system: putting new points on the active plane, placing a line tangent to a circle through a point and specifying a point as the centroid of a polygon each model basic mathematical inferences. Designers do more than make collections of propositions: they make *proofs* of their designs. By using constructions such as grids, snaps, circle intersections and tangents, they construct mathematical proofs that the designs thus specified are consequent to their base assumptions. Of course, designers seldom look at their work in this way, and mathematicians might cringe to consider such special-purpose constructions as meaningful proofs. But the analogy stands; in some sense, designers "do" mathematics. Practically though, designers *use* mathematics more than they *do* mathematics. To use mathematics is to begin with established mathematical fact and to rely on it to make a construction or, even more loosely, as a metaphor for a design move. To do mathematics is to derive theorems (new mathematical facts) by inference from prior known statements. The difference is in both intent and practice. A designer sets out to create a design, a description of a special artefact suited to purpose. A mathematician seeks to discover new and general facts from old or new paths of inference to already known facts. Design work is more like McCullough's (1998) digital craft and less like Lakatos's (1991) cycle of proof and refutation. We can quibble about how similar (or not) these two acts are, but there is an essential difference in license taken and understanding sought. Using mathematics to do design requires far less understanding than doing mathematics for its own sake. Note the word "requires". Some designers choose to delve into the mathematics of their work. Sometimes such apparent distraction becomes core to developing a body of work. Other times it follows the time-honoured tradition of curiosity as its own reward.

A sequence of Rasmi domes in *Nasir-Al-Molk Mosque,* Shiraz, Iran. Projection from a base drawing is the main generator for Rasmi domes.
Source: Babak Nikkhah Bahrami.

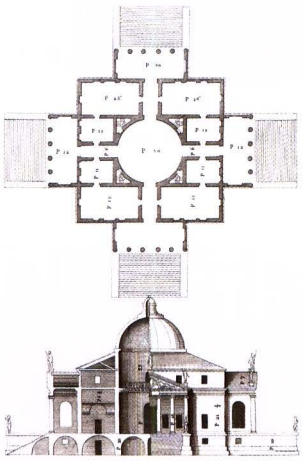
Design has always had practitioners who take more than slight steps towards mathematical maturity. Gothic buildings can be understood and were evidently designed as complex sequences of geometric construction proceeding from a few key dimensions. Traditional Persian Rasmi domes result from projecting a drawing onto a predetermined dome geometry. In Persian, the verb for drawing and the word "rasmi" have the same linguistic root. DaVinci's Vitruvian Man drawing is the centrepiece of a collection of notes on Vitruvius's *The Ten Books on Architecture* (Pollio, 2006).



Da Vinci's *Vitruvian Man.*
Source: Luc Viatour.

Palladio (1742; 1965) expounded on and (sometimes) used proportional systems in his building plans and elevations. Antoni Gaudí limited his form-finding mostly to developable surfaces, to great sculptural effect. Le Corbusier espoused *The Modular*, a manifesto on the play of the golden ratio $\phi = (1 + \sqrt{5})/2$, also the solution to the equation $1/\phi = \phi/(1 + \phi)$, and in turn the division of a line segment such that the small and large parts are in the same ratio as the large part to the whole. Canadian architect James W. Strutt (see Figure 3.2.6) based

his life's work on the play of sphere and polyhedral packings and their duals. Geometric construction and visual clarity are signatures for Foster + Partners. These are statements of historical fact, not judgments on the work obtained. Valuing design for using geometry is circular reasoning at best.

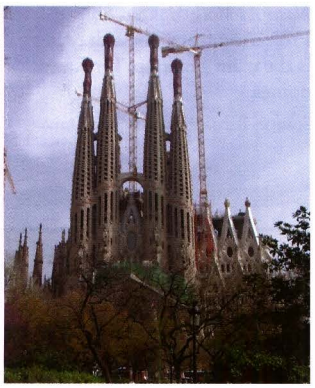


Palladio's *Villa Rotunda.*
Source: Palladio (1965).

Parametric systems can make such mathematics active. By coding theorems and constructions into propagation graphs and node update methods, designers can experience mathematical ideas at play. The once dry ideas of surface normals, cross products, tangencies, projections and plane equations become an essential part of the modeler's repertoire. Active and visual mathematics can become means and strategy to the ends of design.

Modern mathematics is too vast for the lifetime of a single mind. Indeed, it seems too vast for an entire industry. New geometric operators appear slowly in CAD, leaving much design possibility unexplored. For example, in 2009, the mesh subdivision and refinement techniques common in animation systems were only beginning to appear in CAD. The field of computational geometry provides such basic constructs as convex hulls, Voronoi diagrams and Delauney triangulations that would enable new avenues of exploration were they in the CAD toolkit. Parametric modeling enables new mathematical play. Designers know about something these other fields – their demands may well push system developers to richer tools. Section 3.3.5 outlines some of the new strategies in contempory parametric design. Chapter 6 explains some of the fundamental mathematics needed to master parametric modeling.

3.2.6 Thinking algorithmically



Geometrically Antoni Gaudí's, *Temple Expiatori de la Sagrada Família* is an exploration of the possibilities of developable surfaces.
Source: Paolo de Reggio.

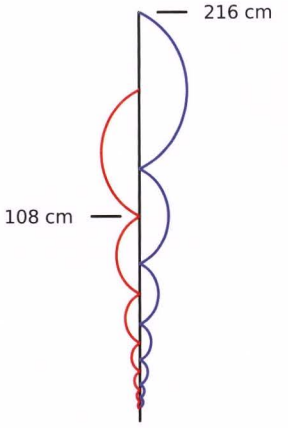
A parametric design is a graph. Its graph-dependent nodes contain either or both update methods and constraint expressions. Both are *algorithms* and can be changed by users, at least in principle. Long practice in using, programming and teaching parametric systems shows that, sooner or later, designers will need (or at least want) to write algorithms to make their intended designs.

It is useful to consider what an algorithm is. There are many definitions. Berlinski (1999) (whose book you should read!) writes on page *xix*

> An algorithm is
>
> a finite procedure,
> written in a fixed symbolic vocabulary,
> governed by precise instructions,
> moving in discrete steps, 1,2,3,...,
> whose execution requires no insight, cleverness, intuition, intelligence or perspicuity,
> and that, sooner or later, comes to an end.

Berlinski's definition is less formal than those you will find in dictionaries and computer science texts, but contains all of the accepted essential elements of an algorithm. Design highlights two of its aspects. The first is "procedure": an algorithm is a process that must be specified step-by-step. Designers largely describe objects rather than processes. The second is "precise": one misplaced character means that an algorithm likely will not work. In contrast, designerly representations are replete with imprecision – they rely on human readers to interpret marks appropriately. It is hardly surprising then that many designers encounter difficulty in integrating algorithmic thinking into their work, in spite of over 30 years of valiant attempts to teach programming in design schools. It is even less surprising that computer-aided design relegates programming to the background. Almost all current systems have a so-called *scripting language*. These are programming languages; developers call them scripting languages to make them appear less foreboding. In almost all of these, to use the language your must remove yourself from the actual task and your accustomed visual, interactive representation. You must work in a domain of textual instructions. This is not surprising either – algorithmic thinking differs from almost all other forms of thought. But the sheer distance between representations familiar to designers and those needed for algorithms exacerbates the gap.

In both conventional and parametric systems, the scripting language can be used to make designs. The language provides functions that can add, modify or erase objects in a model. In addition, parametric systems bring the algorithm closer to design models. They do this by localizing algorithms in nodes of a graph, either as constraint expressions or as update methods. However, designers still must grasp and use algorithmic thought if they are to get the most out of such systems. Chapter 4 summarizes the programmer's craft and shows how and why programming is built into parametric modeling.



The core of the Modulor. The *red series* originates with dimension of 108 cm (putatively at navel height); the *blue series* with 216 cm (the top of an outstretched arm). Each is geometric in the golden ratio $\frac{1+\sqrt{5}}{2}$. Corbusier rounded imperfectly—the Modulor dimensions are not in the golden ratio to the nearest integer.

James W. Strutt's *Rochester House* is based on a close packing of rhombic dodecahedra.

Source: James W. Strutt Family.

The *Albion Riverside* apartments by Foster + Partners combines multiple trigonometric functions to compose an overall form.

Source: Chris Kench.

3.3 New strategies

Conceiving data flow; dividing to conquer; naming; and thinking abstractly, mathematically and algorithmically form the base for designers to build their parametric craft. In this section, I describe strategies that my research group has observed over several years of running courses and workshops in parametric modeling. Our observation techniques have ranged from informal interaction and journaling to structured *participant observer* studies (Qian et al., 2007).

3.3.1 Sketching

The sketch occupies a near sacred place in the design pantheon. A library of books attests to its importance to design, extolls its protean virtues and urges students to learn this all-important skill. Toothy paper and the 2B pencil are among the saints of architectural hagiography. Irony aside, all design teachers know that the student who sketches well tends to do well in the studio; and

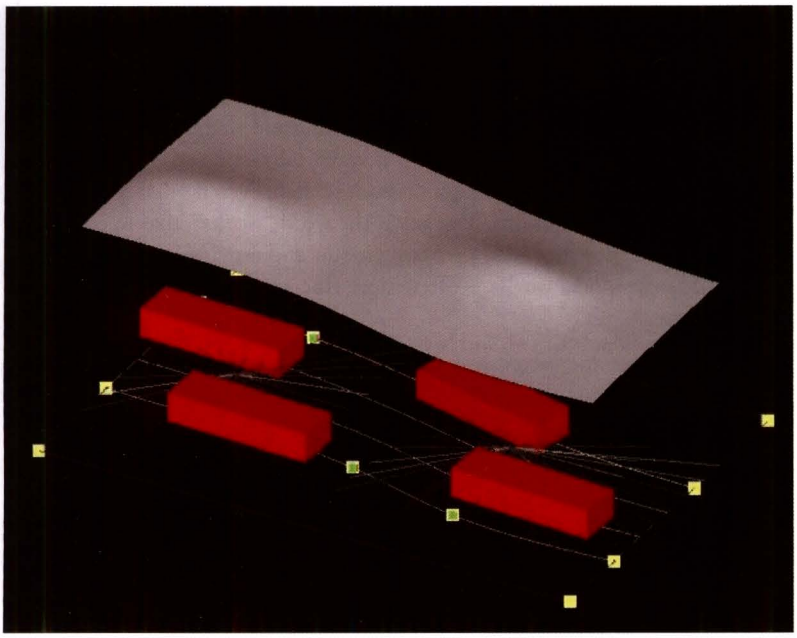
that pencil sketching remains a vital and important tool for design. But what is a sketch? In *Sketching User Experiences*, Bill Buxton (2007) crafts a thorough argument for the qualities and uses of sketches in interaction design. In the chapter *The Anatomy of Sketching* (pp. 111–120), he posits 11 qualities of design sketches. For Buxton sketches are (or have) *quick; timely; inexpensive; disposable; plentiful; clear vocabulary; distinct gesture; minimal detail; suggest and explore rather than confirm; appropriate degree of refinement;* and *ambiguity.* Of these, only clear vocabulary, distinct gesture and appropriate degree of refinement make any reference to the media conveying a sketch. All of the other eight (and much of these three) refer rather to the role of sketches in the design process. Buxton does not have the only or final word on sketching in design, but his voice is both recent and clear. To paraphrase his words: Designers have always sketched. It is how they do their work.

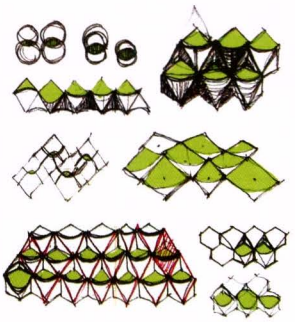
We have known since McLuhan that media and content deeply intertwine; the carrier and carried cannot be pulled apart. Well-mastered, the skill of pencil sketching meets all of Buxton's criteria. But when *taken in their own terms,* so do other media and tools. Unencumbered with the 2B religion, students use the media at hand, and today such media are mostly digital. These fresh newcomers consistently do work that meets all of Buxton's criteria – see Figure 3.12. And their eyes are different. What old-timers like Buxton and I might see as overly determined and graphically definite, the new generation sees as ambiguous and free. If you don't like it, change it! The digital generation might well add the word *dynamic* to Buxton's list.

Parametric models are, by their nature, dynamic. Once made, they can be rapidly changed to answer the archetypal design question: "What if...?" Sometimes a single model replaces pages of manual sketches. On the other hand, parametric models are definite, complex structures that take time to create. Too often, they are not quick. A challenge for system developers is to enable rapid modeling, so that their systems can better serve sketching in design.

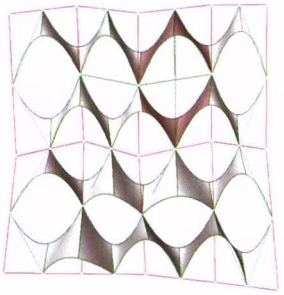
3.3.2 Throw code away

Designers do design, not media. Unless they get seduced by the siren of the parametric tool, they model just what they need to the level of confidence and completeness they need. From project to project, day to day or even hour to hour, they tend to rebuild rather than reuse. In stark contract, much of the toolkit of computer programming (and parametric modeling is programming) aims at making clear code, reducing redundancy and fostering reuse. In the world of professional programming, these aims make eminent sense. In the maelstrom of design work, they give way to such simple devices as copying, pasting and slightly modifying entire blocks of code. Professional programmers would be horrified by such acts. Designers are delighted if the resulting model works, right now.

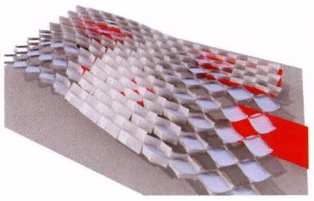




(a)



(b)



(c)

3.12: The initial sketches ((a) & (b) in the margin) led to this parametric sketch, whose purpose was to build and understand how a surface could react to objects underneath it. The screen elements spanning the surface (c) were built, literally, on top of the sketch. The relatively low resolution of the sketch image reflects its ephemeral role in the design process – whatever gets saved in the moment determines the historical record.
Source: Mark Davis and Stephen Pitman.

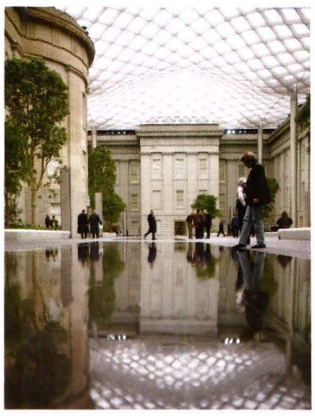
At the 2007 ACADIA conference, Brady Peters presented a paper on the design and construction of a roof over the courtyard of the Smithsonian Institution Patent Office Building (Peters, 2007) by Foster + Partners. During his talk, he showed some of the computer code that generated the design alternatives. It was highly repetitive. Entire blocks of almost identical code appeared again, and again. To an audience question (OK, it was from me) about why he, as a skilled programmer, would not have made his code more clear, he responded simply "I didn't need to do that." Peters wasn't being lazy or uncraftful; he was being a designer. Throw-away code is a fact of parametric design.

3.3.3 Copy and modify

Designers may throw their own models away, but will invest considerable time in finding existing models and using them in their own context. This is hardly surprising. References such as *Architectural Graphics Standards* (Ramsay and Sleeper, 2007a) and their recent digital versions (Ramsay and Sleeper, 2007b)

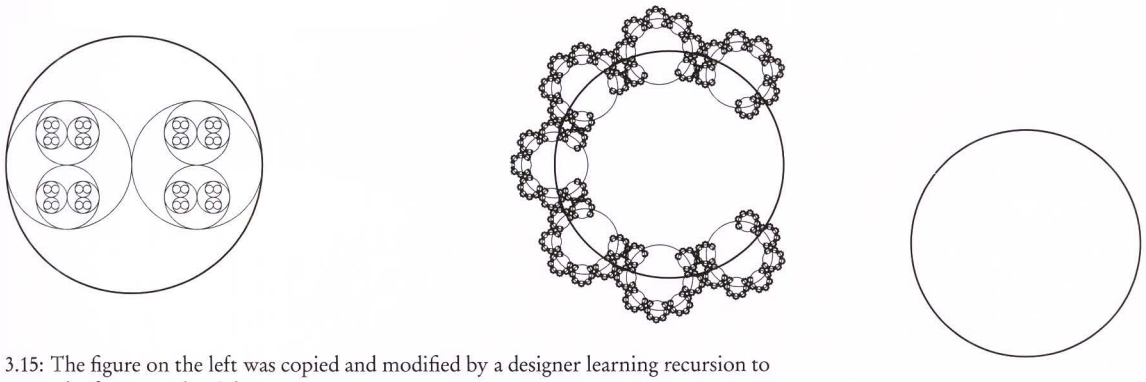
provide exemplary details that, for much design, are the foundation for detailed work. In an engineering design domain, Gantt and Nardi (1992) report script finding and reuse as an important mode of work. Given the additional work that must go into a parametric model, we should expect to see an intellectual trade in models and techniques. As both a learning and enabling tool, existing code reduces the job of making a model. It is typically easier to edit and change code that works than it is to create code from scratch, even if what it produces differs from current intentions. The key is the word "works", that is, code that produces a result. Starting with a working model and moving in steps, always ensuring that the model works, is often more efficient than building a model from scratch.

3.13: The roof over the courtyard of the *Smithsonian Institution Patent Office Building.*
Source: Nigel Young / Foster + Partners.



3.14: Details of the Smithsonian courtyard roof.
Source: Nigel Young / Foster + Partners.

Copy-and-modify is the flip side of throw-away code. Designers show natural reluctance to invest sufficient work in making code that will be clear to others, but they are happy to use such code when it is available. This makes "good" code a treasured community resource. The copy-and-modify strategy requires a community of practice that generates the code. The World Wide Web fosters such communities. Enabled by the fact that pages are written in human-readable HTML (and other languages), web designers often mine existing pages for code snippets that show how to achieve a particular effect. In design, the practice communities are less well developed, but such groups as SmartGeometry have built necessary precursor networks. Vendor publications of books with worked examples partially fill the need for such models and code.



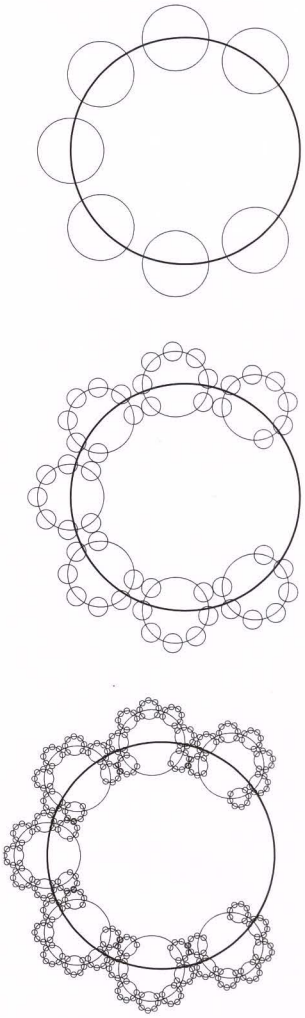
3.15: The figure on the left was copied and modified by a designer learning recursion to create the figure on the right.
Source: Dieter Toews.

3.3.4 Search for form

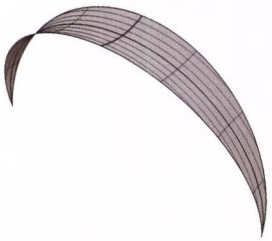
Parametric modeling opens new windows to design. Nowhere is this more evident than with curves and surfaces. These are naturally parametric objects; mathematically they are defined as parametric functions over sets of control points. Conventional systems provide these mathematically motivated controls. In contrast, parametric systems enable a new set of controls to overlay the basis controls. This creates endless opportunities to explore for forms that are not practically reachable otherwise. To the technically-minded, such exploration can appear as play that is both aimless and ungrounded. A broader and longer perspective reveals serious purpose in the play. The history of design can be read as a constantly changing process of exploring for new form-making ideas, using whatever tools and intellectual concepts are at hand. New languages and styles of design require such exploratory play, especially at their early stages. Figure 3.17 shows recent exploratory work by Aranda\Lasch.

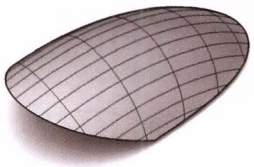
3.3.5 Use mathematics and computation to understand design

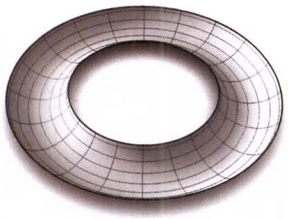
Understanding mathematics (especially geometry) and computation can bring some design concepts into sharp focus. Working with such formal descriptions restricts the range of forms that can be expressed, but links them in a common logic that may be worth the cost paid. For example, taking sections of a toroidal surface yields a surprisingly rich language of form, with the benefit of planar faceting and a limited set of edge lengths for facets.

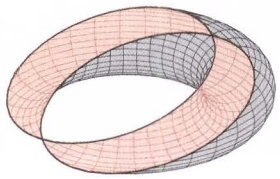


3.16: The same design shown at increasing recursive depth.

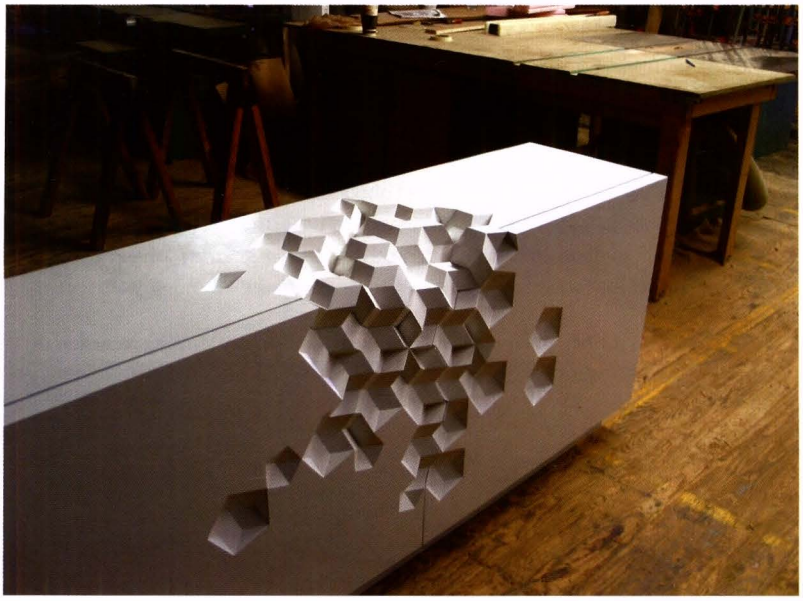






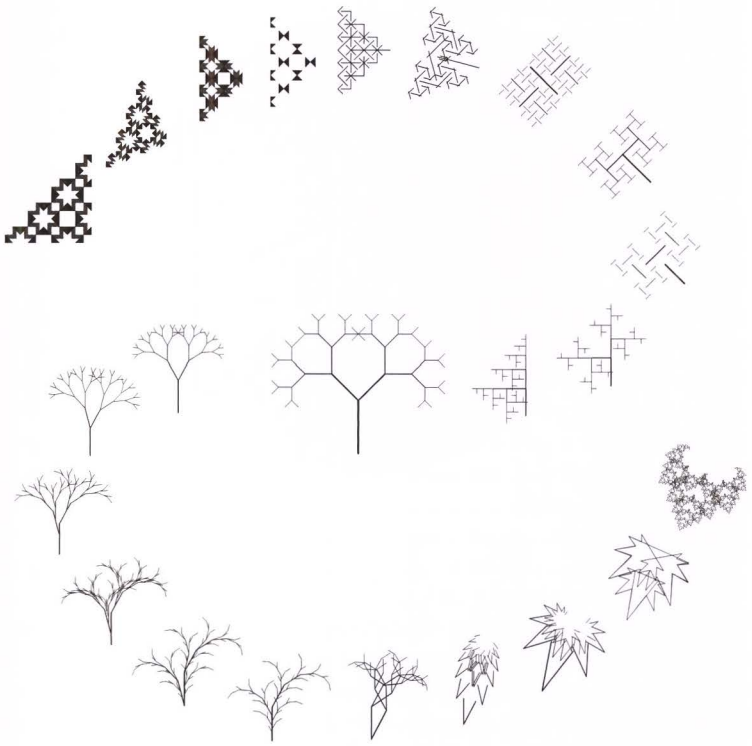


3.18: Sections cut from a torus.



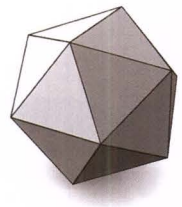
3.17: The quasi-series is about the pursuit of orders that are rigorously modular but wild – almost out of order. Quasicrystals, a new phase of matter discovered in 1984, represent this kind of material structure that hovers on the edge of falling apart. Unlike a regular crystal, whose molecular pattern is periodic (or repetitive in all directions), the distinctive quality of a quasicrystal is that its structural pattern never repeats the same way twice. It is endless and uneven, but it can be described by the arrangement of a small set of modular parts. This furniture piece explores an aperiodic assembly in wood.
Source: Aranda\Lasch, fabrication by James Moore.

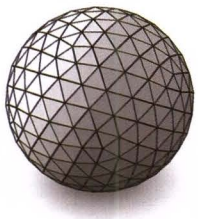
Sometimes you need to understand the underlying mathematics to effectively create a model. Hierarchy is a time-honoured architectural design strategy, yet has limited support in most CAD systems. The intellectual key to hierarchy is recursion, which occurs when a program invokes itself. (See Section 8.16 below and Figure 3.15.) With recursion, parts can be made to directly resemble the wholes they compose.



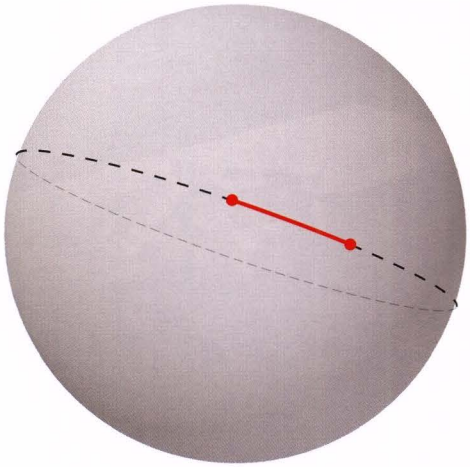
3.19: A single recursive structure with minor variations produces a wide range of designs. The central diagram demonstrates the basic tree structure of two motifs replicating at each recursive level. Along the upper branch, the location of the motifs changes with each successive figure. For the last four figures, the motif changes from a line to a triangle and only the final level of motifs appears. In the lower branch, only location changes; the line motif remains the same. In the final figure on this branch, the number of levels in the recursion increases and only the final level of motifs appears.
Source: Woodbury (1993).

A geodesic curve is the shortest path on a surface that joining two points $\dot{p}$ and $\dot{q}$ also on the surface. For spheres, the geodesic curve between $\dot{p}$ and $\dot{q}$ is the shortest arc linking the two points taken from the *great circle* defined by the two points and the sphere centre. Discrete points along a sphere geodesic curve can be found by projecting points on the 3D line between $\dot{p}$ and $\dot{q}$ to the sphere's surface. Geodesic meshes can be generated by subdividing polyhedral faces and then projecting the new vertices onto the sphere. Figure 3.18 shows that successive subdivisions produce more sphere-like forms. On the other hand, subdivision can be cleanly understood as a recursive operation. Even such a qualitative understanding of geodesic concepts enables complex form making.





3.21: Starting with an icosahedron (20 equilateral triangular faces), successive subdivisions of each triangle make geodesic meshes that better approximate a sphere.



3.20: A geodesic curve on sphere is a fragment of a great circle.

Many contemporary parametric designers are exploring mathematically-based design strategies. I cannot help but notice a pattern. They browse the web and their social networks for math, picking ideas and playing them into design work. The reasoning chains (from pigeon-hole principle to permutations to binomial theorem to permutation cycles...) that structure classical mathematics learning and understanding are seldom part of the game. Designers mostly enter above the foundations, understand something of mathematical mechanism and move on to use it in design. Of course, there is co-evolution here. The web enables casual access (try doing it in a physical library of mathematics texts) and itself is conditioned by such use. It should come as no surprise that the constructive, recursive definition of the Bézier curve is much more common on the web than the Bernstein basis definition, in spite of revealing less mathematically. Visual construction trumps mathematical inference for design. As with algorithms, designers exhibit a copy-and-modify style in using mathematics. Even more quickly, they encounter limits. Just as a program may not even compile in the face of a minor coding error, a mathematical formula or theorem may break down completely with a seemingly minor change. There are no Pythagorean triplets for cubes, that is, there are no integer solutions to $a^3 + b^3 = c^3$. (This is an instance of Fermat's Last Theorem, proved only in 1995, that is, there is no integer solution to $a^n + b^n = c^n$ with $n > 2$.)

The mathematical knowledge available in libraries, especially those of research universities, staggers the mind. Much of this material can be accessed only by physically visiting, and relatively little can be understood without concentrated and sustained study of mathematical foundations. The World Wide Web and interactive software for working with mathematics have thrown open doors through which come a large crowd unfamiliar to the mathematically astute.

Online resources such as Wolfram's MathWorld (Weisstein, 2009) and many university courses provide immediate access to (sometimes carefully constructed) explanations that can help bridge to mathematical understanding. Packages such as Mathematica® and Maple™ are to mathematics what parametric modeling is to design – by making math active, they enable exploration and discovery.

3.3.6 Defer decisions

In design, accuracy measures how the design relates to the thing being designed. Precision measures how design parts relate to each other. Conventional systems require geometric precision and provide tools such as snaps to help achieve it. Without precise size and location, models look messy. They do not have the ambiguity and appropriate refinement of a sketch; they are just messy. I argue that, more than anything else, this need to commit to specific locations at the outset of modeling is what is least sketch-like about computer-aided design. A clear exception is the implicit modeling toolset widely used in animation and gaming. Implicit surfaces lie "somewhere near" their generating objects and provide rules to merge "nearby" surfaces together. Implicit modeling removes both the need for precision and the possibility of accuracy.

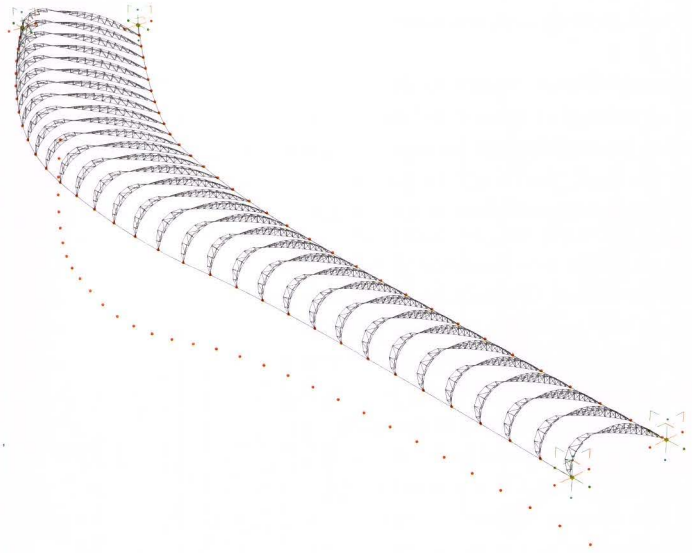
Parametric modeling introduces a new strategy: *deferral.* A parametric design commits to a network of relations and defers commitment to specific locations and details. The system maintains the prior decisions made. Deferral pervades parametric practice. Those new to parametric modeling often ask how to locate their initial points and lines. Those teaching delight in the answer: "It doesn't matter; you can change that later."

One of the earliest (and effective!) demonstrations of parametric modeling in architecture was the International Terminal Waterloo by Nicholas Grimshaw & Partners (see Figures 3.23 to 3.22). Lars Hesselgren crafted the original model in the I_EMS system. More than 15 years later Robert Aish used a similar model to demonstrate the CustomObjects system (which later became Generative-Components™). A salient site condition is that the train track curves through the station. A parametric model need not be initially constrained by this curve; fitting it to location can be deferred. Changing the order in which modeling and design decisions can be made is both a major feature of and deliberate strategy for parametric design. Indeed, a principal financial argument for parametric modeling is its touted ability to support rapid change late in the design process.

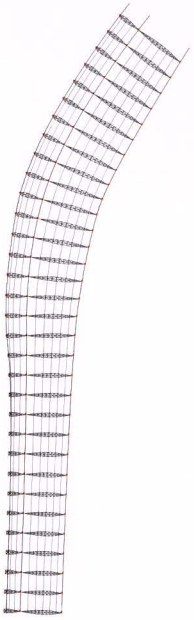
The Eden Project also by Nicholas Grimshaw & Partners, (see Figure 3.26), combined parametric modeling and geodesic geometry to address an unusual problem. The site was a quarry that remained active until very late in the design process. Consequently ground levels could not be predicted in advance. The geodesic geometry made it easy to extend and rearrange partial spheres, while parametric modeling shortened revision cycles.

3.22: The *International Terminal Waterloo* by Nicholas Grimshaw & Partners.
Source: James Pole.

3.23: The *International Terminal Waterloo* by Nicholas Grimshaw & Partners. The station was designed around an existing path for the track system.
Source: Jo Reid and John Peck.



3.24: Using parametric modeling, the exact location of the structure can be changed at the very end of the modeling process.



3.25: Plan view of a parametric model of Waterloo Station.

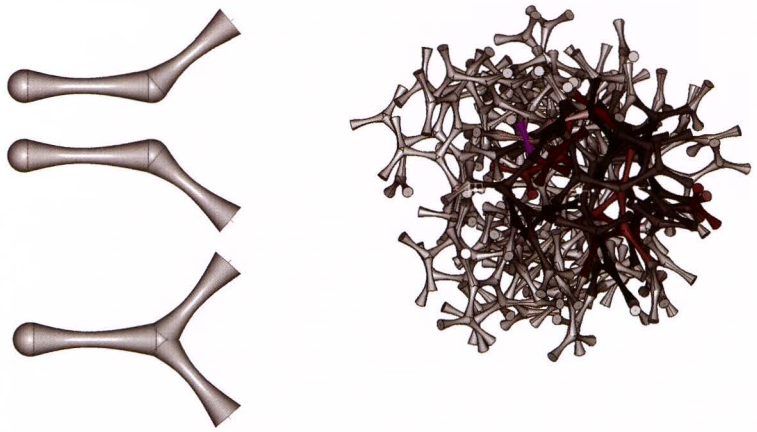
3.26: The *Eden Project*, Cornwall, United Kingdom.
Source: ©2006 Jürgen Matern (http://www.juergen-matern.de).

3.3.7 Make modules

Propagation graphs can, and do, get big. Large size increases system update times and, more importantly, makes models hard to understand. Copying parts of a large graph and reconnecting them elsewhere in a model is prohibitively difficult. Reducing graph complexity and enabling reuse are the main reasons that systems universally provide module-making tools. The names and details of these vary from system to system, but their essence is the same. They provide

a means to encapsulate a sub-graph as a single node with its own set of node-independent (input) parameters. Copying and reuse then reduces to making a copy of a single node and reconnecting its inputs as needed.

It takes much effort to make a module work well and communities of practice develop surprisingly sophisticated module-making techniques. Almost always, the process iterates; through successive attempts modelers converge on stability. Later in this book, the PLACE HOLDER pattern on page 218 shows one such community-developed technique for placing modules onto complex geometric constructions.



3.27: In a complex design with repeating elements, modules are a near-essential part of the modeling process. The design on the right is a programmatically arranged complex of the three modules on the left.
Source: Martin Tamke.

3.3.8 Help others

In *Gardeners and Gurus: Patterns of Cooperation among CAD Users*, Gantt and Nardi (1992) use the concept of the *gardener* to describe internal developers (and extenders) of CAD systems who are supported by their organization. Perhaps the peak example of gardening in architectural design in 2010 is the Specialist Modeling Group at Foster + Partners. This group employs many strategies to enable complex geometric and computational design work throughout the firm. From 2003 to the time of writing (2010) such gardening at a community scale was clearly evident in the SmartGeometry organization, in which more than 20 tutors volunteer a week of their time, year-after-year, to mentor students and practitioners new to parametric modeling. Of course, other rewards are at play: such events are superb places to meet peers, scout for employees and check out the latest work. From their offices and studios, some parametric practitioners (naming one or a few would be unfair – there are just too many good ones)

freely share code and insight into modeling tasks. To them flow the rewards not just of fame, but more importantly, of new problems and approaches to solutions. Formalized or not, helping others is a clear strategy to at least some aspects of mastery.

3.3.9 Develop your toolbox

The parametric medium is complex, perhaps more so than any other media in the history of design. Using it well necessarily combines conceiving data flow; new divide-and-conquer strategies; naming; abstraction; 3D visualization and mathematics; and thinking algorithmically. These are the basics, and mastery requires more. We can expect that new technique and strategy will flow from the practices and schools that invest time and effort in the tools. Between the basics and the designs that are the focus and aim of professional work, lies a largely unexplored territory of what might be called *parametric craft*. I choose the word "craft" on purpose, to align with Malcolm McCullough's (1998) case for a developing digital craft. Some of his examples were of parametric models. Understandably, given the date and breadth of the book, McCullough merely hinted at the richness of parametric technique.

We can expect explorers in the new territory of parametric craft. Unlike the medieval sailors in whose portolanos we can see cartography slowly develop, the current explorers can learn from other fields that have undertaken similar voyages of discovery. There are numerous books on spreadsheets; some, like Monahan (2000), focus on strategies for spreadsheet design. Borrowing partly from film, computer animation has grown an extensive repertoire of technique. Software engineering has forged and polished a powerful tool. *Software Design Patterns* describe fragments of systems both functionally (by what they do) and structurally (by how they are composed of simpler structures). Their origins lie in architecture, particularly in Alexander's many works on pattern languages. In software, though, design patterns have a new and philosophically different logic and application. They have come to occupy a pragmatic place between the technical description of computer languages and the overall organization of a complex computer program. In software, design patterns record demonstrably useful ideas for system design. I have both adopted and adapted software design patterns as a basis for expressing the new parametric craft.

We can expect, as with the medieval nation-states, that some of the parametric portolanos will be kept strictly private. But practices and universities alike will come to use and value only those that are public. I devote much of this book to a small, initial set of design patterns. My aim is to begin what I hope will be a long and fruitful process of developing an explicit, shareable and learnable craft of parametric design. Before patterns must come programming and geometry – the practical manifestations of algorithms and mathematics for much of design. Explaining particular patterns relies on a few key ideas from each of these very large fields.