A Visual Approach for Exploring Computational Design

by

Miranda Clare McGill

Diploma in Professional Practice (RIBA Part III), University of Manchester, England, 1998
Bachelor of Architecture (RIBA Part II), University of Manchester, England, 1997
BA (Hons) Architecture (RIBA Part I), University of Manchester, England, 1994

SUBMITTED TO THE DEPARTMENT OF ARCHITECTURE IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE IN ARCHITECTURE STUDIES
AT THE
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

JUNE 2001

Signature of Author [illegible]
Miranda C. McGill
Department of Architecture
May 24th, 2001

Certified by [illegible]
Terry W. Knight
Associate Professor of Design and Computation
Department of Architecture
Thesis Advisor

Accepted by [illegible]
Roy Strickland
Principal Research Scientist
Department of Architecture
Chair, Committee on Graduate Students

Thesis Reader	William J. Mitchell Professor of Architecture and Media Arts and Sciences Dean, School of Architecture and Planning
Thesis Reader	Edith K. Ackermann Visiting Professor Department of Architecture

A Visual Approach for Exploring Computational Design

by

Miranda Clare McGill

Submitted to the Department of Architecture in Partial Fulfillment of the Requirements
for the Degree of Master of Science in Architecture Studies
June 2001

Abstract

This thesis concerns the use of computers for learning about computational design and shape grammars. It discusses how software can be developed to create "microworlds" for designing, and how to take into account the needs of designers whilst retaining a transparency of process in computational design. The broader context pertains to the learning and practice of design. Through analysis of computation in a workshop setting, consideration is given to the role of the computer as a facilitator for designing with shape grammars.

Prototype software for facilitating the learning of shape grammars, called *Shaper2D*, was created as a focus for this study. It is written in the Java programming language for cross-platform compatibility, and is available both as an applet and stand-alone application.

Thesis Advisor: Terry W. Knight

Title: Associate Professor of Design and Computation

Acknowledgements and Thanks to:

Terry Knight for her invaluable guidance and encouragement, and for urging me to pursue *Shaper2D* further (I'm glad I did!)

William Mitchell, for being my academic advisor and thesis reader, and for all his suggestions and advice during the past two years.

Edith Ackermann, my thesis reader, for her wisdom and knowledge about psychology and learning theory, both before and during the thesis months.

Mark Tapia, Research Scientist, MIT, for his programming advice and enthusiasm.

Gabriela Celani, PhD Candidate, Design Technology, MIT, for inviting me to be her TA for the Remote Collaborative Workshop, and for her support during the workshop, despite being so ill!

Federico Casalegno, visiting social scientist, MIT, for his advice and assistance with my evaluations of the remote collaborative workshop.

Mitchel Resnick for taking the time to talk with me about Constructionist learning tools.

The students in the Remote Collaborative Workshop, the follow-up study, and the Design Inquiry class, for all their enthusiasm and hard work, and without who I would not have had the opportunity to test *Shaper2D*.

The members of the spring 2001 Computational Design II class, for their thoughtful comments and suggestions.

Megan Yakeley for setting me on this path back in 1995, and for being my friend and mentor.

The Rotary Foundation, for awarding me an Ambassadorial Scholarship (which enabled me to finally come to MIT!), with a special thank-you to **Ainsley Donaldson**, for her friendship and support, and **Hoylake and Wesy Kirby Rotary Club**, for believing in me in the first place.

Oliver Dial for his enduring patience, support and understanding before, during and after the *Shaper2D* programming months (and for helping me with my mathematics revision!)

My friends back home in Wirral, as well as Liverpool, Milton Keynes, London and Italy, the **CSB** (you know who you are), **my new friends in the USA** (Massachusetts and California), and the **THDA**, for reminding me that there is life beyond my computer and 77 Massachusetts Avenue!

And finally, a big thank-you to my family, especially my **Mum** (who has been busy studying for her Bachelor's degree), my sister **Tanya** (who has been busy studying for her law exams), my **Nanna**, and **Sadie** -- and, of course, the rest of my family (people, dogs, cats and fish), and people I think of as family -- who are always there for me, no matter what.

Contents

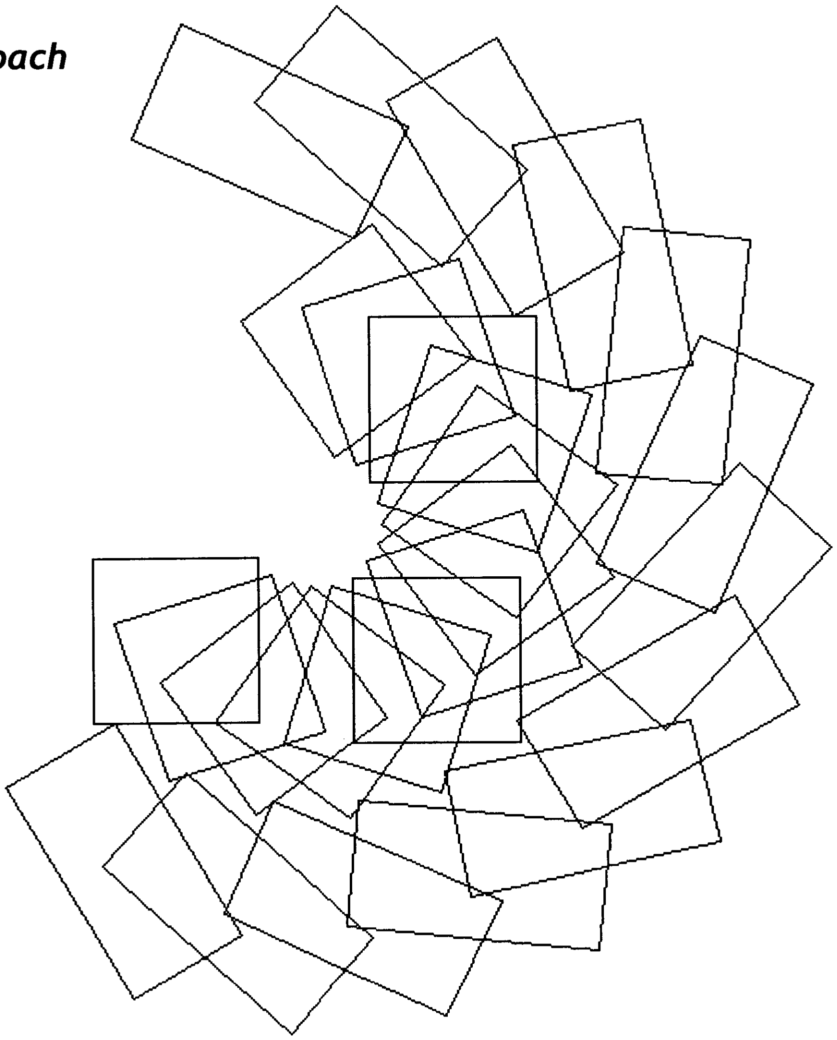
Illustration Credits

Permission was granted by the students to reprint their work.
All chapter cover images by the author.

Figure 1, 3.	Miranda McGill, B.Arch Thesis Project, University of Manchester, 1997.
Figure 2.	*DBN*, "Making Commands", http://dbn.media.mit.edu/examples2.html
Figure 4.	Diagram of Bitmap.
Figure 5.	Cabri-Géomètre, University Joseph Fourier, Grenoble, France & CNRS.
Figure 6.	MicroWorlds™, Version 2.03, © LOGO Systems Inc., 1998.
Figure 7.	Miranda McGill, example of a basic shape grammar.
Figure 8.	Alvar Aalto, three works.
Figure 9.	GEdit, © Mark Tapia, 1998.
Figure 10.	3D Shaper, © Yufei Wang, 1998.
Figure 11.	Shaper2D Applet, © Miranda McGill, 2001.
Figures 13-18.	Shaper2D Application, © Miranda McGill, 2001.
Figures 19-21.	AutoCAD 2000® by AutoDesk®.
Figure 22.	Soohyun Chang, Site Plan.
Figure 23.	Kristie Tate, Site Plan.
Figure 24.	Miguel Chacon, Site Plan.
Figure 25.	Audrey Snyder, Site Plan.
Figure 26.	Miranda McGill, Study A in-class exercises.
Figure 27.	Myoungkeun Kim, Site Plan.
Figure 28.	Prerna Sood, Site Plan.
Figure 29.	Konstantinos Tsakonas & Pamela Campbell, Site Plan.
Figure 30.	Konstantinos Tsakonas & Pamela Campbell, Site Plan.
Figures 31-32.	Samples from Study B, Exercises 1 and 2
Figure 33.	Gustavo Rodriguez, *Shaper2D* design exercise.
Figure 34.	Ian Sheard, *Shaper2D* design exercise.
Figures 35-36.	Xiaohua Sun, *Shaper2D* design exercise.

Introduction

A New Approach



A New Approach

The research presented in this thesis comprises the union of three themes:

- Exploring visual algorithmic processes
- Considering the notion of Constructionist and situated methods for learning design
- Creating and testing a visually dynamic tool for using shape grammars

Many of the previous studies concerning Constructionist teaching and learning techniques use mathematical education as the subject matter (e.g. Papert, 1980; Noss & Hoyles, 1996). Building upon previous research conducted to explore the role of computers in design education (for example, Yakeley, 2000), this thesis will examine specifically how they might be used in the Constructionist learning and practice of shape grammars.

To explore this hypothesis further, I developed a computer program called *Shaper2D* that is designed to facilitate learning and designing with shape grammars. This software was developed to employ an intuitive, visual interface that facilitates a "learning by designing" approach for shape grammar education.

Proponents of shape grammars make the argument that shape grammars empower the designer to consider design possibilities that might otherwise have been neglected, and to make explicit the designer's thought process. Contrary to this, "traditional" designers are concerned about the apparently restrictive and arbitrary nature of this system for design. This thesis does not dwell on this debate in any detail, for this is a research study in itself. Nor does it describe the intricacies of shape grammar theory, as there exists already a large body of research that presents the various concepts pertinent to the methodology of computational design and shape grammars. Instead, the objective is to consider the role that the computer plays in the Constructionist learning and practice of shape grammars, and argues that it can be a liberating medium for designers wishing to use this particular methodology.

This thesis has been divided into three chapters. The first chapter considers the question of what computational design is, how it differs from computation for drafting, and how it inspires creativity. In order to place *Shaper2D* in the context of learning by designing, a summary of several learning theories, including Constructivism/Constructionism, socio-constructivism and situated learning, are given alongside a review of other approaches to computational design. The chapter then introduces shape grammars as an example of computational design and ends with an overview of computer mediated shape grammars.

Chapter two discusses the development of *Shaper2D*, together with its pedagogical and design relevance for using shape grammars. Previous computer implementations of shape grammars are examined in order to situate *Shaper2D* in its intended context. The design and implementation of the software is then described, including an outline of how the program works.

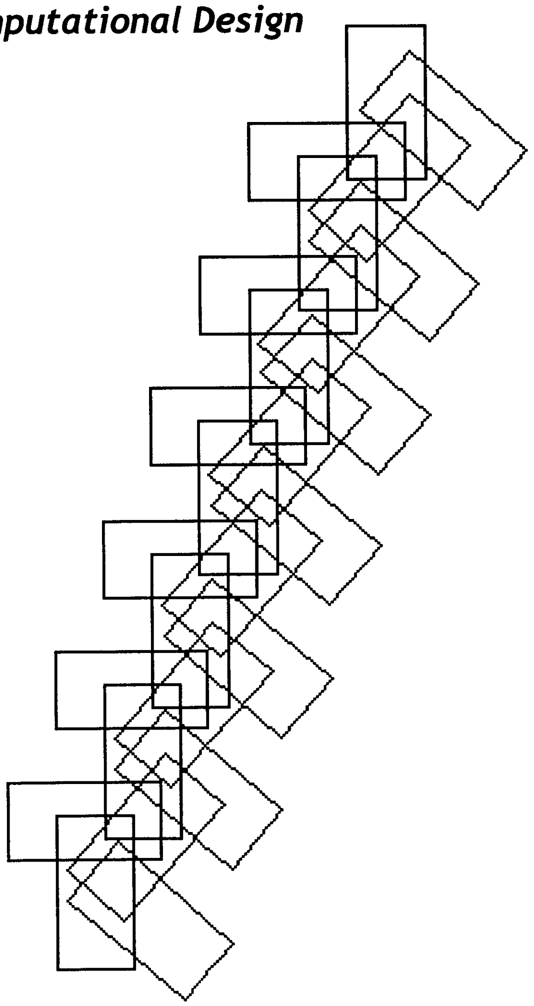
The third chapter gives an account of *Shaper2D*'s use in three studio situations. After a brief summary of the research methods used for analyzing the data gathered, a description of each of the studies is given, together with an evaluation, and concluding remarks. The first and third studies took place as part of a regular MIT class, whilst the second was a follow-up study that took place independent from any formal teaching environment.

"...One must know and understand the alphabet before words can be formed and a vocabulary developed; one must understand the rules of grammar and syntax before sentences can be constructed; one must understand the principles of composition before essays, novels, and the like can be written. Once these elements are understood, one can write poignantly or with force, call for peace or insight to riot, comment on trivia or speak with insight and meaning."

(Ching, 1979, p.11)

Chapter One

Learning About Computational Design



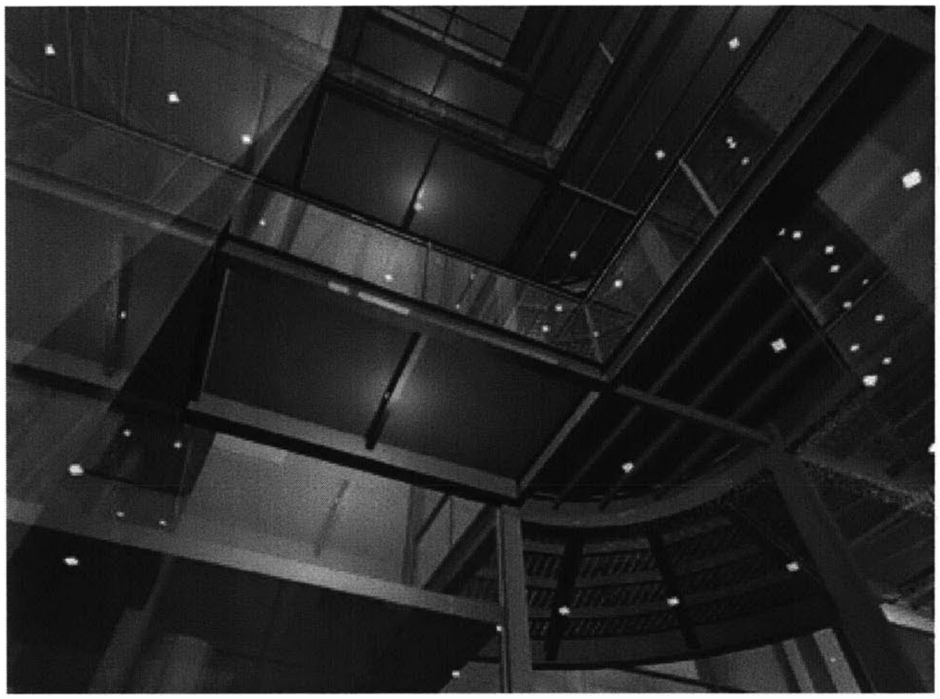


Figure 1. Miranda McGill, Computer rendering of atrium space, thesis project, 1997.

What is Computational Design?

Algorithms for Designing

The MIT Encyclopedia of Cognitive Sciences (1999) has a good description of an algorithm,

> "The essential feature of an algorithm is that it is made up of a finite set of rules or operations that are unambiguous and simple to follow... usually algorithms are thought of as recipes, methods, or techniques for getting computers to do something, and when restricted to computers, the term "algorithm" becomes more precise, because then "unambiguous and simple to follow" means "a computer can do it." The connection with computers is not necessary, however. If a person equipped only with pencil and paper can complete the operations, then the operations constitute an algorithm."

Computational design is an algorithmic system for the synthesis and analysis of designs. The above excerpt demonstrates that algorithms are not necessarily computer-implemented, so it follows that computers are not requisite for computational design. In the context of this thesis, for computational design read rules, algorithms and instructions.

Computation for Designing versus Computation for Modeling

It is presented here that computational design is for generating designs or ideas for designs. On the other hand, computation for modeling—or drafting—is for the production or representation of designs.

To differentiate between computation for designing and computation for modeling, recall the traditional studio presentation. The designer is usually more concerned with exhibiting a final product than with displaying the process employed in order to arrive at that design (Yakeley, 2000). Nowadays, the designer often uses computer aided design (CAD) software to create a comprehensive three-dimensional representation of the scheme, and subsequently generates appealing renderings to assist the critic with their perception of what the "real thing" might look like (see figure 1). Although concept sketches might also be included in the presentation, it is unlikely that they accurately document the path that the designer followed when developing the design. Computational design, in distinction to computational modeling, espouses the evaluation and representation of the design process, as it provides an essential visual guide for understanding and monitoring the design—and the designer. Thus, the design process is made explicit which facilitates the "tweaking" process, as the designer can view the different stages of design development, and can investigate in more detail how changing one parameter can influence the final outcome.

Computation for modeling can allow you to generate an information database for a building, or other, design. Being able to control all the areas of a building from a central knowledge database is a rapidly emerging area of research—make one change to a particular part of a design and the effects of that change automatically propagate throughout the entire design.

A development of the digital paper and pencil paradigm is the computational environment where the designer can perform the same tasks as with CAD drafting tools, but with added functionality such as dynamic updating of drawings in response to design revisions, calculating areas, and estimating quantities. The designer does not create a separate three-dimensional CAD model, but develops one on the fly, using some of the principles that she would use if building a physical model—without the limitations of the physical world. Some very sophisticated parametric design software (for example *Revit*) is being developed that aims to encapsulate all the CAD functionality the designer would wish to use. The CAD program recognizes lines as representations of more complicated physical objects. A design is stored in a relational database rather than as independent lines, and the drawing is synthesized from this information. These approaches can affect alternative design strategies that the designer takes when drawing or modeling with a CAD. Essentially, however, in terms of what the outcome is—be it a detailed construction drawing or a life-like rendering—the guiding "drafting" principle remains the same.

The work that is presented here is not a system for the digital representation of an idea. Representing design ideas electronically is made possible with conventional CAD programs such as *AutoCAD* and *Microstation*, which could be described as automated drawing boards or electronic drafting tools.

Increasingly, however, research is being done to investigate the use of computer applications to generate and explore ideas instead of representing them.

Creativity in Computational Design

Creativity involves the development of original, imaginative ideas, usually for a particular purpose. Margaret Boden (1990) puts it succinctly when referring to computation for creativity: "...creativity is a matter of using one's computational resources to explore, and sometimes to break out of, familiar conceptual spaces." (p.108). It can be situated in both cognitive and social domains, as John Gero (1996) explains,

> "The social view holds that creativity is only embodied in an artifact or an act and is a characteristic bestowed on that artifact or act by society. This view has a degree of attractiveness as it divorces the creator from the 'created' and only assesses the 'created'. The cognitive view holds that there are some things about the cognitive acts of the creator which play a significant role in the creativeness of the resulting artifact or act."

Design is a deliberate, goal-oriented, yet open-ended activity. It is usually performed in conjunction with a given set of constraints or conditions, which form the design problem, and which often reflect social, economic, symbolic or functional issues (Ching, 1979). These constraints may be explicit, such as those in a design brief, physical attributes of the site, or environmental characteristics that must be accounted for. Constraints may also be implicit in aspects of the system of design and methods of representation.

Why Use Computation for Design?

One question that is frequently asked about computation is: What the point of using computational design if you already know what you want to do?

Computation can be used to generate multiple design ideas or solutions, instead of just one. It can also provide a greater depth of understanding of a design. It allows the designer to make more informed decisions when developing and revising a design, during the entire design process, from inception, through evolution, to the conclusion.

"Learning by Doing"

The Design Process

When an architect is posed with a problem, invariably the first thing she does is to pick up a pen or pencil and start sketching, both in order to gain a better grasp of the problem and to develop a personal understanding of the issues that may be involved. Learning about design is as much the process of formulating the problems as the development of a solution space.

Design has always been a hands-on activity, even since the advent of the computer and CAD. As Wolff-Michael Roth (2001) states, "Professional design and design activities in schools are fundamentally situated and distributed activities." (p.216) That is to say, design is not taught in a detached classroom setting, but instead is located in authentic studio activity, using the "master and apprentice" archetype (Lave and Wenger, 1991).

It may be useful to reflect on several areas of learning research in order to shed light on the design process. The approach discussed in this thesis combines several aspects of contemporary learning research: Constructivism/Constructionism, socio-constructivism, situated learning, and motivation, each of which will be introduced, and described briefly, below.

Constructivism and Constructionism

> "The student does not just passively take in knowledge, but actively constructs it on the basis of his/her prior knowledge and experiences."
>
> (Piaget, 1972)

Constructivist learning theory posits that learners actively construct their own understanding using existing knowledge as a starting point. It focuses on the processes involved with the constructive assimilation of knowledge. This approach to learning differs from the traditional view that knowledge is information, presented by a teacher or textbook, to be absorbed by the student. Instead, knowledge is experience, and the student is proactive in developing her own understanding of a subject or task.

Seymour Papert's Constructionism expands Piaget's Constructivism by adding the idea that this constructive process happens best where the learner is engaged in the process of building an artifact—something external, tangible and demonstrable (Papert, 1993). More than Piaget, Papert focuses on the process of externalization of ideas so central in design (Ackermann, TBA).

Megan Yakeley (2000) states simply that, "...design is a constant learning activity within a Constructionist framework." Although the basics of computational design—for example, two-dimensional shape grammars—can be described to the student, it is generally accepted that the only way they can be fully understood is through experimentation.

Collaborative / Cooperative Learning: Socio-Constructivism

Although this thesis embraces the Constructionist approach to learning computational design, the role that collaborative, or cooperative, learning can play should not be overlooked—a major advantage of interactive, computer-mediated learning. One of the reasons for developing a computer program for working with shape grammars was that, by utilizing current technologies (for example, *Microsoft NetMeeting*), the program can be shared with remote parties. Stephen Balkcom (1992) states that,

> "Cooperative learning is a successful teaching strategy in which small teams, each with students of different levels of ability, use a variety of learning activities to improve their understanding of a subject. Each member of a team is responsible not only for learning what is taught but also for helping teammates learn, thus creating an atmosphere of achievement."

When students work cooperatively or collaboratively, rather than laboring over a problem individually, they are able to share the processes of constructing their ideas. Each team member brings her own perspective, so that meaning can be negotiated and solutions generated through a shared understanding of the problem. This means that students are able to view their peers as valuable resources, through peer coaching or 'mutual tutoring', leading to a shared sense of progress and achievement (Strommen, 1999).

Whilst it has been noted that talk can play a central role in collaboration (Hennessy and Murphy, 1999), the natural mode of communication for designers is visual rather than textual or verbal. The process of externalization is a key part of designing. That is to say, "externalizing internalized ideas", or making thoughts visually explicit. When making sense of a design problem or explaining a problem to a fellow designer, the designer's usual habit is to pick up a pencil (or other drawing implement) and sketch her thoughts. Being able to share a graphical computer program with geographically dispersed teammates in the same way that you can share a pencil with a collocated teammate can become a vehicle for testing ideas, with the transfer of control over the program being analogous to handing over a pencil.

Situated Learning

The situated learning paradigm focuses on the notions of authenticity and communities of practice (Brown et al., 1989). In other words, the activity in which the knowledge is developed and used is an integral part of learning, and should occur in culture and context. These cultures and contexts, moreover, need to be genuinely in tune with the learners' interests.

According to Jean Lave and Etienne Wenger (1991), knowledge can be derived from peripheral observation of activities in progress, where the apprentice watches the master and gradually builds her own repertoire of skills situated in that context. This then affords the learner a place in the social organization of the activity, which is amplified over time. Authentic activity is vital for motivation and successful learning.

Motivation

One of the most influential factors in the decision to create a tool for facilitating shape grammar learning is the idea of motivation. The learning environment has a crucial bearing on generating motivation (Brown et al., 1989). Sanna Järvelä and Markku Niemivirta (1999) develop this claim by arguing that, "...the learning environment... should provide the learner with opportunities to test and try out his new conceptual understanding in various applied circumstances like problem solving." (p.58)

They go on to say,

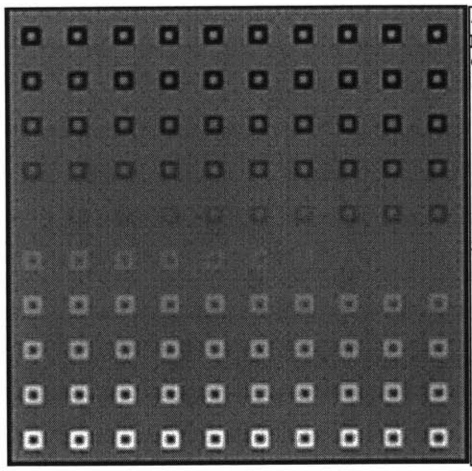
> "Making learning tasks more challenging and authentic, and increasing the possibilities brought on by technology has not only changed students' learning processes but also increased the complexity of learning situations. Thus, a learning situation, for a learner, is not merely a mental performance but also a motivational challenge."
>
> (*ibid*, p.59)

The work presented here was developed in response to a need for an alternative method to encourage, or motivate, students when learning the basic concepts of shape grammar theory.

Other Approaches to Computational Design

Product and Process

Other research initiatives in computational design have been concerned with the process of design. However, the principal focus can vary between the procedural aspects and the final product. In addition, the computation tends to be textual—actual computer programming using a particular language, such as Python, Pascal or VectorScript, as opposed to visual. Two examples of contemporary computational design research are *Design By Numbers* (Maeda, 2000) and



```
// Making commands
paper 50
command square x y s c
{
   pen c
   line x y x (y+s)
   line x (y+s) (x+s) (y+s)
   line (x+s) (y+s) (x+s) y
   line (x+s) y x y
}
repeat A 0 9
{
   repeat B 0 9
   {
      square (A*10+3) (B*10+3) 3
(A+B*10)
   }
}
```

Figure 2. Example of *Design by Numbers*: Final design and associated code.

Digitally Mediated Design (Yakeley, 2000). As will be seen, these examples differ from the approach taken in this thesis in that *Shaper2D* is an exclusively visual environment.

In *Design By Numbers* (DBN), John Maeda developed a programming language as an introduction to, and a tool for, teaching the "idea" of computational design to designers and artists. Although Maeda argues that process is central to the practice of *DBN*, the finished designs remain the primary focus (see figure 2). However, Maeda is not completely disinterested in the design process, as the entire programming language was created specifically for this design approach. While process has not been discussed explicitly, it can be deduced from the various results that have be produced using different programming constructs (Maeda, 1999). It would be interesting to study the impact of these programming constructs on what is produced.

With *Digitally Mediated Design*, Megan Yakeley is concerned with the process of generating a design solution. The aim of *DMD* is to explore the use of computers and computer programming within CAD environment, to aid original design and provide a scaffold for the designer's development and understanding of her own unique process of design (Yakeley, 2000). Yakeley's argument is that the computer is a "tool for thought" that can help to make explicit the thought processes that are evident in the design studio. In brief, the designer writes a computer program using VectorScript (a procedural scripting language which is part of the *VectorWorks* CAD software) that provides a formal description of a design, and executes it in order to generate a representation of that idea. Revisions are then made to the program, either to tweak the design or debug the code, and it is run again.

Necessary Constraints for Computational Design

Limitations are inherent to any computational system for designing, in that they form the boundaries of an approach. Igor Stravinsky (1947), when talking about his approach to composing music, is vehement in his conviction of the necessity of constraints, "The more art is controlled, limited, worked over, the more it is free." (p.63)

He expands on this statement by stating that,

> "...I experience a sort of terror when, at the moment of setting to work and finding myself before the infinitude of possibilities that present themselves, I have the feeling that everything is permissible to me. If everything is permissible to me, the best and the worst; if nothing offers me any resistance, then any effort is inconceivable, and I cannot use anything as a basis, and consequently every undertaking becomes futile."

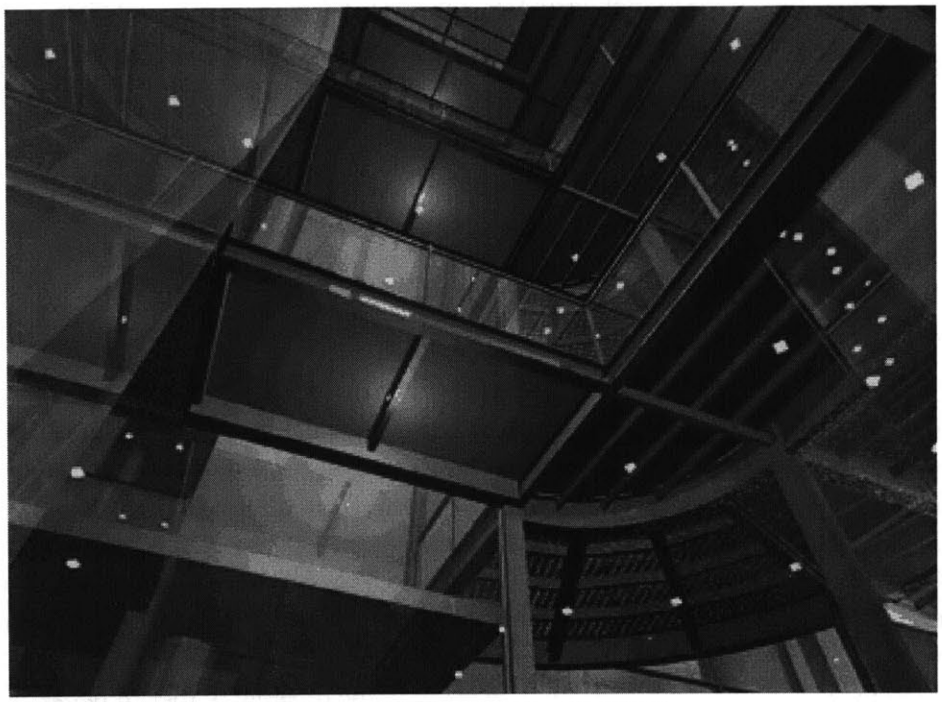


Figure 3. Bitmap image of figure 1 (4 bits per pixel -- 16 colors).

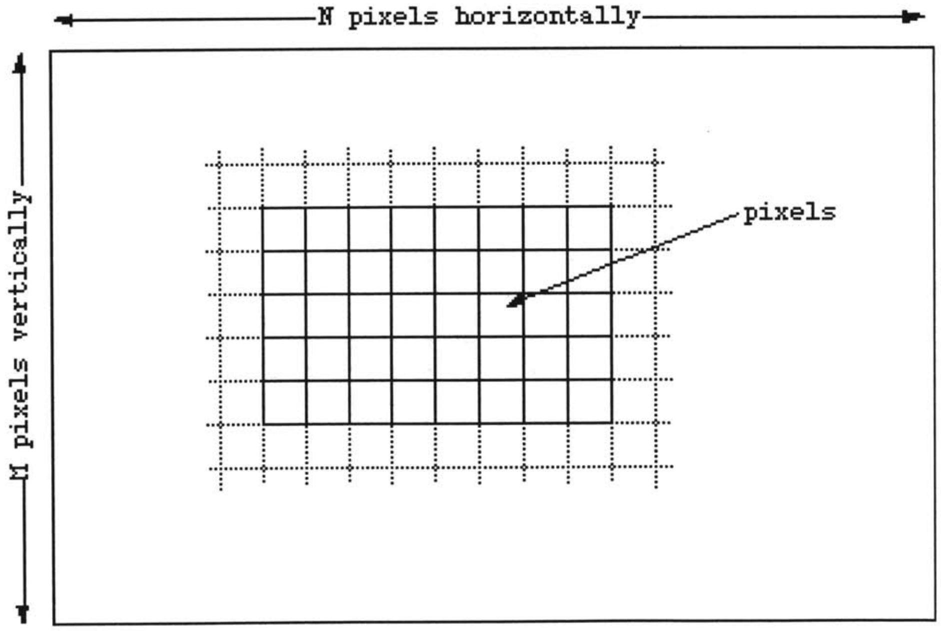


Figure 4. Diagram of a bitmap.

And continues,

> "My freedom... consists in my moving about within the narrow frame that I have assigned myself for each one of my undertakings.
>
> "I shall go even further: my freedom will be so much the greater and more meaningful the more narrowly I limit my field of action and the more I surround myself with obstacles. Whatever diminishes constraint, diminishes strength. The more constraints one imposes, the more one frees one's self of the chains that shackle the spirit."
>
> (*ibid*, p.65)

In keeping with Stravinsky's philosophy, this thesis contends that constraints and limitations are beneficial to design. Limitations can help form a valuable framework in which ideas can be explored and expressed—how you can design and what you can produce with the system that you are using. Maeda's and Yakeley's research do not tackle the issue of potential constraints that may be imposed on the designer in any great detail, whereas it is asserted here that with these methods for designing, the designer is always faced with limitations imposed by the programming language being used.

Microworlds for Computational Design

The term microworld was first coined by Seymour Papert as a description of self-contained worlds for exploring geometrical and mathematical ideas (Papert, 1980). They immerse the student in a specific environment where they can investigate new ideas. Microworlds emphasize particular constructs and operations, thus supporting and encouraging certain styles of thinking. Put another way, the user is unable to do everything, as she is constrained by deliberate, pre-determined (consciously or unconsciously) limitations built into the system. In essence, the source of power and interest in microworlds is their structure, which is organized in response to the constraints. However, there is always a trade-off between freedom and constraints. For example, a bitmap (see figure 3) the simplest digital image, could be interpreted as a microworld. A bitmap, sometimes called raster image or frame buffer, is the display memory for a raster display device (Mitchell et al., 1987). Its microworld consists of a pattern comprising pixels, or individual dots, arranged into a grid of rows and columns to form a graphics image (see figure 4). A value for each pixel is stored in the computer memory as one or more bits of data. The more bits of data, the more shades of gray or colors can be represented by the bitmap. However, other than storing a limited amount of graphics information, the bitmap cannot be manipulated further due to its structure. This demonstrates how, in order for a microworld to be useful in relation to some purpose or goal, the right kind of balance between structure and open-endedness is needed.

It is argued here that the concept of a microworld can used to shed light on the various approaches that have been taken in design, both computational and otherwise.

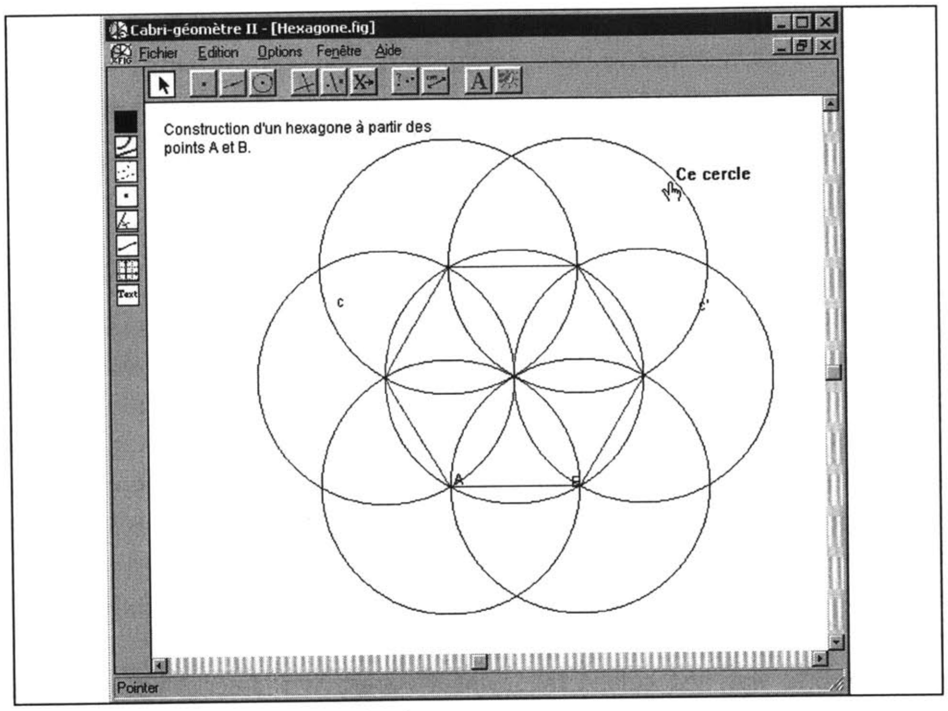
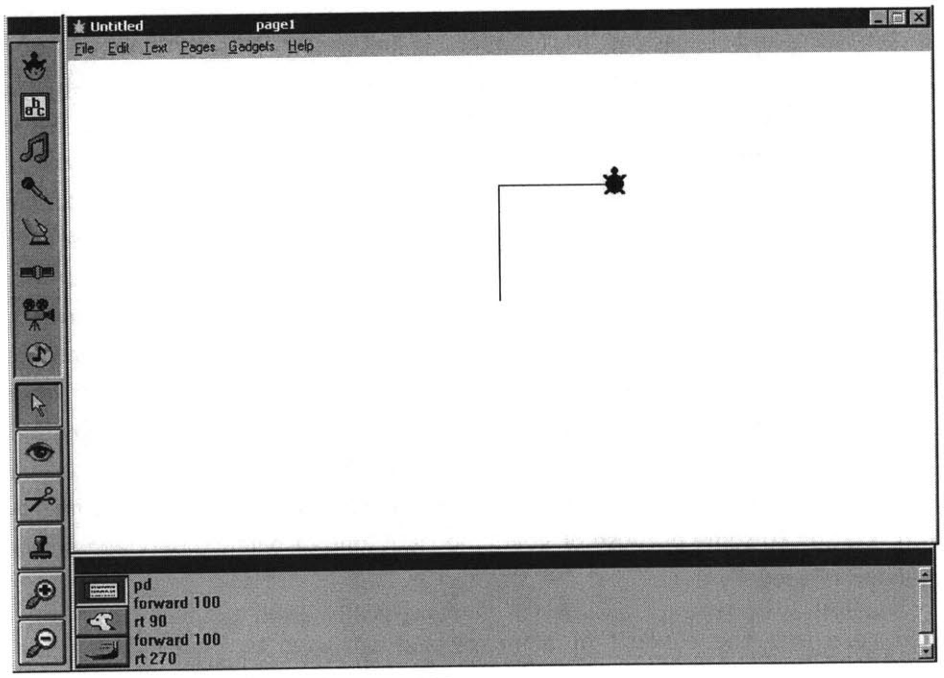

Cabri-géomètre II - [Hexagone.fig]
Fichier Edition Options Fenêtre Aide
Construction d'un hexagone à partir des points A et B.
Ce cercle
C
C'
A
B
Pointer

Figure 5. Screenshot of *Cabri-Géomètre.*



```
pd
forward 100
rt 90
forward 100
rt 270
```

Figure 6. Screenshot of *MicroWorlds LOGO.*

All the media used for design, from traditional pen and paper to sophisticated computer programs, locate the designer in a microworld. With pen and paper, the designer is confined by a two-dimensional surface of some fixed area, while with wooden blocks she cannot construct gravity-defying objects. Microworlds based on computer languages tend to emphasize programming constructs, for example, recursion, conditionals and loops—such that the limitations of the language provide the boundaries of the microworld. Designers often work in very restrictive microworlds in order to refine their particular design methodology. *Shaper2D*, the program described in this thesis, is a microworld for exploring basic, non-parametric shape grammar theory, just as the aforementioned approaches, *DBN* and *DMD*, are microworlds for exploring graphic design and the design process respectively.

The *Shaper2D* microworld emphasizes spatial relations, rules and recursion, and is arguably more restrictive than the microworlds of *DBN* and *DMD*. With these systems, the designer is limited to the confines of the particular programming language and/or CAD environment that she is using, but beyond that she can create an infinite number of designs and shapes. The designer can create an infinite number of designs using *Shaper2D*, but in this instance she is limited to a palette of four pre-programmed shapes (with their respective labeling positions), two rules, and a predetermined maximum number of iterations.

Visual versus Textual Computation

One of the prevailing views of computers is that they are formal symbol manipulators (MITECS, 1999), where symbol refers to "...any causally efficacious internal token of a concept, name, word, idea, representation, image, data structure, or other ingredient that represents or carries information about something else."

Computation is usually associated with numbers and text. A computer program comprises a sequence of textual, symbolic or numerical instructions. The computation addressed in this research concerns visual, or spatial, algorithms. It is important to note the differentiation between traditional symbolic computation, and the less conventional approach of computing with spatial information.

Tools for Learning Geometry

To understand the different microworlds for computational design it may be helpful to consider the following microworlds for learning geometry. As with computational design, the process used for construction and manipulation of objects provides the main difference between geometry microworlds.

The difference between a "dynamic geometry system", which comprises an interface that permits direct construction and manipulation of objects, and a programming environment is the visual as opposed to symbolic interactivity experienced by the user (Healy & Hoyles, TBA).

Computer code makes explicit the sequence of moves a user makes when deriving a solution to a problem, whereas a dynamic geometry system produces instant feedback whenever the user makes a modification. To illustrate this distinction, consider the *Cabri-Géomètre* (Cabri) and *Turtle LOGO* systems.

An example of a dynamic geometry system is *Cabri* (see figure 5). *Cabri* allows the user to draw a visual description of geometric objects and physically manipulate them while maintaining their geometric relationships. On the other hand, *Turtle LOGO* is a programming environment that requires the user to type in a series of instructions to create and manipulate objects. *Turtle LOGO* (see figure 6) is a very powerful environment for experimenting with certain kinds of mathematical operations. It provides a symbolic language with which users can write (and debug) formal descriptions of geometric objects, which are then be rendered visually by the program.

Shape Grammars as an Example of Computational Design

Algorithms for Spatial Design

Most familiar computational algorithms are symbolic. Rules are given using text, numbers or symbols. Output is also textual, or text that is translated into something visual. In contrast, shape grammar rules and computations are spatial. Shape grammars are visual, spatial algorithms for creating, or computing, and understanding designs. Simply put, they are a visual implementation of computational design.

Shape Grammars

George Stiny and James Gips first presented shape grammar theory in the early 1970s, and stated that,

> "Shape grammars are similar to phase structure grammars, which were introduced by [Noam] Chomsky in linguistics. Where phase structure grammars are defined over an alphabet of symbols and generate one-dimensional strings of symbols, shape grammars are defined over an alphabet of shapes and produce n-dimensional shapes."
>
> (*ibid*, 1972, p.127-128)

Shape grammars have been proven to be equivalent in computational power to Turing machines, (Stiny, 1975). Knight (1994) describes them as "...an established paradigm in the study of form and composition". They are used to formulate a 'language' of designs through the iterative application of a set of shape rules. Shape grammars can be descriptive—used to analyze and describe existing designs—or generative—used to generate original designs.

Although we generally expect computation to be numerical and quantitative, shape grammars bring together both spatial and formal elements. This is an unfamiliar condition, as traditionally people tend to deal separately with these two kinds of materials.

A shape grammar is defined in terms of shapes and shape rules. A shape comprises any finite grouping of points, lines, planes and solids. Shape rules are instructions that are applied recursively to generate a design.

A shape computation is a sequence of designs, beginning with an initial shape. Each design is created from the previous design by the application of a rule. A computation describes the process used for design development.

Design 1 $\Rightarrow$ Design 2 $\Rightarrow$ Design 3 $\Rightarrow$ Design 4

There are different types of shape grammars, such as basic, non-parametric, and parametric, defined by various restrictions on rules and rule applications. The work presented here considers only basic shape grammars (Knight,1998b). Basic grammars are a highly restricted type of shape grammar that are deterministic and non-parametric. For a more detailed discussion about this and other types of shape grammars, see Knight (1994; 1998b).

With basic shape grammars, rules are applied in sequence from first to last, and then repeated again any number of times. The computation continues recursively until the designer finds a satisfactory design. Basic grammar development (see figure 7) comprises:

- shapes
- spatial relations
- shape rules
- shape grammar
- designs

All the stages are performed in the context of a particular agenda or design problem, as is the case with traditional design. Input from the outside drives the development of a shape grammar, so although it might appear to be a linear, fixed process, the designer is able to enter at any stage and make revisions, as she would do in a non-computational situation.

The elementary unit for a basic grammar is a shape (see figure 7a). Spatial relations (see figure 7b) represent local relations between shapes, in other words, how the shapes are arranged. Given two shapes A and B, a spatial relation between the shapes is denoted as **A + B**. Two rules that can be defined in terms of the spatial relation **A + B** are:

$$\mathbf{A \rightarrow A + B}$$

$$\mathbf{B \rightarrow B + A}$$

These rules specify how to add a shape in accordance with the spatial relation.

A rule can be applied in different orientations. The various spatial transformations that take the shape on the left-hand side of the rule and match it with a similar shape in a design

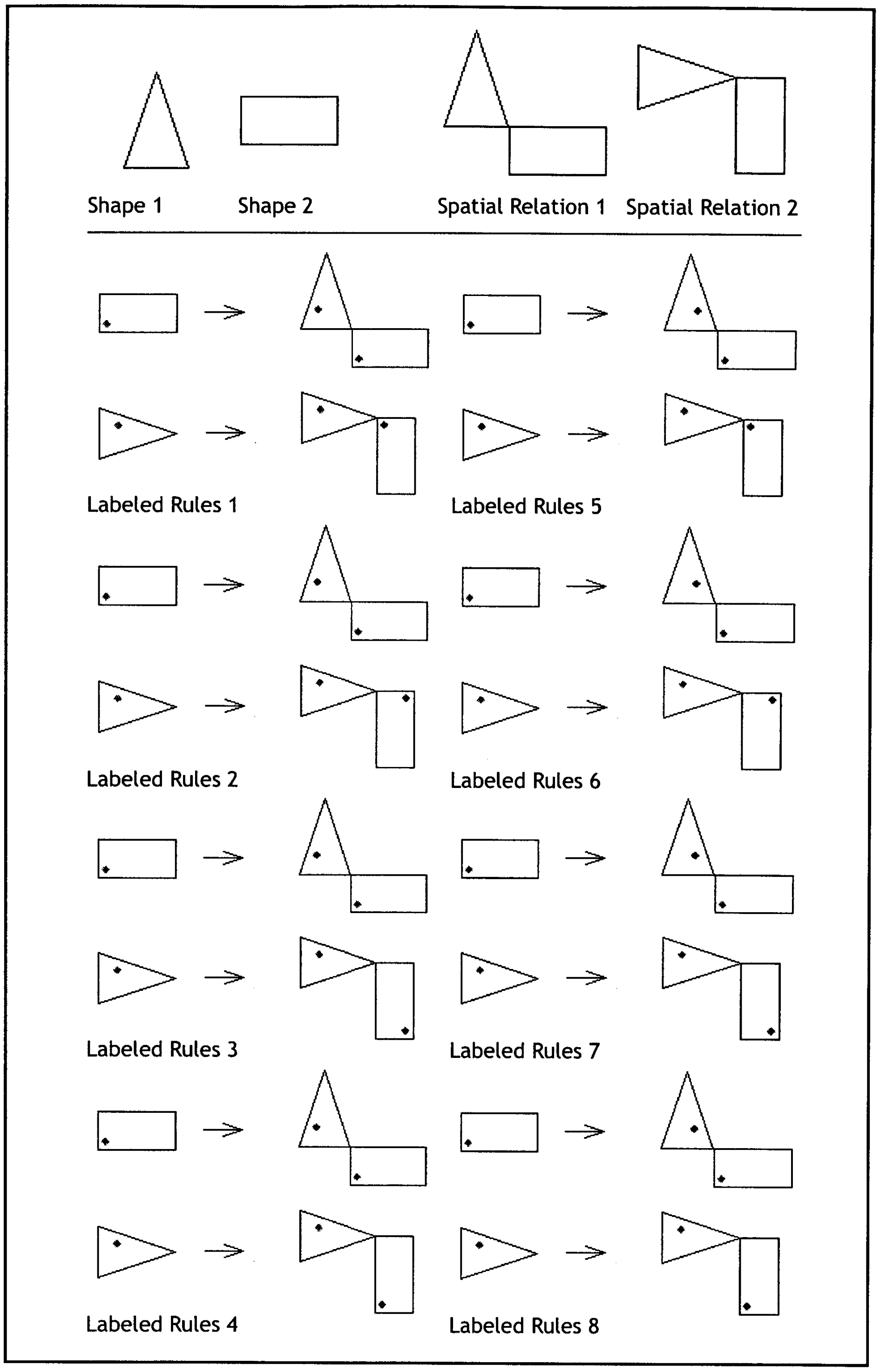


Figure 7. An example of the development of basic shape grammars.
7a: Shapes 1 and 2 | 7b: Spatial Relation 1 and 2 | 7c: Labeled Rules 1-8

determine the different ways a rule can be applied. These transformations include translation, rotation, reflection and scale. The number of different designs that can be produced by the rule applications—the design space—is calculated in terms of the symmetries of the component shapes in the rules. The symmetry of the shape on the left-hand side of the rule determines the number and type of matching transformations. For example, a rule based on spatial relation composed of two squares can be applied in eight different ways because the order of symmetry of a square is eight.

The different ways of applying a rule can be distinguished and controlled using labels. Labels remove the symmetry of the shapes, so that a rule applies in only one way. The number of labeled rules (see figure 7c) relates to the number of designs that can be generated, as each labeled rule generates a different design. When applying a labeled rule, the labeled shape A is matched with a labeled shape in the design, and labeled shape B is then added.

Basic grammars are deterministic. The designer has no choice of rules to apply or how to apply them. Although basic grammars can include any number of rules from any number of spatial relations, the designer is placed in a constrained environment. However, rather than limiting creativity, this limitation can liberate the designer by allowing her to explore possibilities that hitherto would not have been contemplated without the use of computation.

Shape Grammars in Design

Gero (1996) comments that, "Creative designing... can be described as perturbing the schema to produce unexpected and incongruous results." This theme of emergence is central to computational design and shape grammar theory. Predicting the outcome of a rule application becomes more difficult as the generative power of a shape grammar is augmented (Knight, 1998). This leads to new, interesting, unpredictable designs that are situated in the designer's distinctive approach.

To formalize a particular design approach, (consciously or subconsciously) the designer may seek to experiment with new designs based on a given spatial arrangement, thus developing an adaptive design process that generates a body of related designs which are variations upon a theme. An example of this can be seen by analyzing the work of Alvar Aalto (see figure 8).

Shape grammars can be seen as creative in a more pragmatic sense. When first defining a vocabulary of shapes, the designer has to make a conscious decision about which shapes to use and which to leave out. She might make this choice on a purely aesthetic basis, or perhaps the shapes might be chosen because of some contextual reason. For example, site conditions and constraints could lead the designer to try a combination of different shapes in order to suit a particular need.

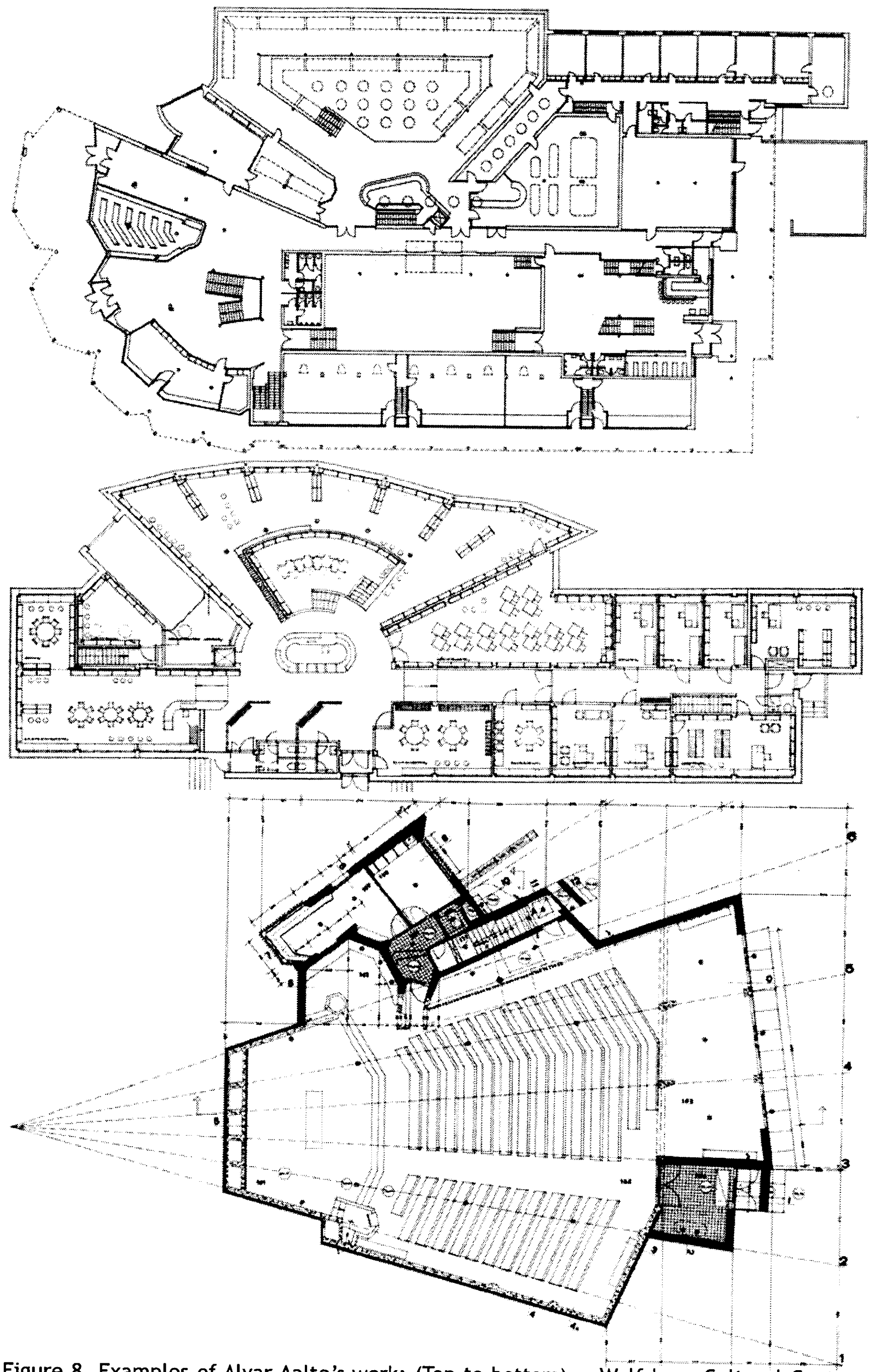


Figure 8. Examples of Alvar Aalto's work: (Top to bottom) -- Wolfsburg Cultural Center, Germany; Library in Seinäjoki; Parish Center, Wolfsburg, Germany.

The choice of shape can characterize a designer's work (for example, Gehry), whereas sometimes the spatial relations between the shapes can epitomize the zeitgeist of a particular movement (for example, the Deconstructivists during 1980s/90s).

The way these shapes are relayed spatially also forms part of the creative process. The designer deliberately imposes constraints on the design, both through the juxtaposition of the shapes (spatial relations) and by applying shape rules. These rules provide a chance to fine-tune a design. After considering the scope of the design space, the designer imparts her creativity by choosing which design to pursue.

In essence, despite the constraints of utilizing shape grammars during the creative process, the rigor of limitations can help free the designer to make more considered design choices. It should be remembered that the generation of a grammar is itself the product of creative thought and experimentation

Computer Mediated Shape Grammars

Hand computation and Computer computation—Complementary Systems?

"The idea behind digital computers may be explained by saying that these machines are intended to carry out any operations which could be done by a human computer."

(Turing, 1950, p.436)

Although it has been widely agreed that hand computation is essential for a comprehensive understanding of the concepts involved in shape grammar[1], This thesis poses the argument that traditional analog systems limit the student's imagination by inhibiting experimentation. The systems presently used in the classroom for introducing shape grammars are tracing paper, for learning two-dimensional computation, and wooden "Froebel" blocks, for three-dimensional computation. This is not to say that hand computation cannot be used for experimentation. Indeed, it has been used successfully both for analyzing designs and developing new designs (for examples see Knight, 1998b and Knight, 2000). However, the contention here is that the computer enables the designer to explore the different possibilities quickly, something that would not be possible by hand.

A computer can substantiate design possibilities much more quickly than a person could. Sometimes, the computer performs the calculations very quickly whilst the designer's mind works more slowly, so hand computation can help to make the process more transparent. On the other hand, if the designer conceives of an idea, the computer can allow her to explore it and come up with many design possibilities in a short space of time. But making selections can be difficult, when so many possibilities become available. How can the designer choose which direction to take?

In principle, given time and patience, a designer can perform any kind of computation by hand. Computing by hand can be a long, laborious task, one that is not, arguably, conducive to the exploration of shape grammars. When a computer is used to facilitate shape manipulation and computation, judgements can be made so that the process becomes qualitative rather than quantitative. That is to say, allowing the designer to make investigations with shape grammars at a speed comparable to thinking about an idea and immediately sketching it with pen and paper encourages a qualitative response to designs produced. Instead of becoming married to an imperfect scheme that may have taken a long time to produce by hand, the designer is freed by the computer from this design encumbrance and advocates experimenting in order to derive a satisfactory proposal.

However, a counter-argument can be made to using the computer. In order to judge which way to proceed in a computation, the designer needs to be knowledgeable of the principles involved before utilizing computer applications. She must understand what the rules are doing before making her decision, and avoid the compulsion to make arbitrary, uneducated decisions through "playing around" with the software. One of the aims of this thesis is to explore this dichotomy further.

Using the computer can open up designing with shape grammars, as designs tend to get very complicated very quickly. Designing can thus become more comprehensible—and designers are encouraged to persevere with an idea rather than give up due to frustration or impatience.

Effect of media on the design

It is interesting to consider the effect that media has on the design process. In the studio, creating physical representations of a design from only paper or wood can supress the designer's imagination due to the limitations imposed by the material. Conversely, exploring a range of materials and microworlds can amplify the designer's comprehension of an idea. In his study of the Reggio Emilia pre-school system in Italy, which advocates an artistic, hands-on approach to idea exploration using a variety of media, George Forman (1994) observes a way around this problem,

> "...one, find the best medium for expressing an idea and, two, make creative compromises with the medium at hand. The compromises often yield totally new perspectives on the theme or concept being represented. Furthermore... children should learn to traffic between a variety of media. The traffic across media will help children produce new representations informed by old ones, and to revisit and revise these old representations as well."
>
> (*ibid*, p.39)

Although Foreman is talking about pre-school children, it is easy to see how the same can be said for the approach that designers take. We know that designers have traditionally worked with different media, both two-dimensional and three-dimensional, for developing ideas. Each microworld that a designer uses has certain limitations, within a given context, that can help with the perception and development of an idea. However, it is beneficial to extend the range of media in order to gain a comprehensive understanding of that idea.

Foreman goes on to say,

> "Media also differ in their modularity, their persistence across time, and the amount of physical feedback they provide."
>
> (*ibid*, p.40)

This thesis argues that computer implementations of shape grammars allow the designer to explore options and emergent possibilities that would not have been feasible or indeed visible when using a by-hand method.

Shape Grammar Interpreters

It was natural that the progression would be made from hand computation to using computer programs for experimenting with shape grammars. Shape grammar interpreters automate the process of applying a shape grammar to create designs. Spatial relations and rules are input by the designer either visually or symbolically, and are subsequently encoded and executed by the program. Terry Knight (1998b) points out that,

> "...the theoretical and computational foundations of shape grammars have proceeded more or less independently, and in advance, of programming issues... The appeal and success of shape grammars is undoubtedly due to a carefully wrought theoretical basis. However, continued growth in applications demands catch-up work in programming."
>
> (*ibid*, p.87)

The first computer programs developed for exploring shape grammar concepts were not concerned with the importance of a user interface, thus restricting their use to experienced shape grammarians and programmers. These interpreters initially concentrated on implementing any two-dimensional shape grammar, but were later followed by systems for implementing three-dimensional shape grammars.

Mark Tapia's program *GEdit* (1999) was the first two-dimensional shape grammar interpreter that emphasized usability through the development of a visual user interface. Subsequent to this, Yufei Wang wrote *3D Shaper* (1999), a three-dimensional interpreter, which includes a static user interface that allows the designer to input numerically values for the shapes, spatial relation and rules.

More recently, interpreters have been written at MIT, which are installed as plug-ins for *AutoCAD* (for example, Gabriela Celani's *AutoGrammar*, Luis Romão's *Shaper 2.5D*). These enable the user to experiment with two- and three-dimensional shape grammars directly within a preexisting CAD program.

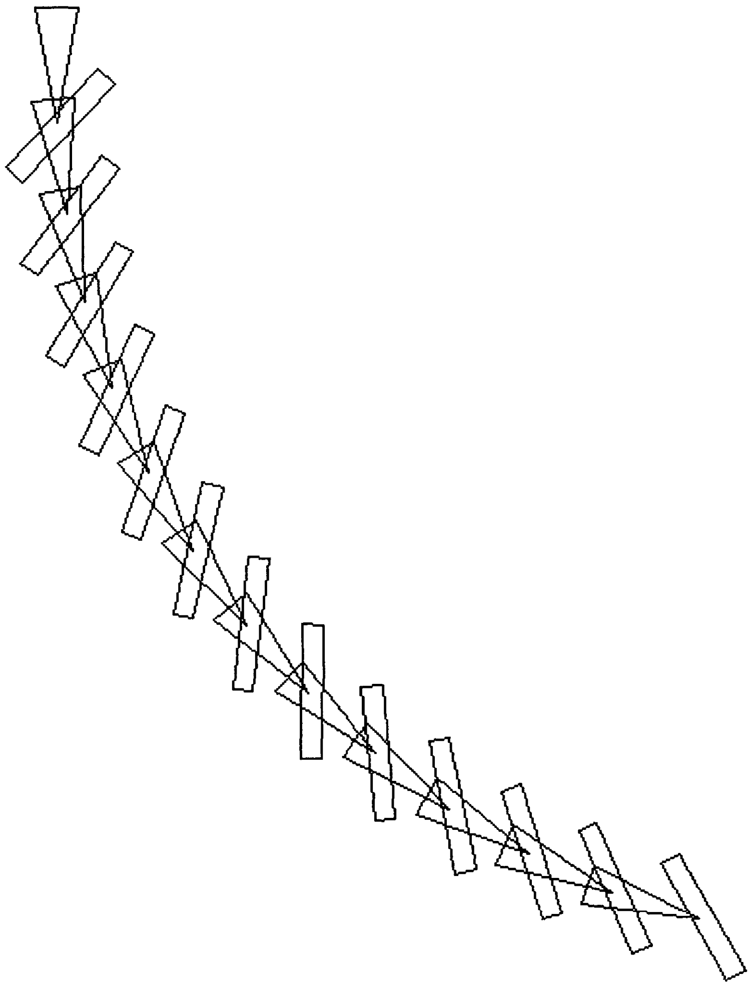
Presently, the preferred approach to the development of shape grammar interpreters is to design specific tools for specific functions, rather than "universal tools". The program presented in this thesis, *Shaper2D,* was developed by the author as a dynamic, visual interpreter for exploring basic, two-dimensional shape grammars.

"All pictoral form begins with the point that sets itself in motion... The point moves... and the line comes into being-the first dimension. If the line shifts to form a plane, we obtain a two-dimensional element. In the movement from plane to spaces, the clash of planes gives rise to body (three-dimensional)... A summary of the kinetic energies which move the point into a line, the line into a plane, and the plane into a spatial dimension."

Paul Klee

Chapter Two

Shaper2D: A Dynamic Shape Grammar Intrepreter



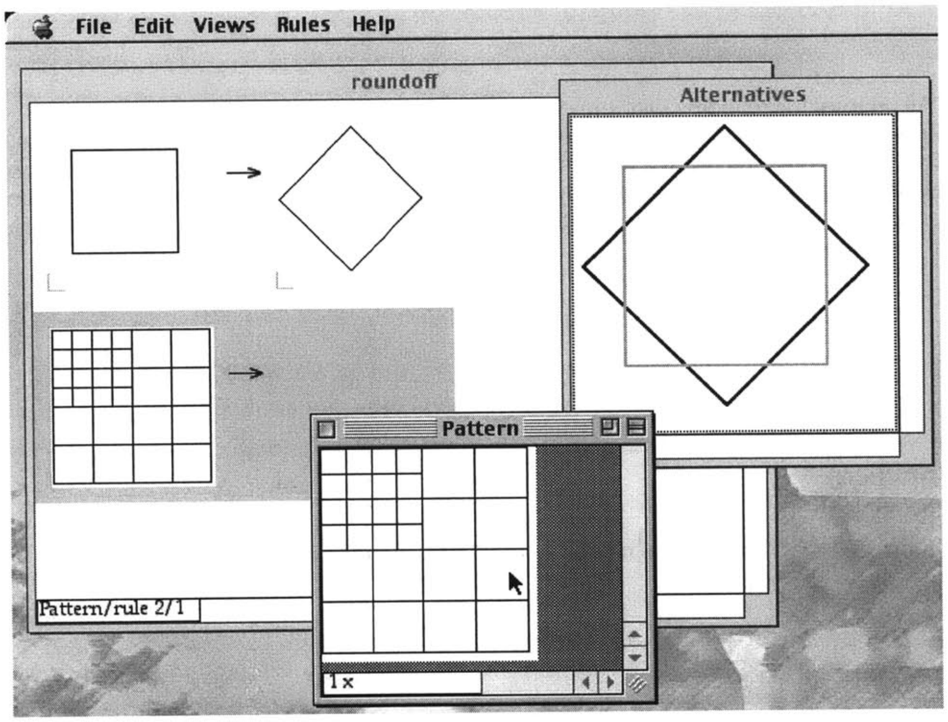
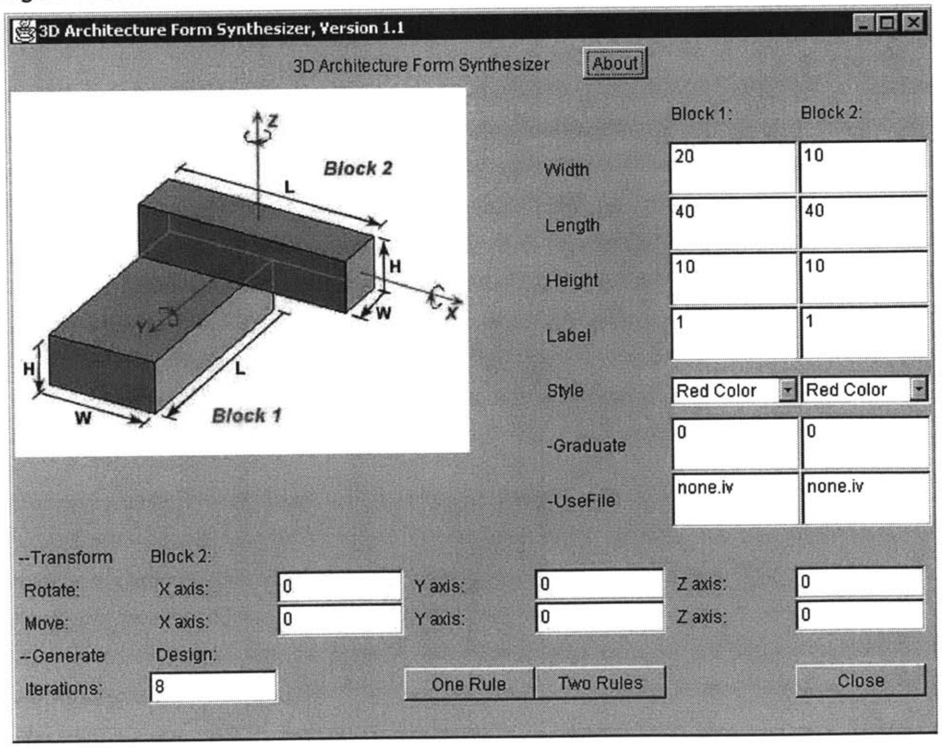


Figure 9. Screenshot of *GEdit*.


3D Architecture Form Synthesizer, Version 1.1
3D Architecture Form Synthesizer
About
Z
L
Block 2
H
W
X
Y
H
L
W
Block 1
One Rule
Two Rules
Close

	Block 1:	Block 2:
Width	20	10
Length	40	40
Height	10	10
Label	1	1
Style	Red Color	Red Color
-Graduate	0	0
-UseFile	none.iv	none.iv

--Transform Block 2:
Rotate: X axis: 0 Y axis: 0 Z axis: 0
Move: X axis: 0 Y axis: 0 Z axis: 0
--Generate Design:
Iterations: 8

Figure 10. Screenshot of *3D Shaper* Interface.

Computer implementations of Shape Grammars

Overview

Shaper2D is not designed to displace user contribution to a design. Instead its aim is to create an engaging environment where the designer is encouraged to explore and experiment with basic, deterministic, non-parametric shape grammars. Previously developed software for experimenting with two-dimensional and three-dimensional shape grammars provide powerful tools to explore fundamental concepts, but are not conducive to practicing, or playing with, shape grammars. That is to say, although they have been used on occasion for classroom teaching, their role in the design studio has been less successful. Various reasons have been cited for this, including the lack of a "designer-friendly" interface, no instant feedback, and an unrestricted workspace that is only truly useful to more advanced uses of shape grammars. These are not criticisms of the software, as each application has some audience in current shape grammar pedagogy and research. However, the deficiencies listed above are those that *Shaper2D* seeks to address.

Other Implementations

To frame the motivation behind the development of *Shaper2D*, it might be helpful to consider two recent implementations of computerized shape grammars, *GEdit* and *3D Shaper*.

The first computer implementation of shape grammars that included a viable user interface was the Apple Macintosh based *GEdit* (see figure 9) developed by Mark Tapia (1999). This self-contained program for generating two-dimensional, non-parametric shape grammars emphasizes that the "...drawing is the computation" (P.67). Tapia sought to minimize user distraction, both in terms of obscuring parts of the program and perverting the design flow, by limiting the use of drop-down menus and dialog boxes. He instead implemented an object-specific radial menu, which only appears when needed to ensure that the user's focus is not removed from the design process. This is a program for shape grammar experts, since it is very open-ended and allows for the free exploration of almost any kind of non-parametric, two-dimensional shape grammar.

A more recent implementation concerned three-dimensional shape grammars. Yufei Wang (1999) developed *3D Shaper* to simulate the manipulation of wooden Froebel blocks. This is a very powerful program that enables the user to experiment with three-dimensional grammars in ways that are difficult to visualize mentally, and impossible to perform physically. *3D Shaper* was written in Java, and was limited originally to being run on SGI computers[1]. It comprises a static interface (see figure 10) that demands that the user type in numerical parameters to determine the size and type of shapes, as well as the spatial relation between the shapes. The program calculates the rules and the results of computations, and writes these as SGI open inventor files. In order to view the results, which are three-dimensional representations, the user must open a secondary program (such as *IV Viewer* or TGS *3Space Assistant*), requiring that the design process be broken up by the need to switch between programs. The designer is also obligated to adjust to the two different information display modalities, which are relative to the current program—symbolic (*3D Shaper*) and visual (the viewer).

What is different about *Shaper2D*?

Previously, programs have been developed expedite the process of design using shape grammars. However, either they lacked a user interface, were too general, or did not account for the dynamic, interactive environment sought by a designer. *GEdit*, whilst introducing the notion of an elegant interface, is a very general microworld. This is suitable for advanced shape grammarians wishing to visualize a particular derivation, but impractical for novice users who require a more constrained environment. For many architects, conceptual designing is fluid and dynamic—with *3D Shaper*, inputting numbers then opening up a separate viewer in order to see the design, does not tally with the designer's habitual design process. *3D Shaper* is an invaluable tool for experimenting with three-dimensional, orthogonal shape grammars, however, the static interface is not analogous to the design process, and hence does not facilitate quick, experimental design.

Shaper2D is a localized shape grammar microworld for generating designs using very restricted kinds of shape grammars. It was written to overcome the platform-specific limitations imposed by previous interpreters. The application has been run successfully under the major operating systems (Windows, Mac OS[2] and Linux), and the applet runs under any web browser capable of running Java™ 2[3] (such as *Netscape 6, Internet Explorer 5, Opera 5*) irrespective of the operating system being used.

The Development of *Shaper2D*

Origins

The purpose of *Shaper2D* is to provide a situated, constructive approach to learning about and understanding computational design using shape grammars. Its premise is that by using the program to tackle a design problem, the user has the opportunity to construct both an advanced understanding of what shape grammars are and how they can be utilized in a real-world situation.

Shaper2D was developed as a computational tool for designing, which can be used intuitively by a designer. Hence it endeavors to provide an interface that gels with the designer's *mode d'emploi*. As discussed in the previous chapter, the usual software that designers utilize tends to facilitate the drawing-up process rather than stimulate experimentation with design concepts. The prototype software presented here, on the other hand, is intended for experimentation with computational design. It was borne out of a need for a tool to encourage and facilitate experimentation with basic, two-dimensional shape grammars (Knight, 1998b). This microworld was designed to be playful, and to engage the user with a meaningful, supportive learning environment, providing the means to create and explore new ideas about shape grammar theory. The limitations built into the system are deliberate—to constrain the user to the exploration of a small sub-section of this computational design methodology.

Is *Shaper2D* a pedagogical, learning or design tool?

The program was originally conceived as a strictly pedagogical tool for assisting shape grammar teaching. However, after considering the differences between teaching and learning, the purpose of the software was revised to that of a learning tool. When differentiating between teaching and learning, John Holt (1989) argues vehemently that, "Learning is not the product of teaching," and, "teaching does not make learning... Learners make learning. Learners create learning" (p.160). This is a moot point. In any case, *Shaper2D* aims to help users construct their own understanding of shape grammars.

It is important to note the difference between a pedagogical tool and a tool for designing. Once development of the program had commenced, it was realized that the didactic focus was too narrow, and that the interactivity of the program lent itself to more practical applications—in other words, it was useful for designing with shape grammars.

There was a difference of opinion about what *Shaper2D* would be used for: learning, designing, or "learning by designing". This debate provided the framework for testing the program and developing a notion about its place in shape grammar pedagogy.

Shaper2D and Constructionist Theory

Shaper2D was developed in correspondence with the Constructionist philosophy of "learning by doing". In essence, this program is designed to facilitate the comprehension of shape grammar concepts through situated, real-time experimentation.

It is important to realize that there is a difference between knowing a fact and being able to perform something using an acquired skill—in other words, "knowing that" and "knowing how" (Papert 1993). To draw that distinction Mitchel Resnick et al. (1996) provide an interesting metaphor of learning to play the piano as opposed to merely listening to music on a stereo,

> "The stereo has many attractions: it is easier to play and it provides immediate access to a wide range of music. But "ease of use" should not be the only criterion. Playing the piano can be a much richer experience. By learning to play the piano, you can become a creator (not just a consumer) of music, expressing yourself musically in ever-more complex ways. As a result, you can develop a much deeper relationship with (and deeper understanding of) music."

With shape grammars, it is one thing to be able to reproduce a design using a given spatial relation or rule, and another to be able to use the concepts in an educated way. By generating or analyzing new or unfamiliar designs, this verifies that the knowledge is "owned". The fundamental purpose of *Shaper2D* is to encourage designers to "play shape grammars" rather than just regurgitate shape grammar concepts mindlessly.

Designing Design-Mediating Software

The aim of *Shaper2D* was to create a program that would enable a constructive understanding of a computational design process, rather than create a program to automate the drafting process. This provided the motivation to situate the creation of this visual, computational design program in a shape grammar context.

It was important to examine the means by which design software is created, from a designer's, rather than programmer's, viewpoint. In order to achieve this, it was necessary to develop a practical understanding of computer programming. It is important to remember that this study is not solely about the development of a new piece of software for use in the design classroom, but is about developing software for a specific purpose.

Implementing *Shaper2D*

Why Java?

The Java programming language was used as it lent itself to object-oriented behavior of the program. It was the natural choice for creating both a stand-alone application and a program that can be run on the internet.

Interface and Interactivity

The interface was a crucial aspect of *Shaper2D's* development. Several researchers in the field of shape grammars had noted that whilst useful implementations of shape grammars have been developed, little attention had been focused on the interface (March, 1997; Knight, 1998b; Tapia, 1999). This is surprising given the prospective audience—that is, designers.

Visual seductiveness is very important to designers. When developing this program, many graphic and interface design issues had to be considered. Primarily, the interface had to embrace the inherent visual uniqueness of shape grammars. The decision to exclude any need for manual typing-in of parameters or instructions was made at the program's inception. It was important to develop a program based on a visual interface without any need to input numbers to generate designs.

> "Transparency in its simplest form may just imply that the inner workings of an artifact are available for the learner's inspection: the black box can be opened, it can become a "glass box". But there is more to understanding the use and significance of an artifact: knowledge within a community of practice and ways of perceiving and manipulating objects characteristic of community practices are encoded in artifacts in ways that can be more or less revealing."
>
> (Lave & Wenger, 1991, p.102)

Without doubt, the most transparent way of doing shape grammars is when computations are done by hand. *Shaper2D* strives to make the shape grammar process as transparent as possible through interactivity and dynamic response. However, because the actual computations are concealed the program could be described as a black box. Rule applications are hidden from the user. Physically flipping and rotating drawings on tracing paper, or manipulating three-dimensional wooden blocks, makes the act of performing the spatial transformations more personal—the designer has direct contact with the processes involved. In the classroom, using shape grammars thus becomes a social activity. Students are aware of peripheral activities and can observe easily and directly how others carry out rule applications and design generation. A community of practice is formed.

The utilization of a dynamic interface endeavors to offset the inherent black box disadvantages, so that the designer is informed of the effect of her actions immediately. The user can experiment quickly with different rules and iterations in order to compare the outcomes of different ideas and grammar applications. The program can also be shared using third-party software applications such as Microsoft NetMeeting, which allows collocated and remote designers to form virtual communities of practice when experimenting with new concepts and designs.

The Mathematical Engine

The shape grammar rules applied by *Shaper2D* are linear transformations. The program calculates these linear transformations using standard algebraic techniques. Transformations, including rotation and translation, are represented by matrices acting on augmented vectors. The use of augmented vectors is a bookkeeping technique, which allows rotation and translation to be combined into a single matrix, rather than having to keep track of each transformation separately. This simplifies inverting the transformations and calculating the combination of transformations (to see how this is accomplished, refer to the *Shaper2D* source code, listed in Appendix A).

The Application and Applet

Shaper2D was developed as a Java applet (see figure 11) as well as a stand-alone application (see figure 12). This decision was based on the advantages of using the internet for enhancing portability and cross-platform compatibility.

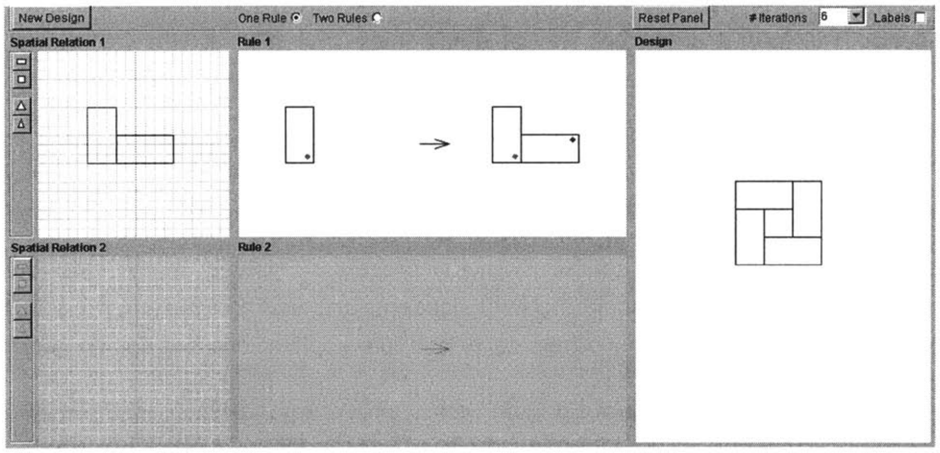


Figure 11. The *Shaper2D* Applet.

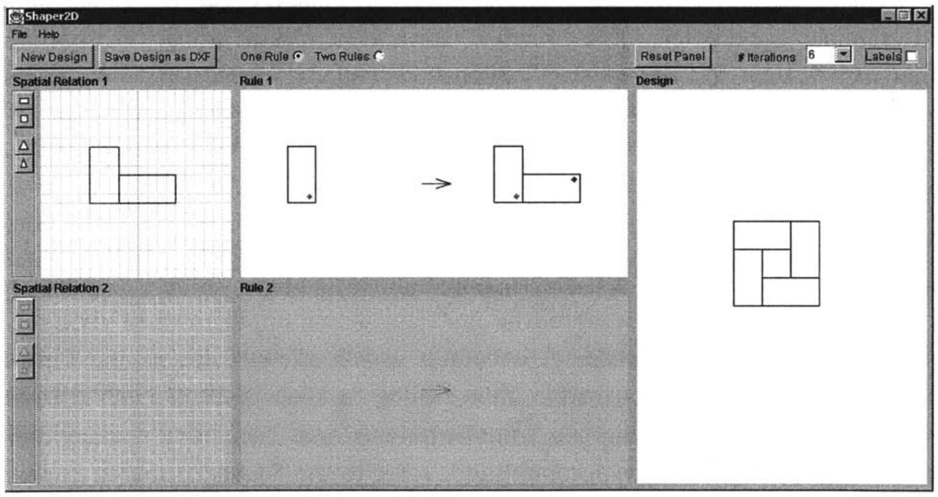


Figure 12. The *Shaper2D* Application.

On the other hand, the application has the advantage that once the user derives a design that she would like to keep she is able to save it as a DXF (Data Exchange File). This enables the design to be imported into third-party CAD applications such as *AutoCAD* or *Microstation.*

How to use Shaper2D: An Introduction to the User Interface

Overview

The *Shaper2D* user interface is composed of a menu bar, five panels and two toolbars for selecting different shapes. Two of the panels are grayed out by default when the program is first run. This was a pedagogical decision to emphasize the difference between using one rule and using two rules.

Whenever the user manipulates a shape in a Spatial Relation panel, the Rule panel(s) and Design panel update in real-time. This enables the user to get a dynamic response to her actions. Similarly, when the position of a label is changed in the rule window, the iteration window updates immediately, so that the effect of the rule change on the design can be seen straight away. This is a key feature of the program, as it enables the user to consider the implications of each design move instantaneously.

If the user elects to start a design from scratch, she clicks on the 'New Design' button that restores the default spatial relation. For reassurance, the program will ask for confirmation of this decision so the user has a level of protection against losing information by accidentally pressing it.

To provide an in-depth guide to *Shaper2D*, each panel will be described as follows:

- Spatial Relation Panel: Selecting Different Shapes
- Spatial Relation Panel: Translating and Rotating Shapes
- Spatial Relation Panel: Resizing Shapes
- Rule Panel
- Design Panel

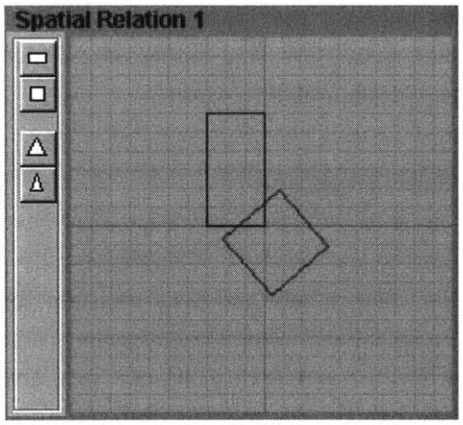


Figure 13. *Shaper2D*, Spatial Relation Panel: Selecting Different Shapes.

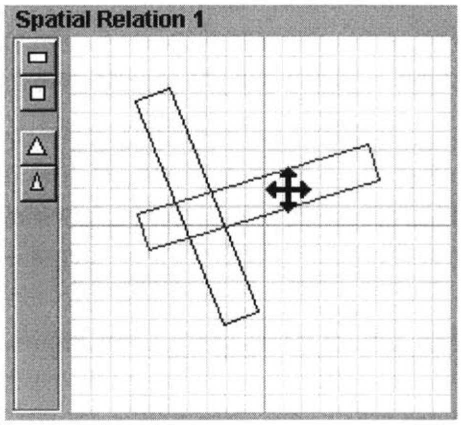


Figure 14. *Shaper2D*, Spatial Relation Panel: Translating Shapes.

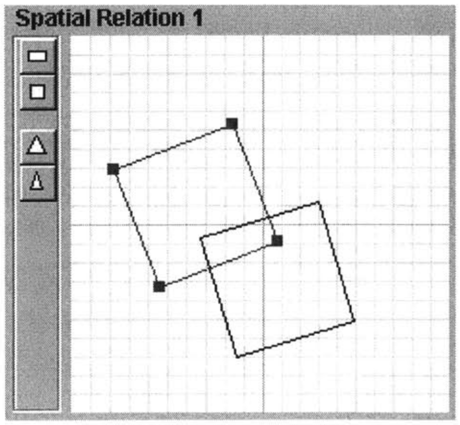


Figure 15. *Shaper2D*, Spatial Relation Panel: Resizing Shapes.

Spatial Relation Panel: Selecting Different Shapes

When one rule is selected, the user can only use one shape, whereas when two rules is selected she can choose two different shapes. Shapes are selected by pressing the buttons on the toolbar adjacent to the Spatial Relation panel (see figure 13). Each button corresponds to the shape icon displayed on it. Currently, there is a choice of four shapes: square, rectangle, equilateral triangle and isosceles triangle.

Spatial Relation Panel: Translating and Rotating Shapes

The spatial relation panels are for the selection and manipulation of shapes and spatial relations. Although the user can translate (move) and rotate both shapes in a relation, she is only able to resize (scale) the original shape in that panel. This shape is highlighted in blue. To emphasize the selected shape for translation or rotation, the shape is highlighted in red when the cursor is passed over it. To translate a shape, the user moves the cursor to the center of the desired shape. She can see that the cursor has changed to a "translate" cursor, indicating that she can now translate the shape (see figure 14). The shape is translated by pressing the left mouse button and dragging the shape to where the user wants it, before releasing the mouse button. To rotate a shape, the user moves the cursor to the center of the shape, but this time presses the right mouse button (or shift plus pressing on the mouse button if she is using Mac OS). To rotate the shape, the user drags the mouse around either clockwise or counter-clockwise. Once she has arrived at the desired rotation, the mouse button is released.

Spatial Relation Panel: Resizing Shapes

When the cursor passes over a corner of a shape, handles appear to inform the user that she has the option to resize (scale) the shape (see figure 15). To resize, the user moves the cursor to the desired corner, then holds down on the left mouse button and drags the corner to where she wants it, before releasing the mouse button. This is the same for each shape. However, how each shape is resized depends on that particular shape. Resizing squares and equilateral triangles is straightforward, as the shape retains the correct proportions as it is increased and decreased in size. Rectangles and isosceles triangles display different characteristics as the aspect ratio can be changed. However, a rectangle cannot be transformed into a square. This is a deliberate gesture, to emphasize to the user the particular symmetries inherent in each shape, and to reduce confusion when looking at the rules and the design. Similarly, an isosceles triangle cannot be changed into an equilateral triangle—if this is attempted, the isosceles triangle snaps back into an isosceles triangle. The reason for this restriction of shape manipulation is the shape's order of symmetry, which will be discussed shortly.

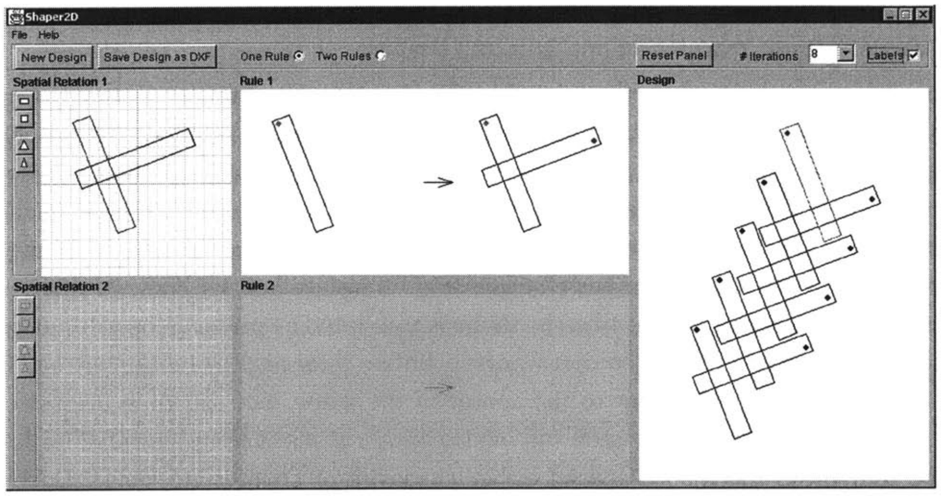


Figure 16. *Shaper2D*, Rule Panel: One Rule, with Labels selected in Design Panel.

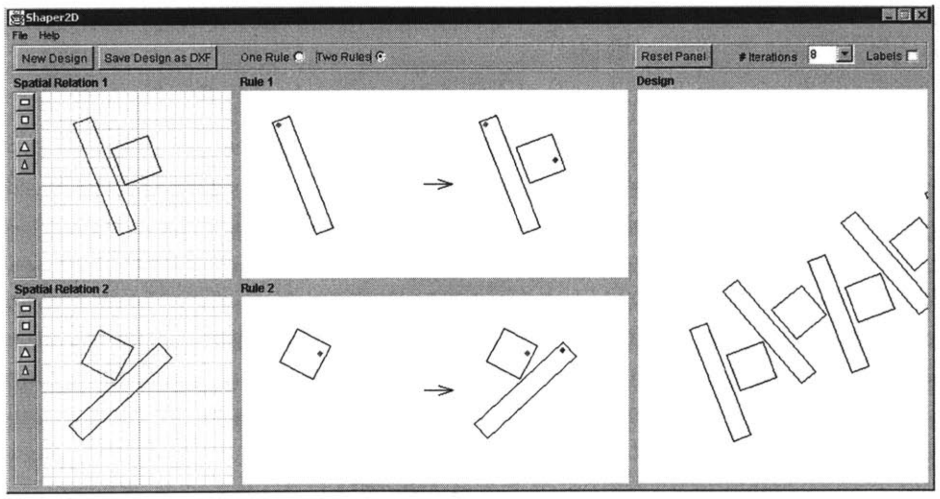


Figure 17. *Shaper2D*, Rule Panel: Two Rules.

Rule Panel

As mentioned previously, one rule or two rules can be used for deriving designs. With one rule, the user is limited to using one shape (see figure 16), whereas with two rules the user can use two different shapes (see figure 17). Although basic shape grammars can include any number of rules, it was decided to limit the number to two for pedagogical reasons. When the "two rules" button is pressed the "Spatial Relation 2" and "Rule 2" panels are activated. The user can then manipulate shapes in both spatial relation panels, and modify the labels in both rule panels.

To modify a label position, the user first moves the cursor over the shape containing the red label, so that the shape highlights in red. She then clicks on the shape with the left mouse button, and can see the label changing positions. To get an idea of the effect that this is having on the design, the user can look at the Design panel, where the design changes dynamically every time the label is repositioned. The number of positions that the label can be repositioned depends on the shape selected. This is because of the orders of symmetry inherent in the different shapes. The number of label positions we have to choose from are:

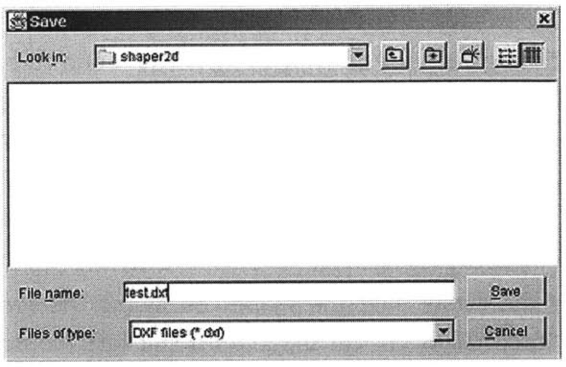
- Rectangle -- 4
- Square -- 8
- Equilateral Triangle -- 6
- Isosceles Triangle -- 2

Design Panel

The Design panel displays the design, which is updated dynamically every time a change is made to the Spatial Relation (s) or Rule(s). The number of iterations displayed is changed by clicking on the "# Iterations" pull-down menu. The maximum number of iterations is 25. The user might find it helpful to look at how the labels are positioned on the shapes during the derivation of a design. To do this, she clicks on the "Labels" checkbox—the labels from the previous rule iterations in black, while the current shape in the design is shown in red (see figure 17). If a design is too big to fit into the panel, the move or zoom features of the Design panel can be used.

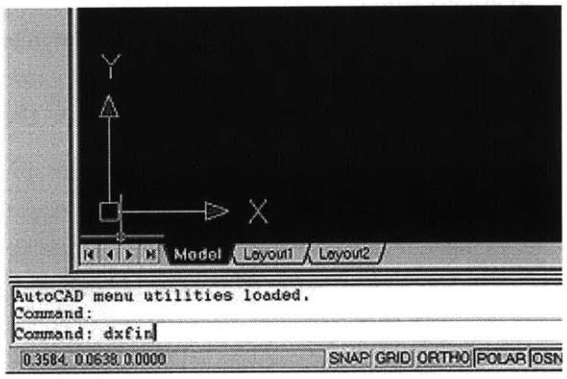
Moving is done using the same principles used for the Spatial Relation panel. Zooming is achieved by pressing on the right mouse button (or shift + mouse button for Mac OS) and dragging the mouse up or down, depending on whether the user wants to zoom in or out. The cursor changes to an "up-down" arrow when zooming is enabled, as a reminder of which way to drag the cursor. The zoom is set when the mouse button is released.

If the design is accidentally lost (by moving it off the panel, or zooming in or out by too much), the user should click on the 'Reset Panel' button—this repositions the design to the origin and restores the default zoom setting. As mentioned earlier in this section, if the *Shaper2D* application is being used, there is the additional option for saving the design as a DXF for importing into a CAD program such as *AutoCAD*. Details of importing into *AutoCAD* can be found below. To save a design, the user clicks on the "Save Design as DXF" button, and saves the design in the usual way (by giving it a name and placing it into the desired directory).



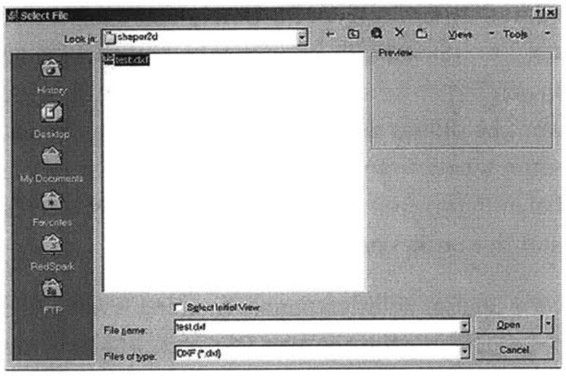
Look in: shaper2d
File name: test.dxf
Files of type: DXF files (*.dxf)

Figure 18. Saving as DXF from Shaper2D.



```
AutoCAD menu utilities loaded.
Command:
Command: dxfin
0.3584, 0.0638, 0.0000
```

Figure 19. Importing DXFs into AutoCAD.



Look in: shaper2d
test.dxf
File name: test.dxf
Files of type: DXF (*.dxf)

Figure 20. AutoCAD DXF Dialog Box.

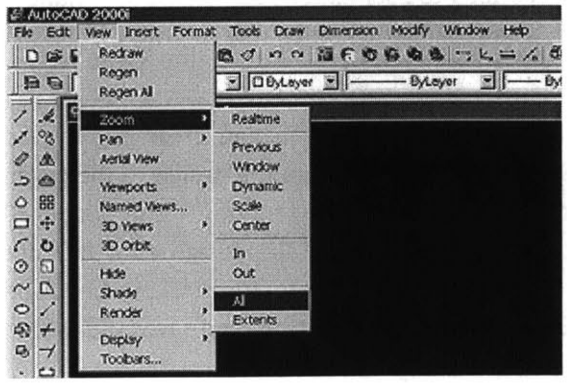


Figure 21. AutoCAD Zoom > All.

Beyond *Shaper2D*

Saving Designs

Shaper2D comes in two flavors—the applet and the application. The advantages of the applet are that it can be run over the internet if the correct Java plug-in has been installed. This allows users who may not be able to download and run the application to have access to the software. However, the applet version is limited, as it does not allow the user to save her designs.

A sample session is provided to demonstrate the ease of exporting designs into *AutoCAD* for further manipulation or insertion into a site plan or other design.

An Example: Exporting DXFs into *AutoCAD*

1. The user generates a design using *Shaper2D.*

2. To save the design, the user clicks on the "Save Design as DXF" button. The Save dialogue box pops up (see figure 18), and she can save the design wherever she chooses (e.g. on the C:\ drive or on floppy disk). If the user tries to overwrite an existing file with a new one, she is asked to confirm or cancel this operation.

3. The user opens *AutoCAD* (or any CAD program that allows DXF imports). To import a DXF file (see figure 19), the user types into the command line at the bottom of the screen: "`dxfin`"

The following dialog box then appears (see figure 20).

4. The user finds the directory where the DXF file was saved, selects the file she would like to import, and clicks Open. The file will then appear in the *AutoCAD* display window.

5. In *AutoCAD r.14* the design should be centered on the screen. However, in *AutoCAD 2000,* the display window may have to be re-centered. To do this, the user selects View > Zoom > All, as shown (see figure 21).

6. The design should then appear in the center of the display window.

Saving Spatial Relations and Rules

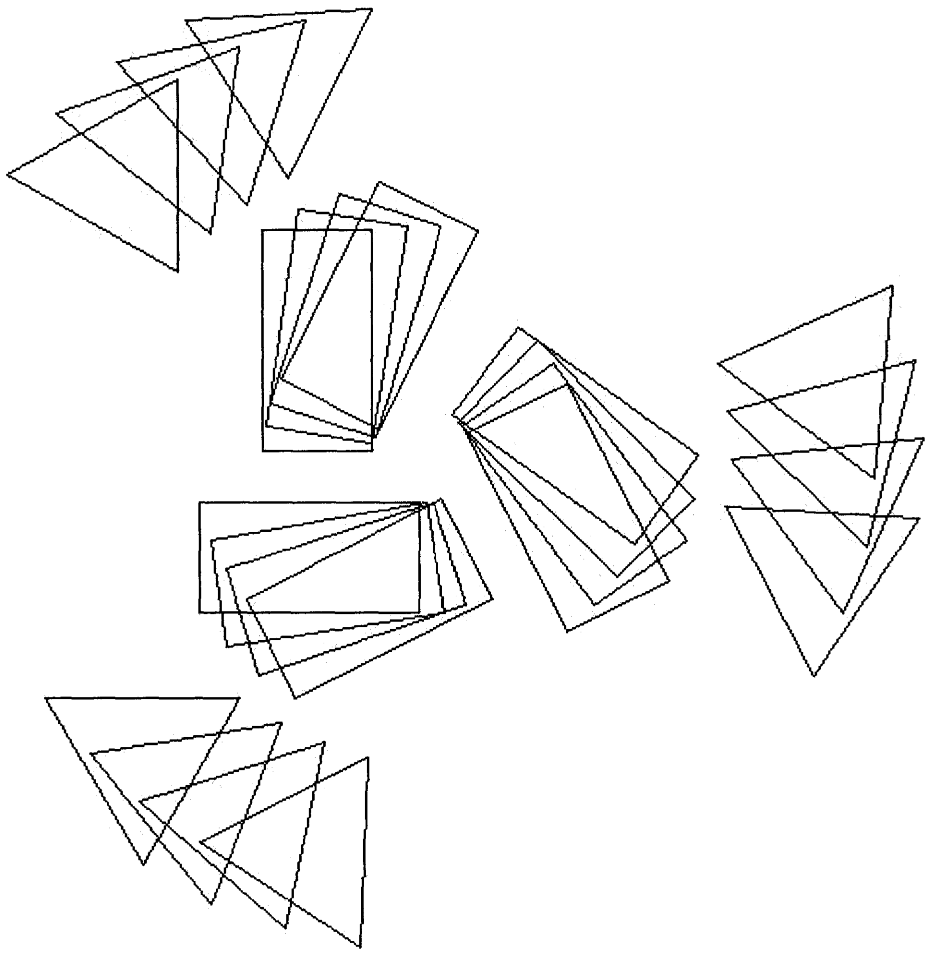
In response to feedback generated by the workshops, which will be discussed in the next chapter, it is planned that the next version of Shaper2D will also incorporate the facility to save the spatial relation and rules, for later "tweaking", thus expanding the program's educational value. Presently, to go back and revise the design, the user must reverse-engineer the spatial relation and rules from the saved design. In the future, the user will have the option to either save the spatial relation and/or rule in a specific format that permits re-opening of files in Shaper2D, or to save them as DXFs.

"Any new shape grammar interpreter... should be easy to use for the creation and execution of shape grammars; it should be visual and intuitive, reflecting the innate character of shape grammars; it should be sufficiently powerful to support interesting applications, but the emphasis should be on ease of use, elegance of design, robustness, and openness to improvements. Programs should be implemented in a language that allows them to be used on all common platforms."

(March, 1997)

Chapter Three

Shaper2D in the Studio



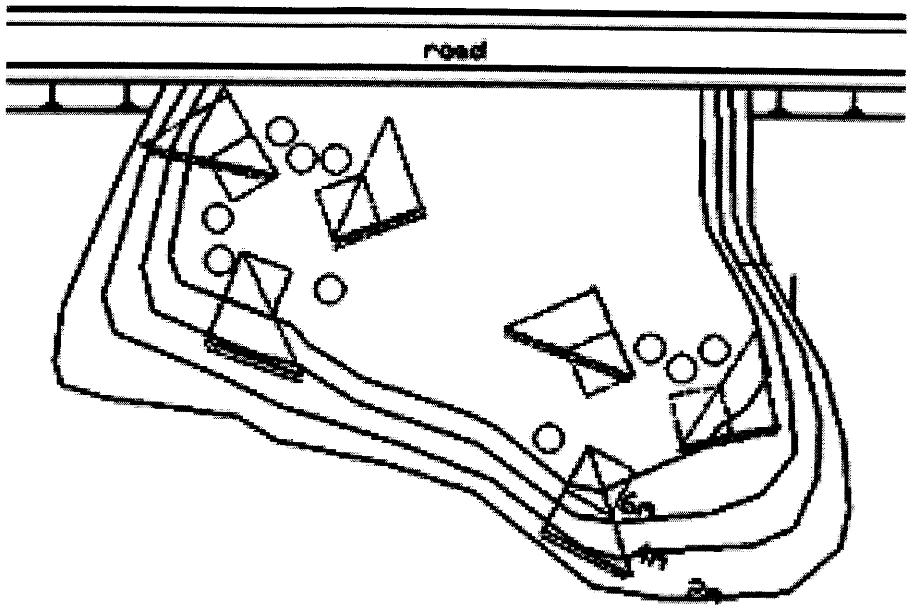


Figure 22. Study A: Soohyun Chang | Site Plan generated using *Shaper2D*, from Remote Collaborative Workshop.

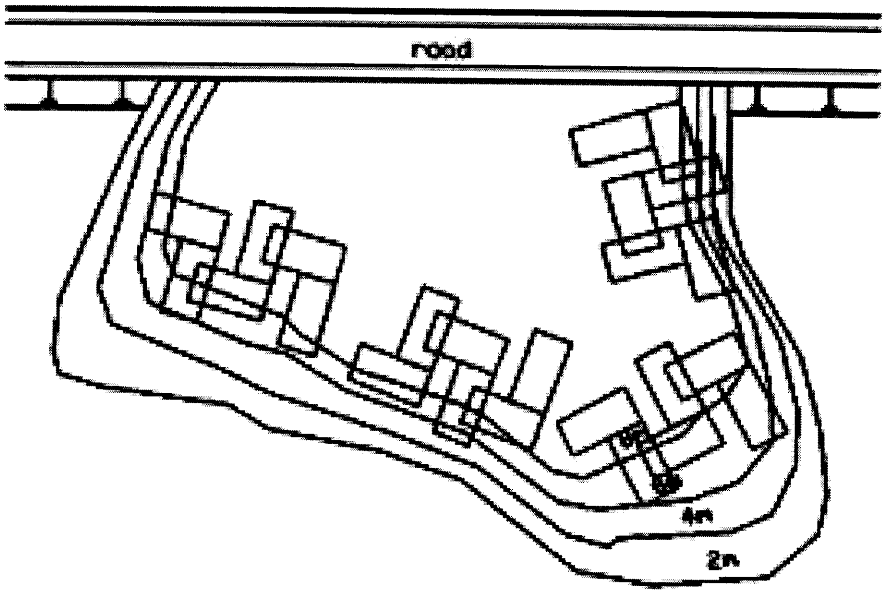


Figure 23. Study A: Kristie Tate | Site Plan generated using *Shaper2D*, from Remote Collaborative Workshop.

Purpose and Background to the Shaper2D Studies

In order to test Shaper2D, three studies were undertaken. The broad purpose of the studies was to consider the role of a process-specific computer program for facilitating the Constructionist learning and understanding of shape grammars. All three studies were located at the MIT Department of Architecture.

For each study, exercises were devised to assess and develop the student's comprehension of shape grammar theory. The exercises situated the exploration in a specific activity.

The first study, study A, took place during the spring 2001 semester, as part of the MIT/University of Miyagi Remote Collaborative Workshop. This was the first time that *Shaper2D* had been used in a practical (as opposed to demonstrative) studio situation. The aim of the study was to determine whether *Shaper2D* could be utilized for its intended purpose as a tool for learning.

Study B, the follow-up study, was set up in response to the findings from study A. Its objective was to test whether hand computation is a necessary prerequisite to using computers for learning shape grammars.

The third study, study C, was in response to studies A and B. Instead of dwelling on the question of doing shape grammars by hand or computer, this study sought to test Shaper2D as a design rather than pedagogical tool. It took place during two sessions of the spring 2001 Design Inquiry seminar.

This thesis is a qualitative research study, rather than a precise, scientific, quantitative investigation. I thus followed the Grounded Theory approach (Glaser & Strauss, 1967; Glaser, 1999) in order to develop and explore my research interests and theories. I decided that the rigorous method of statistical analysis, a method usually associated with quantitative research, was not appropriate for this investigation. One of the fundamental principals of the grounded theory approach is that the researcher does not need to know, and should not force, the precise focus of the investigation beforehand. Instead, she should be open to emergent hypotheses. Given the scope of my research, this seemed to be the most natural methodology to use.

All participants in the studies were graduate or undergraduate architecture students. The students were required to synthesize and assimilate the new shape grammar concepts presented to them with previous knowledge acquired during their architectural education and training. With this in mind, an attempt was made to situate the exercises in authentic design activities, as far as was feasible and relevant to the particular study. The studies were all located in typical architecture studios augmented by computers and other technological equipment.

It is not expected that *Shaper2D* will be relegated to a subordinate role in shape grammar learning. Rather, it is hoped that this program will help to foster enthusiasm amongst designers for exploring the possibilities available with computational design, by capitalizing on the advantages of computer power over hand computation—especially when it comes to more complex design generation. It was with this in mind that Shaper2D was used in the three studies that will be described below.

In order to provide a foundation for this research investigation, the research strategies will first be presented. This introduction includes a brief overview of grounded theory, a summary of the data collection methods used, and a discussion of the recognized limitations of the studies with proposed remedies for further studies.

Data Analysis

Grounded Theory

Grounded theory is an inductive research methodology that was first presented in the late 1960s by Dr. Barney Glaser and Dr. Anslem Strauss. By using this methodology it is possible to develop theories that are empirically grounded in the data being collected (Glaser & Strauss, 1967). In other words, it is a "systematic generation of theory from data[1]". Each step of the approach is systematic—data collection, data analysis, etc.—so it is possible to see exactly the process involved in developing a theory.

Glaser and Strauss (1967) acknowledge that an inductive study includes much deduction, and point out that theoretical sensitivity is inherently deductive. However, this deduction is based on carefully inducted, or generated, ideas from which the researcher deduces what she might discover next and, consequently, she conducts further studies in these areas. This empowers the researcher to delimit her investigation and develop a better research focus. The iterative process of conceptual deduction therefore enables increasingly better induction.

Data Collection

The process of data collection differed for each of the three studies according to the aims of the study, the context, and the number and location of the participants.

Study A, the remote collaborative workshop, included a non-anonymous, post-assignment software evaluation questionnaire (an anonymous summary of which is included in Appendix B). Informal conversations were also held with both MIT and Miyagi students (in person and via *NetMeeting* respectively) after the workshop, in order to gain a more in-depth insight into the affordances and limitations of the Shaper2D software and collaborative learning experience.

Study B, the follow-up study, was more formal and required MIT approval which was duly granted (see Appendix C for sample forms). The session was tape recorded for later analysis. After the exercises, there was a general discussion about the session and the strengths and limitations of hand and computer computation.

Study C, part of the spring 2001 MIT Design Inquiry seminar, was more informal. Data collection took place in the form of the post-assignment presentation and round table discussion.

Limitations of the Studies

To yield more accurate results for the three studies, it would have been preferable to conduct each with an additional control group that approached the identical problem from a contrasting viewpoint. To clarify, for the first two experiments, one half of the subjects would have been exposed to hand computation prior to using the computer and one half would not, and for the third study the computer would not have been used at all by half of the class. However, this was not possible due to the lack of fundamental resources—time constraints and a limited number of participants.

It would also have been useful to interview the participants prior to commencement of the study, in order to gauge a notion of the participants' intentions and expectations.

The first and third studies were part of regular MIT courses, which may have also influenced the outcome of the exercises, as the students were all aware that their work might be tied to the outcome of their final grade for the class. The students' motivation might have been different had these studies been undertaken outside the class, as the second study was.

Study A: The Remote Collaborative Workshop

A.1 Motivation for the Study

The purpose of using Shaper2D in the workshop was to establish the validity of this software as a tool for facilitating the learning of shape grammars.

A.2 Details of the Study

Description

A description of the workshop from the course website:

> "An intensive 9 day ... remote collaborative workshop involving MIT and Miyagi University in Japan. The objective is to develop a small housing project using shape computation as a design methodology. Students will use and test new interactive software for designing, sharing applications with overseas partners, presenting projects on an Internet workspace, and critiquing design proposals through the web and other advanced digital technologies."
>
> (from the course website, http://www.mit.edu/~4.184/)

The workshop instructors at MIT were Gabriela Celani, Design Technology PhD candidate, Terry Knight, Associate Professor of Design and Computation, and William Mitchell, Dean, School of Architecture and Planning. The author was the teaching assistant. Professor Riusuke Naka

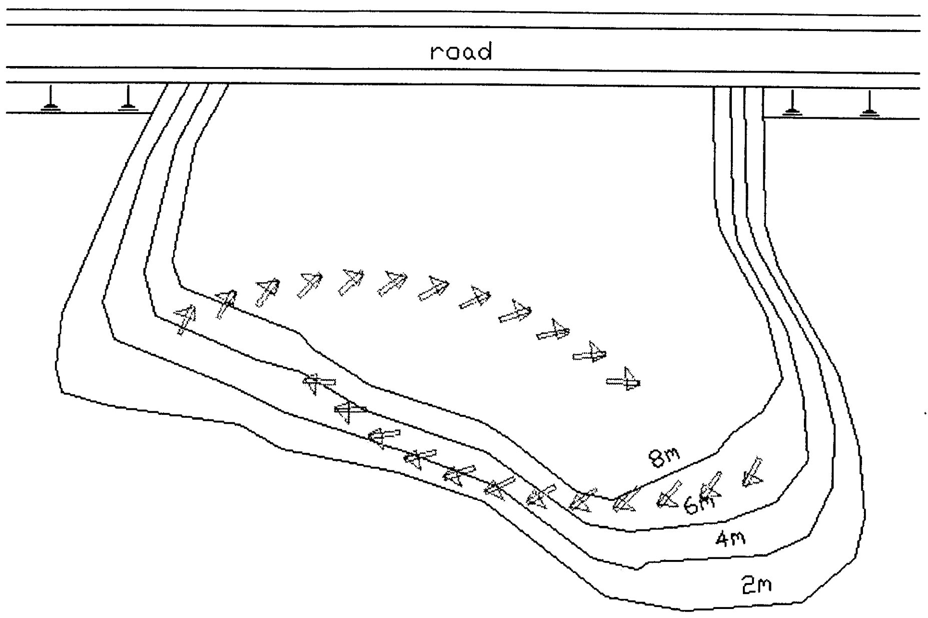


Figure 24. Study A: Miguel Chacon | Site Plan generated using *Shaper2D*, from Remote Collaborative Workshop.

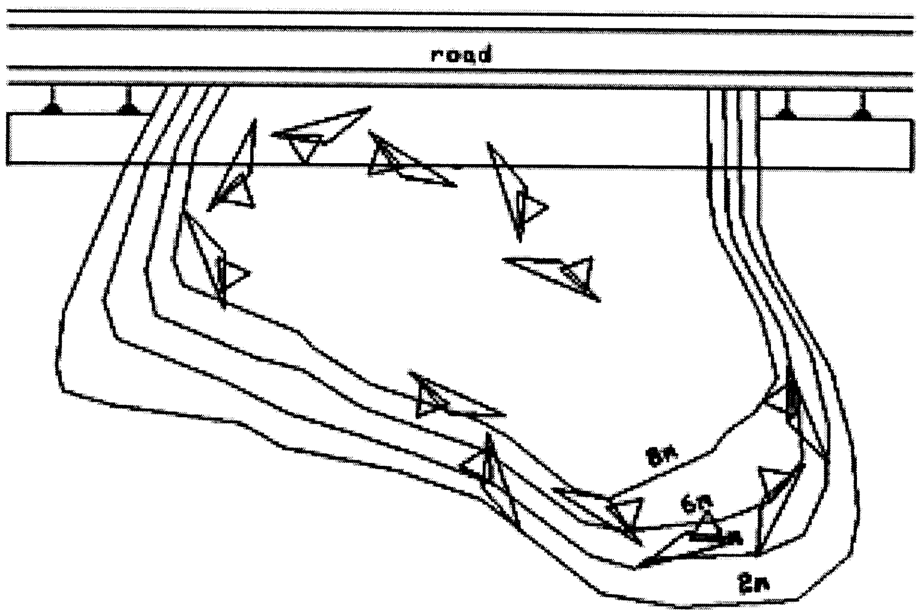


Figure 25. Study A: Audrey Snyder | Site Plan generated using *Shaper2D*, from Remote Collaborative Workshop.

coordinated the class at Miyagi, with the assistance of Professor Roberto Okada. There were also two observers—Federico Casalegno, visiting social scientist to MIT, and Hiroto Kobayashi, a visiting scholar who was able to assist with translations.

A specific aim of the workshop was to design the layout and massing for family townhouses, using shape grammars as the generative design methodology. Two sites were given—Long Island, Boston Bay, Massachusetts, and Izumi Park Town, Sendai City, Miyagi Prefecture, Japan. (see Celani, forthcoming, for a comprehensive description of the workshop).

It was proposed that shape grammars would be used for two purposes:

1. Using *Shaper2D* to generate the site layout for the houses (two-dimensional shape grammars)
2. Designing the houses using *3D Shaper* and *AutoGrammar* (three-dimensional grammars)

The brief for both sites required that the houses contain at least: living room, three or four bedrooms, one or more bathrooms, kitchen, laundry area, external parking space for 1 or 2 cars. The only difference being the two briefs was the gross floor area (GFA). For the houses located in Long Island, the GFA was required to be between 180 and 220m^2, whereas the GFA for the houses in Sendai was to be between 100 and 150m^2. Whilst this is a minor difference in brief requirement, inevitably the approach to each townhouse arrangement and development would be different, due to the site area and the differing geographical—and therefore social—contexts. For the first assignment using Shaper2D, which will be described shortly, the details of the brief were probably somewhat superfluous. However, the nature of the workshop was to situate the practice of shape grammars in an authentic activity, so it was important for the student to bear these details in mind from the outset, just as they would for a real project, as they provided context for the designs.

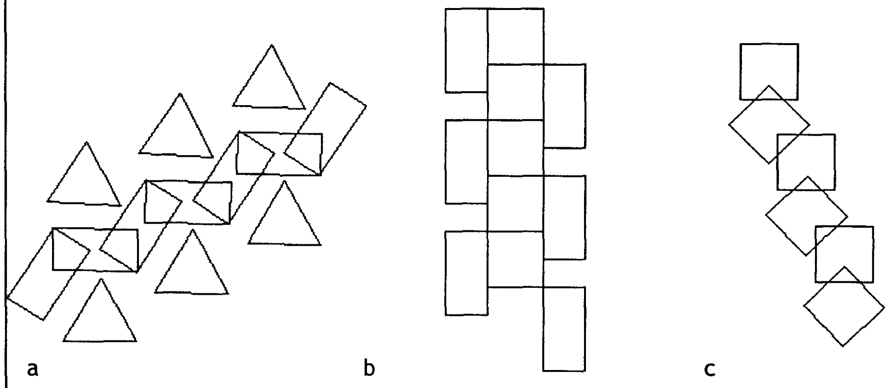
The MIT students were designing for a site in Japan, and the Miyagi students were designing for a site in the US, so in order to satisfy the cultural requirements of the brief the teams were expected to collaborate by using each other as "on-line consultants". This was in order to gain an insight into the characteristics of Japanese and American homes and families, respectively. As the brief merely set out the quantitative aspects of the project, discussion was needed in order to articulate the necessities for family housing—such as the hierarchy of rooms, size of laundry room, location of bedrooms and size and number of bathrooms. Due to the brevity of the project, designs were not expected to comply with building code regulations, however, it was expected that the designs produced would be as realistic as possible.

The Participants

Eight teams took part in the workshop—four from MIT (A, B, C and D) and four from Miyagi (A, B, C and D). MIT teams A and B each consisted of two undergraduate students—one sophomore and one senior, while teams C and D each comprised two graduates—an MArch student and a SMArchS

Exercise 1

Without resizing any shapes (rotating and translating is allowed), and by using one rule only, create the following designs:



Exercise 2

Without resizing any shapes (rotating and translating is allowed), and by using two rules, create the following designs:

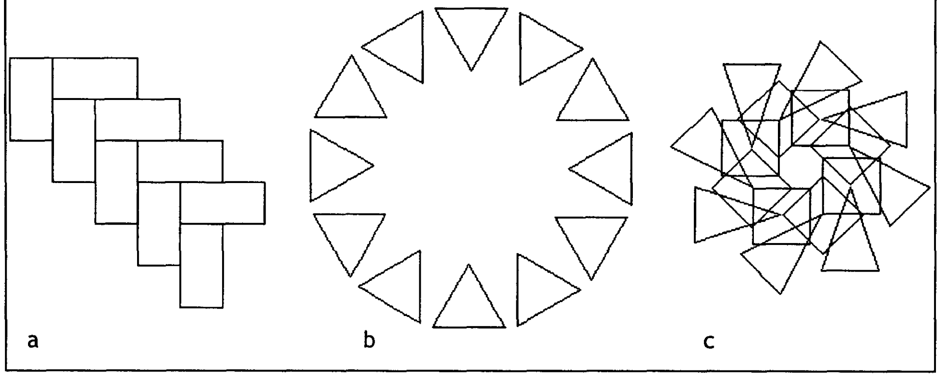


Figure 26. Study A, in-class exercises.

student. A PhD applicant also took part as a special student. Each Miyagi team comprised four undergraduate students—one sophomore and two seniors, plus a translator.

Approximately half of the students from each university had either heard about shape grammars or had had some experience with them. One MIT student had some previous experience with shape grammars, and six Miyagi students claimed to have some experience, with one Miyagi student claiming to have had considerable experience. Some of the Miyagi students with previous experience of shape grammars had taken part in the previous year's remote collaborative workshop. All previous experience was noted and accounted for in the evaluation of the workshop. A better test of the software would have required that all students were new to the methodology.

Among the MIT students, only one was fluent in spoken and written Japanese, whereas the Miyagi students were of mixed ability, with most students classifying themselves as poor English speakers. The Japanese students were divided almost equally between those with good and poor written English.

All the students were experienced with computers, and were generally evenly split between some and considerable experience.

Shaper2D Tutorial, Exercises, and Assignment

The *Shaper2D* session followed the introductory presentation of shape grammars given by Professor Terry Knight. This first session provided an opportunity for the students to perform several exercises using hand computation to establish an elementary understanding of basic shape grammars. Tracing paper and pencils comprised an analog microworld, as the exercises required tracing over spatial relations in order to physically apply the rules. The second workshop session introduced *Shaper2D,* the first of three computer implementations of shape grammars described above, and built on the knowledge developed from the hand computation.

After a short tutorial given to the local and remote teams using *PictureTel,* the students were asked to perform some simple exercises using *Shaper2D*. The goal of these exercises was for the students to "reverse-engineer" six designs (see figure 26) created with *Shaper2D* to figure out the spatial relations and rules that were used to generate each design. The first three designs were produced using one rule, and the next three designs used two rules for design generation. These exercises involved:

- selecting either One Rule or Two Rules
- choosing the appropriate shape(s)
- translating the shape(s) to get the correct spatial relation
- changing the label position(s) to get the correct transformation
- setting the correct number of iterations in the Design panel

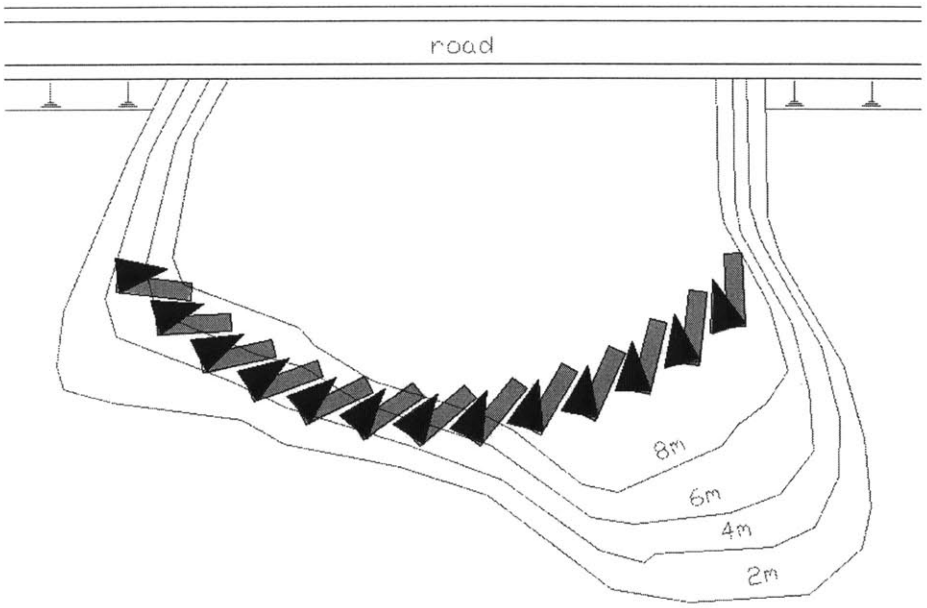


Figure 27. Study A: Myoungkeun Kim | Site Plan generated using *Shaper2D*, from Remote Collaborative Workshop.

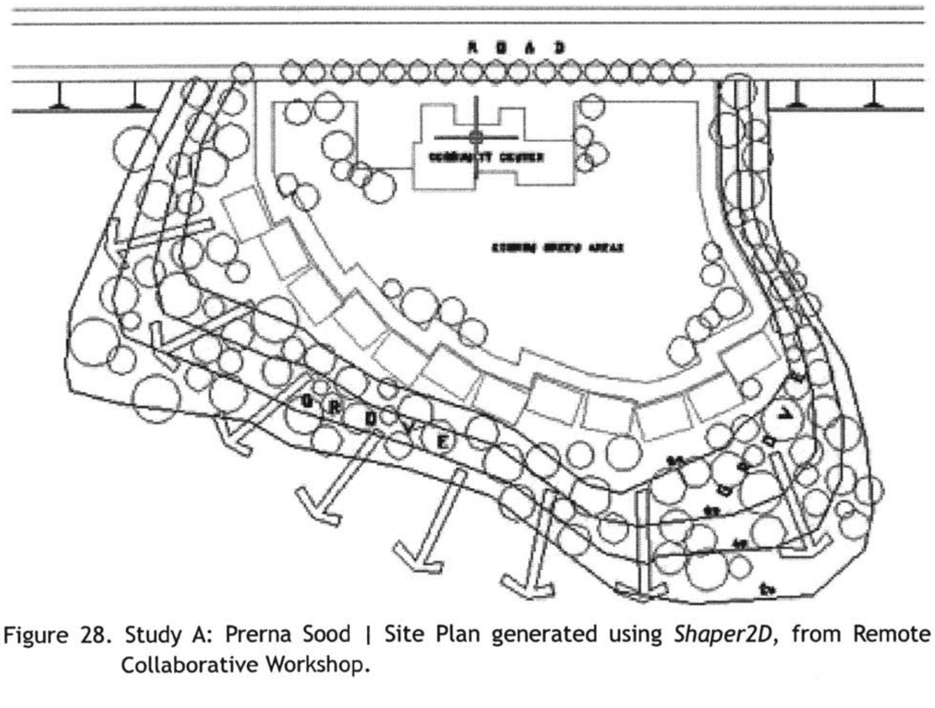


Figure 28. Study A: Prerna Sood | Site Plan generated using *Shaper2D*, from Remote Collaborative Workshop.

The homework assignment was given in the *Shaper2D* session after the in-class exercises. The principal completion criteria for the assignment was well defined in that the requirement was to develop a several conceptual housing arrangements for the given site (see figures 22-25 and 27-30 for examples of student work). However, the students were implicitly asked to tackle a subordinate task. This task was a requirement of the overall workshop design problem—acceptable design solutions required students to discover information about culture and family from their remote teammates (or consultants). This had the positive side effect of empowering the students to take control over the design process.

Software Used

During the course of the workshop the students were introduced to three new programs, or microworlds, for exploring shape grammars: *Shaper2D, 3D Shaper*, and *AutoGrammar*. Each of the three programs was used for a particular design exercise: *Shaper2D* for the site layout, *3D Shaper* for the massing of the townhouses, and *AutoGrammar* for the detailed design of the townhouses. Each microworld built on the knowledge constructed by the previous one in order to present a more advanced investigative environment. In the case of *Shaper2D*, this microworld built on the knowledge constructed from the hand computation exercises.

Each assignment, including the main workshop design problem, required the use of computer aided design software (CAD). For this purpose, *AutoCAD* and *3D Studio Max* were used. As *Shaper2D* allows the user to export designs as DXFs, other CAD programs could have been used, such as *Microstation* (favored by the Miyagi students), however it was decided for reasons of continuity and consistency to focus on two programs only. To export three-dimensional models from the open inventor viewer, required by *3D Shaper* (see Wang, 1998, for more details), *3D Studio Max* was required due to the restricted file types. Gabriela Celani developed *AutoGrammar* as a plug-in for *AutoCAD*.

The modalities of communication included both asynchronous and synchronous interaction, with certain applications (specifically the MIT-developed online collaboration portal *ArchNet*) providing dual functionality. For the tutorials and in-class exercises, *Microsoft NetMeeting* and *PictureTel* were used, whereas *ArchNet* was designated for file sharing and online presentations/ pinups. *PictureTel* was primarily used for lecture and tutorial transmissions, whereas NetMeeting was for application sharing, text and voice chat. The application sharing proved somewhat successful for the *Shaper2D* exercises. However, the students eventually had to resort to textual typed conversations instead of using the microphone and headsets. Poor sound quality rendered voice chatting almost impossible. The students used e-mail for communication and file sharing between workshop sessions, although *ArchNet* was also used to a lesser extent for the latter (for a more detailed overview of the software, see Celani, 2001).

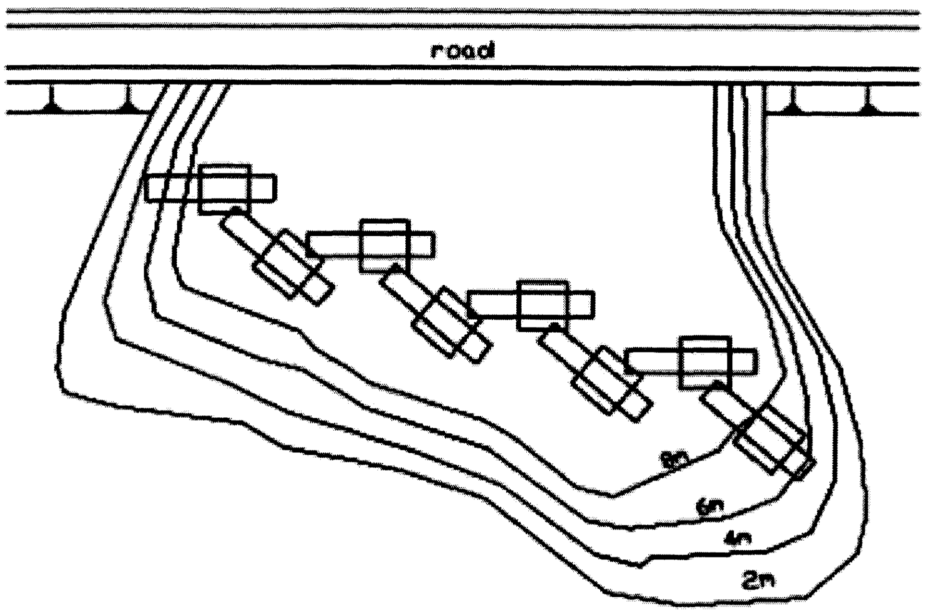


Figure 29. Study A: Konstantinos Tsakonas & Pamela Campbell | Site Plan generated using *Shaper2D*, from Remote Collaborative Workshop.

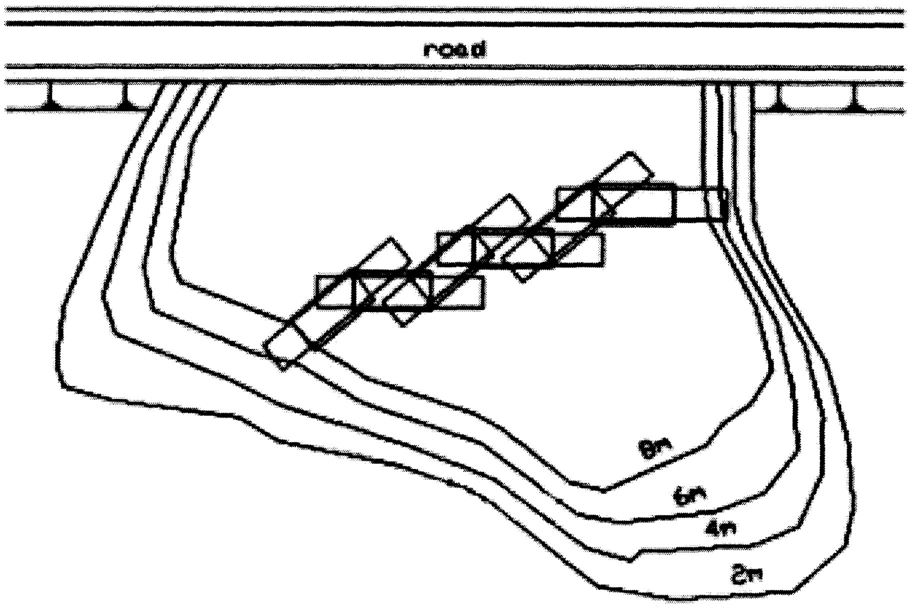


Figure 30. Study A: Konstantinos Tsakonas & Pamela Campbell | Site Plan generated using *Shaper2D*, from Remote Collaborative Workshop.

A.3 Evaluation of the Study

The MIT and Miyagi students both found Terry Knight's introductory lecture understandable, and, with the exception of three Miyagi students, found the hand computations easy. The majority of the students professed to have a comprehensive idea about shape grammar concepts after the lecture.

After the *Shaper2D* tutorial and in-class exercises, an interesting and important aspect of shape grammar computation arose during the analysis of the solutions. When discussing the rules, one of the MIT teams commented that they had come up with alternative rules that generated an identical design to the given design generated using my rules. This was a valuable contribution to the class, as it was able to confer a sense of belonging to the students within the shape grammar culture. It helped them to identify themselves as legitimate, albeit novice, shape grammarians.

The students were expected to produce several designs for housing layouts using *Shaper2D* as the principal design resource. This expectation was fulfilled, although the Miyagi students only presented one design rather than the two alternatives presented by the MIT teams. It was decided that this was probably due to a miscommunication about the assignment requirements. The designs presented by the students for the *Shaper2D* assignment demonstrated a varying degree of aptitude for designing with shape grammars.

It was interesting to see how *Shaper2D* was used in the student's final designs, as many of them had commented that the speed and feedback provided by the program facilitated quick explorations of ideas. Half the teams explicitly stated that *Shaper2D* contributed to their final designs. Students commented that the program was useful for the site plan, although the lack of site context in the design panel made it difficult to immediately associate the designs generated with the site. Scale was also an issue. As one student stated, "Being able to insert the site plan, it's very important when you're using it in architectural layering... you'd never start sketching without having a map or underlay as a base plan. And from a scale point of view as well... they are programs where you can get carried away without thinking too much about it, and to keep it real, it needs to be slightly more like a traditional medium."

As far as increased motivation is concerned, positive comments were made by many students about the advantages of using *Shaper2D*, for example, "It made it a lot quicker-it can get monotonous when you're working with boxes and things... ". Another student mentioned that, "...building upon doing it by hand, it's an amazing tool to be able to understand things quicker than you would by hand in terms of getting faster responses and being able to see much quicker, where the label changes would affect your design." A further comment was, "I think the computer is good for fine-tuning-it's very fast. By hand it's so slow."

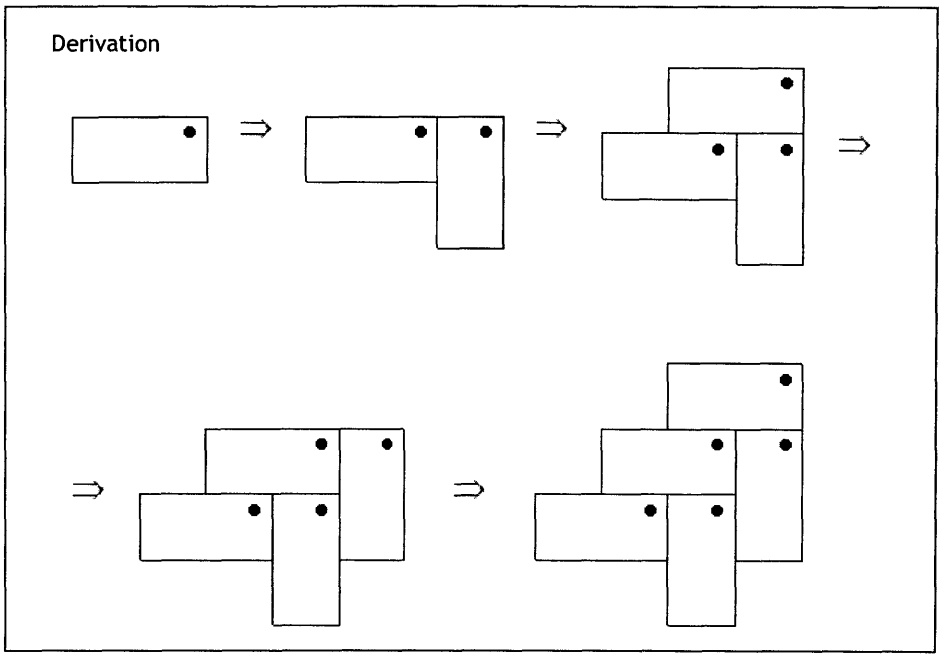
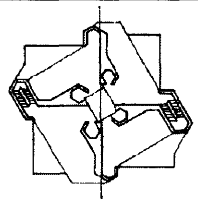
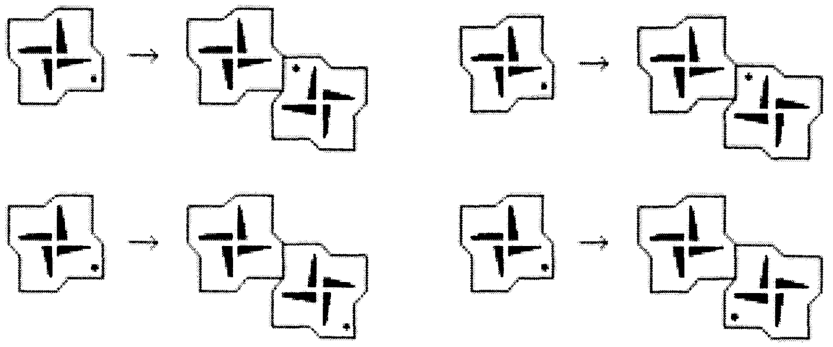


Figure 31. Study B: Example solution from exercise 1.



Left is Frank Lloyd Wright's plan for St.Mark's-in-the-Bouwerie apartment building (1929).

The St. Mark's plan became the basic compositional element in subsequent projects by Wright: an apartment house in Chicago (1930), the Crystal Heights hotel and apartments (1939), and the Price Tower (1952-56).



For each rule above, generate a design by applying the rule recursively to an initial shape which is the shape on the left-side (a total of four new designs).

Figure 32. Study B: Sample problem from exercise 2.

On using *Shaper2D* and doing hand computations, one student said, "I think that working on trace was pretty important, it laid a good foundation, but after that *Shaper2D* was really good at helping us just do stuff, showing us how shape grammars actually worked."

A.4 Conclusions Drawn from the Study

The workshop highlighted a key aspect of *Shaper2D* that had been overlooked—that it can be used as a design tool, rather than purely educational software. This was signified by suggestions to include additional shapes and rules, and a facility to import site plans into the design window.

Overall, the students claimed that it was important to learn shape grammars by using hand computation in order to understand the concepts fully. One student commented, "I think you really need to understand the concepts using the tracing paper before using Shaper2D, otherwise you won't understand what's happening." Another student explained in more detail, "The whole idea of doing it first by hand is very important—you're thinking when you're doing it. When you use the computer you can take it further and the computer does all the thinking, and you just play around—that's the thing that actually processes it."

However, it was unclear whether the students genuinely benefited from learning about shape grammars by hand prior to using *Shaper2D* as no control group was used. This provided the impetus to instigate a follow-up experiment.

Study B: The Follow-up Study

B.1 Motivation for the Study

The results of the remote collaborative workshop were inconclusive as far as hand versus computer computation was concerned, so it was decided to conduct a further study of *Shaper2D*. Previous introductory shape grammar classes, including Terry Knight's lectures during the Remote Collaborative Workshop, used hand computation as an introduction to the fundamental concepts of shape rule application. Instead of using the computer for extended in-depth investigations, this study reversed the conventional teaching order so that the computer was used for learning the concepts prior to hand computation.

The purpose of this follow-up study was, therefore, to test the necessity of learning shape grammar concepts through hand computation before moving on to using computers for further exploration of the concepts. This study was a single session, and a design component was not included, so it was limited to being conducted in a theoretical exploration. It also served as another opportunity to collect feedback about the learning and interface aspects of *Shaper2D*.

B.2 Details of the Study

Description

This study took place during a single three-hour evening session, and was held in one of the MIT Department of Architecture studios.

> "This study comprises a 'crash course' in computational design and the use of a computer program, "Shaper2D", designed as a pedagogical/practical application for teaching and experimenting with two-dimensional shape grammars. The purpose of the study is to ascertain whether shape grammars can be taught and understood without any previous experience with hand computation."
>
> (from the experiment protocol, see Appendix E)

The study was a condensed version of the first two sessions of the remote collaborative workshop. It began with a short presentation by the author. The presentation was based on Terry Knight's "Introduction to Shape Grammars" lecture, and introduced the history, concepts and uses of shape grammars. Examples of analytic shape grammars developed for various designs, both architectural (including Palladian villa grammar and Queen Anne houses grammar), and non-architectural (paintings by Kandinsky) were presented (for further information about these grammars, see Stiny, 1976; Flemming, 1987; Knight 1989). In addition, examples were shown of original architectural designs that were generated using shape grammar methodology.

After the presentation, the subjects were asked to complete two short shape grammar exercises: the first using *Shaper2D*, and the second using hand computation (see figures 31 and 32). In the previous study, the problems included in the first exercise were completed using hand computation as a precursor to using the computer. In order to assess their understanding of the spatial transformations and rule applications involved, the subjects had to explain verbally their approach to the exercises.

To make a further assessment of the subjects' comprehension of shape grammar concepts, the second exercise required the subjects to complete several problems using hand computation[2]. This exercise was designed to see how well the subjects could tackle shape grammar derivations without the aid of a computer program. Again, in order to gauge the subjects' understanding of the shape grammars, they were asked to talk through how they set about completing the problems.

Finally, a short discussion took place at the end of the session to evaluate and reflect on the exercises, hand computation, and *Shaper2D*.

The Participants

The experiment involved two students in the SMArchS program in the MIT Department of Architecture. To preserve anonymity, they will be referred to as subjects A and B. There were to have been three participants, but due to unforeseen circumstances the third student was unable to take part in the study, and because of time constraints it was impossible to find a replacement. It was decided to continue the experiment, but the limitation of only having two subjects meant that the results were not as conclusive as had been hoped. Participation in the study was on an entirely voluntary basis, and was mandated by the MIT Committee on the Use of Humans as Experimental Subjects (see Appendix C for samples of the forms).

Selection criteria for the participants included design, preferably architectural, training and no previous exposure to shape grammars in a studio environment. Both students were somewhat aware of shape grammar research through projects they had seen, but neither knew anything about the concepts involved.

It should be noted the results of this experiment may have been affected by the fact that the subjects were colleagues of the author.

B.3 Evaluation of the Study

The subjects were able to complete the first exercise without too much difficulty, although the descriptions given as to how they derived the designs varied in terms of precision and accuracy. Subject A demonstrated a good understanding of the spatial transformations involved in the rule applications and was able to elucidate the procedures to Subject B who took longer to perceive what was happening.

> "It seems there are two rules, one after the other... it's [reflection] and a rotation... It rotates and then has to be on the other one. So this piece rotates... the whole piece rotates and [reflects] so is that, and then this one goes here. And then this one has to rotate *shtuck!* again."
>
> (Subject A)

The second exercise reversed the comprehension of the concepts established during the previous exercise. This time, subject B quickly completed all the set problems, whilst subject A struggled with the rule applications and found it difficult to understand the spatial transformations being performed. Subject B explained the solutions to subject A, who eventually claimed to have an understanding of the processes involved.

Subject B felt that developing a greater familiarity with *Shaper2D* was beneficial to being able to explore its potential,

> "I think that there was definitely a learning curve involved with understanding how to use the tool. Because once you understood how to use the tool, I think you can manipulate it to your advantage, you could then do... certain things that you probably weren't able to do if you didn't understand the tool that well."
>
> (Subject B, during the end-of-session interview)

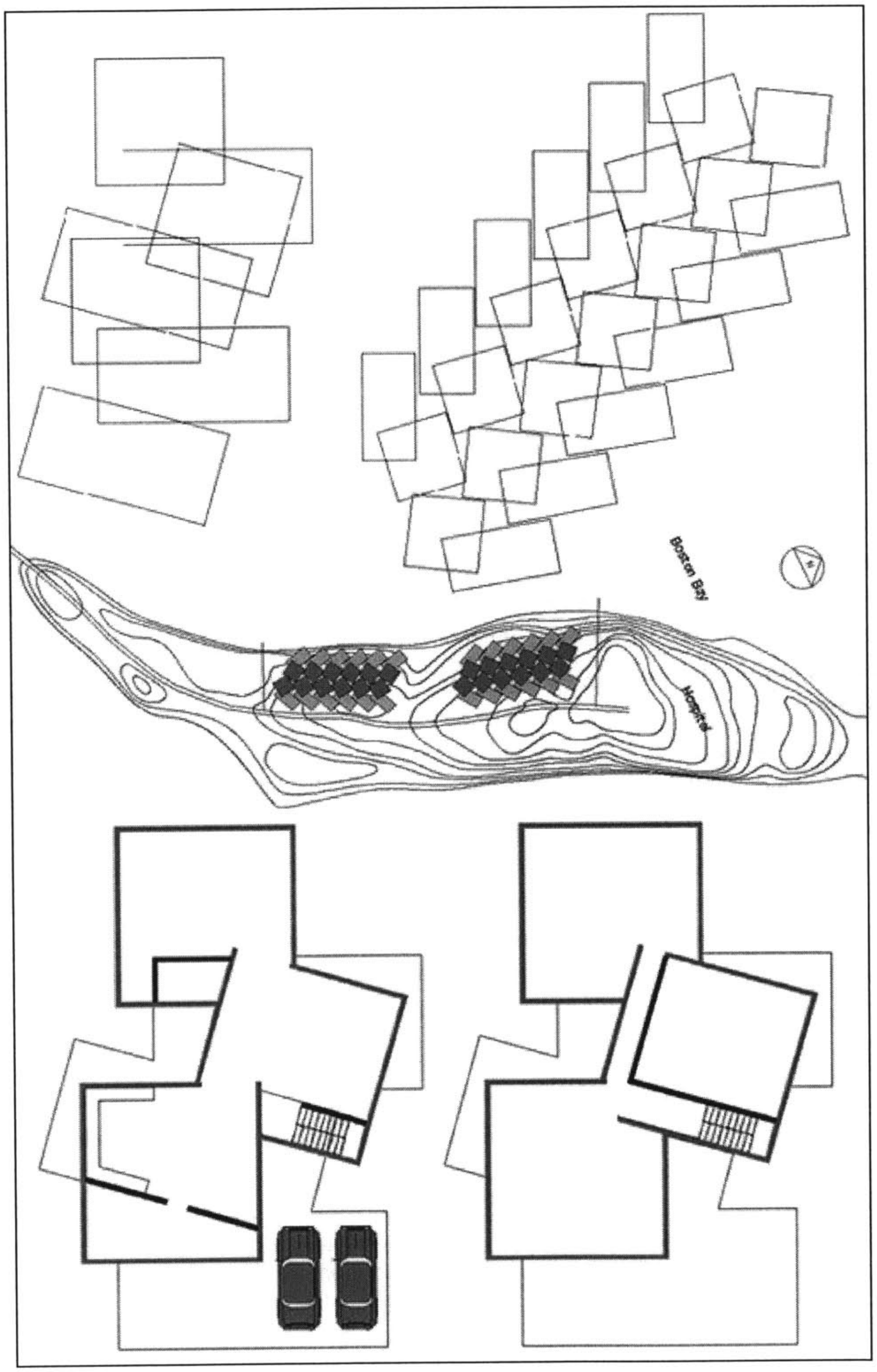


Figure 33. Study C: Gustavo Rodriguez.

B.4 Conclusions Drawn from the Study

Due to the limited number of participants, conclusive evidence could not be drawn as to whether hand computation is necessary as a precursor to using the computer, in order to fully understand shape grammar concepts.

> Subject A was able to explain thought processes clearly when using *Shaper2D*, but experienced difficulties when doing hand computation.
>
> "I do understand the movement. But... because it's just trace I got, like, lost... I completely forgot about the computational aspect."
>
> (Subject A, during the end-of-session interview)

The reverse situation happened in the case of subject B.

> "The hand computation made it very easy to see what was happening... and [Shaper2D] is probably a pretty elegant...interface for what it needs to do. Because the paper is tactile... because you're able to flip it (the trace) it's very obvious what's happening."
>
> "...maybe the trace would have been as tough, the other way around, and then the computer program would have been really easy to understand. But since we had done the computer program and understood, okay we can manipulate the points on the geometric shape and then see what happens there... it became very straightforward when we got to the paper."
>
> (Subject B, during the end-of-session interview)

Both subjects expressed a belief in the benefits of using hand and computer computation when using shape grammars,

> "I would think a combination of both is much more interesting, in a way. With [Shaper2D] you can see all the different alternatives possible. If you print [a design or a spatial relation], you can take a trace and play with that and continue. Play with both [Shaper2D and tracing paper]. A hybrid is somehow more important."
>
> (Subject A, during the end-of-session interview)

Although this study cannot state categorically whether hand computation is necessary, it has certainly indicated a need for further research studies.

The study was not considering Shaper2D in a design context, however, comments that the subjects made about constraints of rule application and using the design as a base for concept development suggested the need to explore the use of Shaper2D as a design tool.

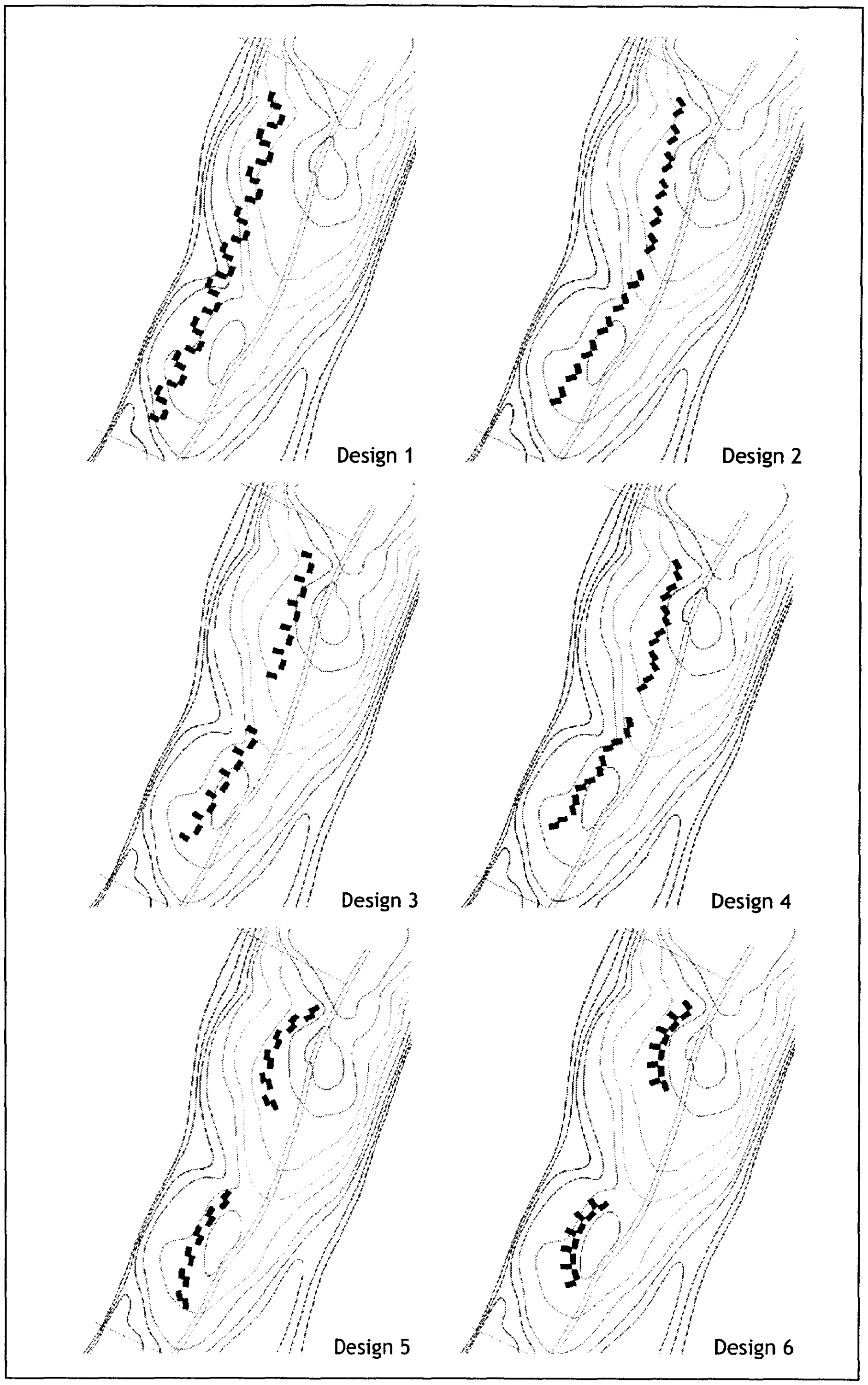


Figure 34. Study C: Ian Sheard.

Study C: The Design Inquiry Seminar

C.1 Motivation for the Study

The third occasion to test *Shaper2D* was during an informal, two-session experiment, which was part of a regular MIT course during the spring 2001 semester. The course, "Design Inquiry", taught by Professor William Porter, is a required subject for all students in the SMArchS Design Technology program. The broad purpose of this study was to investigate how a pre-conceived design intention could be implemented, if at all, for a site development problem using basic shape grammar concepts. *Shaper2D* was used as the principal design resource. My specific research interest lay in how *Shaper2D* can be utilized as a design tool. This interest was motivated by comments made by students in the remote collaborative workshop, that *Shaper2D* was a practical rather than learning tool. The benefits of doing shape grammars by hand or by computer were not considered.

C.2 Details of the Study

Description

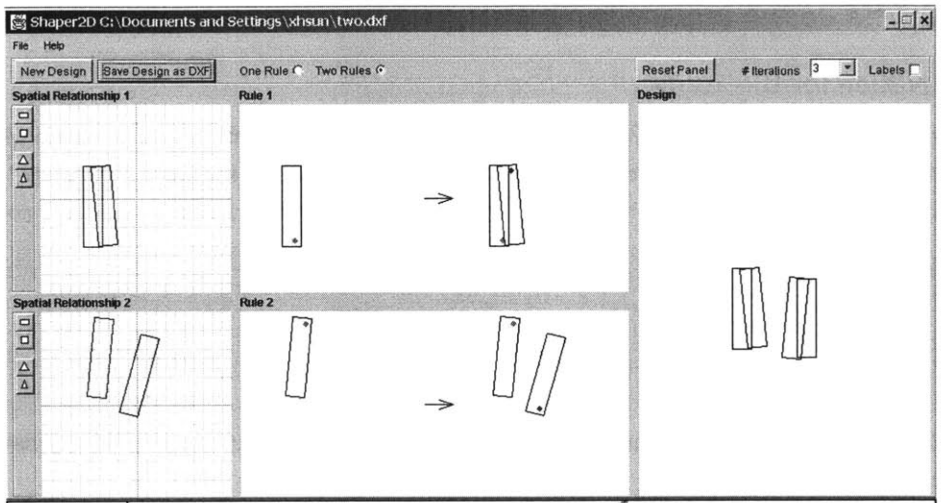
The study took place over two sessions. In the first session, the students were provided with a short introduction to shape grammars, *Shaper2D*, and a design brief. After the introductory presentation, the students attempted several short computations by hand. To conclude this part of the session, a brief account was about previous projects that were developed using shape grammars.

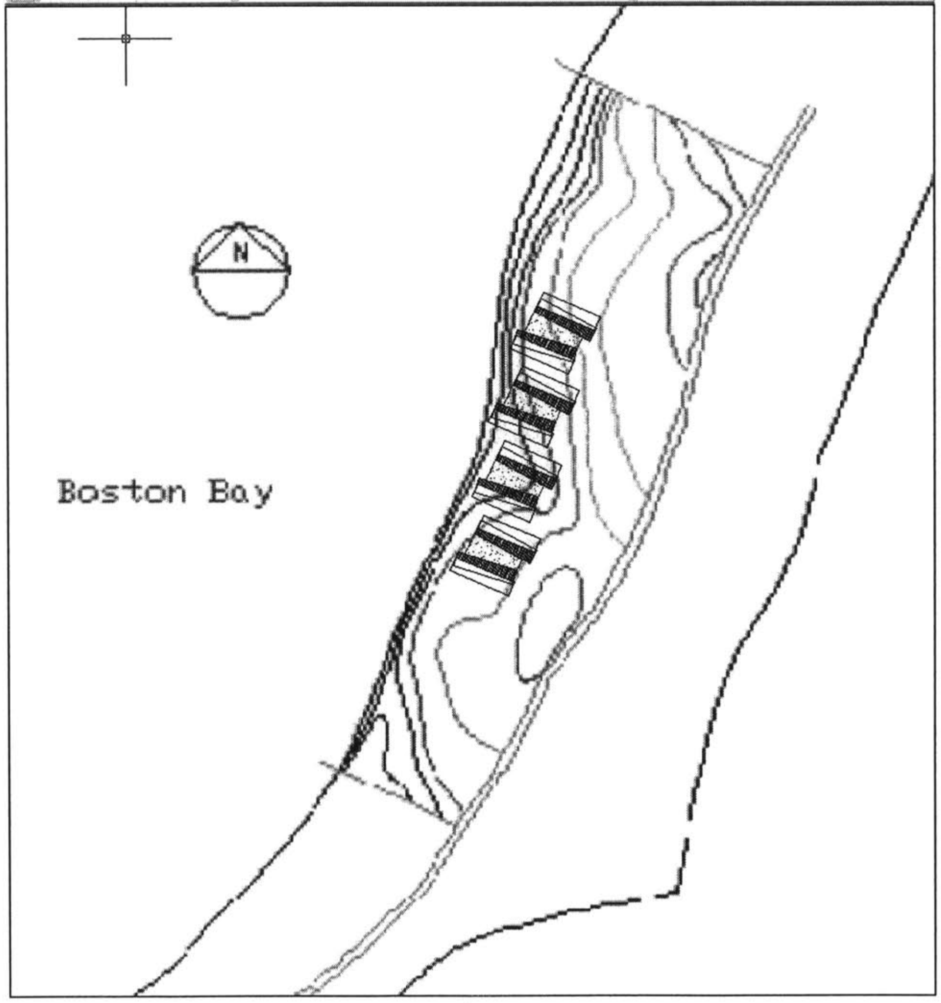
The *Shaper2D* applet and application was then demonstrated using the same *Shaper2D* tutorial given during the Remote Collaborative Workshop.

A homework assignment was then given. The design brief used was the same as that for the Remote Collaborative Workshop, except this time the MIT students were designing for the Boston site. The students were also not required to present detailed plans, only an overview of their design process that led them to the proposed site layout, and a summary of their experience using *Shaper2D*. In the second session, the students were required to present their work.

The Participants

The participants in the study were all graduate students, predominantly studying for the SMArchS degree in the field of Design Technology, at the MIT Department of Architecture. However, several students were from other MIT disciplines and departments, including Building Technology, Urban Studies and Planning, and the MIT Media Lab. The students completed the exercises and design assignment as homework.





Figures 35 (top), 36 (bottom). Study C: Xiaohua Sun.

At least one student had previous experience with shape grammars, and the majority of students were aware of shape grammar research from seeing previous projects.

C.3 Evaluation of the Study

Ten students in total completed the homework assignment and presented their designs to the class. Out of these students, seven used Shaper2D for part of the design process. Three students elected to use a different program for the assignment.

The students were asked to make a note of their intentions prior to commencing the design exercise, in order to compare their original thoughts with what was actually accomplished. Several students shared the frustration of having a clear notion about what they wanted to achieve, and then having to find a rule to describe this idea—in particular, they sought to generate designs that followed the site contours. One team of two students observed that shape grammars are very helpful when the designer doesn't have a fixed idea, but once the designer has an idea, the use of shape grammars can provoke frustration. This was a comment on the theory of shape grammars as a design process, as opposed to *Shaper2D* in particular. The majority of the students were happy with using *Shaper2D* as a way of speeding up the sketching and design generation process. However, two students talked about difficulties they experience when trying to devise rules that would generate a curve, but this was to do with their overall understanding of shape grammars, and was not due to a limitation imposed by *Shaper2D*.

Another frustration voiced by several students was the necessity of having to keep *Shaper2D* open at the same time as a third-party CAD program displaying the site plan, in order to associate a context with the designs. One student complained that, "...at the moment, it's like designing in a void." This echoed complaints made by students from the remote collaborative workshop in asserting the need for context when designing.

It was interesting to see the methods used by the students in order to come up with a suitable site plan. A timeline in images was presented by one student, to give a visual description of his design development (see figure 34), specifically with reference to his quest to find a curvilinear design. Another student, who approached the assignment with no fixed design intentions, exploited a practical application of shape grammars. By taking the opportunity to apply two rules, she assigned each rule a specific function—the first was used for the individual plan of the housing blocks, and the second for the actual site layout (see figures 35 and 36). A more adventurous student elected to consider the environmental (light and solar gain) conditions, using Shaper2D to explore a design solution. He also used Shaper2D to develop a both a site layout and a more detailed house plan (see figure 33). He commented that although the designer develops the rules, that the point of using shape grammars is to mimic the designer's design process. He felt too constrained when using the shape grammars, and expressed relief when he was able to explore the design using (created in Shaper2D) third-party CAD software.

Being able to keep "a memory of each design as you go along", as a student suggested, was a feature of Shaper2D that the students felt was lacking. If they derived an appealing design, reverse engineering was necessary in order to generate it again later.

The constraints of using shape grammars as a design method was noted, although an interesting observation was made that there exists a dichotomy of constraints: the constraints imposed by the designer on the tool, or method, and the constraints that the tool imposes on the designer.

C.4 Conclusions Drawn from the Study

In the post-study discussion, which took place during the following Design Inquiry session, one student commented that, "...designing is the process of deciding what's best out of an array of possibilities."

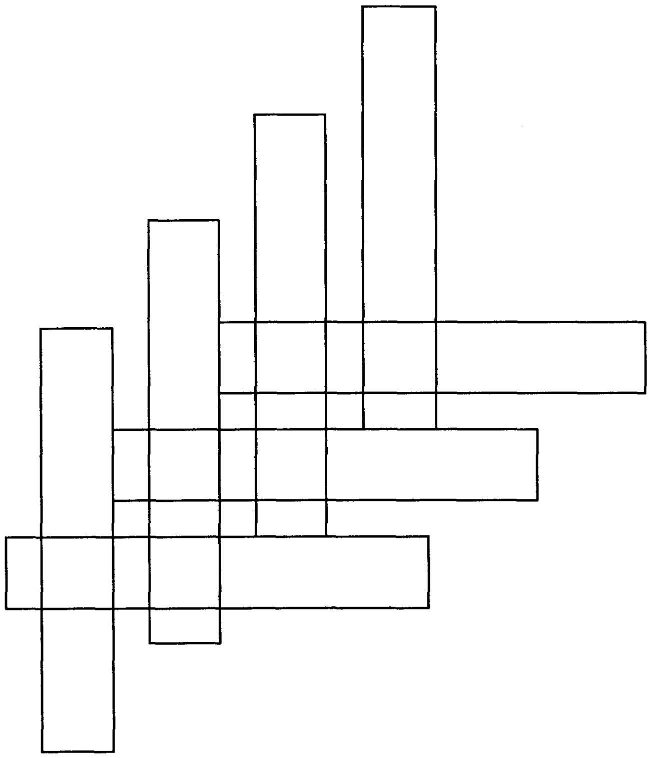
A possible conclusion from this discussion is that a designer develops a personal relationship with a design tool, leading to an individual belief in her experience with the tool. However, a designer's fluency with a preferred design tool or method is usually not vocalized. With reference made to shape grammars, and their place in the design process, it was commented that when a designer becomes fluent in using a particular tool she is able to better consider all the possibilities available to her. Different exploratory tools or microworlds are needed to derive a better understanding of the objects that are being created.

The designer will develop her own criteria for using different microworlds, which could include convenience and how that environment relates to the particular stage reached in the design process. *Shaper2D*, and shape grammars as a whole, provides one particular microworld, or technical domain, for design exploration. During every stage of the design process, certain constraints guide—these allow the designer to explore certain aspects of the design.

A question was raised which could motivate further research: how do you try to exploit each microworld for what it can contribute to the process?

Conclusions

Observations, Reflections and Future Work



Observations and Reflections

There is no clear distinction between *Shaper2D* as a learning or a design tool. As it could be argued that design exploration is a constructive mode of learning, the program satisfies both objectives. After much thought—and observation of students' experimentation with the program—I have concluded that the overlap between it as a learning tool and a practical, designing, tool is not a detrimental attribute. Rather, it presents an analogy with the architect's paradigm for designing described in chapter one.

From the workshop and follow-up study, it appears that the students generally view hand computation as playing a vital role in learning computational design. However, from my observations, I also discovered that the computer is also considered an imperative part of the process, as it allows students to explore grammars further than they would have been inclined to do by hand.

An interesting observation made from the workshop and the design inquiry seminar is that computational design can make explicit a designer's design process. Some designers approach a problem with a mind to exploring it, looking for unexpected solutions. Others like to have a greater level of constraint—in other words, they prefer to work within a more limited environment, such as a microworld. These two approaches can also go hand-in-hand—some designers preferring to have more control of the design while wanting to explore possible alternatives.

Future Research

Improvements to *Shaper2D*

When *Shaper2D* was first developed, I did not perceive it as a practical design tool. Yet the studies show that this is a potential function. Users have praised its speed and interactivity, while pointing to its drawbacks.

Several improvements and additions have been suggested that would improve *Shaper2D*. Additions that I have considered incorporating into a future version of *Shaper2D* include:

- The incorporation of the site plan into the design panel
- Being able to save the rules and reload them at a later date
- The inclusion of a zoom feature for the spatial relation and rule panels
- Increasing the number of shapes available

Other suggestions have been made, that on reflection I have decided would radically change the nature of the program. These include:

- Increasing the number of rules that can be applied
- Allowing the user to design new shapes
- Permit using designs as emergent shapes and plugging them in as new spatial relations

Although these would all be useful features, *Shaper2D* was purpose-written as a basic shape grammar interpreter. Implementation of the latter suggestions would result in a program more advanced than I had intended.

In addition to emphasizing the aspects of shape grammars that were described earlier, it should be noted that Shaper2D has proven to be robust in the teaching/experimentation situations. Throughout these extensive periods of testing, only one bug was discovered, which was quickly fixed. This highlights an important aspect of all software development—reliability. It can be argued that it is better to have several properly functioning, albeit limited, tools than one all encompassing, but poorly designed, "super tool". Consider the Swiss army knife, a portable tool that comprises many tools. It is a useful gadget to have handy when camping, for menial tasks like opening a bottle or a tin can, cutting string or sawing a small branch. However, if more involved work was required, it would be far more beneficial to have access to a "real" version of one of the tools.

Merging hand computation with computer computation

Currently in the field of computational design, working by hand is seen as being distinct from working on a computer. The approaches are different, in terms of how the user accesses the particular medium—be it a pen, a wooden block or a piece of computer software. We know that analog and digital systems have their strengths and limitations, yet they are persistently independent, largely because it has been inconceivable to think of them otherwise. However, after working with both computers and paper, and seeing how designers work with the various media in different design scenarios (learning and practical), I would like to propose a merging of the methodologies.

With current technology and research this proposal is not as far-fetched as it might seem at first. Many students have asked for a dynamic tool for exploring three-dimensional shape grammars. After considering their comments about the importance of doing hand computation before using a computer, and to understand physically the spatial transformations being applied, it would be interesting to merge physical and virtual.

I suggest two possible approaches to this problem. One would be to embed computer chips into wooden Froebel blocks[1]. The spatial relations between the blocks would then be manipulated on a digital surface that recognized the blocks[2], such as the "active table" currently being developed as part of the MIT House_*n* research[3]. Spatial transformations would be computed by a computer and with designs generated by the resultant rule applications being projected onto a screen. A drawback to this would be that it would still be impossible to explore designs generated by intersecting blocks. Gravity would also be a limitation.

A way of overcoming physical constraints and the laws of gravity would be to instead manipulate "virtual" Froebel blocks. This would be possible using an interactive three-dimensional virtual reality system such as the Responsive Workbench, currently being researched at Caltech[4]. A description of the interface follows,

> "The user of the workbench interacts with the virtual objects on the workbench just as they would with actual objects on an actual workbench... In order to create the 3D environment, users wear shutter glasses to view computer-generated stereoscopic images that are projected [onto] the tabletop display surface by a projector/mirror system. The user's head position and orientation are tracked to create the correct perspective for the computer to use when rendering the environment. An input device is also tracked by the system allowing the users to interact with objects in the tabletop environment."
>
> (from the Responsive Workbench web site, http://www.multires.caltech.edu/~rwbdemo/)

This has advantage that two people are able to use it at once. Users can see what others are doing and touching. Like the previous proposal, the generated designs could be displayed onto an adjacent surface or monitor. An application of this would be to introduce a more interactive "hands on" element to remote workshops, thus inducing a greater sense of collaboration, whereby a student at MIT and a student in Miyagi could simultaneously explore three dimensional shape grammars together.

It is only a question of time before a veritable synthesis of "by hand" and "by computer" computation is devised and implemented. If shape grammars are to be seen as a viable design methodology for designers and architects, it is crucial that the interface evolves.

End Notes

Chapter One

1. Hand computation, using tracing paper, played an important role in developing Shaper2D.

Chapter Two

1. However, due to the nature of the programming language—and with a slight modification to the code that does not affect the intended functionality—it can now also be accessed by any computer running the Microsoft Windows operating system with the Java™ Runtime Environment installed.

2. Mac OSX. Previous releases of the Apple Macintosh operating system do not support Java™ 2.

3. For older browsers, the Java™ plug-in can be downloaded from the Sun website at http://java.sun.com/products/plugin/

Chapter Three

1. Dr. Barney Glaser, from an interview with Dr. Andy Lowe, c.1999.

2. These problems were devised by Terry Knight and were included in the first Remote Collaboration Workshop in spring 2000.

Conclusions

1. Or other media. Froebel blocks are used as examples due to their place in current shape grammar pedagogy.

2. Refer to "Luminous Room", and other current research being conducted by the Tangible Media Group at the MIT Media Lab, http://tangible.media.mit.edu/

3. See http://architecture.mit.edu/house_n/

4. See http://www.multires.caltech.edu/~rwbdemo/

Shaper2D Code

```
//Title:        Shaper2D
//Version:      2.1
//Copyright:    Copyright (c) 2001
//Author:       Miri McGill
//Company:      MIT
//Description:  Interactive panel for
//              drawing shapes
package Shaper2D;

import java.awt.*;
import java.awt.event.*;
import java.awt.event.InputEvent;
import javax.swing.*;
import Jama.*;

public class DrawShapes extends JPanel
       implements MouseListener,
       MouseMotionListener, Constants
{
  private Rules childRules;
  private Transform trans;
  private Iteration childIter;
  public Shape shapeOrig, shapeDest;
  public ShapeBar shapeBar;

  Shape testR1, testR2, testT;
  Transform testTransform1;

  public Shape movedShape = null;
  public Shape rotatedShape = null;
  public int onVertex = Shape.NO_VERTEX;

  public int dragStartX, dragStartY, dx,
       dy, rotateStartX, rotateStartY;
  public double theta;

  private int grid = 10;
  private boolean gridOn = true;
  private boolean active = true;


/* ------------------------
   Constructors
 * ------------------------ */

  public DrawShapes()
  {
    //setCursor(new
       Cursor(Cursor.CROSSHAIR_CURSOR));
    try
    {
      jbInit();
    }
    catch(Exception e)
    {
      e.printStackTrace();
    }
  }

/* ------------------------
   Component initialization
 * ------------------------ */

  private void jbInit() throws Exception
  {
    shapeOrig = DEFAULT_RECTANGLE.copy();
    shapeDest = DEFAULT_RECTANGLE.copy();
    trans = new Transform(shapeOrig,
       shapeDest);

    addMouseMotionListener(this);
         addMouseListener(this);
  }

/* ------------------------
   Mouse event handlers
 * ------------------------ */

  public void mouseClicked(MouseEvent e)
  {
  }
  public void mouseEntered(MouseEvent e)
  {
  }
  public void mouseExited(MouseEvent e)
  {
  }

  public void mousePressed(MouseEvent e)
  {
    if((e.getModifiers() & e.BUTTON1_MASK)
       != 0
    && (e.getModifiers() & e.SHIFT_MASK)
       == 0)
    {
      // Move Vertex
      if(onVertex != Shape.NO_VERTEX)
      {
        shapeOrig.setSelected(true);
        shapeDest.setSelected(false);
      }
      // Move Shape
      else if(shapeOrig.hitTest(e.getX(),
       e.getY()))
      {
        movedShape = shapeOrig;
        dragStartX = e.getX();
        dragStartY = e.getY();
        shapeOrig.setSelected(true);
        shapeDest.setSelected(false);
        setCursor(CURSOR_MOVESHAPE);
      }
      else if(shapeDest.hitTest(e.getX(),
       e.getY()))
      {
        movedShape = shapeDest;
        dragStartX = e.getX();
        dragStartY = e.getY();
        shapeDest.setSelected(true);
        shapeOrig.setSelected(false);
        setCursor(CURSOR_MOVESHAPE);
      }
      repaint();
    }
    else if((e.getModifiers() &
       e.BUTTON1_MASK) == 0
           ||((e.getModifiers() &
       e.BUTTON1_MASK) != 0
```

```
            && (e.getModifiers() &
        e.SHIFT_MASK) != 0))
      {
        // Rotate Shape
        if(shapeOrig.hitTest(e.getX(),
        e.getY()))
        {
          rotatedShape = shapeOrig;
          rotateStartX = e.getX();
          rotateStartY = e.getY();
          shapeOrig.setSelected(true);
          shapeDest.setSelected(false);
          setCursor(CURSOR_ROTATESHAPE);
        }
        else if(shapeDest.hitTest(e.getX(),
        e.getY()))
        {
          rotatedShape = shapeDest;
          rotateStartX = e.getX();
          rotateStartY = e.getY();
          shapeDest.setSelected(true);
          shapeOrig.setSelected(false);
          setCursor(CURSOR_ROTATESHAPE);
        }
        repaint();
      }

/*
    // FOR DEBUGGING
    System.out.print("Mouse Pressed ");
    System.out.print(e.getX());
    System.out.print(",");
    System.out.print(e.getY());
    System.out.print("\n");
    System.out.println(e.getModifiers());
*/
  }

  public void mouseReleased(MouseEvent e)
  {
    if(!this.contains(e.getX(), e.getY()))
    {
      return;
    }
    else if(onVertex != Shape.NO_VERTEX)
    {
      shapeOrig.moveVertex(onVertex,
        e.getX(), e.getY());
      shapeOrig.finalizeVertex();
      shapeOrig.setSelected(false);
      shapeOrig.setHighlighted(true);
      childIter.changeShape(this,
        shapeOrig);
      updateDrawing();
      onVertex = Shape.NO_VERTEX;
    }
    // Move Shape
    else if(movedShape != null)
    {
      dx = (int)(e.getX() - dragStartX);
      dy = (int)(e.getY() - dragStartY);
      movedShape.translateShape(dx, dy);
      movedShape.setSelected(false);
      movedShape.setHighlighted(true);
      updateDrawing();
      movedShape = null;
    }
    // Rotate Shape
    else if(rotatedShape != null)
    {
      theta = (Math.atan2
        (rotatedShape.center[0] - e.getX(),
         rotatedShape.center[1] - e.getY()))
            - (Math.atan2
        (rotatedShape.center[0] -
              rotateStartX,
        rotatedShape.center[1] -
              rotateStartY));
      theta = Math.toDegrees(theta);
      theta = Math.toRadians
        (Math.round(theta));
      rotatedShape.rotateShape(theta);
      rotatedShape.setSelected(false);
      rotatedShape.setHighlighted(true);
      updateDrawing();
      setCursor(CURSOR_DEFAULTSHAPE);
      rotatedShape = null;
    }
/*
    // FOR DEBUGGING
    System.out.print("Mouse Released ");
    System.out.print(e.getX());
    System.out.print(",");
    System.out.print(e.getY());
    System.out.print("\n");
*/
  }

  public void mouseDragged(MouseEvent e)
  {
    if(!this.contains(e.getX(), e.getY()))
    {
//    mouseReleased(e);
//    shapeDest.setSelected(false);
//    shapeOrig.setSelected(false);
      return;
    }
    else if(onVertex != Shape.NO_VERTEX)
    {
      shapeOrig.moveVertex
        (onVertex, e.getX(), e.getY());
      childIter.changeShape
        (this, shapeOrig);
      updateDrawing();
      dragStartX = e.getX();
      dragStartY = e.getY();
    }
    else if(movedShape != null)
    {
      dx = (int)(e.getX() - dragStartX);
      dy = (int)(e.getY() - dragStartY);
      movedShape.translateShape(dx, dy);
      updateDrawing();
      dragStartX = e.getX();
      dragStartY = e.getY();
    }
    else if(rotatedShape != null)
    {
      theta = (Math.atan2
        (rotatedShape.center[0] - e.getX(),
        rotatedShape.center[1] - e.getY()))
            - (Math.atan2
      (rotatedShape.center[0] -
             rotateStartX,
      rotatedShape.center[1] -
             rotateStartY));
    theta = Math.toDegrees(theta);
    theta = Math.toRadians
      (Math.round(theta));
    rotatedShape.rotateShape(theta);
    updateDrawing();
    rotateStartX = e.getX();
    rotateStartY = e.getY();
  }
}

/**/
public void mouseMoved(MouseEvent e)
{
  int temp = shapeOrig.hitTestVertex
      (e.getX(), e.getY());

  if(temp != Shape.NO_VERTEX)
  {
    shapeOrig.setHighlighted(true);
    shapeDest.setHighlighted(false);
    setCursor(CURSOR_DEFAULTSHAPE);
    onVertex = temp;
    repaint();
  }
  else
  {
    if(onVertex != Shape.NO_VERTEX)
    {
      onVertex = Shape.NO_VERTEX;
      setCursor(CURSOR_MOVESHAPE);
      repaint();
    }

    else if(shapeOrig.hitTest
      (e.getX(), e.getY()))
    {
      if(!shapeOrig.getHighlighted())
      {
        shapeOrig.setHighlighted(true);
        shapeDest.setHighlighted(false);
        setCursor(CURSOR_MOVESHAPE);
        repaint();
      }
    }
    else if(shapeDest.hitTest
      (e.getX(), e.getY()))
    {
      if(!shapeDest.getHighlighted())
      {
        shapeOrig.setHighlighted(false);
        shapeDest.setHighlighted(true);
        setCursor(CURSOR_MOVESHAPE);
        repaint();
      }
    }
    else
    {
      setCursor(CURSOR_DEFAULTSHAPE);
      if(shapeOrig.getHighlighted())
      {
        shapeOrig.setHighlighted(false);
        repaint();
      }
      if(shapeDest.getHighlighted())
      {
        shapeDest.setHighlighted(false);
        repaint();
      }
    }
  }
}

/* ------------------------
   Public Methods
 * ------------------------ */

public void updateDrawing()
{
  trans = new Transform
      (shapeOrig, shapeDest);
  if (childIter != null) childIter.setRule
      (this, shapeOrig, trans);
  if (childRules != null)
      childRules.setSeedShape
             (shapeOrig, shapeDest);
  repaint();
}

/** set the drawing window's status
*/
public void setActive(boolean newState)
{
  active = newState;
  if(active)
  {
    addMouseMotionListener(this);
         addMouseListener(this);
  }
  else
  {
    removeMouseMotionListener(this);
         removeMouseListener(this);
  }
  childRules.setActive(active);
  shapeBar.setActive(active);
  repaint();
}

/** set instance of Iteration to childIter
*/
public void setIterate(Iteration iter)
{
  childIter = iter;
  childIter.setRule
      (this, shapeOrig, trans);
}

/** set instance of Rules to childRules
*/
public void setIterate(Rules iter)
{
  childRules = iter;
  childRules.setSeedShape
      (shapeOrig, shapeDest);
}
```

```
/** replace original shape with a
  * different shape
*/
public void changeShapeOrig(Shape s)
{
  s.moveShape(shapeOrig);
  shapeOrig = s;
  childIter.changeShape(this, s);
  childRules.setSeedShape
      (shapeOrig, shapeDest);
  repaint();
}

/** replace destination shape with a
  * different shape
*/
public void changeShapeDest(Shape s)
{
  s.moveShape(shapeDest);
  shapeDest = s;
  childRules.setSeedShape
      (shapeOrig, shapeDest);
  s.setSelected(false);
  s.setHighlighted(false);
  repaint();
}

/** Setup background grid in the Graphics
  * context
*/
public void drawGrid(Graphics g)
{
  int panelW = getWidth();
  int panelH = getHeight();

  g.setColor(COLOR_GRID_MINOR);
  for(int i = grid;
      i < panelW; i += grid)
    g.drawLine(i, 0, i, panelH);
  for(int i = grid;
      i < panelH; i += grid)
    g.drawLine(0, i, panelW, i);

  g.setColor(COLOR_GRID_MAIN);
  for(int i = grid*10;
      i < panelW; i += grid*10)
    g.drawLine(i, 0, i, panelH);
  for(int i = grid*10;
      i < panelH; i += grid*10)
    g.drawLine(0, i, panelW, i);
}

/** Draw shapes and background grid in
  * the Graphics context
*/
public void paint(Graphics g)
{

  if(active)
  {
    g.clearRect(0, 0, getWidth(),
      getHeight());
    this.drawGrid(g);
    if(shapeOrig != null)
    {
      if(!shapeOrig.getSelected() &&
        !shapeOrig.getHighlighted())
          shapeOrig.currentColor =
               Color.blue;
        if(onVertex != Shape.NO_VERTEX)
          shapeOrig.drawCornerHandle(g);
        shapeOrig.drawShape(g);
      }
      if(shapeDest != null)
        shapeDest.drawShape(g);
    }
    else
    {
      g.setColor(COLOR_DISABLED);
      g.fillRect(0, 0, getWidth(),
getHeight());
      this.drawGrid(g);
    }
  }

}
```

```
//Title:        Shaper2D
//Version:      2.1
//Copyright:    Copyright (c) 2001
//Author:       Miri McGill
//Company:      MIT
//Description:  Constructs and applies a
//              transformation matrix
package Shaper2D;

import java.awt.*;
import Jama.*;

public class Transform
{
  private Matrix trans;

/* ------------------------
   Constructors
 * ------------------------ */

  /** Construct the transformation matrix
    * from X * A = X'
    * Where A is the transformation trans,
    * s1 = X and s2 = X'
  @param s1  The source shape, X
  @param s2  The destination shape, X'
  @return    The transformation matrix, A
             = X(inverse) * X'
  @see       times in Jama.Matrix
  */
  public Transform(Shape s1, Shape s2)
  {
    trans = s1.location.inverse
        ().times(s2.location);
//    trans.print(new
java.text.DecimalFormat(),10);
  }

  /** Constructs a product transformation
    * for the iteration
  */
  public Transform
        (Transform t1, Transform t2)
  {
    trans = t2.trans.times(t1.trans);
  }


  // This would have been a pain to do
  // any other way...
  // It makes a transform that performs t1
  // and t2, where t0 is a relative
  // transformation between the coordinated
  // systems where t1 and t2 are defined.
  public Transform(Transform t1, Transform
        t2, Transform t0)
  {
    trans = t0.trans.times
        (t2.trans.times(t0.trans.inverse
        ().times(t1.trans)));
  }

/* ------------------------
   Public Methods
 * ------------------------ */

/**
  * Method to apply the transformation to
  * a given shape
 @param s   The shape that the
           transformation is applied to
 @return    The transformed shape
 @see       times in Jama.Matrix
*/
  public void apply(Shape s)
  {
    s.location = s.location.times(trans);
    s.basisI = s.location.getArray()[0];
    s.basisJ = s.location.getArray()[1];
    s.center = s.location.getArray()[2];
    s.updateShape();

    //Matrix transShape = s.location;
    //transShape = transShape.times(trans);
    //s.location = transShape;
  }
}
```

```
//Title:        Shaper2D
//Version:      2.1
//Copyright:    Copyright (c) 2001
//Author:       Miri McGill
//Company:      MIT
//Description:  Draws shape grammar rules
package Shaper2D;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import Jama.*;

public class Rules extends JPanel implements
       Constants
{
  public Shape seedShape1, seedShape2;
  public DrawShapes parentDrawShape;
  int nIters;
  int translation = 220;
  private boolean active = true;

/* ------------------------
   Constructors
 * ------------------------ */
  public Rules()
  {
  }

  public Rules(DrawShapes parent)
  {
    parentDrawShape = parent;
    parent.setIterate(this);
    try
    {
      jbInit();
    }
    catch(Exception e)
    {
      e.printStackTrace();
    }
  }

/* ------------------------
   Component initialization
 * ------------------------ */

  private void jbInit() throws Exception
  {
    //this.setTitle("Shaper2D Rules");
    //this.setLocation(350, 80);

    this.setSize(new Dimension(400, 250));
    nIters = 2;

    this.addMouseListener
        (new java.awt.event.MouseAdapter()
    {
      public void mousePressed
        (MouseEvent e)
      {
        this_mousePressed(e);
      }

      public void mouseReleased
        (MouseEvent e)
      {
        this_mouseReleased(e);
      }
    });
    this.addMouseMotionListener(new
        java.awt.event.MouseMotionAdapter()
    {
      public void mouseMoved(MouseEvent e)
      {
        this_mouseMoved(e);
      }
    });
  }

/* ------------------------
   Mouse event handlers
 * ------------------------ */

  void this_mousePressed(MouseEvent e)
  {
    /*
    if(seedShape1.hitTest(e.getX(),
        e.getY()))
      {
        seedShape1.rotateLabel();
        updateDrawing();
      }
    else
    */
    if(seedShape2.hitTest((e.getX()) -
        translation, e.getY()))
    {
      parentDrawShape.shapeDest.
        rotateLabel();
      updateDrawing();
      seedShape2.setHighlighted(true);
    }
  }

  void this_mouseReleased(MouseEvent e)
  {
  }

  void this_mouseMoved(MouseEvent e)
  {
    if(seedShape2.hitTest(e.getX() -
        translation, e.getY()))
    {
      if(!seedShape2.getHighlighted())
      {
        seedShape2.setHighlighted(true);
        repaint();
      }
    }
    else if(seedShape2.getHighlighted())
    {
      seedShape2.setHighlighted(false);
      repaint();
    }
  }

/* ------------------------
   Public Methods
 * ------------------------ */

  public void updateDrawing()
{
  parentDrawShape.updateDrawing();
}

/** set the rule window's status
*/
public void setActive(boolean newState)
{
  active = newState;
  repaint();
}

/** Notifies iteration of seed shape
*/
public void setSeedShape
      (Shape s1, Shape s2)
{
  seedShape1 = s1.copy();
  seedShape2 = s2.copy();
  seedShape1.setSelected(false);
  seedShape2.setSelected(false);
  repaint();
}

/** Draws the arrow for the rule windows
*/
public void drawArrow(Graphics g)
{
  drawArrow(g, Color.black);
}

public void drawArrow
      (Graphics g, Color color)
{
  g.setColor(color);
  g.drawLine(195, 100, 225, 100);
  g.drawLine(210, 105, 225, 100);
  g.drawLine(210, 95, 225, 100);
}

/** Setup the iteration in the Graphics
  * context
*/
public void drawRule(Graphics g)
{
  Shape iterShape1 = seedShape1;
  Shape iterShape2 = seedShape2;
  // draw the left side of the rule
  iterShape1.drawShape(g);
  g.setColor(Color.gray);
  iterShape1.drawLabel(g);

  g.translate(translation, 0);
  // draw the right side of the rule
  iterShape1.drawShape(g);
  g.setColor(Color.gray);
  iterShape1.drawLabel(g);
  iterShape2.drawShape(g);
  g.setColor(Color.red);
  iterShape2.drawLabel(g);
}

/** Draw the iteration in the Graphics
  * context
*/
public void paint(Graphics g)
{
  if(active)
  {
    g.clearRect(0, 0, getWidth(),
      getHeight());
    drawArrow(g);
    drawRule(g);
  }
  else
  {
    g.setColor(COLOR_DISABLED);
    g.fillRect(0, 0, getWidth(),
      getHeight());
    drawArrow(g, COLOR_LINE_DISABLED);
  }
}

}
```

```
//Title:        Shaper2D
//Version:      2.1
//Copyright:    Copyright (c) 2001
//Author:       Miri McGill
//Company:      MIT
//Description:  Draws the iteration nTimes
package Shaper2D;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;

public class Iteration extends JPanel
        implements MouseListener,
        MouseMotionListener, Constants
{
  public int nIters;

  private DrawShapes drawShapes1,
        drawShapes2;
  private Shape seedShape1, seedShape2;
  private Transform trans1, trans2;
  private int numRules = 1;
  private int xOffset, yOffset;
  private int dragStartX, dragStartY, zoomY;
  private double zoomFactor = 1.0;
  private boolean pan = false;
  private boolean zoom = false;
  private boolean drawLabels = false;


/* ------------------------
   Constructors
 * ------------------------ */

  public Iteration(DrawShapes parent1,
        DrawShapes parent2)
  {
    setCursor(CURSOR_MOVEITER);
    drawShapes1 = parent1;
    drawShapes2 = parent2;
    parent1.setIterate(this);
    parent2.setIterate(this);
    try
    {
      jbInit();
    }
    catch(Exception e)
    {
      e.printStackTrace();
    }
  }

/* ------------------------
   Component initialization
 * ------------------------ */

  private void jbInit() throws Exception
  {
    nIters = 7;
    addMouseMotionListener(this);
         addMouseListener(this);
    resetOrigin();
  }

/* ------------------------
   Mouse event handlers
 * ------------------------ */

  public void mouseClicked(MouseEvent e)
  {
  }

  public void mouseEntered(MouseEvent e)
  {
  }

  public void mouseExited(MouseEvent e)
  {
  }

  public void mouseMoved(MouseEvent e)
  {
  }

  public void mousePressed(MouseEvent e)
  {
    dragStartX = e.getX();
    dragStartY = e.getY();
    // Pan
    if((e.getModifiers() & e.BUTTON1_MASK)
        != 0
    && (e.getModifiers() & e.SHIFT_MASK)
        == 0)
    {
      pan = true;
    }
    // Zoom
    else if((e.getModifiers() &
        e.BUTTON1_MASK) == 0
          ||((e.getModifiers() &
        e.BUTTON1_MASK) != 0
          && (e.getModifiers() &
        e.SHIFT_MASK) != 0))
    {
      setCursor(CURSOR_ZOOMITER);
      zoom = true;
    }
  }

  public void mouseDragged(MouseEvent e)
  {
    if(pan)
    {
      xOffset += e.getX() - dragStartX;
      yOffset += e.getY() - dragStartY;
    }
    else if (zoom)
    {
      zoomFactor -= (e.getY() - dragStartY)
        * ZOOM_SPEED;
      if(zoomFactor <= 0.0)
        zoomFactor = ZOOM_SPEED;
    }
    dragStartX = e.getX();
    dragStartY = e.getY();
    repaint();
  }

  public void mouseReleased(MouseEvent e)
  {
    if(pan)
    {
      xOffset += e.getX() - dragStartX;
      yOffset += e.getY() - dragStartY;
      pan = false;
    }
    else if (zoom)
    {
      zoomFactor -= (e.getY() - dragStartY)
        * ZOOM_SPEED;
      if(zoomFactor <= 0.0)
        zoomFactor = ZOOM_SPEED;
      setCursor(CURSOR_MOVEITER);
      zoom = false;
    }
    repaint();
  }

/* ------------------------
   Public Methods
 * ------------------------ */

  public void resetOrigin()
  {
    xOffset = getWidth()  / 2;
    yOffset = getHeight() / 2;
    zoomFactor = 1.0;
  }

  /** Sets the number of iterations
  */
  public void setIters(int iters)
  {
    nIters = iters;
    repaint();
  }

  /** Sets the number of rules
  */
  public void setNumRules(int n)
  {
    numRules = n;

    switch(n)
    {
      case 1:
        drawShapes2.setActive(false);
        drawShapes1.changeShapeDest
              (seedShape1.copy());
        seedShape2 = seedShape1;
        repaint();
        break;

      case 2:
        drawShapes2.setActive(true);
        seedShape2 = drawShapes2.shapeOrig;
        drawShapes1.changeShapeDest
              (seedShape2.copy());
        repaint();
        break;
    }
  }

  /** Notifies iteration of grammar rule
  */
  public void setRule(DrawShapes src, Shape
      s, Transform t)
  {
    if (src == drawShapes1)
    {
      seedShape1 = s;
      trans1 = t;
    }
    else
    {
      seedShape2 = s;
      trans2 = t;
    }
    repaint();
  }

  /**
  */
  public void changeShape(DrawShapes src,
      Shape s)
  {
    if (src == drawShapes1)
    {
      seedShape1 = s;
      drawShapes2.changeShapeDest(s.copy());
    }
    if (src == drawShapes2 || numRules
       == 1)
    {
      seedShape2 = s;
      drawShapes1.changeShapeDest(s.copy());
    }
    repaint();
  }

  /** Show the labels
  */
  public void showLabels()
  {
    drawLabels = !drawLabels;
    repaint();
  }

  /** Draw the iteration in the Graphics
    * context
  */
  public void draw1RuleIteration(Graphics g)
  {
    Shape iterShape = seedShape1.copy();
    iterShape.setSelected(false);

    for (int i = 1; i <= nIters; i++)
    {
      if (drawLabels)
      {
        iterShape.currentColor =
               Color.black;
        if (i == nIters)
          iterShape.currentColor =
               Color.red;
      }
      else
      iterShape.currentColor = Color.black;
      /* // For using color in the
         // iteration  feature to be added
        iterShape.currentColor = new
Color(255 - (i - 1) * 255 /
      (nIters - 1), 0, (i - 1) * 255 /
              (nIters - 1));
    */
    iterShape.drawShape(g);
    if (drawLabels)
      iterShape.drawLabel(g);
    trans1.apply(iterShape);
  }
}

public void draw2RuleIteration(Graphics g)
{
  Shape iterShape1 = seedShape1.copy();
  Shape iterShape2 = seedShape2.copy();
  Shape iterShape;
  Transform combined = new Transform
      (trans1, trans2, new Transform
              (iterShape1,iterShape2));

  trans1.apply(iterShape1);
  iterShape2.moveShape(iterShape1);
  iterShape1 = seedShape1.copy();

  iterShape1.setSelected(false);
  iterShape2.setSelected(false);

  for (int i = 1; i <= nIters; i++)
  {
    // checking to see if i is odd
    if (i%2 == 1)
      iterShape = iterShape1;
    else
      iterShape = iterShape2;

    if (drawLabels)
    {
      iterShape.currentColor =
             Color.black;
      if (i == nIters)
        iterShape.currentColor =
             Color.red;
    }
    else
    iterShape.currentColor = Color.black;
    /* // For using color in the
       // iteration
      iterShape.currentColor = new
     Color(255 - (i - 1) * 255 /
      (nIters - 1), 0, (i - 1) * 255 /
              (nIters - 1));
    */
    iterShape.drawShape(g);
    if (drawLabels)
      iterShape.drawLabel(g);
    combined.apply(iterShape);
  }
}

/** Draw the iteration in DXF format
*/
public void draw1RuleIterationDXF
      (PrintStream pout)
{
  Shape iterShape = seedShape1.copy();
  iterShape.setSelected(false);
```

```
  for (int i = 1; i <= nIters; i++)
  {
    iterShape.drawShapeDXF(pout);
    iterShape.drawLabelDXF(pout);
    trans1.apply(iterShape);
  }
}

public void draw2RuleIterationDXF
      (PrintStream pout)
{
  Shape iterShape1 = seedShape1.copy();
  Shape iterShape2 = seedShape2.copy();
  Shape iterShape;
  Transform combined = new Transform
      (trans1, trans2, new Transform
              (iterShape1,iterShape2));

  trans1.apply(iterShape1);
  iterShape2.moveShape(iterShape1);
  iterShape1 = seedShape1.copy();

  iterShape1.setSelected(false);
  iterShape2.setSelected(false);

  for (int i = 1; i <= nIters; i++)
  {
    if (i%2 == 1)
      iterShape = iterShape1;
    else
      iterShape = iterShape2;


    if (drawLabels)
    {
      iterShape.currentColor =
             Color.black;
      if (i == nIters)
        iterShape.currentColor =
             Color.red;
    }
    else
    iterShape.currentColor = Color.black;
    /* // For using color in the
       // iteration
      iterShape.currentColor = new
             Color(255 - (i - 1) * 255 /
             (nIters - 1), 0, (i - 1) *
             255 / (nIters - 1));
    */
    iterShape.drawShapeDXF(pout);
    iterShape.drawLabelDXF(pout);
    combined.apply(iterShape);
  }
}


/** Paint the iteration in the Graphics
  * context
*/
public void paint(Graphics g)
{
  g.clearRect(0, 0, getWidth(),
     getHeight());
  g.translate(xOffset, yOffset);
  ((Graphics2D) g).scale(zoomFactor,
        zoomFactor);
    g.translate(-drawShapes1.getWidth()
        / 2, -drawShapes1.getHeight() / 2);

    if(numRules == 1)
      draw1RuleIteration(g);
    else
      draw2RuleIteration(g);
  }

  /** Print the iteration as DXF output
  */
  public void printDXF(String name)
  {
    try
    {
      FileOutputStream fout = new
        FileOutputStream(name);
      PrintStream pout = new
        PrintStream(fout);
      printDXFHeader(pout);

      if(numRules == 1)
        draw1RuleIterationDXF(pout);
      else
        draw2RuleIterationDXF(pout);

      printDXFFooter(pout);
      pout.close();
      fout.close();
    }
    catch(Exception e)
    {
      System.out.print("Couldn't write to
        file");
    }
  }

  /** Print the DXF header
  */
  private void printDXFHeader(PrintStream
        pout)
  {
    pout.println("0");
    pout.println("SECTION");
    pout.println("2");
    pout.println("ENTITIES");
  }

  /** Print the DXF footer
  */
  private void printDXFFooter(PrintStream
        pout)
  {
    pout.println("0");
    pout.println("ENDSEC");
    pout.println("0");
    pout.println("EOF");
  }
}
```

```
//Title:        Shaper2D
//Version:      2.1
//Copyright:    Copyright (c) 2001
//Author:       Miri McGill
//Company:      MIT
//Description:  Abstract class for
//              constructing shapes
package Shaper2D;

import java.awt.*;
import java.util.*;
import Jama.*;
import java.io.*;

abstract public class Shape implements
       Constants
{
  protected Color currentColor =
       Color.black;
  protected Color lineColor = Color.black;
  protected Color selectColor = Color.red;
  protected Color highlightColor =
       Color.red;
  protected Color fillColor = Color.gray;

  public static int NO_VERTEX = -1;
  public boolean filled = false;
  public boolean selected = false;
  public boolean highlighted = false;
  public boolean isMirrored = false;
  // label diameter
  public double labelD = 6.0;
  // label radius
  public double labelR = labelD / 2.0;

  Matrix location = new Matrix(3, 3);
  double basisI[] = location.getArray()[0];
  double basisJ[] = location.getArray()[1];
  double center[] = location.getArray()[2];

/* ------------------------
   Constructors
 * ------------------------ */

  public Shape()
  {
    setLineColor(Color.black);
    setSelected(false);
    setHighlighted(false);
    initializeBasis();
  }

  public Shape(Color lineColor)
  {
    setLineColor(lineColor);
    setSelected(false);
    setHighlighted(false);
    initializeBasis();
    /*
    createHandles();
    */
  }

  public Shape(Color lineColor, Color
       fillColor)
  {
    setLineColor(lineColor);
    setFillColor(fillColor);
    setSelected(false);
    setHighlighted(false);
    initializeBasis();
  }

  /** Copy Constructor
  */
  public Shape(Shape src)
  {
    lineColor = src.lineColor;
    currentColor = src.currentColor;
    selectColor = src.selectColor;
    highlightColor = src.highlightColor;
    fillColor = src.fillColor;

    filled  = src.filled;
    selected = src.selected;
    highlighted = src.highlighted;

    location = src.location.copy();
    basisI = location.getArray()[0];
    basisJ = location.getArray()[1];
    center = location.getArray()[2];
  }

/* ------------------------
   Public Methods
 * ------------------------ */

  /** Dummy copy (cloning) method to be
    * overridden by subclasses of shape
  */
  public Shape copy()
  {
    return null;
  }

  public void initializeBasis()
  {
    basisI[0] = 1.0;
    basisI[1] = 0.0;
    basisI[2] = 0.0;
    basisJ[0] = 0.0;
    basisJ[1] = 1.0;
    basisJ[2] = 0.0;
    center[0] = 0.0;
    center[1] = 0.0;
    center[2] = 1.0;
    isMirrored = false;
  }

  /** Methods for determining color of
    * line, selected & highlighted shape
  */
  public void setLineColor(Color lineColor)
  {
    this.lineColor = lineColor;
        }

        public void setFillColor(Color
        fillColor)
  {
    this.fillColor = fillColor;
    if(fillColor != null)
      filled = true;
        }

  public void setSelectColor(Color
        selectColor)
  {
    this.selectColor = selectColor;
        }

  public void setHighlightColor(Color
        highlightColor)
  {
    this.highlightColor = highlightColor;
  }

  /** Methods for determining whether shape
    * is selected
  */
  public void setSelected(boolean selected)
  {
    this.selected = selected;
    if(selected)
      currentColor = selectColor;
    else
      currentColor = lineColor;
  }

  public boolean getSelected()
  {
    return selected;
  }

  /** Methods for determining whether shape
    * is highlighted
  */
  public void setHighlighted(boolean
        highlighted)
  {
    this.highlighted = highlighted;
    if(highlighted)
      currentColor = highlightColor;
    else
      currentColor = lineColor;
  }

  public boolean getHighlighted()
  {
    return highlighted;
  }


  /** Translate the shape by some distance
  */
  public void translateShape(double dx,
        double dy)
  {
    center[0] += dx;
    center[1] += dy;
    updateShape();
  }

  /** Rotate the shape by some angle
  @param theta     The angle of rotation
  */
  public void rotateShape(double theta)
  {
    double x, y;

    x = basisI[0];
    y = basisI[1];
    basisI[0] = (x * Math.cos(theta)) + (y
        * Math.sin(theta));
    basisI[1] = (y * Math.cos(theta)) - (x
        * Math.sin(theta));

    x = basisJ[0];
    y = basisJ[1];
    basisJ[0] = (x * Math.cos(theta)) + (y
        * Math.sin(theta));
    basisJ[1] = (y * Math.cos(theta)) - (x
        * Math.sin(theta));
    updateShape();
  }

  /** Mirror the shape about the X axis
  */
  public void mirrorShapeX()
  {
    basisJ[0] = -basisJ[0];
    basisJ[1] = -basisJ[1];
    isMirrored = !isMirrored;
    updateShape();
  }

  /** Mirror the shape about the Y axis
  */
  public void mirrorShapeY()
  {
    basisI[0] = -basisI[0];
    basisI[1] = -basisI[1];
    isMirrored = !isMirrored;
    updateShape();
  }

  /** Takes a shape and moves it to the
    * location of another shape --
    * this is a way to change one shape
    * into another shape for the iteration
  */
  public void moveShape(Shape s)
  {
    location = s.location.copy();
    isMirrored = s.isMirrored;
    basisI = location.getArray()[0];
    basisJ = location.getArray()[1];
    center = location.getArray()[2];
    updateShape();
  }

  /** Placeholder updateShape method -- for
    * derived classes
  */
  public void updateShape()
  {
  }

  /** Placeholder method for rotating the
label -- for derived classes
  */
  public void rotateLabel()
  {
  }
```

```
/** Hit test method
@return boolean
*/
public boolean hitTest(int x, int y)
{
  return false;
}

/** Hit test method for the vertices
@return boolean
*/
public int hitTestVertex(int x, int y)
{
  return NO_VERTEX;
}

/** Placeholder method for moving a
  * shape's vertex
*/
public void moveVertex(int i, int x,
      int y)
{
}

/** Placeholder method for checking a
  * shape's constraints
*/
public void finalizeVertex()
{
}

/** Method for filling a shape
@return boolean
*/
public boolean getFilled()
{
  return filled;
}

/** Placeholder method for drawing the
  * shape in the graphics context
*/
public void drawShape(Graphics g)
{
}

/** Placeholder method for drawing the
  * label in the graphics context
*/
public void drawLabel(Graphics g)
{
}

/** Placeholder method for drawing a
  * resize handle in the graphics context
*/
public void drawCornerHandle(Graphics g)
{
}

/** Draw the shape in DXF format
*/
public void drawShapeDXF(PrintStream pout)
{
}
```

```
  /** Draw the label in DXF format
  */
  public void drawLabelDXF(PrintStream pout)
  {
  }

}
```

```
//Title:        Shaper2D
//Version:      2.1
//Copyright:    Copyright (c) 2001
//Author:       Miri McGill
//Company:      MIT
//Description:  Abstract class for
//              constructing polygonal
//              shapes
package Shaper2D;

import java.awt.*;
import Jama.*;
import java.io.*;

public class PolyShape extends Shape
implements Constants
{
  protected Polygon poly;

/* ------------------------
   Constructors
 * ------------------------ */

  public PolyShape()
  {
  }

  public PolyShape(Color lineColor)
  {
    super(lineColor);
  }

  public PolyShape(Color lineColor, Color
       fillColor)
  {
    super(lineColor, fillColor);
  }

  /** Copy constructor
  @param src       instance of PolyShape
  */
  public PolyShape(PolyShape src)
  {
    super((Shape)src);
  }


/* ------------------------
   Public Methods
 * ------------------------ */

  /** Hit test method to see if shape
    * contains mouse pointer
  */
  public boolean hitTest(int hitX, int hitY)
  {
    return poly.contains(hitX, hitY);
  }

  /** Hit test method to see if mouse
    * pointer touches a vertex
  */
  public int hitTestVertex(int hitX, int
       hitY)
  {
    int d = 5;
  int dx;
  int dy;

  for(int i = 0;  i < poly.npoints; i++)
  {
     dx = (hitX - poly.xpoints[i]);
     dy = (hitY - poly.ypoints[i]);

     if ((d * d) >= (dx * dx) +
      (dy * dy))
       return i;
  }
  return NO_VERTEX;
}

/** Method for moving a shape's vertex
*/
public void moveVertex(int i, int x,
      int y)
{
  poly.xpoints[i] = x;
  poly.ypoints[i] = y;
}

/** Draw a resize handle in the graphics
  * context
*/
public void drawCornerHandle(Graphics g)
{
  for(int i = 0; i < poly.npoints; i++)
  {
    g.setColor(Color.red);
    g.fillRect(poly.xpoints[i] - 3,
      poly.ypoints[i] - 3, 6, 6);
  }
}

/** Draw the shape in the Graphics context
*/
public void drawShape(Graphics g)
{
  g.setColor(currentColor);
  if (poly != null)
  {
    g.drawPolygon(poly);
  }
  /*
  if(filled == true)
  {
    g.setColor(fillColor);
    g.fillPolygon(poly);
  }
  */
}

/** Draw the shape in DXF format
*/
public void drawShapeDXF(PrintStream pout)
{
  if (poly != null)
  {
    pout.println("0");
    pout.println("POLYLINE");
    pout.println("10");
    pout.println("0");
    pout.println("20");
    pout.println("0");
    pout.println("8");
    pout.println(LAYER_SHAPE);
    pout.println("70");
    pout.println("1");
    pout.println("66");
    pout.println("1");

    for(int i=0;  i < poly.npoints; i++)
    {
      pout.println("0");
      pout.println("VERTEX");
      pout.println("8");
      pout.println(LAYER_SHAPE);
      pout.println("10");
      pout.println(poly.xpoints[i]);
      pout.println("20");
      pout.println(-poly.ypoints[i]);
    }
    pout.println("0");
    pout.println("SEQEND");
  }
 }
}
```

```
//Title:        Shaper2D
//Version:      2.1
//Copyright:    Copyright (c) 2001
//Author:       Miri McGill
//Company:      MIT
//Description:  Subclass of PolyShape for
//              constructing rectangles
package Shaper2D;

import java.awt.*;
import Jama.*;

public class MyRectangle extends PolyShape
implements Constants
{
  public double halfWidth, halfHeight;

/* ------------------------
   Constructors
 * ------------------------ */

  public MyRectangle()
  {
  }

  /** Constructs a rectangle with a starting
    * point, width, height & rotation
    *
  @param xOrig     The x coordinate for
                   the rectangle's origin
  @param yOrig     The y coordinate for
                   the rectangle's origin
  @param w         The width of the
                  rectangle
  @param h         The height of the
                   rectangle
  @param theta     The angle of rotation
  */
  public MyRectangle(int xOrig, int yOrig,
        int w, int h, double theta)
  {
    makeRect(xOrig, yOrig, w, h, theta);
    makePoly();
  }

  /** Constructs a rectangle with a starting
    * point, width, height, rotation &
    * line color
  @param xOrig     The x coordinate for
                   the rectangle's origin
  @param yOrig     The y coordinate for
                   the rectangle's origin
  @param w         The width of the
                   rectangle
  @param h         The height of the
                   rectangle
  @param theta     The angle of rotation
  @param lineColor The line color
  */
  public MyRectangle(int xOrig, int yOrig,
        int w, int h, double theta,
                     Color lineColor)
  {
    super(lineColor);
    makeRect(xOrig, yOrig, w, h, theta);
    makePoly();
  }

  /** Constructs a rectangle with a starting
    * point, width, height, rotation,
    * line & fill color
  @param xOrig     The x coordinate for
                   the rectangle's origin
  @param yOrig     The y coordinate for
                   the rectangle's origin
  @param w         The width of the
                   rectangle
  @param h         The height of the
                   rectangle
  @param theta     The angle of rotation
  @param lineColor The line color
  @param fillColor The fill color
  */
  public MyRectangle(int xOrig, int yOrig,
        int w, int h, double theta,
         Color lineColor, Color fillColor)
  {
    super(lineColor, fillColor);
    makeRect(xOrig, yOrig, w, h, theta);
    makePoly();
  }

  /** Copy constructor
  @param src       instance of MyRectangle
  */
  public MyRectangle(MyRectangle src)
  {
    super((PolyShape)src);
    halfWidth = src.halfWidth;
    halfHeight = src.halfHeight;
    makePoly();
  }

  /** Copy (cloning) method
  @return          a copy of the
                   instance of MyRectangle
  */
  public Shape copy()
  {
    return new MyRectangle(this);
  }

/* ------------------------
   Public Methods
 * ------------------------ */

  /** Create each point and add it to the
    * rectangle
    *
  @param xOrig     The x coordinate for
                   the rectangle's origin
  @param yOrig     The y coordinate for
                   the rectangle's origin
  @param w         The width of the
                   rectangle
  @param h         The height of the
                   rectangle
  @param theta     The angle of rotation
  */
  public void makeRect(int xOrig, int yOrig,
        int w, int h, double theta)
  {
    center[0] = xOrig;
    center[1] = yOrig;
    halfWidth = Math.abs((double)w / 2.0);
    halfHeight = Math.abs((double)h / 2.0);
    rotateShape(theta);
  }

  /** Make the polygon for use in hitTesting
  */
  public void makePoly()
  {
    poly = new Polygon();
    double x1, y1, x2, y2, x3, y3, x4, y4;

    x1 = center[0] + (basisJ[0] *
        halfHeight) - (basisI[0] *
        halfWidth);
    y1 = center[1] + (basisJ[1] *
        halfHeight) - (basisI[1] *
        halfWidth);
    x2 = center[0] + (basisJ[0] *
        halfHeight) + (basisI[0] *
        halfWidth);
    y2 = center[1] + (basisJ[1] *
        halfHeight) + (basisI[1] *
        halfWidth);
    x3 = center[0] - (basisJ[0] *
        halfHeight) + (basisI[0] *
        halfWidth);
    y3 = center[1] - (basisJ[1] *
        halfHeight) + (basisI[1] *
        halfWidth);
    x4 = center[0] - (basisJ[0] *
        halfHeight) - (basisI[0] *
        halfWidth);
    y4 = center[1] - (basisJ[1] *
        halfHeight) - (basisI[1] *
        halfWidth);

    poly.addPoint((int)Math.round(x1),
        (int)Math.round(y1));
    poly.addPoint((int)Math.round(x2),
        (int)Math.round(y2));
    poly.addPoint((int)Math.round(x3),
        (int)Math.round(y3));
    poly.addPoint((int)Math.round(x4),
        (int)Math.round(y4));
  }

  /** Method for updating the shape after
    * it has been manipulated
  */
  public void updateShape()
  {
    makePoly();
  }

  /** Method for rotating the label
  */

  public void rotateLabel()
  {
    if(isMirrored)
      mirrorShapeY();
    else
      mirrorShapeX();
  }

  /** Draw the label in the Graphics context
  @param g         The graphics context
  */
  public void drawLabel(Graphics g)
  {
    double d = labelD;
    double r = labelR;
    double x1, y1;
    x1 = center[0] + (basisJ[0] *
        (halfHeight - d)) - (basisI[0] *
        (halfWidth - d));
    y1 = center[1] + (basisJ[1] *
        (halfHeight - d)) - (basisI[1] *
        (halfWidth - d));
    g.fillOval((int)Math.round(x1 - r),
        (int)Math.round(y1 - r), (int)d,
        (int)d);
  }

  /** Overriding method for moving a
    * rectangle's vertex (corner)
  */
  public void moveVertex(int i, int x,
       int y)
  {
    double dx = x - center[0];
    double dy = y - center[1];
    halfWidth  = Math.round((dx * basisI[0])
       + (dy * basisI[1]));
    halfHeight = Math.round((dx * basisJ[0])
       + (dy * basisJ[1]));

    if(i == 0 || i == 3)
      halfWidth = -halfWidth;
    if(i == 2 || i == 3)
      halfHeight = -halfHeight;

    if(halfWidth < MIN_SIZE)
      halfWidth = MIN_SIZE;
    if(halfHeight < MIN_SIZE)
      halfHeight = MIN_SIZE;

    updateShape();
  }

  /** Method for checking a rectangle's
    * constraints
    * If user turns rectangle into square
    * the shape perturbs back to a
    * rectangle by predefined amount
  */
  public void finalizeVertex()
  {
    if(Math.abs(halfWidth - halfHeight) <
       SHAPE_SLOP)
    {
      if(halfWidth > halfHeight)
        halfWidth = halfHeight + 5;
      else
        halfHeight = halfWidth + 5;
      updateShape();
    }
  }
}
```

```
//Title:        Shaper2D
//Version:      2.1
//Copyright:    Copyright (c) 2001
//Author:       Miri McGill
//Company:      MIT
//Description:  Subclass of PolyShape for
//              constructing squares
package Shaper2D;

import java.awt.*;
import Jama.*;

public class MySquare extends PolyShape
implements Constants
{
  public double halfWidth;

/* ------------------------
   Constructors
 * ------------------------ */

  public MySquare()
  {
  }

  /** Constructs a square with a starting
    * point, width, height & rotation
    *
  @param xOrig     The x coordinate for
                   the square's origin
  @param yOrig     The y coordinate for
                   the square's origin
  @param w         The width of the
                   square
  @param theta     The angle of rotation
  */
  public MySquare(int xOrig, int yOrig,
        int w, double theta)
  {
    makeSquare(xOrig, yOrig, w, theta);
    makePoly();
  }

  /** Constructs a square with a starting
    * point, width, height, rotation &
    * line color
  @param xOrig     The x coordinate for
                   the square's origin
  @param yOrig     The y coordinate for
                   the square's origin
  @param w         The width of the
                   square
  @param theta     The angle of rotation
  @param lineColor The line color
  */
  public MySquare(int xOrig, int yOrig,
        int w, double theta,
        Color lineColor)
  {
    super(lineColor);
    makeSquare(xOrig, yOrig, w, theta);
    makePoly();
  }

  /** Constructs a square with a starting
    * point, width, height, rotation,
    * line & fill color
  @param xOrig     The x coordinate for
                   the square's origin
  @param yOrig     The y coordinate for
                   the square's origin
  @param w         The width of the
                   square
  @param theta     The angle of rotation
  @param lineColor The line color
  @param fillColor The fill color
  */
  public MySquare(int xOrig, int yOrig,
        int w, double theta,
       Color lineColor, Color fillColor)
  {
    super(lineColor, fillColor);
    makeSquare(xOrig, yOrig, w, theta);
    makePoly();
  }

  /** Copy constructor
  @param src       An instance of MySquare
  */
  public MySquare(MySquare src)
  {
    super((PolyShape)src);
    halfWidth = src.halfWidth;
    makePoly();
  }

  /** Copy (cloning) method
  @return          A copy of the
                   instance of MyRectangle
  */
  public Shape copy()
  {
    return new MySquare(this);
  }

/* ------------------------
   Public Methods
 * ------------------------ */

  /** Create each point and add it to
    * the square
    *
  @param xOrig     The x coordinate for
                   the square's origin
  @param yOrig     The y coordinate for
                   the square's origin
  @param w         The width of the
                   square
  @param theta     The angle of rotation
  */
  public void makeSquare(int xOrig,
        int yOrig, int w, double theta)
  {
    center[0] = xOrig;
    center[1] = yOrig;
    halfWidth = Math.abs((double)w / 2.0);
    rotateShape(theta);
  }

  /** Make the polygon for use in hitTesting
  */
  public void makePoly()
{
  poly = new Polygon();
  double x1, y1, x2, y2, x3, y3, x4, y4;

  x1 = center[0] + (basisJ[0] * halfWidth)
      - (basisI[0] * halfWidth);
  y1 = center[1] + (basisJ[1] * halfWidth)
      - (basisI[1] * halfWidth);
  x2 = center[0] + (basisJ[0] * halfWidth)
      + (basisI[0] * halfWidth);
  y2 = center[1] + (basisJ[1] * halfWidth)
      + (basisI[1] * halfWidth);
  x3 = center[0] - (basisJ[0] * halfWidth)
      + (basisI[0] * halfWidth);
  y3 = center[1] - (basisJ[1] * halfWidth)
      + (basisI[1] * halfWidth);
  x4 = center[0] - (basisJ[0] * halfWidth)
      - (basisI[0] * halfWidth);
  y4 = center[1] - (basisJ[1] * halfWidth)
      - (basisI[1] * halfWidth);

  poly.addPoint((int)Math.round(x1),
      (int)Math.round(y1));
  poly.addPoint((int)Math.round(x2),
      (int)Math.round(y2));
  poly.addPoint((int)Math.round(x3),
      (int)Math.round(y3));
  poly.addPoint((int)Math.round(x4),
      (int)Math.round(y4));
}

/** Method for updating the shape after
  * it has been manipulated
*/
public void updateShape()
{
  makePoly();
}

/** Method for rotating the label
*/

public void rotateLabel()
{
  if(isMirrored)
  {
    rotateShape(Math.PI / 2);
    mirrorShapeY();
  }
  else
    mirrorShapeX();
}

/** Draw the label in the Graphics context
@param g         The graphics context
*/
public void drawLabel(Graphics g)
{
  double d = labelD;
  double r = labelR;
  double x1, y1;
  x1 = center[0] + (basisJ[0] * (halfWidth
      - (d * 2))) - (basisI[0] * (halfWidth
      - d));
  y1 = center[1] + (basisJ[1] * (halfWidth
      - (d * 2))) - (basisI[1] * (halfWidth
      - d));
  g.fillOval((int)Math.round(x1 - r),
      (int)Math.round(y1 - r), (int)d,
      (int)d);
}

/** Overriding method for moving a
  * square's vertex (corner)
*/
public void moveVertex(int i, int x,
     int y)
{
  double r;
  r = Math.sqrt((((x - center[0]) * (x
      - center[0])) + ((y - center[1]) *
      (y - center[1]))) / 2.0);
  if(r < MIN_SIZE)
    r = MIN_SIZE;
  halfWidth = Math.round(r);
  updateShape();
}
}
```

```
//Title:        Shaper2D
//Version:      2.1
//Copyright:    Copyright (c) 2001
//Author:       Miri McGill
//Company:      MIT
//Description:  Subclass of PolyShape for
//              constructing equilateral
//              triangles
package Shaper2D;

import java.awt.*;
import Jama.*;

public class MyEqTriangle extends PolyShape
       implements Constants
{
  public double triSize;
  private double x1, y1, x2, y2, x3, y3;

/* ------------------------
   Constructors
 * ------------------------ */

  public MyEqTriangle()
  {
  }

  /** Constructs an equilateral triangle
    * with a starting point, size
    * and rotation
  @param xOrig      The x coordinate for
                    the triangle's origin
  @param yOrig      The y coordinate for
                    the triangle's origin
  @param size       radius of circle which
                    circumscribes the
                    triangle
  @param theta      The angle of rotation
  */
  public MyEqTriangle(int xOrig, int yOrig,
        int size, double theta)
  {
    makeEqTriangle(xOrig, yOrig, size,
        theta);
    makePoly();
  }

  /** Constructs an equilateral triangle
    * with a starting point, size,
    * rotation & line color
  @param xOrig      The x coordinate for
                    the triangle's origin
  @param yOrig      The y coordinate for
                    the triangle's origin
  @param size       radius of circle which
                    circumscribes the
                    triangle
  @param theta      The angle of rotation
  @param lineColor The line color
  */
  public MyEqTriangle(int xOrig, int yOrig,
        int size, double theta,
        Color lineColor)
  {
    super(lineColor);
    makeEqTriangle(xOrig, yOrig, size,
theta);
    makePoly();
  }

  /** Constructs an equilateral triangle
    * with a starting point, size,
    * rotation, line color & fill color
  @param xOrig      The x coordinate for
                    the triangle's origin
  @param yOrig      The y coordinate for
                    the triangle's origin
  @param size       radius of circle which
                    circumscribes the
                    triangle
  @param theta      The angle of rotation
  @param lineColor The line color
  @param fillColor The fill color
  */
  public MyEqTriangle(int xOrig, int yOrig,
        int size, double theta,
        Color lineColor, Color fillColor)
  {
    super(lineColor, fillColor);
    makeEqTriangle(xOrig, yOrig, size,
        theta);
    makePoly();
  }

  /** Copy constructor
  @param src        instance of MyEqTriangle
  */
  public MyEqTriangle(MyEqTriangle src)
  {
    super((PolyShape)src);
    triSize = src.triSize;
    makePoly();
  }

  /** Copy (cloning) method
  @return           A copy of the
                    instance of MyEqTriangle
  */
  public Shape copy()
  {
    return new MyEqTriangle(this);
  }

/* ------------------------
   Public Methods
 * ------------------------ */

  /** Make the triangle by creating each
    * point and adding it to the triangle
  @param xOrig      The x coordinate for
                    the triangle's origin
  @param yOrig      The y coordinate for
                    the triangle's origin
  @param size       radius of circle which
                    circumscribes the
                    triangle
  @param theta      The angle of rotation
  */
  public void makeEqTriangle(int xOrig,
        int yOrig, int size, double theta)
  {
    center[0] = xOrig;
  center[1] = yOrig;
  triSize = size;
  rotateShape(theta);
}

/** Make the polygon for use in hitTesting
  * Math.round() used to overcome problem
  * of rounding errors
@see            Math.round() in
                java.math
*/
public void makePoly()
{
  poly = new Polygon();

  double a = triSize * ((Math.sqrt(3.0))
      / 2.0);
  double b = triSize / 2.0;

  x1 = center[0] + (basisJ[0] * triSize);
  y1 = center[1] + (basisJ[1] * triSize);
  x2 = center[0] - (basisJ[0] * b) +
      (basisI[0] * a);
  y2 = center[1] - (basisJ[1] * b) +
      (basisI[1] * a);
  x3 = center[0] - (basisJ[0] * b) -
      (basisI[0] * a);
  y3 = center[1] - (basisJ[1] * b) -
      (basisI[1] * a);

  poly.addPoint((int)Math.round(x1),
      (int)Math.round(y1));
  poly.addPoint((int)Math.round(x2),
      (int)Math.round(y2));
  poly.addPoint((int)Math.round(x3),
      (int)Math.round(y3));
}


/** Method for updating the shape after
  * it has been manipulated
*/

public void updateShape()
{
  makePoly();
}


/** Method for rotating the label
*/
public void rotateLabel()
{
  if(isMirrored)
  {
    rotateShape(- Math.PI * 2.0 / 3.0);
    mirrorShapeY();
  }
  else
    mirrorShapeY();
}

/** Draw the label in the Graphics context
@param g         The graphics context
*/
public void drawLabel(Graphics g)
{
  double d = labelD;
  double r = labelR;
  double xL = Math.sin(15.0 * Math.PI
      / 180.0) * (d * 2.0);
  double yL = Math.cos(15.0 * Math.PI
      / 180.0) * (d * 2.0);
  double x, y;
  //double a = triSize * (
      (Math.sqrt(3.0)) / 2.0);
  //double b = triSize / 2.0;

  x = center[0] + (basisJ[0] *
      (triSize - yL)) + (basisI[0] * xL);
  y = center[1] + (basisJ[1] *
      (triSize - yL)) + (basisI[1] * xL);
  g.fillOval((int)Math.round(x - r),
      (int)Math.round(y - r), (int)d,
      (int)d);
}

/** Overriding method for moving an
  * equilateral triangle's vertex
*/
public void moveVertex(int i, int x,
      int y)
{
  double r;
  r = Math.sqrt(((x - center[0]) * (x
      - center[0])) + ((y - center[1]) *
      (y - center[1])));
  if(r < (MIN_SIZE * 1.5))
    r = (MIN_SIZE * 1.5);
  triSize = Math.round(r);
  updateShape();
}

}
```

```
//Title:       Shaper2D
//Version:     2.1
//Copyright:   Copyright (c) 2001
//Author:      Miri McGill
//Company:     MIT
//Description: Subclass of PolyShape for
//             constructing isoceles tri.
package Shaper2D;

import java.awt.*;
import Jama.*;

public class MyIsoTriangle extends PolyShape
       implements Constants
{
  public double triBase, triHeight;
  private double x1, y1, x2, y2, x3, y3;

/* ------------------------
   Constructors
 * ------------------------ */

  public MyIsoTriangle()
  {
  }

  /** Constructs an equilateral triangle
    * with starting point, sizen & rotation
  @param xOrig     The x coordinate for
                   the triangle's origin
  @param yOrig     The y coordinate for
                   the triangle's origin
  @param base      The length of the base
                   of the triangle
  @param height    height of triangle
  @param theta     The angle of rotation
  */
  public MyIsoTriangle(int xOrig, int yOrig,
        int base, int height, double theta)
  {
    makeIsoTriangle(xOrig, yOrig, base,
        height, theta);
    makePoly();
  }

  /** Constructs an equilateral triangle
    * with starting point, size,
    * rotation & line color
  @param xOrig     The x coordinate for
                   the triangle's origin
  @param yOrig     The y coordinate for
                   the triangle's origin
  @param base      The length of the base
                   of the triangle
  @param height    height of triangle
  @param theta     The angle of rotation
  @param lineColor The line color
  */
  public MyIsoTriangle(int xOrig, int yOrig,
        int base, int height, double theta,
         Color lineColor)
  {
    super(lineColor);
    makeIsoTriangle(xOrig, yOrig, base,
        height, theta);
    makePoly();
  }

  /** Constructs an equilateral triangle
    * with a starting point, size,
    * rotation, line color & fill color
  @param xOrig     The x coordinate for
                   the triangle's origin
  @param yOrig     The y coordinate for
                   the triangle's origin
  @param base      The length of the base
                   of the triangle
  @param height    height of triangle
  @param theta     The angle of rotation
  @param lineColor The line color
  @param fillColor The fill color
  */
  public MyIsoTriangle(int xOrig, int yOrig,
        int base, int height, double theta,
         Color lineColor, Color fillColor)
  {
    super(lineColor, fillColor);
    makeIsoTriangle(xOrig, yOrig, base,
        height, theta);
    makePoly();
  }

  /** Copy constructor
  @param src       instance of MyEqTriangle
  */
  public MyIsoTriangle(MyIsoTriangle src)
  {
    super((PolyShape)src);
    triBase = src.triBase;
    triHeight = src.triHeight;
    makePoly();
  }

  /** Copy (cloning) method
  @return          A copy of the
                   instance of MyEqTriangle
  */
  public Shape copy()
  {
    return new MyIsoTriangle(this);
  }

/* ------------------------
   Public Methods
 * ------------------------ */

  /** Make the triangle by creating each
    * point and adding it to the triangle
  @param xOrig     The x coordinate for
                   the triangle's origin
  @param yOrig     The y coordinate for
                   the triangle's origin
  @param base      The length of the base
                   of the triangle
  @param height    height of triangle
  @param theta     The angle of rotation
  */
  public void makeIsoTriangle(int xOrig, int
        yOrig, int base, int height,
        double theta)
  {
    center[0] = xOrig;
    center[1] = yOrig;
    triBase = base;
    triHeight = height;
    rotateShape(theta);
  }

  /** Make the polygon for use in hitTesting
    * Math.round() used to overcome problem
    * of rounding errors
  @see             Math.round() java.math
  */
  public void makePoly()
  {
    poly = new Polygon();

    double halfBase = triBase / 2.0;
    double halfHeight = triHeight / 2.0;

    x1 = center[0] + (basisJ[0] *
        halfHeight);
    y1 = center[1] + (basisJ[1] *
        halfHeight);
    x2 = center[0] - (basisJ[0] *
        halfHeight) + (basisI[0] *
        halfBase);
    y2 = center[1] - (basisJ[1] *
        halfHeight) + (basisI[1] *
        halfBase);
    x3 = center[0] - (basisJ[0] *
        halfHeight) - (basisI[0] *
        halfBase);
    y3 = center[1] - (basisJ[1] *
        halfHeight) - (basisI[1] *
        halfBase);

    poly.addPoint((int)Math.round(x1),
        (int)Math.round(y1));
    poly.addPoint((int)Math.round(x2),
        (int)Math.round(y2));
    poly.addPoint((int)Math.round(x3),
        (int)Math.round(y3));
  }

  /** Method for updating the shape after
    * it has been manipulated
  */
  public void updateShape()
  {
    makePoly();
  }

  /** Method for rotating the label
  */
  public void rotateLabel()
  {
    mirrorShapeY();
    isMirrored = !isMirrored;
  }

  /** Draw the label in the Graphics context
  @param g         The graphics context
  */
  public void drawLabel(Graphics g)
  {
    double d = labelD;
    double r = labelR;
    double x, y;
    double labelHeight = triHeight -
(triHeight / 4.0);

    x = center[0] - (basisJ[0] * (triHeight
      - labelHeight)) + (basisI[0] * d);
    y = center[1] - (basisJ[1] * (triHeight
      - labelHeight)) + (basisI[1] * d);
    g.fillOval((int)Math.round(x - r),
      (int)Math.round(y - r), (int)d,
      (int)d);
  }

  /** Overriding method for moving an
    * isoceles triangle's vertices
  */
  public void moveVertex(int i, int x,
       int y)
  {
    double dx = x - center[0];
    double dy = y - center[1];
    triHeight = Math.round(2.0 * ((dx *
       basisJ[0]) + (dy * basisJ[1])));

    if(i != 0)
    {
      triBase = Math.round(2.0 * ((dx *
        basisI[0]) + (dy * basisI[1])));
      triHeight = -triHeight;
      if(i == 2)
      {
        triBase = -triBase;
      }
    }
    if(triBase < MIN_SIZE * 2)
      triBase = MIN_SIZE * 2;
    if(triHeight < MIN_SIZE * 2)
      triHeight = MIN_SIZE * 2;

    updateShape();
  }

  /** Method for checking an isoceles
    * triangle's constraints
    * If the user turns the triangle into
    * equilateral triangle, the shape
    * perturbs back to an iso. triangle by
    * a predefined amount
  */
  public void finalizeVertex()
  {
    if(Math.abs(((Math.sqrt(3.0) / 2.0) *
       triBase) - triHeight) < SHAPE_SLOP
       * 5)
    {
      if(triBase > triHeight)
        triBase = triHeight + (2.0 /
       Math.sqrt(3.0)) * (SHAPE_SLOP * 5);
      else
        triHeight = triBase + (SHAPE_SLOP
        * 5);
      updateShape();
    }
  }
}
```

```
//Title:        Shaper2D
//Version:      2.1
//Copyright:    Copyright (c) 2001
//Author:       Miri McGill
//Company:      MIT
//Description:  Toolbar for selecting shapes
//              for the design
package Shaper2D;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;

public class ShapeBar extends JToolBar
implements Constants
{
  public DrawShapes parentDrawShapes;
  JButton rectButton, squareButton,
        eqTriButton, isoTriButton;

/* ------------------------
   Constructors
 * ------------------------ */

  public ShapeBar(DrawShapes parent)
  {
    parentDrawShapes = parent;
    parent.shapeBar = this;

    Icon rectIcon = new RectIcon();
    RectIcon disabledRectIcon = new
        RectIcon();
    disabledRectIcon.lineCol =
        COLOR_LINE_DISABLED;
    disabledRectIcon.fillCol =
        COLOR_FILL_DISABLED;
    rectButton = new JButton(rectIcon);
    rectButton.setDisabledIcon
        (disabledRectIcon);
    rectButton.addActionListener
        (new ActionListener()
    {
      public void actionPerformed
        (ActionEvent e)
      {
        parentDrawShapes.changeShapeOrig
        (DEFAULT_RECTANGLE.copy());
      }
    }
    );
    add(rectButton);

    Icon squareIcon = new SquareIcon();
    SquareIcon disabledSquareIcon = new
        SquareIcon();
    disabledSquareIcon.lineCol =
        COLOR_LINE_DISABLED;
    disabledSquareIcon.fillCol =
        COLOR_FILL_DISABLED;
    squareButton = new JButton(squareIcon);
    squareButton.setDisabledIcon
        (disabledSquareIcon);
    squareButton.addActionListener
        (new ActionListener()
    {
      public void actionPerformed
        (ActionEvent e)
      {
        parentDrawShapes.changeShapeOrig
        (DEFAULT_SQUARE.copy());
      }
    }
    );
    add(squareButton);

    this.addSeparator();

    Icon eqTriIcon = new EqTriIcon();
    EqTriIcon disabledEqTriIcon = new
        EqTriIcon();
    disabledEqTriIcon.lineCol =
        COLOR_LINE_DISABLED;
    disabledEqTriIcon.fillCol =
        COLOR_FILL_DISABLED;
    eqTriButton = new JButton(eqTriIcon);
    eqTriButton.setDisabledIcon
        (disabledEqTriIcon);
    eqTriButton.addActionListener
        (new ActionListener()
    {
      public void actionPerformed
        (ActionEvent e)
      {
        parentDrawShapes.changeShapeOrig
        (DEFAULT_EQTRIANGLE.copy());
      }
    }
    );
    add(eqTriButton);

    Icon isoTriIcon = new IsoTriIcon();
    IsoTriIcon disabledIsoTriIcon = new
        IsoTriIcon();
    disabledIsoTriIcon.lineCol =
        COLOR_LINE_DISABLED;
    disabledIsoTriIcon.fillCol =
        COLOR_FILL_DISABLED;
    isoTriButton = new JButton(isoTriIcon);
    isoTriButton.setDisabledIcon
        (disabledIsoTriIcon);
    isoTriButton.addActionListener
        (new ActionListener()
    {
      public void actionPerformed
        (ActionEvent e)
      {
        parentDrawShapes.changeShapeOrig
        (DEFAULT_ISOTRIANGLE.copy());
      }
    }
    );
    add(isoTriButton);

    this.addSeparator();
    setFloatable(false);
  }

/* ------------------------
   Icon Classes
 * ------------------------ */
```

```
abstract class ShapeIcon implements Icon
{
  public Color lineCol = Color.black;
  public Color fillCol = Color.white;

  public ShapeIcon()
  {
  }
  public int getIconWidth()
  {
    return 8;
  }
  public int getIconHeight()
  {
    return 8;
  }
  public double getIconSize()
  {
    return 6.0;
  }
}

/** Create the Rectangle Icon
*/
class RectIcon extends ShapeIcon
{
  public RectIcon()
  {
  }
  public void paintIcon (Component c,
      Graphics g, int x, int y)
  {
    g.setColor(fillCol);
    g.fillRect(x-2, y+1, getIconWidth()+1,
      getIconHeight()-3);
    g.setColor(lineCol);
    g.drawRect(x-2, y+1, getIconWidth()+1,
      getIconHeight()-3);
  }
}

/** Create the Square Icon
*/
class SquareIcon extends ShapeIcon
{
  public SquareIcon()
  {
  }
  public void paintIcon (Component c,
      Graphics g, int x, int y)
  {
    g.setColor(fillCol);
    g.fillRect(x-1, y, getIconWidth()-1,
      getIconHeight()-1);
    g.setColor(lineCol);
    g.drawRect(x-1, y, getIconWidth()-1,
      getIconHeight()-1);
  }
}

/** Create the Equilateral Triangle Icon
*/
class EqTriIcon extends ShapeIcon
{
  Polygon poly = new Polygon();

  public EqTriIcon()
  {
    makePoly();
  }
  public void makePoly()
  {
    double x1, y1, x2, y2, x3, y3;
    double center = getIconSize() / 2.0;
    double a = getIconSize() *
      ((Math.sqrt(3.0)) / 2.0);

    x1 = center;
    y1 = 0;
    x2 = center + a;
    y2 = center + getIconSize();
    x3 = center - a;
    y3 = center + getIconSize();

    poly.addPoint((int)Math.round(x1),
      (int)Math.round(y1));
    poly.addPoint((int)Math.round(x2),
      (int)Math.round(y2));
    poly.addPoint((int)Math.round(x3),
      (int)Math.round(y3));
  }
  public void paintIcon (Component c,
      Graphics g, int x, int y)
  {
    g.translate(x-1, y-2);
    g.setColor(fillCol);
    g.fillPolygon(poly);
    g.setColor(lineCol);
    g.drawPolygon(poly);
  }
}

/** Create the Isoceles Triangle Icon
*/
class IsoTriIcon extends ShapeIcon
{
  Polygon poly = new Polygon();

  public IsoTriIcon()
  {
    makePoly();
  }
  public void makePoly()
  {
    double x1, y1, x2, y2, x3, y3;
    double center = getIconSize() / 2.0;
    double a = getIconSize() *
      ((Math.sqrt(3.0)) / 2.0);

    x1 = center;
    y1 = 0;
    x2 = center + (a - 2);
    y2 = center + getIconSize();
    x3 = center - (a - 2);
    y3 = center + getIconSize();

    poly.addPoint((int)Math.round(x1),
      (int)Math.round(y1));
    poly.addPoint((int)Math.round(x2),
      (int)Math.round(y2));
    poly.addPoint((int)Math.round(x3),
        (int)Math.round(y3));
    }
    public void paintIcon (Component c,
        Graphics g, int x, int y)
    {
      g.translate(x-1, y-2);
      g.setColor(fillCol);
      g.fillPolygon(poly);
      g.setColor(lineCol);
      g.drawPolygon(poly);
    }
  }


  /** set the drawing window's status
  */
  public void setActive(boolean active)
  {
    squareButton.setEnabled(active);
    rectButton.setEnabled(active);
    eqTriButton.setEnabled(active);
    isoTriButton.setEnabled(active);
    repaint();
  }
}
```

```
//Title:        Shaper2D
//Version:      2.1
//Copyright:    Copyright (c) 2001
//Author:       Miri McGill
//Company:      MIT
//Description:  DXF File Filter
package Shaper2D;

import java.io.File;
import javax.swing.filechooser.FileFilter;

public class DXFFileFilter extends
       FileFilter implements Constants
{

  public DXFFileFilter()
  {
  }

  public String getDescription()
  {
    return "DXF files (*.dxf)";
  }

  public static String getExtension(File f)
  {
    if(f != null)
    {
      String fileName = f.getName();
      int i = fileName.lastIndexOf('.');
      if(i > 0 && i < fileName.length()-1)
        return fileName.substring(i +
       1).toLowerCase();
    }
    return null;
  }

  public static String getExtension
       (String s)
  {
    if(s != null)
    {
      int i = s.lastIndexOf('.');
      if(i > 0 && i < s.length()-1)
        return s.substring(i +
       1).toLowerCase();
    }
    return null;
  }

  public boolean accept(File file)
   {
    if(file.isDirectory())
      return true;
    String extension = getExtension(file);
    return extension != null &&
        (extension.compareToIgnoreCase(DXF)
        == 0);
  }
}
```

```
//Title:        Shaper2D
//Version:      2.1
//Copyright:    Copyright (c) 2001
//Author:       Miri McGill
//Company:      MIT
//Description:  Constants
package Shaper2D;

import java.awt.*;

public interface Constants
{
  public static final Dimension
        DEFAULT_DIMENSION =
        new Dimension(640, 480);
  public static final Dimension
        SCREEN_SIZE =
        Toolkit.getDefaultToolkit().
        getScreenSize();
  public static final int WIDTH = 100;
  public static final int HEIGHT = 100;
  public static final int MIN_SIZE = 10;
  public static final int SHAPE_SLOP = 5;
  public static final double HALF_CIRCLE
        = 180.0;
  public static final double ZOOM_SPEED =
        0.01;
  public static final Color COLOR_GRID_MAIN
        = new Color(190, 190, 190);
  public static final Color COLOR_GRID_MINOR
        = new Color(230, 230, 230);
  public static final Color COLOR_DISABLED =
        new Color(210, 210, 210);
  public static final Color
        COLOR_FILL_DISABLED = new Color(210,
        210, 210);
  public static final Color
        COLOR_LINE_DISABLED = new Color(140,
        140, 140);
  public static Dimension DEFAULT_SCREEN =
        new Dimension(640,480);
  public static String[] itersChoice = {
        "1", "2", "3", "4", "5", "6", "7",
        "8", "9", "10", "11", "12", "13",
        "14", "15", "16", "17", "18", "19",
        "20", "21", "22", "23", "24", "25" };
  public static String LAYER_SHAPE =
        "shape";
  public static String LAYER_LABEL =
        "labels";
  public static String DXF = "dxf";
  public static Shape DEFAULT_RECTANGLE
        = new MyRectangle(65, 90, 60, 30, 90
        * (Math.PI / 180.0));
  public static Shape DEFAULT_SQUARE  = new
        MySquare(60, 60, 40, 0);
  public static Shape DEFAULT_EQTRIANGLE =
        new MyEqTriangle(60, 60, 30, 180 *
        (Math.PI / 180.0));
  public static Shape DEFAULT_ISOTRIANGLE
        = new MyIsoTriangle(60, 60, 40, 60,
        180 * (Math.PI / 180.0));
  public static Shape DEFAULT_SCATRIANGLE =
        new MyScaTriangle(60, 60, 30, 180 *
        (Math.PI / 180.0));
  public static Cursor CURSOR_MOVESHAPE =
        new Cursor(Cursor.MOVE_CURSOR);
  public static Cursor CURSOR_RESIZESHAPE =
        new Cursor(Cursor.NW_RESIZE_CURSOR);
  public static Cursor CURSOR_ROTATESHAPE =
        new Cursor(Cursor.DEFAULT_CURSOR);
  public static Cursor CURSOR_MOVEITER =
        new Cursor(Cursor.HAND_CURSOR);
  public static Cursor CURSOR_ZOOMITER =
        new Cursor(Cursor.N_RESIZE_CURSOR);
  public static Cursor CURSOR_DEFAULTSHAPE =
        new Cursor(Cursor.DEFAULT_CURSOR);
}
```

```
//Title:        Shaper2D
//Version:      2.1
//Copyright:    Copyright (c) 2001
//Author:       Miri McGill
//Company:      MIT
//Description:  Main Shaper2D frame
package Shaper2D;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.borland.jbcl.layout.*;
import javax.swing.border.*;
import java.io.*;

public class Shaper2D extends JFrame
                       implements Constants
{
  public boolean pressed = false;
  public boolean released = false;
  BorderLayout borderLayout1 = new Border
        Layout();
  XYLayout xYLayout1 = new XYLayout();
  XYLayout xYLayout2 = new XYLayout();
  XYLayout xYLayout3 = new XYLayout();
  XYLayout xYLayout4 = new XYLayout();
  XYLayout xYLayout5 = new XYLayout();
  private DrawShapes childDrawShapes1,
        childDrawShapes2;
  private Rules childRule1, childRule2;
  private Iteration childIter;
  private MenuBar menuBar1 = new MenuBar();
  private Menu menuFile = new Menu("File");
  private MenuItem menuFileExit = new
        MenuItem("Exit");
  private Menu menuHelp = new Menu("Help");
  private MenuItem menuHelpRef = new
        MenuItem("Shaper2D Reference");
  private MenuItem menuHelpAbout = new
        MenuItem("About...");
  JPanel mainPanel = new JPanel();
  JPanel menuPanel = new JPanel();
  JButton newDesignButton = new JButton();
  JButton printDXFButton = new JButton();
  JButton resetButton = new JButton();
  JComboBox itersComboBox = new
        JComboBox(itersChoice);
  JCheckBox showLabels = new JCheckBox();
  JLabel labelIters = new JLabel();
  JLabel labelDraw1 = new JLabel();
  JLabel labelDraw2 = new JLabel();
  JLabel labelRule1 = new JLabel();
  JLabel labelRule2 = new JLabel();
  JLabel labelDesign = new JLabel();
  JLabel statusBar = new JLabel();
  JRadioButton rule1Radio = new
        JRadioButton();
  JRadioButton rule2Radio = new
        JRadioButton();
  ButtonGroup ruleGroup = new ButtonGroup();
  ShapeBar shapeBar1;
  ShapeBar shapeBar2;
  String currFileName = null;
  private File lastFile = null;
  JFileChooser fileChooser = new
        JFileChooser();
  DXFFileFilter fileFilter = new
        DXFFileFilter();

/* ------------------------
   Constructors
 * ------------------------ */
  public Shaper2D()
  {

enableEvents(AWTEvent.WINDOW_EVENT_MASK);
      try
      {
        jbInit();
      }
      catch(Exception e)
      {
        e.printStackTrace();
      }
  }

/* ------------------------
   Component initialization
 * ------------------------ */
  private void jbInit() throws Exception
  {
    this.getContentPane().setLayout
       (borderLayout1);
    this.getContentPane().add(mainPanel,
       BorderLayout.CENTER);
    this.setSize(new Dimension(980, 500));
    this.setLocation(150,50);
    this.setTitle("Shaper2D");
    this.setMenuBar(menuBar1);
    this.setResizable(false);
    this.setBackground(Color.white);
    mainPanel.setMinimumSize(new
       Dimension(975, 495));
    mainPanel.setPreferredSize(new
       Dimension(975, 495));
    mainPanel.setLayout(xYLayout2);
    menuPanel.setLayout(xYLayout1);
    menuPanel.setBorder(BorderFactory.
       createEtchedBorder());
    childDrawShapes1 = new DrawShapes();
    childDrawShapes2 = new DrawShapes();
    childRule1 = new
       Rules(childDrawShapes1);
    childRule2 = new
       Rules(childDrawShapes2);
    childIter = new
       Iteration(childDrawShapes1,
       childDrawShapes2);
    setIterate(childIter);
    fileChooser.addChoosableFileFilter
       (fileFilter);
    fileChooser.setFileFilter(fileFilter);
    shapeBar1 = new
       ShapeBar(childDrawShapes1);
    shapeBar2 = new
       ShapeBar(childDrawShapes2);
    shapeBar1.setOrientation
       (JToolBar.VERTICAL);
    shapeBar2.setOrientation
       (JToolBar.VERTICAL);
    showLabels.setHorizontalTextPosition
       (SwingConstants.LEFT);
    showLabels.setText("Labels");
    showLabels.addMouseListener(new
java.awt.event.MouseAdapter()
    {
      public void mouseReleased
        (MouseEvent e)
      {
        showLabels_mouseReleased(e);
      }
    });
    rule1Radio.setHorizontalTextPosition
        (SwingConstants.LEFT);
    rule1Radio.setText("One Rule");
    rule1Radio.setSelected(true);
    childIter.setNumRules(1);
    rule2Radio.setHorizontalTextPosition
        (SwingConstants.LEFT);
    rule2Radio.setText("Two Rules");
    ruleGroup.add(rule1Radio);
    ruleGroup.add(rule2Radio);
    rule1Radio.addActionListener(new
java.awt.event.ActionListener()
    {
      public void actionPerformed
        (ActionEvent e)
      {
        childIter.setNumRules(1);
      }
    });
    rule2Radio.addActionListener(new
java.awt.event.ActionListener()
    {
      public void actionPerformed
        (ActionEvent e)
      {
        childIter.setNumRules(2);
      }
    });
    itersComboBox.setBorder(BorderFactory.
        createLoweredBevelBorder());
    itersComboBox.setSelectedIndex(5);
    itersComboBox.addActionListener(new
java.awt.event.ActionListener()
    {
      public void actionPerformed
        (ActionEvent e)
      {
        itersComboBox_actionPerformed(e);
      }
    });
    menuBar1.add(menuFile);
    menuBar1.add(menuHelp);
    menuFile.add(menuFileExit);
    menuHelp.add(menuHelpRef);
    menuHelp.add(menuHelpAbout);
    menuFileExit.addActionListener(new
java.awt.event.ActionListener()
    {
      public void actionPerformed
        (ActionEvent e)
      {
        fileExit_actionPerformed(e);
      }
    });
    menuHelpRef.addActionListener(new
java.awt.event.ActionListener()
    {
      public void actionPerformed
        (ActionEvent e)
      {
        helpRef_actionPerformed(e);
      }
    });
    menuHelpAbout.addActionListener(new
java.awt.event.ActionListener()
    {
      public void actionPerformed
        (ActionEvent e)
      {
        helpAbout_actionPerformed(e);
      }
    });
    newDesignButton.setMargin
        (new Insets(1, 3, 1, 3));
    newDesignButton.setText("New Design");
    newDesignButton.addActionListener(new
java.awt.event.ActionListener()
    {
      public void actionPerformed
        (ActionEvent e)
      {
        newDesignButton_actionPerformed(e);
      }
    });
    resetButton.setMargin
        (new Insets(1, 3, 1, 3));
    resetButton.setText("Reset Panel");
    resetButton.addActionListener(new
java.awt.event.ActionListener()
    {
      public void actionPerformed
        (ActionEvent e)
      {
        resetButton_actionPerformed(e);
      }
    });
    printDXFButton.setMargin
        (new Insets(1, 3, 1, 3));
    printDXFButton.setText
        ("Save Design as DXF");
    printDXFButton.addActionListener
        (new java.awt.event.ActionListener()
    {
      public void actionPerformed
        (ActionEvent e)
      {
        printDXFButton_actionPerformed(e);
      }
    });
    this.addKeyListener(new
java.awt.event.KeyAdapter()
    {
      public void keyReleased(KeyEvent e)
      {
        this_keyReleased(e);
      }
      public void keyPressed(KeyEvent e)
      {
        this_keyPressed(e);
      }
    });
    this.addMouseListener(new
java.awt.event.MouseAdapter()
{
  public void mousePressed
    (MouseEvent e) {
    this_mousePressed(e);
  }
  public void mouseReleased
    (MouseEvent e) {
    this_mouseReleased(e);
  }
});
labelIters.setText("# Iterations");
labelDraw1.setFont
    (new java.awt.Font
    ("Dialog", 1, 12));
labelDraw1.setText
    ("Spatial Relation 1");
labelDraw2.setFont(new java.awt.Font
    ("Dialog", 1, 12));
labelDraw2.setText
    ("Spatial Relation 2");
labelRule1.setFont
    (new java.awt.Font
    ("Dialog", 1, 12));
labelRule1.setText("Rule 1");
labelRule2.setFont(new java.awt.Font
    ("Dialog", 1, 12));
labelRule2.setText("Rule 2");
labelDesign.setFont(new java.awt.Font
    ("Dialog", 1, 12));
labelDesign.setText("Design");
mainPanel.add(menuPanel, new
    XYConstraints(5, 0, 966, -1));
menuPanel.add(showLabels, new
    XYConstraints(898, 4, 60, 17));
mainPanel.add(labelDesign,
    new XYConstraints(665, 32, -1, -1));
mainPanel.add(labelDraw1, new
    XYConstraints(7, 32, -1, -1));
mainPanel.add(labelDraw2, new
    XYConstraints(7, 252, -1, -1));
mainPanel.add(labelRule1, new
    XYConstraints(245, 32, -1, -1));
mainPanel.add(labelRule2, new
    XYConstraints(245, 252, -1, -1));
mainPanel.add(childDrawShapes1,
    new XYConstraints(35, 50, -1, -1));
mainPanel.add(childDrawShapes2, new
    XYConstraints(35, 270, -1, -1));
mainPanel.add(childRule1, new
    XYConstraints(245, 50, 410, -1));
mainPanel.add(childRule2, new
    XYConstraints(245, 270, 410, -1));
mainPanel.add(childIter, new
    XYConstraints(665, 50, 305, 420));
mainPanel.add(shapeBar1, new
    XYConstraints(5, 50, 25, 200));
mainPanel.add(shapeBar2, new
    XYConstraints(5, 270, 25, 200));
menuPanel.add(newDesignButton, new
    XYConstraints(2, 0, -1, -1));
menuPanel.add(resetButton, new
    XYConstraints(656, 0, -1, -1));
menuPanel.add(printDXFButton, new
    XYConstraints(88, 0, -1, -1));
menuPanel.add(labelIters, new
    XYConstraints(765, 5, 63, 16));
```

```
    menuPanel.add(itersComboBox, new
        XYConstraints(836, 0, 50, -1));
    menuPanel.add(rule1Radio, new
        XYConstraints(238, 5, 74, 15));
    menuPanel.add(rule2Radio, new
        XYConstraints(318, 5, 80, 15));
    childDrawShapes1.setBorder
        (BorderFactory.
        createEtchedBorder());
    childDrawShapes1.setPreferredSize(new
        Dimension(200, 200));
    childDrawShapes1.setLayout(xYLayout3);
    childDrawShapes1.setBackground
        (Color.white);
    childDrawShapes2.setBorder
        (BorderFactory.
        createEtchedBorder());
    childDrawShapes2.setPreferredSize(new
        Dimension(200, 200));
    childDrawShapes2.setLayout(xYLayout3);
    childDrawShapes2.setBackground
        (Color.white);
    childRule1.setBorder(BorderFactory.
        createEtchedBorder());
    childRule1.setPreferredSize(new
        Dimension(245, 200));
    childRule1.setLayout(xYLayout4);
    childRule2.setBorder(BorderFactory.
        createEtchedBorder());
    childRule2.setPreferredSize(new
        Dimension(245, 200));
    childRule2.setLayout(xYLayout4);
    childIter.setBackground(Color.white);
    childIter.setBorder(BorderFactory.
        createEtchedBorder());
    childIter.setSize(new Dimension
        (305, 420));
    childIter.setLayout(xYLayout5);
    childIter.resetOrigin();
    resetDrawing();
  }

  /** set the number of iterations
  */
  public void setNIters(int nIters){
    childIter.setIters(nIters);
  }

  /** set instance of Iteration to childIter
  */
  public void setIterate(Iteration iter) {
    childIter = iter;
  }

/* ------------------------
   Event handlers
 * ------------------------ */
  public void showLabels_mouseReleased
       (MouseEvent e)
  {
    childIter.showLabels();
  }
  void this_mousePressed(MouseEvent e) {
  }
  void this_mouseReleased(MouseEvent e){
  }
  void itersComboBox_actionPerformed
        (ActionEvent e) {
    int choice;
    JComboBox cb = (JComboBox)e.getSource();
    choice = cb.getSelectedIndex() + 2;
    setNIters(choice);
    repaint();
  }

  void newDesignButton_actionPerformed
        (ActionEvent e) {
    if(okToAbandon())
    {
      resetDrawing();
      childIter.resetOrigin();
      currFileName = null;
    }
  }

  void resetButton_actionPerformed
        (ActionEvent e)
  {
    childIter.resetOrigin();
    repaint();
  }

  void printDXFButton_actionPerformed
        (ActionEvent e)
  {
    saveFile();
  }

/* ------------------------
   Methods
 * ------------------------ */
  public void resetDrawing()
  {
    childDrawShapes1.shapeOrig =
        DEFAULT_RECTANGLE.copy();
    childDrawShapes1.shapeDest =
        DEFAULT_EQTRIANGLE.copy();
    childDrawShapes1.shapeDest.
        translateShape(50.0, 45.0);
    childDrawShapes2.shapeOrig =
        DEFAULT_EQTRIANGLE.copy();
    childDrawShapes2.shapeDest =
        DEFAULT_RECTANGLE.copy();
    childDrawShapes2.shapeDest.
        translateShape(50.0, 40.0);
    childDrawShapes1.updateDrawing();
    childDrawShapes2.updateDrawing();
    childIter.setNumRules(1);
    rule1Radio.setSelected(true);
  }

  boolean okToAbandon()
  {
    int value = JOptionPane.
        showConfirmDialog(this,
        "Are you sure you want to start a
        new design?",
        "Shaper 2D",
        JOptionPane.YES_NO_OPTION);

    switch (value)
    {
      case JOptionPane.YES_OPTION:
        // yes, please start new design
        return true;
      case JOptionPane.NO_OPTION:
        // no, abandon edits
        return false;
      default:
        // cancel
        return false;
    }
  }

boolean okToOverwrite()
{
  int value = JOptionPane.
      showConfirmDialog(this,
      "Are you sure you want to overwrite
      this file?",
      "Shaper 2D",
      JOptionPane.YES_NO_OPTION);

  switch (value)
  {
    case JOptionPane.YES_OPTION:
      return true;
    case JOptionPane.NO_OPTION:
      return false;
    default:     // cancel
      return false;
  }
}

/** File | Exit action performed
*/
public void fileExit_actionPerformed
      (ActionEvent e)
{
  System.exit(0);
}

/** Help | About action performed
*/
public void helpAbout_actionPerformed
      (ActionEvent e)
{
  Shaper2D_AboutBox dlg = new
      Shaper2D_AboutBox(this);
  Dimension dlgSize =
      dlg.getPreferredSize();
  Dimension frmSize = getSize();
  Point loc = getLocation();
  dlg.setLocation((frmSize.width -
      dlgSize.width) / 2 + loc.x,
                 (frmSize.height -
      dlgSize.height) / 2 + loc.y);
  dlg.setModal(true);
  dlg.show();
}

/** Help | Reference action performed
*/
public void helpRef_actionPerformed
      (ActionEvent e)
{
  Shaper2D_HelpBox dlg = new
```

```
      Shaper2D_HelpBox(this);
   Dimension dlgSize =
      dlg.getPreferredSize();
   Dimension frmSize = getSize();
   Point loc = getLocation();
   dlg.setLocation((frmSize.width -
      dlgSize.width) / 2 + loc.x,
                 (frmSize.height -
      dlgSize.height) / 2 + loc.y);
   dlg.setModal(true);
   dlg.show();
 }

 /** Saves current file;
   * handles not yet having a filename;
   * reports to statusBar
 */
 boolean saveFile()
 {
     if(lastFile != null)
       fileChooser.setSelectedFile
       (lastFile);
     if (JFileChooser.APPROVE_OPTION ==
       fileChooser.showSaveDialog(this))
     {
       lastFile =
       fileChooser.getSelectedFile();
       currFileName = lastFile.getPath();
       if(fileFilter.getExtension
       (currFileName) == null)
         {
           currFileName =
              currFileName.concat(".dxf");
           lastFile = new
              File(currFileName);
         }
       if(lastFile.exists() &&
              (okToOverwrite() == false))
       {
         this.repaint();
         return false;
       }
       childIter.printDXF(currFileName);
       updateCaption();
       this.repaint();    //repaints menu
       return true;
     }
   return false;
 }

 /** Updates the caption of the application
   * to show the filename.
 */
 void updateCaption()
 {
   String caption;
   if(currFileName == null)
   {
     caption = "Untitled";
       // synthesize the "Untitled" name
       // if no name yet.
   }
   else
   {
     caption = currFileName;
   }
     caption = "Shaper2D " + caption;
     this.setTitle(caption);
   }

   /** Overridden so we can exit on System
     * Close
   */
   protected void processWindowEvent
         (WindowEvent e)
   {
     super.processWindowEvent(e);
     if(e.getID() ==
        WindowEvent.WINDOW_CLOSING)
        fileExit_actionPerformed(null);
   }
   void this_keyReleased(KeyEvent e)
   {
     pressed = false;
     released = true;
   }
   void this_keyPressed(KeyEvent e)
   {
     pressed = true;
     released = false;
   }
 }
```

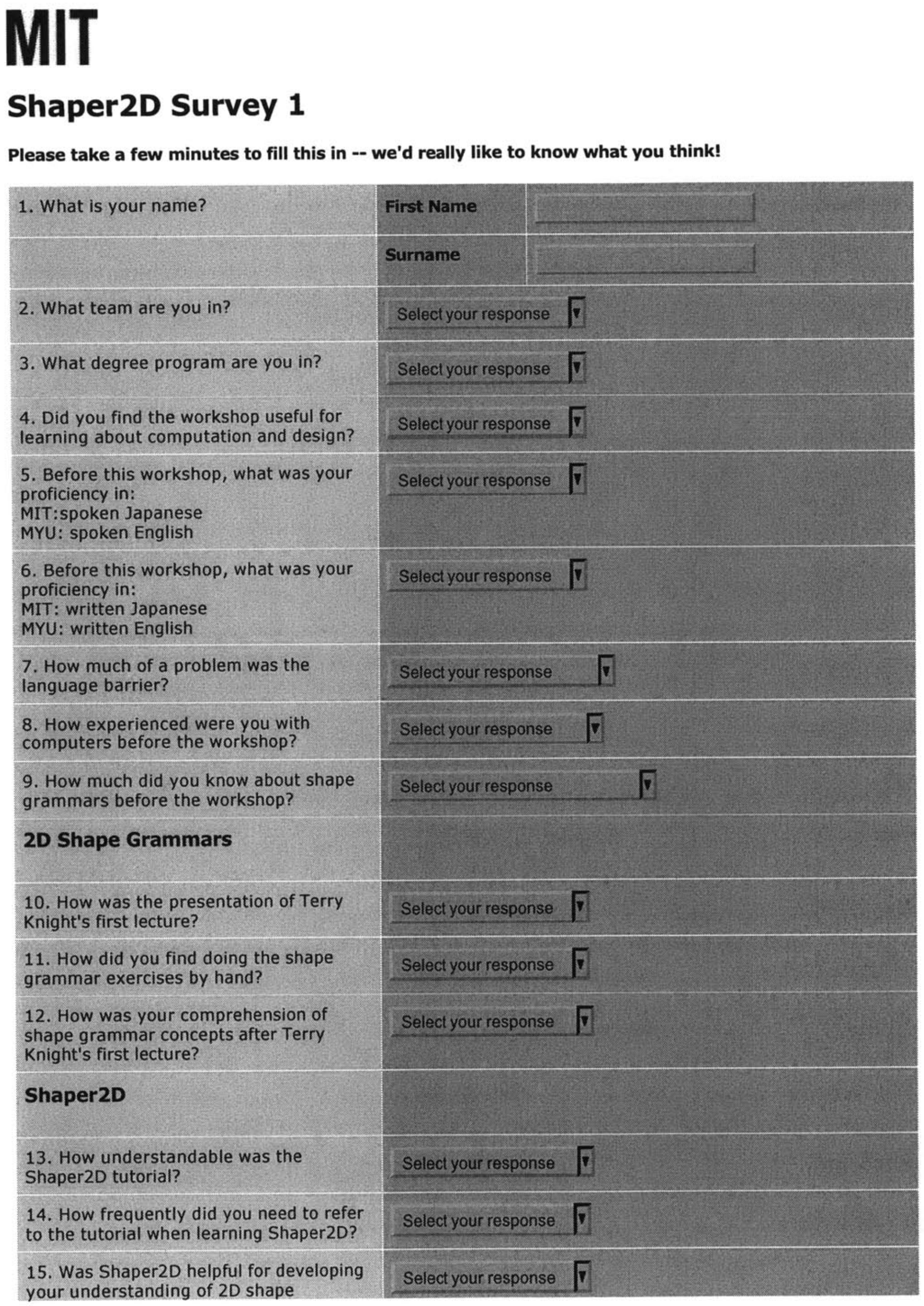
```
//Title:        Shaper2D
//Version:      2.1
//Copyright:    Copyright (c) 2001
//Author:       Miri McGill
//Company:      MIT
//Description:  The Shaper2D Applet
package Shaper2D;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.borland.jbcl.layout.*;
import javax.swing.border.*;
import java.io.*;

public class Shaper2DApplet extends JApplet
       implements Constants
{
  boolean isStandalone = false;
  boolean pressed = false;
  boolean released = false;

  BorderLayout borderLayout1 = new
       BorderLayout();
  XYLayout xYLayout1 = new XYLayout();
  XYLayout xYLayout2 = new XYLayout();
  XYLayout xYLayout4 = new XYLayout();
  XYLayout xYLayout5 = new XYLayout();
  XYLayout xYLayout6 = new XYLayout();

  DrawShapes childDrawShapes1;
  DrawShapes childDrawShapes2;
  Rules childRule1;
  Rules childRule2;
  Iteration childIter;

  JPanel mainPanel = new JPanel();
  JPanel menuPanel1 = new JPanel();
  JButton newDesignButton = new JButton();
  JButton resetButton = new JButton();
  JComboBox itersComboBox = new
       JComboBox(itersChoice);
  JCheckBox showLabels = new JCheckBox();
  JLabel labelIters = new JLabel();
  JLabel labelDraw1 = new JLabel();
  JLabel labelDraw2 = new JLabel();
  JLabel labelRule1 = new JLabel();
  JLabel labelRule2 = new JLabel();
  JLabel labelDesign = new JLabel();
  JRadioButton rule1Radio = new
       JRadioButton();
  JRadioButton rule2Radio = new
       JRadioButton();
  ButtonGroup ruleGroup = new ButtonGroup();

  ShapeBar shapeBar1;
  ShapeBar shapeBar2;

  String product = "Shaper2D";
  String version = "version 2.1";
  String copyright = "Copyright (c) 2001,
       MIT & Miri McGill";
  String comments = "A Simple Intro
       to Shape Grammars";

  String currFileName = null;
  File lastFile = null;

  DXFFileFilter fileFilter = new
       DXFFileFilter();

  //Get a parameter value
  public String getParameter(String key,
       String def)
  {
    return isStandalone ?
       System.getProperty(key, def) :
      (getParameter(key) != null ?
       getParameter(key) : def);
  }

/* ------------------------
   Constructors
 * ------------------------ */

 /** Construct the applet
 */
  public Shaper2DApplet()
  {
  }

  /** Initialize the applet
  */
  public void init()
  {
    try
    {
      jbInit();
    }
    catch(Exception e)
    {
      e.printStackTrace();
    }
  }

/* ------------------------
   Component initialization
 * ------------------------ */

  private void jbInit() throws Exception
  {
    this.getContentPane().setLayout
        (borderLayout1);
    this.getContentPane().add(mainPanel,
       BorderLayout.CENTER);
    this.setSize(new Dimension(980, 475));
    this.setName("Shaper2D");
    this.setBackground(Color.white);

    mainPanel.setMinimumSize(new
       Dimension(975, 475));
    mainPanel.setPreferredSize(new
       Dimension(975, 475));
    mainPanel.setLayout(xYLayout1);
    menuPanel1.setBorder(BorderFactory.
       createEtchedBorder());
    menuPanel1.setMinimumSize(new
       Dimension(975, 30));
    menuPanel1.setPreferredSize(new
       Dimension(975, 30));
    menuPanel1.setLayout(xYLayout2);
```

```
childDrawShapes1 = new DrawShapes();
childDrawShapes2 = new DrawShapes();
childRule1 = new
   Rules(childDrawShapes1);
childRule2 = new
   Rules(childDrawShapes2);
childIter = new
   Iteration(childDrawShapes1,
   childDrawShapes2);
setIterate(childIter);

childDrawShapes1.setBackground
   (Color.white);
childDrawShapes2.setBackground
   (Color.white);
childRule1.setBackground(Color.white);
childRule2.setBackground(Color.white);
childIter.setBackground(Color.white);

shapeBar1 = new
   ShapeBar(childDrawShapes1);
shapeBar2 = new
   ShapeBar(childDrawShapes2);
shapeBar1.setOrientation
   (JToolBar.VERTICAL);
shapeBar2.setOrientation
   (JToolBar.VERTICAL);

showLabels.setHorizontalTextPosition
   (SwingConstants.LEFT);
showLabels.setText("Labels");
showLabels.addMouseListener
   (new java.awt.event.MouseAdapter()
{
  public void mouseReleased
   (MouseEvent e)
  {
    showLabels_mouseReleased(e);
  }
});

this.addKeyListener
   (new java.awt.event.KeyAdapter()
{
  public void keyReleased(KeyEvent e)
  {
    this_keyReleased(e);
  }

  public void keyPressed(KeyEvent e)
  {
    this_keyPressed(e);
  }
});

rule1Radio.setText("One Rule");
rule1Radio.setHorizontalTextPosition
   (SwingConstants.LEFT);
rule1Radio.setSelected(true);
childIter.setNumRules(1);
rule2Radio.setText("Two Rules");
rule2Radio.setHorizontalTextPosition
   (SwingConstants.LEFT);
ruleGroup.add(rule1Radio);
ruleGroup.add(rule2Radio);
rule1Radio.addActionListener
   (new java.awt.event.ActionListener()
{
  public void actionPerformed
   (ActionEvent e)
  {
   childIter.setNumRules(1);
  }
});
rule2Radio.addActionListener
   (new java.awt.event.ActionListener()
{
  public void actionPerformed
   (ActionEvent e)
  {
   childIter.setNumRules(2);
  }
});

itersComboBox.setBorder(BorderFactory.
   createLoweredBevelBorder());
itersComboBox.setSelectedIndex(5);
itersComboBox.addActionListener
   (new java.awt.event.ActionListener()
{
  public void actionPerformed
   (ActionEvent e)
  {
   itersComboBox_actionPerformed(e);
  }
});

newDesignButton.setText("New Design");
newDesignButton.setMargin(new Insets(1,
   3, 1, 3));
newDesignButton.addActionListener
   (new java.awt.event.ActionListener()
{
  public void actionPerformed
   (ActionEvent e)
  {
   newDesignButton_actionPerformed(e);
  }
});
resetButton.setText("Reset Panel");
resetButton.setMargin(new Insets(1, 3,
   1, 3));
resetButton.addActionListener
   (new java.awt.event.ActionListener()
{
  public void actionPerformed
   (ActionEvent e)
  {
   resetButton_actionPerformed(e);
  }
});

this.addKeyListener
   (new java.awt.event.KeyAdapter()
{
  public void keyReleased(KeyEvent e)
  {
   this_keyReleased(e);
  }

  public void keyPressed(KeyEvent e)
  {
    this_keyPressed(e);
  }
});

labelDraw1.setFont(new
    java.awt.Font("Dialog", 1, 12));
labelDraw1.setText("Spatial Relation
    1");
labelDraw2.setFont(new
    java.awt.Font("Dialog", 1, 12));
labelDraw2.setText("Spatial Relation
    2");
labelRule1.setFont(new
    java.awt.Font("Dialog", 1, 12));
labelRule1.setText("Rule 1");
labelRule2.setFont(new
    java.awt.Font("Dialog", 1, 12));
labelRule2.setText("Rule 2");
labelDesign.setFont(new
    java.awt.Font("Dialog", 1, 12));
labelDesign.setText("Design");
labelIters.setText("# Iterations");

mainPanel.add(menuPanel1, new
    XYConstraints(5, 0, 975, -1));
mainPanel.add(labelDraw1, new
    XYConstraints(7, 32, -1, -1));
mainPanel.add(labelDraw2, new
    XYConstraints(7, 252, -1, -1));
mainPanel.add(labelRule1, new
    XYConstraints(245, 32, -1, -1));
mainPanel.add(labelRule2, new
    XYConstraints(245, 252, -1, -1));
mainPanel.add(childDrawShapes1, new
    XYConstraints(35, 50, -1, -1));
mainPanel.add(childDrawShapes2, new
    XYConstraints(35, 270, -1, -1));
mainPanel.add(childRule1, new
    XYConstraints(245, 50, 410, -1));
mainPanel.add(childRule2, new
    XYConstraints(245, 270, 410, -1));
mainPanel.add(shapeBar1, new
    XYConstraints(5, 50, 25, 200));
mainPanel.add(shapeBar2, new
    XYConstraints(5, 270, 25, 200));
mainPanel.add(labelDesign, new
    XYConstraints(665, 32, -1, -1));
mainPanel.add(childIter, new
    XYConstraints(665, 50, 310, 420));
menuPanel1.add(newDesignButton, new
    XYConstraints(2, 0, -1, -1));
menuPanel1.add(rule1Radio, new
    XYConstraints(238, 6, 74, 15));
menuPanel1.add(rule2Radio, new
    XYConstraints(318, 6, 80, 15));
menuPanel1.add(showLabels, new
    XYConstraints(908, 5, 60, 17));
menuPanel1.add(itersComboBox, new
    XYConstraints(850, 1, 50, -1));
menuPanel1.add(labelIters, new
    XYConstraints(778, 5, 63, 16));
menuPanel1.add(resetButton, new
    XYConstraints(655, 1, -1, -1));

childDrawShapes1.setLayout(xYLayout4);
childDrawShapes1.setPreferredSize(new
Dimension(200, 200));
    childDrawShapes1.setBorder
        (BorderFactory.
        createEtchedBorder());
    childDrawShapes2.setLayout(xYLayout4);
    childDrawShapes2.setPreferredSize(new
        Dimension(200, 200));
    childDrawShapes2.setBorder
        (BorderFactory.
        createEtchedBorder());
    childRule1.setBorder
        (BorderFactory.
        createEtchedBorder());
    childRule1.setPreferredSize(new
        Dimension(245, 200));
    childRule1.setLayout(xYLayout5);
    childRule2.setBorder
        (BorderFactory.
        createEtchedBorder());
    childRule2.setPreferredSize(new
        Dimension(245, 200));
    childRule2.setLayout(xYLayout5);
    childIter.setSize(new Dimension(310,
        420));
    childIter.setLayout(xYLayout6);
    childIter.resetOrigin();

    resetDrawing();
  }

/* ------------------------
   Methods
 * ------------------------ */

  public void resetDrawing()
  {
    childDrawShapes1.shapeOrig =
        DEFAULT_RECTANGLE.copy();
    childDrawShapes1.shapeDest =
        DEFAULT_EQTRIANGLE.copy();
    childDrawShapes1.shapeDest.
        translateShape(50.0, 45.0);
    childDrawShapes2.shapeOrig =
        DEFAULT_EQTRIANGLE.copy();
    childDrawShapes2.shapeDest =
        DEFAULT_RECTANGLE.copy();
    childDrawShapes2.shapeDest.
        translateShape(50.0, 40.0);
    childDrawShapes1.updateDrawing();
    childDrawShapes2.updateDrawing();
    childIter.setNumRules(1);
    rule1Radio.setSelected(true);
  }

  boolean okToAbandon()
  {
    int value = JOptionPane.
        showConfirmDialog(this, "Are you
        sure you want to start a new
        design?", "Shaper 2D",
        JOptionPane.YES_NO_OPTION);
    switch (value)
    {
      case JOptionPane.YES_OPTION:
        // yes, please start new design
        return true;
      case JOptionPane.NO_OPTION:
        // no, abandon edits
        return false;

      default:
        // cancel
        return false;
    }
  }

  boolean okToOverwrite()
  {
    int value =
        JOptionPane.showConfirmDialog(this,
        "Are you sure you want to overwrite
        this file?", "Shaper 2D",
        JOptionPane.YES_NO_OPTION);
    switch (value)
    {
      case JOptionPane.YES_OPTION:
        return true;
      case JOptionPane.NO_OPTION:
        return false;
      default:     // cancel
        return false;
    }
  }

  public void about()
  {
    int value =
        JOptionPane.showConfirmDialog(this,
        "Shaper2D - A Simple Introduction to
        Shape Grammars, by Miranda McGill.
        Copyright 2001", "About",
        JOptionPane.CLOSED_OPTION);
  }

  // Saves current file; handles not yet
  // having a filename; reports to
  // statusBar.
  boolean saveFile()
  {
    return false;
  }

/** set the number of iterations
*/

  public void setNIters(int nIters)
  {
    childIter.setIters(nIters);
  }

/** set the instance of Iteration to
childIter
*/
  public void setIterate(Iteration iter)
  {
    childIter = iter;
  }

/* ------------------------
   Applet stuff
 * ------------------------ */

  /** Start the applet
  */
  public void start()
  {
  }

  /** Stop the applet
  */
  public void stop()
  {
  }

  /** Destroy the applet
  */
  public void destroy()
  {
  }

  /** Get Applet information
  */
  public String getAppletInfo()
  {
    return "Applet Information";
  }

  /** Get parameter info
  */
  public String[][] getParameterInfo()
  {
    return null;
  }
  // static initializer for setting look
  // & feel
  static {
    try
    {
      UIManager.setLookAndFeel(UIManager.
        getSystemLookAndFeelClassName());
    }
    catch (Exception e) {}
  }

/* ------------------------
   Event handlers
 * ------------------------ */

  void showLabels_mouseReleased
        (MouseEvent e)
  {
     childIter.showLabels();
  }

  void newDesignButton_actionPerformed
        (ActionEvent e)
  {
    if(okToAbandon())
    {
      resetDrawing();
      childIter.resetOrigin();
      currFileName = null;
    }
  }
  void printDXFButton_actionPerformed
        (ActionEvent e)
  {
    saveFile();
  }

  void resetButton_actionPerformed
        (ActionEvent e)
  {
    childIter.resetOrigin();
    repaint();
  }

  void itersComboBox_actionPerformed
        (ActionEvent e)
  {
    int choice;
    JComboBox cb = (JComboBox)e.getSource();
    choice = cb.getSelectedIndex() + 2;
    setNIters(choice);
    repaint();
  }

  void this_keyReleased(KeyEvent e)
  {
    pressed = false;
    released = true;
  }

  void this_keyPressed(KeyEvent e)
  {
    pressed = true;
    released = false;
  }
}
```

```
//Title:        Shaper2D
//Version:      2.1
//Copyright:    Copyright (c) 2001
//Author:       Miri McGill
//Company:      MIT
//Description:  The Shaper2D Application
package Shaper2D;

import javax.swing.UIManager;
import java.awt.*;

public class Shaper2DApplication implements
       Constants
{
  Shaper2D shaper2D;
  DrawShapes drawShapes;
  Iteration iter;
  Rules rules;
  boolean packFrame = false;

/* ------------------------
   Constructors
 * ------------------------ */

  public Shaper2DApplication()
  {
    Shaper2D shaper2D = new Shaper2D();
    Dimension shaper2DSize =
        shaper2D.getSize();
    // Validate frames with preset sizes
    // Pack frames with useful preferred
    // size info, e.g. from their layout
    if (packFrame)
    {
      shaper2D.pack();
    }
    else
    {
      shaper2D.validate();
    }

    shaper2D.setLocation((SCREEN_SIZE.width
        - shaper2DSize.width) / 2,
        (SCREEN_SIZE.height -
        shaper2DSize.height) / 2);
    shaper2D.setVisible(true);
  }

/* ------------------------
   Main Method
 * ------------------------ */
  public static void main(String[] args)
  {
    try
    {
      UIManager.setLookAndFeel(UIManager.
        getSystemLookAndFeelClassName());
    }
    catch(Exception e)
    {
    }
    new Shaper2DApplication();
  }
}
```

```
//Title:        Shaper2D
//Version:      2.1
//Copyright:    Copyright (c) 2001
//Author:       Miri McGill
//Company:      MIT
//Description:  About Box
package Shaper2D;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;

public class Shaper2D_AboutBox extends
       JDialog implements ActionListener
{
  JPanel panel1 = new JPanel();
  JPanel panel2 = new JPanel();
  JPanel insetsPanel1 = new JPanel();
  JPanel insetsPanel2 = new JPanel();
  JPanel insetsPanel3 = new JPanel();
  JButton button1 = new JButton();
  JLabel imageControl1 = new JLabel();
  ImageIcon imageIcon;
  JLabel label1 = new JLabel();
  JLabel label2 = new JLabel();
  JLabel label3 = new JLabel();
  JLabel label4 = new JLabel();
  BorderLayout borderLayout1 = new
       BorderLayout();
  BorderLayout borderLayout2 = new
       BorderLayout();
  FlowLayout flowLayout1 = new FlowLayout();
  FlowLayout flowLayout2 = new FlowLayout();
  GridLayout gridLayout1 = new GridLayout();
  String product = "Shaper2D";
  String version = "version 2.1";
  String copyright = "Copyright (c) 2001,
       Miri McGill";
  String comments = "A Simple Intro to
       Shape Grammars";

/* ------------------------
   Constructors
 * ------------------------ */

  public Shaper2D_AboutBox(Frame parent)
  {
    super(parent);
    enableEvents
       (AWTEvent.WINDOW_EVENT_MASK);
    try
    {
      jbInit();
    }
    catch(Exception e)
    {
      e.printStackTrace();
    }
    //imageControl1.setIcon(imageIcon);
    pack();
  }

/* ------------------------
   Component initialization
 * ------------------------ */

  private void jbInit() throws Exception
  {
    this.setTitle("About");
    setResizable(false);
    panel1.setLayout(borderLayout1);
    panel2.setLayout(borderLayout2);
    insetsPanel1.setLayout(flowLayout1);
    insetsPanel2.setLayout(flowLayout1);
    insetsPanel2.setBorder(new
       EmptyBorder(10, 10, 10, 10));
    gridLayout1.setRows(4);
    gridLayout1.setColumns(1);
    label1.setText(product);
    label2.setText(version);
    label3.setText(copyright);
    label4.setText(comments);
    insetsPanel3.setLayout(gridLayout1);
    insetsPanel3.setBorder(new
       EmptyBorder(10, 60, 10, 10));
    button1.setText("OK");
    button1.addActionListener(this);
    insetsPanel2.add(imageControl1, null);
    panel2.add(insetsPanel2,
       BorderLayout.WEST);
    this.getContentPane().add(panel1, null);
    insetsPanel3.add(label1, null);
    insetsPanel3.add(label2, null);
    insetsPanel3.add(label3, null);
    insetsPanel3.add(label4, null);
    panel2.add(insetsPanel3,
       BorderLayout.CENTER);
    insetsPanel1.add(button1, null);
    panel1.add(insetsPanel1,
       BorderLayout.SOUTH);
    panel1.add(panel2, BorderLayout.NORTH);
  }

/* ------------------------
   Methods
 * ------------------------ */

  protected void processWindowEvent
       (WindowEvent e)
  {
    if(e.getID() ==
       WindowEvent.WINDOW_CLOSING)
    {
      cancel();
    }
    super.processWindowEvent(e);
  }

  void cancel()
  {
    dispose();
  }

  public void actionPerformed(ActionEvent e)
  {
    if(e.getSource() == button1)
    {
      cancel();
    }
  }
}
```

```
//Title:        Shaper2D
//Version:      2.1
//Copyright:    Copyright (c) 2000
//Author:       Miri McGill
//Company:      MIT
//Description:  Help Box
package Shaper2D;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
import com.borland.jbcl.layout.*;

public class Shaper2D_HelpBox extends
       JDialog implements ActionListener
{
  JPanel panel1 = new JPanel();
  JPanel insetsPanel1 = new JPanel();
  JButton button1 = new JButton();
  BorderLayout borderLayout1 = new
       BorderLayout();
  FlowLayout flowLayout1 = new FlowLayout();
  GridLayout gridLayout1 = new GridLayout();
  JPanel jPanel1 = new JPanel();
  XYLayout xYLayout1 = new XYLayout();
  JLabel jLabel1 = new JLabel();
  JLabel jLabel2 = new JLabel();
  JLabel jLabel3 = new JLabel();
  JLabel jLabel4 = new JLabel();

/* ------------------------
   Constructors
 * ------------------------ */

  public Shaper2D_HelpBox(Frame parent)
  {
    super(parent);
    enableEvents
       (AWTEvent.WINDOW_EVENT_MASK);
    try
    {
      jbInit();
    }
    catch(Exception e)
    {
      e.printStackTrace();
    }
    pack();
  }
/* ------------------------
   Component initialization
 * ------------------------ */

  void jbInit() throws Exception
  {
    this.setTitle("Shaper2D Reference");
    setResizable(false);
    panel1.setLayout(borderLayout1);
    insetsPanel1.setLayout(flowLayout1);
    gridLayout1.setRows(4);
    gridLayout1.setColumns(1);
    button1.setText("OK");
    button1.addActionListener(this);
    panel1.setMinimumSize(new Dimension(350,
        350));
    panel1.setPreferredSize(new
       Dimension(350, 350));
    jPanel1.setLayout(xYLayout1);
    jPanel1.setBackground(Color.white);
    jPanel1.setBorder(BorderFactory.
       createEtchedBorder());
    jPanel1.setMinimumSize(new
       Dimension(350, 300));
    jPanel1.setPreferredSize(new
       Dimension(350, 300));
    jLabel1.setFont(new
       java.awt.Font("Dialog", 1, 14));
    jLabel1.setToolTipText("");
    jLabel1.setHorizontalTextPosition
       (SwingConstants.LEFT);
    jLabel1.setText("How to use Shaper2D");
    jLabel2.setText("Sketch Pane");
    jLabel3.setText("Rules Pane");
    jLabel4.setText("Number of Iterations");
    insetsPanel1.setMinimumSize(new
       Dimension(250, 35));
    insetsPanel1.setPreferredSize(new
       Dimension(350, 35));
    this.getContentPane().add(panel1, null);
    insetsPanel1.add(button1, null);
    panel1.add(jPanel1,
       BorderLayout.CENTER);
    jPanel1.add(jLabel1, new
       XYConstraints(0, 0, -1, -1));
    jPanel1.add(jLabel2, new
       XYConstraints(0, 40, -1, -1));
    jPanel1.add(jLabel3, new
       XYConstraints(0, 108, -1, -1));
    jPanel1.add(jLabel4, new
       XYConstraints(0, 177, -1, -1));
    panel1.add(insetsPanel1,
       BorderLayout.SOUTH);
  }

/* ------------------------
   Methods
 * ------------------------ */
  protected void processWindowEvent
       (WindowEvent e)
  {
    if(e.getID() ==
       WindowEvent.WINDOW_CLOSING)
    {
      cancel();
    }
    super.processWindowEvent(e);
  }

  void cancel()
  {
    dispose();
  }

  public void actionPerformed(ActionEvent e)
  {
    if(e.getSource() == button1)
    {
      cancel();
    }
  }
}
```

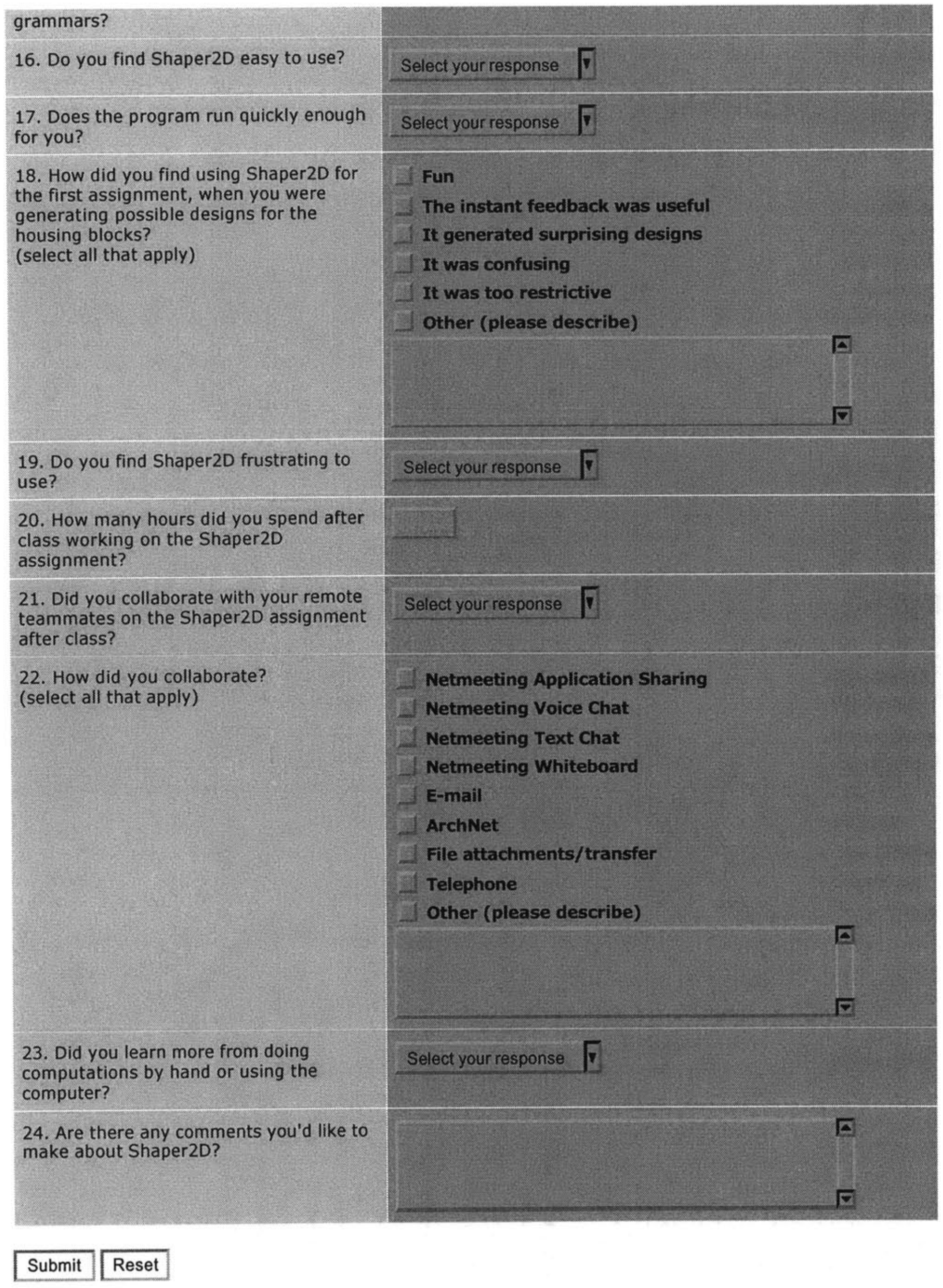


MIT

Shaper2D Survey 1

Please take a few minutes to fill this in -- we'd really like to know what you think!

1. What is your name?	**First Name**	
	Surname	
2. What team are you in?	Select your response	
3. What degree program are you in?	Select your response	
4. Did you find the workshop useful for learning about computation and design?	Select your response	
5. Before this workshop, what was your proficiency in: MIT:spoken Japanese MYU: spoken English	Select your response	
6. Before this workshop, what was your proficiency in: MIT: written Japanese MYU: written English	Select your response	
7. How much of a problem was the language barrier?	Select your response	
8. How experienced were you with computers before the workshop?	Select your response	
9. How much did you know about shape grammars before the workshop?	Select your response	
2D Shape Grammars		
10. How was the presentation of Terry Knight's first lecture?	Select your response	
11. How did you find doing the shape grammar exercises by hand?	Select your response	
12. How was your comprehension of shape grammar concepts after Terry Knight's first lecture?	Select your response	
Shaper2D		
13. How understandable was the Shaper2D tutorial?	Select your response	
14. How frequently did you need to refer to the tutorial when learning Shaper2D?	Select your response	
15. Was Shaper2D helpful for developing your understanding of 2D shape	Select your response	



grammars?	
16. Do you find Shaper2D easy to use?	Select your response
17. Does the program run quickly enough for you?	Select your response
18. How did you find using Shaper2D for the first assignment, when you were generating possible designs for the housing blocks? (select all that apply)	☐ Fun ☐ The instant feedback was useful ☐ It generated surprising designs ☐ It was confusing ☐ It was too restrictive ☐ Other (please describe)
19. Do you find Shaper2D frustrating to use?	Select your response
20. How many hours did you spend after class working on the Shaper2D assignment?	
21. Did you collaborate with your remote teammates on the Shaper2D assignment after class?	Select your response
22. How did you collaborate? (select all that apply)	☐ Netmeeting Application Sharing ☐ Netmeeting Voice Chat ☐ Netmeeting Text Chat ☐ Netmeeting Whiteboard ☐ E-mail ☐ ArchNet ☐ File attachments/transfer ☐ Telephone ☐ Other (please describe)
23. Did you learn more from doing computations by hand or using the computer?	Select your response
24. Are there any comments you'd like to make about Shaper2D?	

survey_summary Page 1 of 5

Summary of Shaper2D Questionnaire

Questions	**MIT**	**MIYAGI**
Before this workshop, what was your proficiency in: MIT: spoken Japanese, MYU: spoken English		
Fluent	1	1
Good		1
Average		3
Poor		7
None	7	
Before this workshop, what was your proficiency in: MIT: written Japanese, MYU: written English		
Fluent	1	
Good		5
Average		1
Poor		6
None	7	
How would you describe the quality of communication with your remote teammates?		
Very good		1
Good	3	2
Average	4	5
Poor	1	2
How much experience did you have with computers before the workshop?		
Considerable	4	6
Some	4	4
Little		2
None		
What computer software/applications have you used before?		
Windows 95/98/2000	8	11
Internet Explorer	8	12
NetMeeting	1	5
Java Applications	5	1
AutoCAD	7	6
Other CAD programs (please describe)	3	7
MIT: Alias Wavefront, Adobe Photoshop, Adobe Illustrator, Adobe Premier, Adobe After Effects, MacroMedia Flash, MacroMedia Dreamweaver, Bentley Microstation **MIYAGI:** Microstation (7 responses), Lightscape, 3D Studio Max, VectorWorks, FormZ (2), Autocad LT, MiniCAD.		
How much experience did you have with shape grammars before the workshop?		
Considerable		1

file://D:\MIRI_WORK\Thesis\survey\survey_summary.html 5/24/01

Little		4
None	7	5
Why did you enroll in this workshop?		
To learn about design and computation	4	9
To learn about shape grammars	5	5
To experience remote collaboration	6	9
To work with students from another university	4	9
To learn more about a different culture	5	12
Other (please describe)	2	3
MIT: I enrolled mostly to learn more about design and computation, and particularly shape grammars, but the remote collaboration was an additional bonus. **MIT:** to probe into the effectiveness of NetMeeting and PictureTel **MIYAGI:** train translation skill **MIYAGI:** I want opportunity to speak English **MIYAGI:** to learn English		

2D Shape Grammars

Questions	**MIT**	**MIYAGI**
How understandable was Professor Terry Knight's first lecture?		
Understood everything	8	3
Understood most		9
Understood little		
Did not understand		
What was your experience of doing the shape grammar computation by hand, using tracing paper?		
Very easy	5	2
Somewhat easy	3	7
Average		1
Difficult		2
What was your understanding of 2D shape grammar concepts after Professor Terry Knight's first lecture?		
Understood everything	8	4
Understood most		7
Understood little		1
Did not understand		
How would you use shape grammars "by hand" again?		
For a site design problem	5	6
For other design problems	4	4

For fun	4	4
Other (please describe)	2	1

MIT: any kind of repetitive design...such as math illustration, general design

MIT: for better understanding of the rules governing the behavior of the shape formation process

MIYAGI: I don't have good idea about 2d grammar, especially by hand. But 2d shaper you made is really wonderful! I might use for some other design.

Shaper2D

Questions	**MIT**	**MIYAGI**
How understandable was the Shaper2D tutorial?		
Understood everything	6	4
Understood most	2	7
Understood little		1
Did not understand		
How frequently did you need to refer to the tutorial when learning Shaper2D?		
Often	1	1
Sometimes	2	8
Rarely	2	3
Never	3	
How helpful was Shaper2D for developing your understanding of 2D shape grammars?		
Very helpful	3	4
Helpful	4	6
Somewhat helpful	1	2
Not helpful		
How easy to use was Shaper2D?		
Very easy	6	6
Somewhat easy	2	2
Average		4
Difficult		
What is your comment on the speed of Shaper2D?		
Very quick	3	3
Quick	3	7
Average	2	1
Slow		1
What is your opinion of using Shaper2D in a design context, such as the first assignment?		
The instant feedback was useful	7	6
It generated surprising designs	3	8

It was too restrictive	1	2
It was frustrating		1
Other (please describe)	3	1
MIT: I had more control over my own design than through 3d shape than any other program **MIT:** It is hard to design without knowing exactly what the end result is going to be. **MIT:** Using Shaper2D in the process of creating a site plan was an excellent way to introduce the program in a real world context. **MIYAGI:** interesting!		
How enjoyable was Shaper2D to use?		
Very enjoyable	4	9
Somewhat enjoyable	4	3
Boring		
What was your understanding of shape grammar concepts after using Shaper2D?		
Understood everything	3	3
Understood most	5	8
Understood little		1
Did not understand		
How many hours did you spend after class working on the Shaper2D assignment?	2.16 hours (average)	2.18 hours (average)
Less than an hour	1	
1 hour	2	5
2 hours	3	3
3 hours		2
More than 3 hours	2	2
Did you collaborate with your remote teammates on the Shaper2D assignment after class?		
Yes – from University	4	7
Yes – from Home		
No	4	5
How did you collaborate?		
NetMeeting Application Sharing	4	6
NetMeeting Voice Chat	2	5
NetMeeting Text Chat	4	8
NetMeeting Whiteboard		
E-mail	5	7
ArchNet	1	3
File attachments/transfer		1
Telephone		1

Other (please describe)	1	
MIT: We had three different correspondences by email, by far the most effective means of communication.		
How would you use Shaper2D again?		
For a site design problem	6	8
For other design problems	3	8
For fun	4	5
Other (please describe)	1	
MIT: I can see myself using Shaper2D for creating a piece of sculpture.		
Did you learn more from doing computations by hand (using tracing paper) or using the computer (using Shaper2D)?		
By hand	5	1
Using the computer		5
Both the same	3	6
Neither		
Are there any comments you'd like to make about Shaper2D?		

MIT:

"it has a lot of potential in it usage. Architecture, design, math, games, etc. The best part of it is that it is so easy to learn and allows much control over the design"

"It is fast for creating the designs, but sometimes, if running on the web, the Java can freeze, or maybe it overloads the system. I had much better luck running the program directly, not on the web. It's an awesome program! Good work Miranda. The quick feedback doesn't force you to make a commitment to a design, like 3dshaper can - for that you have to be willing to wait and reopen windows. Also, the interface is really easy to use. I had no problems using it, and I think even without a tutorial I could have figured it out pretty quickly."

"This workshop was excellent!!! I learned a lot. And, I have many chances to think about computer generated shapes and usage of computer as well."

"I wish we could import shapes from AutoCAD as in the case of AutoGrammar that would be great..."

"It would definitely be very helpful in an architectural design context to be able to import the site plan into the program."

"It is a good program that has development potential. If it can produce formations that can be used in site plans in a convincing manner, it can become popular."

MIYAGI:

"I am very surprised to hear that you made program shaper 2d. This software is very useful."

"Generating shape is very easy with 2d shaper but it doesn't help very much for understanding or "feeling" grammar. Anyway, it helps us to design."

"I want to we can use more rules (3or4)"

Experimental Protocol

Shaper2D: A New Approach to Learning and Practicing Computational Design

Principal Investigator: Terry Knight
Associate Investigator: Miranda McGill

Description of Proposed Study

This study comprises a 'crash course' in computational design and the use of a computer program, "Shaper2D"*, designed as a pedagogical/practical application for teaching and experimenting with two-dimensional shape grammars. The purpose of the study is to ascertain whether shape grammars can be taught and understood without any previous experience with hand computation.

Prior to the commencement of the study, the participants will be asked to read and sign consent forms that explicitly request permission for the use of audio taping (see attached consent form).

The session will be approximately two-three hours long. It will involve a short presentation outlining the theory of shape grammars, their history and applications. It will then be followed by guided exercises with simple shape computations using the computer program. The participants will then be asked to complete a short exercise using the computer, during which time they can ask the investigator any necessary questions, at any time, pertaining to the exercise and/or theory. Once this exercise has been completed, the participants will then be asked to complete a short exercise using hand computation instead of the computer. Again, questions can be posed at any time to the investigator. Each exercise will be done independently by each participant.

A short interview will be conducted with each participant at the end of the study.

* The computer program can be viewed/used online at: http://architecture.mit.edu/~miri/shaper2d/

Proposed Interview Questions

- Did you find the session useful for learning about computation and design?
- Was Shaper2D helpful for learning the concepts of two-dimensional shape grammar?
- How do you feel your understanding of the concepts being taught were affected by experimentation with Shaper2D?
- What is your opinion of doing shape grammars by hand (using tracing paper) vs. by computer?
- Would you have preferred to do the hand computation before using the computer program?

Consent Form

Shaper2D: A New Approach to Learning and Practicing Computational Design

Principal Investigator: Terry Knight

Associate Investigator: Miranda McGill

Outline of Study

This study comprises a 'crash course' in computational design and the use of a computer program, "Shaper2D", designed as a pedagogical/practical application for teaching and experimenting with two-dimensional shape grammars. The purpose of the study is to ascertain whether shape grammars can be taught and understood without any previous experience with hand computation.
The session will be approximately two/three hours long. It will involve a short presentation outlining the theory of shape grammars, their history and applications. It will then be followed by guided exercises with simple shape computations using the computer program. The participants will then be asked to complete a short exercise using the computer, during which time they can ask the investigator any necessary questions, at any time, pertaining to the exercise and/or theory. Once this exercise has been completed, the participants will then be asked to complete a short exercise using hand computation instead of the computer. Again, questions can be posed at any time to the investigator. Each exercise will be done independently by each participant.

A short interview will be conducted with each participant at the end of the study.

Consent Information

I understand that participation in this study is voluntary and that I am free to withdraw my consent and to discontinue participation in the project or activity at any time without prejudice to myself. During the interview, I understand that I may decline to answer any questions, again without prejudice.

In the unlikely event of physical injury resulting from participation in this research, I understand that medical treatment will be available from the MIT Medical Department, including first aid emergency treatment and follow-up care as needed, and that my insurance carrier may be billed for the cost of such treatment. However, no compensation can be provided for medical care apart from the foregoing. I further understand that making such medical treatment available; or providing it, does not imply that such injury is the Investigator's fault. I also understand that by my participation in this study I am not waiving any of my legal rights.

I understand that I may also contact the Chairman of the Committee on the Use of Humans as Experimental Subjects, MIT 253-6787, if I feel I have been treated unfairly as a subject.

I understand that the session and interviews will be audiotaped. After the study, the Associate Investigator will have control over the audiotapes. Once the study has been completed and the session has been analyzed, the audiotapes will be detroyed. This will happen no later than May 11, 2001.

☐ By checking this box, I agree to the audiotaping of this study and the subsequent interview. I understand that I have the right to withdraw the audiotapes at ANY time, even after the study is complete. I also understand that the investigator may ask to quote any comments I make in future research publications, and that I will inform the investigator if I wish to remain anonymous in such publications.

Signed Date

References & Bibliography

Ackermann, E. (TBA, paper in preparation) Constructivism or Constructionism: What's the difference?, Massachusetts Institute of Technology, Cambridge, MA.

Arnow, D.M. & Weiss, G. (2000) *Introduction to Programming using Java: An Object Oriented Approach*, New York, NY: Addison-Wesley

Balkcom, S. (1992) Cooperative Learning: What Is It?, *Office of Educational Research and Improvement (ED)*, Washington, DC.

Boden, M.A. (1990) *The Creative Mind: Myths and Mechanisms*, London, England: Wiedenfield and Nicholson.

Brown, J.S., Collins, A. & Duguid, P. (1989) Situated Cognition and the Culture of Learning, *Educational Researcher*, vol. 18, no. 1, pp. 32-42.

Bruner, J.S. (1966) *Toward a Theory of Instruction*, Cambridge, MA: Harvard University Press.

Celani, G. (2001, paper in preparation) MIT/Miyagi Remote Collaborative Workshop: Computational Design for Housing, Cambridge, MA: Massachusetts Institute of Technology.

Ching, F.D.K (1979) Architecture: Form, Space, and Order, New York, NY: Van Nostrand Reinhold.

Cole, P.M. (1994) Finding a Path Through the Research Maze, *The Qualitative Report*, vol. 2, no. 1.

Copeland, B.J. (1997) The Church-Turing Thesis, University of Canterbury, in the *Stanford Encyclopedia of Philosophy*, http://plato.stanford.edu/entries/church-turing/

Dietrich, E. (1999) Algorithm, in *The MIT Encyclopedia of the Cognitive Sciences*, http://cognet.mit.edu/MITECS/Entry/dietrich

Flanagan, D. (1997) *Java in a Nutshell: A Desktop Quick Reference*, Sebastopol, CA: O'Reilly & Associates, Inc.

Flemming, U. (1987) More Than the Sum of Parts: The Grammar of Queen Anne Houses, *Environment and Planning B: Planning and Design*, vol. 14, pp. 323-350.

Foreman, G. (1994) Different Media, Different Languages, in Katz, L. & Cesarone, B. (eds), *Reflections on the Reggio Emilia Approach*, ERIC, University of Illinois at Urbana-Champaign.

Gardner, H. (1993) *The Unschooled Mind: How Children Think and How Schools Should Teach*, New York, NY: Basic Books.

Gero, J.S. (1994) Towards A Model Of Exploration In Computer-Aided Design, in J. S. Gero and E. Tyugu (eds) *Formal Design Methods for CAD*, Amsterdam, Holland: North-Holland, pp. 315-336.

Gero, J.S. (1996) Computers and Creative Design, Key Centre of Design Computing, Department of Architectural and Design Science University of Sydney, Australia.

Glaser, B.G. & Strauss, A.L. (1967) *The Discovery of Grounded Theory*, Chicago, IL: Aldine.

Glaser, B. (c.1999) an interview with Dr. Andy Lowe, produced by the University of Sterling, http://www.groundedtheory.com/vidseries1.html

Healy, L. & Hoyles, C. (TBA, paper in preparation) Software Tools for Geometrical Problem Solving: Potentials and Pitfalls, Institute of Education, University of London, England.

Hennessy, S. & Murphy, P. (1999) The Potential for Collaborative Problem Solving in Design and Technology, *International Journal of Technology and Design Education*, vol. 9, pp.1-36.

Holt, J.C. (1989) *Learning All the Time*, Cambridge, MA: Perseus Books.

Järvelä, A. & Niemivirta, M. (1999) The changes in learning theory and the topicality of the recent research on motivation, *Research Dialogue in Learning and Instruction*, vol. 1, pp.57-65

Kinach, B.M. (1995) Grounded Theory as Scientific Method: Haig-Inspired Reflections on Educational Research Methodology, *Philosophy of Education.*

Knight, T.W. (1989) Color Grammars: Designing with Lines and Colors, *Environment and Planning B: Planning and Design*, vol. 16, pp. 417-449.

Knight, T.W. (1994) Shape Grammars and Color Grammars in design, *Environment and Planning B: Planning and Design*, vol. 21, pp. 705-735.

Knight, T.W. (1998a) Designing a Shape Grammar: Problems of Predictability, in Gero, J.S. & Sudweeks, F. (eds), *Artificial Intelligence*, Kluwer Academic Publishers.

Knight, T.W. (1998b) Shape Grammars, *Planning and Design*, Anniversary Issue, pp. 86-91.

Knight, T.W. (2000) Shape Grammars in Education and Practice: History and Prospects, *http://www.mit.edu/~tknight/IJDC/ (online paper)*, Department of Architecture, MIT.

Lave, J. & Wenger, E. (1991) *Situated Learning: Legitimate Peripheral Participation*, New York, NY: Cambridge University Press.

Leblanc, S., Saury, J., Sève, C., Durand, M. & Theureau, J. (2001) An analysis of a user's exploration and learning of a multimedia instruction system, *Computers & Education*, vol. 36, pp.59-82

March, L. (1997) A White Paper: Shape Computation, *From the proceedings of the Symposium on Design and Computation, University of California: Los Angeles.*

Mitchell, W.J. (1993) A computational view of design creativity, in J. S. Gero and M. L. Maher (eds), *Modeling Creativity and Knowledge-Based Creative Design*, Hillsdale, NJ: Lawrence Erlbaum, pp. 25-42.

Norman, D.A. (199) *The Design of Everyday Things*, New York: Doubleday.

Papert, S. (1993) *Mindstorms: Children, Computers and Powerful Ideas*, New York, NY: Basic Books.

Piaget, J. (1972) *Psychology and Epistemology: Towards a Theory of Knowledge*, London, England: Penguin.

Reddy, M.J. (1993) The Conduit Metaphor, in Ortony, A (ed.), *Metaphor and Thought*, Cambridge, England: Cambridge University Press, pp.284-310.

Resnick, M., Bruckman, A. & Martin, F. (1996) Pianos Not Stereos: Creating Computational Construction Kits, *Interactions*, vol. 3, no. 6.

Roth, W. (2001) Modeling Design as situated and distributed process, *Learning and Instruction*, vol. 11, pp.211-239.

Sanders, Ken (1996) *The Digital Architect*, New York, NY: John Wiley & Sons, Inc.

Smith, B.C. (1999) Computation, in *The MIT Encyclopedia of the Cognitive Sciences*, http://cognet.mit.edu/MITECS/Entry/smithbc2.html

Stiny, G. (1975) *Pictoral and Formal Aspects of Shape and Shape Grammars*, Basel, Germany: Birkhäuser Verlag.

Stiny, G. (1976) Two Exercises in Formal Composition, *Environment and Planning B*, vol. 3, pp.187-210.

Stiny, G. (1978) An Evaluation of Palladian Plans, *Environment and Planning B*, vol. 5, pp.199-206.

Stiny, G. (1980) Introduction to Shape and Shape Grammars, *Environment and Planning B*, vol. 7, pp.343-451.

Stravinsky, I (1947) *Poetics of Music*, Cambridge, MA: Harvard University Press.

Strommen, E.F., & Lincoln, Bruce (1992) Constructivism, Technology, and the Future of Classroom Learning, *Children's Television Workshop, New York*.

Tapia, M.A. (1999) A Visual Implementation of a Shape Grammar System, *Environment and Planning B: Planning and Design*, vol. 26, pp. 59-73.

Turing, A.M. (1950) Computing Machinery and Intelligence, *Mind*, vol. 59, pp.433-460.

Vygotsky, L.S. (1978) *Mind in Society: The Development of Higher Psychological Processe*s, with Cole, M., John-Steiner, V., Scribner, S., & Souberman, E. (eds), Cambridge, MA: Harvard University Press.

Yakeley, M.W. (2000) Digitally Mediated Design: Using Computer Programming to Develop a Personal Design Process, *PhD, Department of Architecture*, Cambridge, MA: Massachusetts Institute of Technology.