Algorithmic Craft: Tools and Practices For Creating Useful and Decorative Objects With Code

by

Jennifer Jacobs

Submitted to the Department of Media Arts and Sciences
in partial fulfillment of the requirements for the degree of

Masters of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2013

Author .....
Department of Media Arts and Sciences
May 18, 2013

Certified by .....
Leah Buechley
Associate Professor of Media Arts and Sciencesr
Thesis Supervisor

Certified by .....
Mitchel Resnick
Academic Head, Program in Media Arts and Sciences
Thesis Supervisor

Certified by .....
Neri Oxman
Assistant Professor of Media Arts and Sciences
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Algorithmic Craft: Tools and Practices For Creating Useful and Decorative Objects With Code

by

Jennifer Jacobs

Submitted to the Department of Media Arts and Sciences
on May 18, 2013, in partial fulfillment of the
requirements for the degree of
Masters of Science

Abstract

The accessibility, diversity, and functionality of modern computer systems make computer programming (hereafter programming) useful in many realms of human study and advancement. Visual and physical art, craft, and design are interrelated domains that offer exciting possibilities when combined with programming. Unfortunately, use of programming is currently limited as a medium for art and design, especially by young adults and amateurs. Many potential users view programming as highly specialized, difficult, inaccessible, and only relevant as a career path in science, engineering or business fields, rather than as a mode of personal expression. Despite this perception, programming has the potential to correspond well with traditional, physical art-making practices. By forging a strong connection between programming and the design and fabrication of personally relevant physical objects, it may be possible to foster meaningful experiences in both programming and design for novice practitioners. The combination of digital fabrication technologies with computational design serves as one such connection.

Thesis Supervisor: Leah Buechley
Title: Associate Professor of Media Arts and Sciencesr

Thesis Supervisor: Mitchel Resnick
Title: Academic Head, Program in Media Arts and Sciences

Thesis Supervisor: Neri Oxman
Title: Assistant Professor of Media Arts and Sciences

"The type of work which modern technology is most successful in reducing or even eliminating is skillful, productive work of human hands, in touch with real materials of one kind or another. In an advanced industrial society, such work has become exceedingly rare. A great part of the modern neurosis may be due to this very fact; for the human being enjoys nothing more to be creatively, usefully, productively engaged with both his hands and his brains."

E.F. Schumacher, Small is Beautiful (1973)

Acknowledgments

To come....

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Computation is a driving force in our world. The power and ubiquity of modern computer systems have made the skill of computer programming (hereafter programming) relevant to a wide range of human studies and disciplines. Most commonly, programming is viewed as an essential component of science, engineering, and business related applications[6]. As a result, many nascent programmers view programming as highly specialized, difficult, inaccessible, and only relevant as a career path in those particular fields. In reality, computation is a broad discipline with many applications, ranging from professional to personal. Foremost programing can serve as a medium for personal expression, through applications in art and design. With the emergence of digital fabrication technology, In addition, programing provides the means to design and produce useful objects and devices, not only on at an industrial scale, but also on a personal and individual scale [**?**]. When programing is used create unique, functional physical objects, new possibilities emerge in the way people design, the types of objects people create, and role programing can play in peoples' lives. Computational design, the practice of programming to create form, structure and ornamentation, is a new way to design. When paired with digital fabrication technology, computational design allows people to make physical objects by writing code. My objective is to examine the combination of computational design, digital fabrication and traditional arts and crafts for the production of functional decorative objects. I define this domain with the term algorithmic craft. In this thesis, I will define the affordances of algorithmic craft and describe the development and dissemination of three tools to support novice practitioners in this domain. The process of bridging the spaces between textual programming language, visual design, and physical construction however, is not self-evident and raises many practical and theoretical questions. What are the important design principles to consider when creating programming environments for physical design? How do we compellingly link textual code with visual designs and what are the appropriate intersection points between textual manipulation and visual manipulation? What support is required to help people move back and forth from programming to

building real objects in a way that is comfortable, expressive and pleasurable? How can we remove the technical challenges in translating code into an object that can be successfully fabricated, while still supporting a wide variety of design styles, aesthetics and approaches? Finally, how can we interlink the often disparate processes of physical prototyping with digital design and programming in a way that creatively reinforces both physical and virtual modes of working?

Outline general thesis here...

Chapter 2

Motivation and Background

"The mathematicians patterns, like the painters or the poets must be beautiful; the ideas like the colours or the words, must fit together in a harmonious way. Beauty is the first test: there is no permanent place in the world for ugly mathematics."

G. H. Hardy [5]

2.1 Computational Design

The practice of computational design is fundamentally different from contemporary design. Due to the multitudes of approaches among different designers, it ultimately futile to attempt to describe a single standard design process. It is however useful to point out several key features of computational design that stand in contrast with the conventional design.

Both computational and conventional designers begin with a design problem. Conventional designers often proceed by roughing out a number of specific solutions to this problem. These early solutions are evaluated against one another for their successes and drawbacks, and from this evaluation a smaller set of more refined solutions are produced. This iterative process may continue, often through the incorporation of outside feedback, until a single solution is reached that is sufficiently refined and successful in addressing the initial problem. This highly simplified summary approximately describes the conventional iterative design process. Computational design incorporates many of the iterative principles of conventional design, but differs significantly in its approach. Rather than begin by developing a concrete initial solution to the initial design problem problem, the computational designer must first formalize the elements of the problem into a set of rules. The designer then creates a system based on these rules that capable of producing a variety of solutions, depending on the input criteria it is given. In its simplest form, this system may consist of a single algorithm with static input and limited output solutions. More frequently however, the computa-

tional design process produces complex systems that act on upon a wide range of input criteria and parameters, and can produce nearly infinite number of design solutions. Iteration in computational design then takes the form of incremental adjustments to the system. Naturally, many of the solutions produced by an initial system fail to address original design problem. By sampling a number of outputs from a system, the designer can "tweak" or make adjustments to the rules that govern the system, eventually resulting in more and more desirable output solutions. This process of sampling and tweaking is continued until the designer is satisfied with a given range of outputs. The designer can then vary the input to the system and can select among the resulting solutions.

While the process of computational design can be distinguished from conventional design, the two fields are compatible with one another. Aside from a wholly different approach, computational design can also be considered as a means of extending traditional design practice through several key affordances: These include the following:

- **Precision:** Computation supports high levels of numerical precision with relatively little effort on the part of the designer.
- **Automation:** Computation allows for rapid automation of repetitive tasks. Automation often plays a key role in enabling the development and transformation of complex patterns and structures, through the combination of large numbers of simple elements in an ordered and structured manner.
- **Generativity and randomness:** Computation allows for the programmer to create algorithms which when run, allow for the computer to autonomously produce unique and often unexpected designs.
- **Parameterization:** Computation allows users to specify a set of degrees of freedom and constraints of a model and then adjust the values of the degrees of freedom while maintaining the constraints of the original model [7].
- **Documentation and remixing:** Computationally generated designs are generated by a program, which can be shared with and modified by other designers. Because these programs are often text-based, they also serve as a form of documentation of the design process.

In combination with these affordances however, computational design also incorporates a number of challenges in the design process that are not present in traditional design:

- **Formalizing complex problems** As design problems grow in complexity, formalizing the problem in a manner that can be expressed programmatically becomes increasingly challenging. Writing an algorithm to generate a visual pattern is relatively simple, however writing a program to incorporate that pattern into the design of an entire garment is non-trivial.

- **Creating singularities:** A designer will often choose to deviate from a set pattern or structure at specific points in order to create a special emphasis in that area. Because computational design is governed by a systematized ruleset, the methods of breaking these rules at arbitrary points is are often unclear and tedious to implement.

- **Selecting a final design:** The systematic approach to computational design gives the designer the ability to produce extremely large numbers of solutions to a single design problem. While this is useful in situations where multiple solutions are required, when a single design must be chosen, the process of deciding on a solution is often difficult and sometimes arbitrary, especially if the decision is based on aesthetic criteria.

2.2 Digital Fabrication

expand sectionAlthough computational design must be conducted on a computer to some degree, the artifacts generated by computational design are not restricted to the screen. Digital fabrication technology provides the opportunity to translate programmatically generated designs to physical form. Digital Fabrication is the process of using computer-controlled machines to fabricate objects specified by a digital design file or tool path. The machines that encompass digital fabrication range from 3D printers, laser cutters, and computer numerically controlled (CNC) milling machines, to vinyl cutters, CNC embroidery machines and knitting machines, and even inkjet printers. Digital fabrication shares many of the affordances of computational design. In particular, it allows for the creation of physical objects of great complexity without formal skill in craft or extensive manual labor. Digital fabrication also allows for the rapid production of small volumes of similar or identical objects. Lastly, because the artifacts produced through digital fabrication are derived from digital files, anyone with access to the file, and a similar fabrication machine can create a copy of the object, or incorporate elements of it into a new design.

Digital fabrication is also compelling for its own reasons. Currently, digital fabrication machines are rapidly decreasing in price and increasing in availability [**?**]. As a result, we are seeing the emergence of personal fabrication, wherein sophisticated manufacturing technologies are becoming available to regular people [**?**]. Excluding personal 3D printers which are generally limited to a few varieties of ABS plastic, most personal fabrication machines can work with a wide range of materials. Laser cutters work well with traditional materials such as wood, paper and cloth. Vinyl cutters can also be used on cloth and paper, as well as cut vinyl patterns which can be used for screen printing. The current stage of personal fabrication is estimated to be at the same place as personal computing in the 1970s [**?**] add section on computer aided design

2.3 Algorithmic Craft

The conjunction of computational design and digital fabrication has the potential to allow individuals to use programming to express their aesthetic concerns in the creation of objects. This is important because aesthetic expression through design is a substantial part of intellectual development [4] and an important part of peoples lives. These machines offer the potential to extend and innovate traditional forms of design, constructing and crafting by allowing for greater levels of automated complexity and precision in physical objects, and correspond well with the practice of programming. importance of materials- craft as domain of materiality, design and programing as abstractions in many cases, but deal very directly with materiality when applied to the real world- craft offers direct connection to this - citation from rosner article craft vs design

2.4 Challenges in broad participation in computational design and digital fabrication

Despite the opportunity for casual, non-professional engagement in computational design and digital fabrication, this domain is largely limited to experts and professionals for a number of reasons. In a practical context, new practitioners in this field are confronted with the difficult process of translating their code-based design to a format that is compatible with the target fabrication machine. Furthermore, the challenges involved in designing complex objects from multiple digitally fabricated parts are extremely difficult to tackle for casual users. There are also severe limitations on computational design software for novices capable of supporting digital fabrication. As we discuss in the following related work section, the majority of traditional CAD tools do not contain computational design capablities that are accessible to nicest users. Similarly, novice oriented programing environments lack the functionality to allow novices to produce designs that are suitable for fabrication. More broadly, there are significant perceptual barriers to participation. There persists among the general public a limited perception of the applications of programing. Many people consider programing to be irrelevant to their interests, and therefore lack motivation to pursue what they perceive to be a highly specialized and difficult undertaking [12]. There are also prevailing perceptions of digital fabrication which may hinder casual engagement. Personal fabrication technology is often portrayed as a precursor to the production of replicator-like technology which can instantiate literally anything by building it directly from atoms. This projection of future technology is exciting to think about, but I argue that it also acts as a barrier to immediate widespread engagement with existing forms of digital fabrication, by setting up unreal expectations for this technology and portraying it as technology that facilitates new forms of consumerism, as opposed to being a new tool for personal creation and expression. This perspective also eliminates the need or desire for human engagement

in the fabrication process, eliminating the entry points for craft:

*A central element of these and other visions of the future is that craft is done for us: Kitchens tell us what and how to cook, eliminating the creativity and pleasure of cooking from scratch with whats on hand; object printers create flawless prototypes, eliminating messily glued-together chipboard and toothpicks. In this new world, craft becomes fetishthe proudly displayed collection of vinyl records shelved alongside an iPod and digital files [**?**].*

There is also the tendency to trivialize the hobbyist applications of digital fabrication when analyzed in a research context. In the domain of Human Computer Interaction (HCI), researchers often focus on the hedonistic properties technologically oriented DIY practices as opposed to the utility of the resultant artifacts or their ability to generate profit. Pleasure and self-expression are central components of hobbyist and craft-oriented computation and digital fabrication, however these qualities do not come at the cost of generating artifacts that are practical, functional, and sellable [**?**]. The trend of separating hobbyist practice as merely fun in contrast to professional practical applications overshadows some of the most interesting practical possibilities that emerge through amateur use of this technology.

Chapter 3

Related Tools and Research

There are numerous forms of CAD software and programing environments. Within the realm of computational design and digital fabrication, there are 4 primary categories of existing tools that directly relate to my study of computational design and digital fabrication: professional computational-design tools, entry-level programing environments, and novice-oriented computer-aided-design (CAD) tools. While certain qualities are shared between these categories, several key distinctions exist between each group of tools.

3.1 Professional Computational Design Tools

A couple of forms of professional computational-design tools exist. Foremost, many popular graphic-user-interface (GUI) CAD applications include a feature that allows the user to automate certain elements of the program through scripting or programing. For example, in Adobe software like Photoshop and Illustrator, it is possible to write JavaScript-based programs to automate various application procedures. Similarly, 3D modeling tools such as Maya and Blender feature the ability to script behaviors in languages that are syntactically similar to Perl and Python respectively. This scripting is usually omitted from the primary menus and interfaces of the applications that feature it. There are also professional tools that are explicitly developed for computational design. The most prominent example is Grasshopper, a third-party add-on for the Rhinoceros 3D modeling tool. Grasshopper is a data-flow programing environment that lets users combine a variety of modules and blocks to create and adjust 3D models in Rhino. A textual coding module is also available and allows users to integrate C# scripts using the Rhino API into their patch, although the user must have an understanding of the basic principles of programing in order to effectively use this functionality.

DesignScript, a more recent computational design tool, developed by Autodesk, is a domain specific text-based programing environment and language that contains methods to generate and

manipulate geometric models that are compatible with existing Autodesk applications. DesignScript itself functions as an add on to the Autodesk AutoCad software. DesignScript is intended for use by experienced designers and 3d modelers who posses a range of programming expertise. The language syntax is based on C#, however it features the ability to operate in both associative and imperative paradigms, in an effort to support a pedagogical transition between basic and more complex forms of computational design [**?**].

Lastly, OpenSCAD is a script-based constructive solid geometry modeling tool developed specifically for CAD applications. OpenSCAD contains a custom programing language in which the user can create descriptions of 3d models in a textual format, and display them by compiling the script. This scripting behavior provides the user with precise control over the modeling process and enables the creation of designs that are defined by configurable parameters, however this control comes at the cost of requiring the user to be familiar with textual programing an scripting. In fact, OpenSCAD is explicitly developed for programmers and relies on textual input exclusively as the mechanism for design, In addition, unlike the prior tools mentioned, OpenSCAD is both free and open source, and many variations and derivatives of it exist [**?**].

In the context of digital fabrication, one of the most important elements of these professional tools is their ability to import and export a wide variety of file formats, thus facilitating the transitions between a digital design and the required file type for a specific fabrication tool. Despite their power, and due to their high cost and complex feature set, these professional tools are extremely difficult for amateurs to access and use. It is also important to note that with the exception of OpenSCAD, the examples listed are only available as plugins or add-ons or are developed to supplement an existing graphical tool, rather than serve as the primary method of design. In some cases this status as a form secondary functionality adds a set of practical barriers to independent use. The scripting tools in illustrator and photoshop are difficult to locate, Grasshopper only functions on Windows versions of Rhino, and Design Script requires the prior purchase of AutoCAD to operate. Although these practical barriers can be overcome, their existence often prevents less experienced users from gaining access. In addition, the positioning of computational functionality as secondary to the primary method of design points to a larger ideological classification of these forms of design as a specialized and exclusive, rather than a primary method of design.

3.2 Entry-level CAD Tools

A subset of CAD tools have also been created that are designed to be more accessible to a wider range of people. These tools provide an option for individuals who lack the experience and access to professional level tools, however they also provide an opportunity for more casual participation in CAD. SketchUp is a 3d modeling tool developed by Google to enable easier forms of 3D model-

ing. Although SketchUp was not explicitly created to allow people to design for CNC and digital fabrication, several 3rd-party add ons exist that allow users to export designs to file formats that are compatible with a variety of fabrication machine [**?**]. TinkerCad is another 3d modeling tool designed for entry level users. As opposed to SketchUp, TinkerCad is explicitly developed to assist in designing for 3d printers and has built in functionality to allow users to export their designs to the .stl format which is compatible with 3D printing [**?**]. AutoDesk has also produced several entry level 3d-modeling applications as a part of their 123D series. Many of these applications are designed to interface with digital fabrication, including 123D Make which allows users to convert stock or uploaded 3d models into a series of flat parts which can be fabricated on 2-axis machines like laser cutters, and 123D Creature, which enables users to design a variety of creatures from a set of basic parts and then order a 3d printed model of their finished creature [**?**]. AutoDesk Research has also developed MeshMixer, an application for the intuitive merging and manipulating high resolution triangle meshes. MeshMixer was released to the public and has since become a popular 3d design tool for hobbyist 3D printer users. All of the entry level tools listed above vary in their specific approach to creating more accessible forms of CAD. In general they feature a trade off between limited functionality and power, in favor of a simplified tool set and an easier learning curve. Despite these restrictions, it is possible to use these entry level tools to develop highly complex and sophisticated models show example image of mesh mixer model. A more serious limitation of these tools is their ephemerality. Because entry level CAD tools are often free, and more frequently web based applications, it is common for them to suddenly become unavailable or no longer supported by the company that produces them. Tinkercad serves as a recent example of this wherein the parent company decided to transition to focusing on professional-level CAD tools and as a result, closed down the Tinkercad website and cut off access to the application footnote about tinkercad recently being acquired by autodesk. Several of these entry-level tools feature some form of scripting or programing functionality. A plugin for Sketchup allows users to automate certain actions by using the Ruby-based SketchUp API. TinkerCad allows users to create Shape Scripts, which are parametric models defined by javascript code. MeshMixer has an C++ API which is not yet publicly available, but is provided to interested parties upon request. While these computational tools suggest compelling possibilities, similar to the professional level tools listed above, they are positioned as secondary ways of interacting, and are much less deliberate than the primary features of the application.

3.3 Learning-Oriented programming tools

In addition to entry level CAD tools, a number of tools and applications have been created to introduce inexperienced programmers to the realm of computer science. Logo, a computational drawing

program, serves as the seminal novice programming language founded on principles of constructionism and embodiment [9]. The Scratch visual programming language is a notable successor to Logo, and allows users to create interactive projects by combining command blocks rather than writing textual code [11]. Alice is another programming environment that relies on visual programming, but is targeted towards an older user group than Scratch [3]. Turtle Art [16] and Design Blocks [2] are two visual programming languages inspired by Logo that are designed specifically for visual composition. Processing is a text-based programming environment designed for easy learning, and directed toward artists, designers, and inexperienced programmers [10]. Logo, Turtle Art, Design Blocks, and Processing facilitate computational drawing and, therefore, can be viewed as computational-design environments. There remains a gap, however, between novice-oriented programing environments and the novice-oriented CAD tools. In direct contrast to the novice oriented CAD tools described in the preceding section, although learning oriented programing tools can provide an excellent platform for generating digital computational design work, they often lack explicit features for generating and exporting designs that are compatible digital fabrication. It is possible to create work-arounds to this. For example in processing, users can download and install community-created libraries that allow for .stl, .dxf and .pdf export, enabling a sub group of users to use processing as a design tool for 3d printing, and laser cutting. The independent development of export functionality for tools like Processing demonstrates that there is significant interest in combining computational design and digital fabrication. If we wish to open this space for entry level practitioners however, we must design tools that exhibit the tools and techniques for computational design for fabrication as their primary functionality.

3.4 Novel Fabrication and CAD tools

In addition to these tools, there are a number of research projects involving novel forms of fabrication and software tools that demonstrate new approaches for computational design and digital fabrication. Sketch It, Make It is a 2D CAD tool that allows users to constrain their designs through gestures made using a digital drawing tablet [14]. Spatial Sketch is a tool that allows users to create abstract 3D sketches via their gestures, and then translates the sketches into a set of slices, which can be fabricated and combined into a finished piece [17]. SketchChair allows users to design their own chair by sketching with a computer stylus [13]. The resultant design can then be cut on a computer-numerical controlled (CNC) milling machine and assembled into a 3D object. SketchChair includes a simulation tool that allows users to test the usability of their chairs before they cut them. FlatCAD seeks to connect programming and digital fabrication and allows users to build customized construction kits with a laser cutter by programming in FlatLang, a novice-oriented programing language modeled on Logo [8]. Spirogator is a processing based tool that allows users to digitally

customize a set of hypotrochoid geared-drawing tools and then view a simulation of those tools in action. The user then has the option of either exporting the resulting design generated by the digital gears and fabricating it directly, or exporting the file paths for the gears themselves, and fabricating them on a laser cutter, to be used as physical drawing tools [**?**]. These examples share several important elements. They are restricted to a relatively narrow domain, or end product, but still support a wide range of design variation and personal expression within this domain. They contain intuitive and familiar methods of interaction often in the form of sketching, moving sliders. They contain explicit features for making the process of digital fabrication easier for new practitioners, and reduce the possibility of creating designs that will be infeasible to fabricate or are physically unstable. Spirogator and Sketch Chair's simulation tools are particularly interesting in this regard, as they assist the user in predicting some of the behavior of the resultant physical artifact prior to its fabrication. These qualities of domain-specificity, design flexibility, intuitive interaction and give this a better name practical support for fabrication are properties that should be incorporated in future tools in this area. /Garment-Creation CAD and Fabrication tools Sensitive Couture parsing patterns into 3d garments Sketch Based Garment Design Art quilt?

Chapter 4

Objectives

As indicated by the analysis of existing CAD and computational design tools. Many wonderful options exist to support novice entry into computer science. In addition, new tools are emerging to support novices in Computer Aided Design and digital fabrication. At this point however, is a lack of tools, which attempt to bridge these two spaces and make it feasible for people without significant technological experience to combine computational design and digital fabrication. The goal of my masters thesis is to explore and evaluate possible methods of bridging this space by developing tools that allow for casual, craft-oriented applications of digital fabrication and computational design. My objective is to better understand the relationship between textual programing language and visual design, investigate the iteration between code and physical object and examine strategies that support independent amateur use of computational design and digital fabrication.

4.1 Functional Properties of Algorithmic Crafting Tools

Based on my examination of related CAD and computational design tools I hypothesized that the following functional properties are necessary for an effective algorithmic craft software:

- **Emphasis on computational design:** Programing should be the chief method of generating and manipulating designs, and this focus should be reflected in the interface of the software.
- **Novice-oriented programing syntax:** The programing syntax and application programing interface (API) should be designed for novice programmers, and should be limited to methods and structures relevant to the task of design.
- **Design methods that facilitate digital fabrication:** The software should include programing and drawing methods that allow for the production of designs that are suitable for digital fabrication, including shape boolean operations, and support for exporting to relevant file formats.

- **Prioritization of visualization and simulation:** Users should be given ample and highly responsive visual feedback to inform their programming decisions.
- **Simple workflow from software to fabrication:** The transition from the design tool to the fabrication device should require as few intermediary steps as possible.
- **Free and open-source:** The software should be freely available, and able to function on multiple platforms with low requirements for computational processing power to afford high levels of access to casual users. If possible, the software should also be open source, in order to encourage the proliferation of additional novice oriented tools that can be developed for the specific needs of distinct user groups.
- **Domain specificity:** In order to ensure the usability of the software, it should be constrained to a the design of a particular set of end products, or crafting techniques. Or, if the software is general purpose, it should be packaged with a set of well documented example designs and projects that clearly demonstrate its key applications.
- **Fabrication and craft-specific documentation:** In addition to the documentation of the application and programing language, the fabrication and crafting techniques that are compatible with the software should be thoughtfully documented and provided to users. This documentation should include details on suitable materials, fabrication machine access and settings, and tutorials on the craft components of example projects.

4.2 Evaluation Criteria

In addition to these functional properties, I generated a set of evaluation criteria for any prospective algorithmic crafting software. A successful tool should produce the following results:

- **Allow users to successfully create physical artifacts:** The artifacts themselves should be both durable and used or in the creators life after completion.
- **Afford a wide degree of variation in design and expression:** The personal stylistic and aesthetic preferences of the creator should be apparent in the resultant artifact.
- **Enable people to understand the functionality and utility of the programs they write** Individuals should emerge from the process with a general understanding of some of the key components of computer programing, with an ability to articulate how these components function in their design.
- **Allow users to create objects and designs they would have difficulty generating with conventional techniques** The tool should enable the use of the affordances of computa-

tional design expressed in the Section 2.1, specifically precision, visual complexity, generativity and stylistic abstraction.

- **Engender in users a positive, enjoyable experience:** The tool and subsequent crafting activities should be pleasurable.
- **Foster a sense of confidence:** After working with the tool, people should have increased confidence in their ability to successfully program, design, and use digital fabrication tools.

4.3 Design Tools and Evaluation Methodology

Over the course of my thesis, I developed and tested three successive algorithmic crafting software tools, Codeable Objects, a domain specific programing library for the design and production of lamps, Soft Objects, an expanded version of Codeable Objects aimed at computational fashion design and DressCode, an integrated programing and visual design environment.

Each tool was evaluated during one or more workshops. I documented each workshop through pre and post surveys, interviews, and photographs of student projects. The surveys were aimed at understanding participants previous experience in programming and design, their interest in and attitudes toward programming and design (before and after the workshops), and their engagement in and enjoyment of the workshops. Pre-surveys were administered at the start of the workshops and focused on participants previous experience and attitudes. They also asked students to describe their opinions about how programing and craft could be combined, and how they felt programming could extend or limit creativity. Post-surveys were administered at the termination of the workshops and contained attitudinal questions that were matched to the pre-surveys. In addition, post surveys contained a range of written questions asking the participants to describe their opinion of the success of their projects and their experience using Codeable Objects, Soft Objects or DressCode respectively. In-person interviews were conducted with the participants in the fashion workshop. These interviews lasted an average of 15-30 minutes and were audio recorded and transcribed. During the interviews, the participants were asked to describe their experience in the workshop and talk about the process of conceptualizing, designing and producing their garments. They were asked to describe what they enjoyed, what was difficult for them, and what they felt they had learned through this process. Survey and verbal interview responses and project outcomes were then analyzed to determine if the essential qualities outlined in the requirements section (above) were achieved. We also used this information to identify recurring and prominent themes in participants experiences.

In the following three chapters, I detail the development, feature set and evaluation of each design tool.

Chapter 5

Codeable Objects

Codeable Objects is computational design tool that allowed people to design a laser cut lamp. The choice of a lamp allowed for a relatively broad design space wherein aesthetics were a primary

consideration, while still retaining the qualities of functionality and utility in the finished product. Lamps possess an established function, but offer a great deal of flexibility and personal freedom in the aesthetics and form. In addition, there is an established history of creating DIY lamps via digital fabrication. The Instructables community tutorial website has an entire section devoted to DIY lamps, and many examples of patterns that use a laser cutter for fabrication.

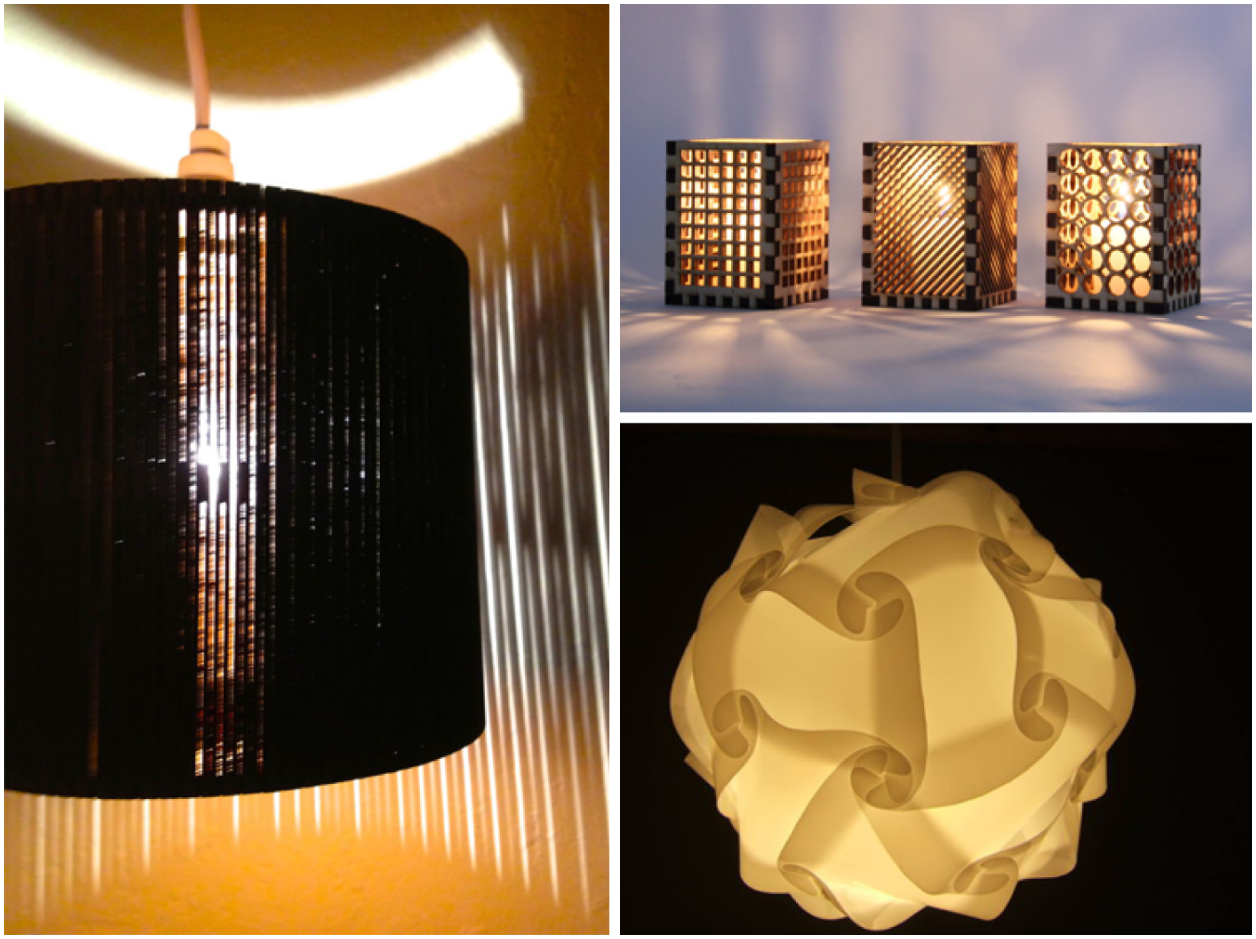


Figure 5-1: a selection of laser cut lamps from Instructables

5.1 Motivation

One of the restrictions of many of these examples is that they require the person making the lamp to directly emulate the design provided by the creator of the tutorial. If the person wishes to deviate from the original design, they need to use a CAD tool like Adobe Illustrator or Solid Works[1]. As mentioned in Section 3.1, professional CAD tools like Solid Works are often difficult to access and use for casual practitioners. In addition, during my personal experience in using a non-parametric

tool like illustrator to design, I often found I had to resort to fabricating numerous sample pieces of in order to ensure the joints and form would function correctly in the final piece. If I made a mistake, or decided I wanted to modify the design, I lost time and materials in the fabricating process, and had to endure the tedious process of adjusting correcting each individual part.

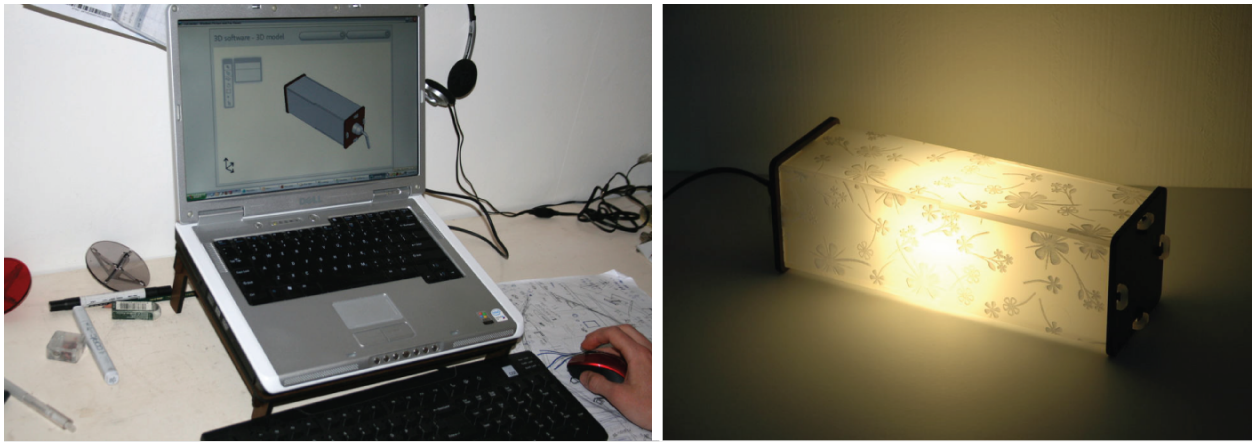


Figure 5-2: Instructables lamp tutorial with SolidWorks design process

One of the most frequent applications of a laser cutter is to create 3D forms by assembling 2D press fit pieces in a frame-like structure. I found that when creating 3D forms that were curved, it was extremely challenging in traditional 2D CAD software to correctly size and design parts which would fit the faces of the form. This was particularly relevant to lamp design, wherein it was necessary to create shades to diffuse the light. The shades also provided an excellent space for incorporating styles and patterns into the lamp. The combined tasks of simplified design and customization, parametric manipulation, and the calculation and conversion of a 3D form to 2D parts indicated that computational design would be a good match for the task of designing and fabricating a laser cut lamp.

5.2 Tool Description and workflow

The objective of the first version of Codeable Objects was simple: to create a tool that allowed to design a custom lamp by describing the form and the pattern of the shades, which they could then fabricate and assemble. The lamp itself was comprised of 4 basic parts, a wooden press fit frame, a set of vellum pieces that fit over the frame to act as a shade,a set of cardstock pieces with a pattern that fit over the shades, and a commercial made light fixture that fit into the frame (see figure: 5-3.)

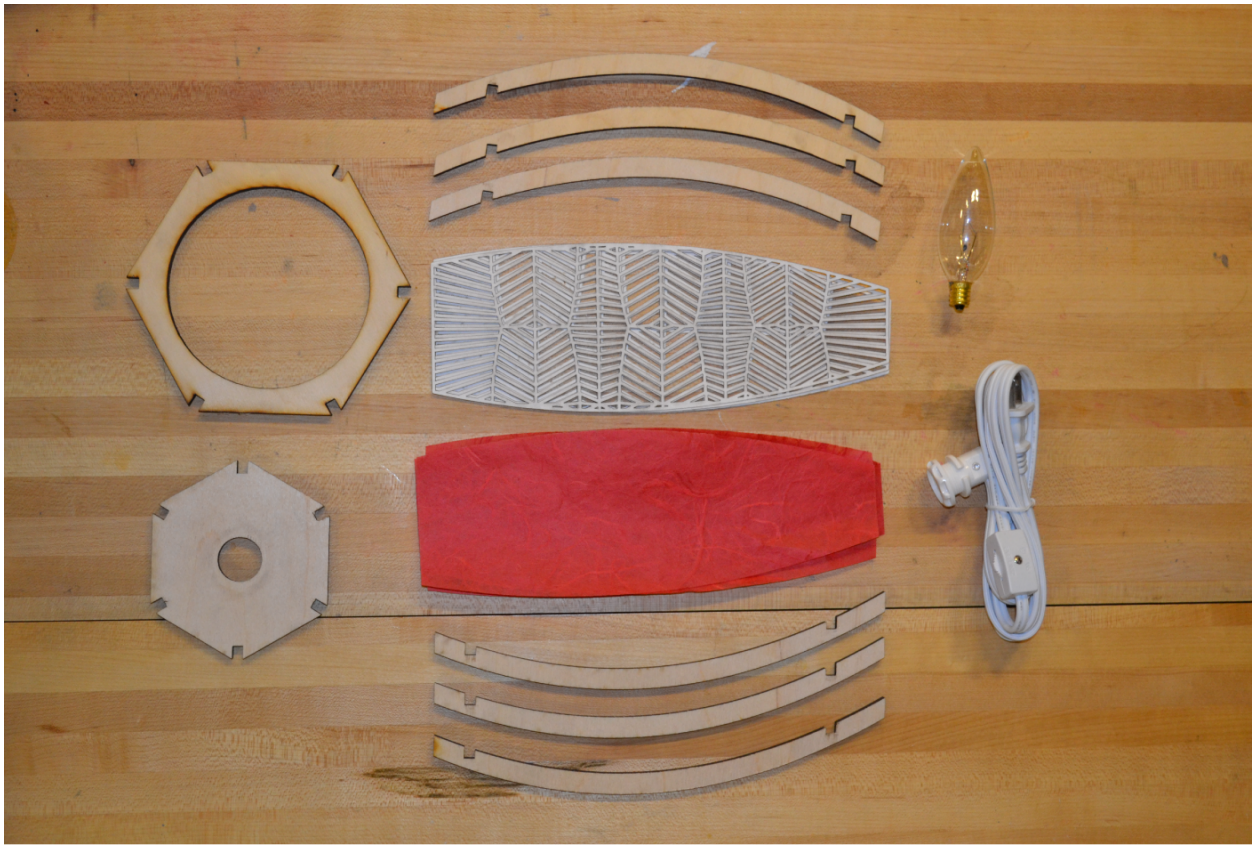


Figure 5-3: the individual parts of a lamp

Codeable Objects was developed as a programing library for Processing and contained a set of pre-defined programing methods that allow the user to describe the lamp, and define the tool paths for all three materials. The first version of the library was somewhat rough. All design took place via textual programing, and keyboard commands. Within the Processing IDE one imports and initialize the controller class of the library, and uses it to call four main functions that determine the height, top width, middle width and bottom width of the lamp. These 4 parameters are used to determine the form of the lamp, by generating the equation of a parabola with 3 intersection points. By rotating this parabola round the y-axis, it was possible to generate a closed 3-dimensional ellipsoid form. The library also provided access to an additional set of methods that control over a number of other parameters in describing the form of the lamp, including the number of sides, the resolution of the curve and the position of the internal structural supports. To facilitate the construction process, notches are automatically generated in all of the individual parts to allow the form of the lamp to be press-fit together. The inclusion of this feature gives the user freedom to customize the shape of their lamp, without having to worry about the mechanics of construction. The library determines the correct position of the notches by calculating appropriate angle for each individual notch and determining the correct edge of intersection for each tool path based on this angle.

Codeable Objects also includes a second set of programing methods that allow users to describe the decorative components of the lamp by specifying coordinates in polar or Cartesian space. Upon compilation, the coordinates are used by the application to calculate a design using a Voronoi diagram. A Voronoi diagram is a geometric subdivision of space that generates a quadrants based on a given point set according to the equidistant boundaries between all the points [4]. When the diagram is calculated, each segment is checked for intersection or containment with the polygon. Segments with both endpoints within the polygon are preserved unchanged, while segments with only one endpoint inside the diagram are clipped at the appropriate edge of intersection, by checking their angle against the angle of the points of the edges of the boundary. Segments which have both points outside of the polygon are checked for intersection using the segment intersection algorithm and either clipped according to their intersection points or removed altogether if they lack an intersection (fig: 5-4).

Once the code is compiled, a graphic preview is displayed. For the pilot version, users could use key-commands to toggle between a view of the form of the 3D form of lamp, the voronoi-diagram pattern, and a 2D preview of the press fit parts (fig:5-5.) A final key-command allowed for the resultant design files to be exported as three separate pdfs, containing the paths for the press-fit frame, the shades, and the pattern files.

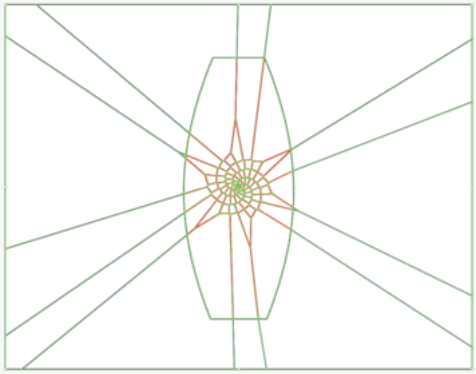


Figure 5-4: Algorithim for constraining the voronoi diagram within the shade

5.3 Evaluation

Using this basic pilot library, the first evaluation of Codeable Objects was conducted with a group of nine graduate students, ranging in age from 24-34, who engaged in a six-hour workshop. Five participants were women. According to self-reported pre-survey data, all but one of the participants were intermediate to experienced programmers. Five of the nine had previous experience with Processing. In contrast, participants indicated they had little or no prior experience in design. What experience they had was primarily gained in high school art classes and college elective courses. During the workshop, each participants engaged in the design and fabrication of a lamp. Participants received programing instruction in the use of Codeable Objects and a basic explanation of the principles behind the geometry of the lamp. The pilot version of Codeable Objects was packaged with a set of example programs that contained the basic code for initializing the library and defining

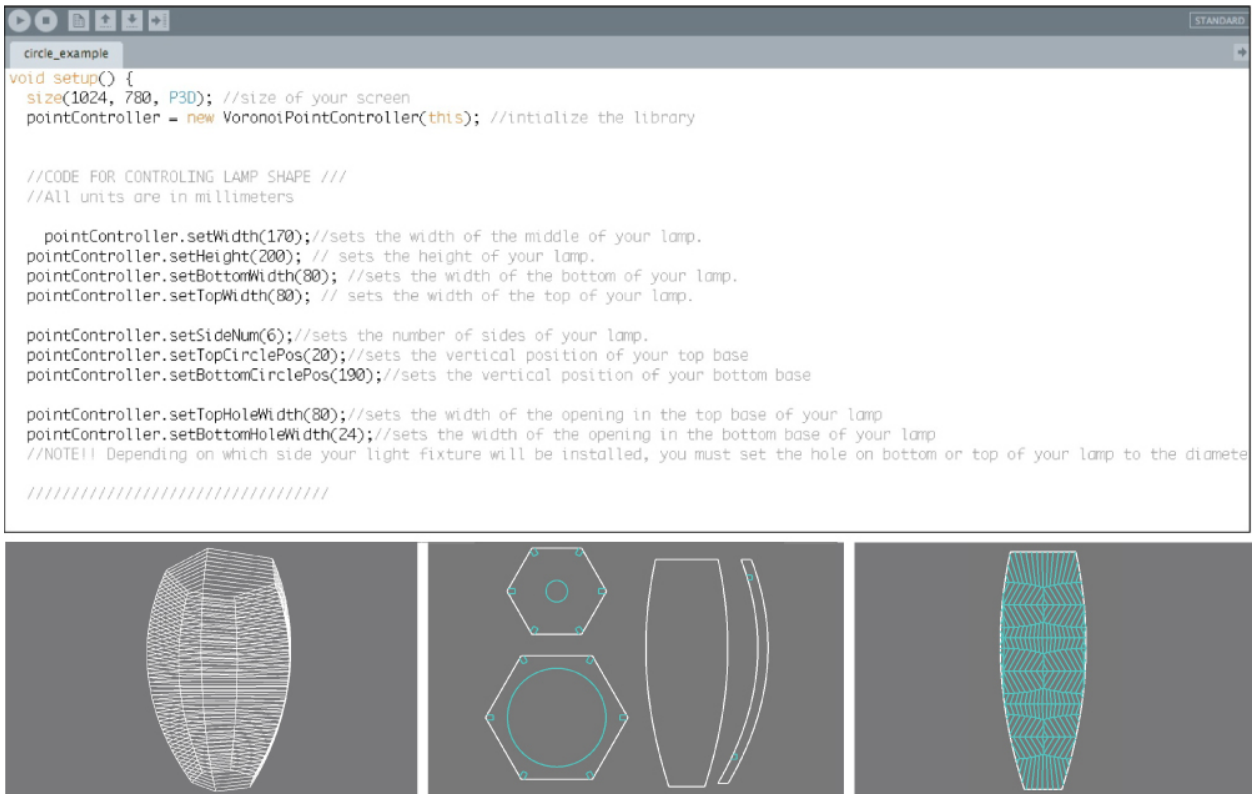


Figure 5-5: The first version of Codeable Objects, with only text-based interaction

the parameters of the lamp, a long with a variety of point generation methods. Examples included algorithms to generate spirals, circles and sine and cosine wave distributions of points. Participants were also provided with access to materials, and received training in the use of the laser cutter. Participants were given approximately four hours to design the structure and ornamentation of their lamp, followed by instruction on and access to the laser cutter. After cutting, participants were provided instructions about how to assemble their lamps.

5.4 Results

All but one of the participants in the Lamp workshop successfully completed their lamp. The one exception was a user who wished to incorporate a specialized light fixture into their piece, but unfortunately damaged her parts while waiting for the fixture to arrive. Participants with little or no prior programing experience primarily relied upon tweaking or remixing the example programs to design the form and pattern of their lamp, whereas those more experienced in programing experimented extensively with the library to produce a wide range of forms and patterns. One participant wrote a program that decomposed a black and white image into a point cloud and used that as the basis

Figure 5-6: Several of the finished lamps from the first workshop

for her pattern. Another participant wrote a program that used a Gaussian distribution of points to achieve the gradual variation he desired in his final pattern.

The physical assembly process required additional time beyond the duration of the workshop for most participants. This can partially be attributed to the bottleneck on the laser cutter, however the design and crafting components of the project took longer than expected. Despite this, all the participants returned after the workshop to complete their projects, and each participants indicated on the survey that they were able to complete a finished product to their satisfaction. The physical objects produced were both attractive and functional; participants displayed their lamps in their offices and homes after completion. One participant returned several days later to build a second lamp so that he would have a matching set for his bedside tables (figure:5-6.)

5.5 Discussion

Because of their prior expertise, the experiences of the majority of the participants in the first study are not indicative of the feasibility of Codeable Objects for novice programmers. Their experiences provide valuable contrast to the experience of the novice coders in the successive workshops however, and provide important information about the usability and workflow of the software. Despite their experience in programing however, the experienced programmers in the lamp workshop exhibited limited knowledge of computational design prior to the start of the workshop. When asked in the pre-workshop surveys how they thought programing, design and craft could be combined, the general response was either uncertain, or as method to create dynamic interactivity, rather than a tool for the design of form and pattern:

"You can combine software and hardware and make craft more dynamic (e.g. sensors). [Lamp Participant pre 1]

"[Programing] gives [you] the ability to make something dynamic. [Lamp Participant pre 3]

Following the workshop, the participants were generally pleased with the creative affordances of the tool, and described how the software enabled them to expand their programing abilities to the realm of art and craft with greater success:

"I think programming makes designing more accessible because you don't have to be able to draw or paint. [Lamp participant post 4]

"I love the idea of being able to combine my interest in programming for creative expressions. [Lamp participant post 6]

There was also an awareness among several participants about the practical benefits of combining computational design and digital fabrication:

"I understand now how programming can be used for quick prototyping and mockups that can be used to inform final design decisions. This is easy [and] helpful when using physical materials where mistakes can be costly. [Lamp participant post 2]

"Using programming in the design process adds some exciting and unique capabilities over traditional design and crafting, including mixing in different algorithm and ideas from other existing software, and rapid prototyping of complex designs."[Lamp participant post 6]

From these responses, it is apparent that even among experienced programmers, algorithmic craft has the potential to expand people's understanding of the applications of programming and motivate them to apply computation to other forms of production and expression. There were also elements of the process and tool that were problematic for the participants. It became immediately clear during the workshop that textual programing was not the optimal method of modifying the form of the lamp. Many of the participants became frustrated about having to set the parameters and then wait for the compilation process to complete before they could view the resulting form. This issue was addressed partly in subsequent versions of the tool by replacing the textual parameters with a set of sliders in the compiled application, which would adjust the form in real time, across each of the views (figure: 5-7.)

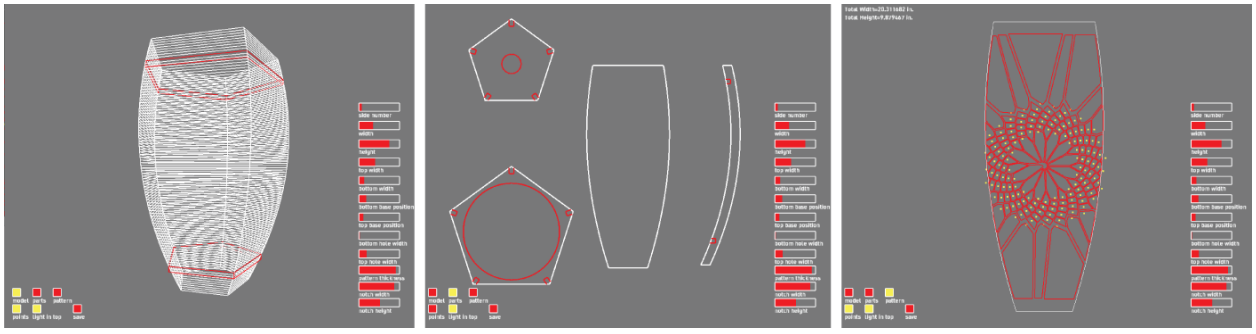


Figure 5-7: Revised graphic view with sliders

The textual programing method proved to be useful in the context of the point specification for the pattern. The simple method of specifying points in a programing context, allowed for the wide degree of variation and approaches in the resulting designs. If the tool had relied on a more standard set of graphical user interface(UI) components, like sliders to control the point generation, it is doubtful that the same range and creativity could have been achieved. On the other hand, it was clear that the less experienced programmers had more difficulty deliberately designing the patterns of their lamps, and relied primarily on adjusting and remixing existing examples.

Several participants also put forth detailed critiques of the programming process, which brought into focus concerns about the practice of computational design itself. One participant reacted against defining the generative qualities of the Voronoi diagram patterns as a design method:

'"Changing the parameters didn't always generate the pattern you have in mind. It was more like generating a few semi-random patterns and you choose one that looks good. It is rather a trying-and-choosing rather than designing /making something you planned to have. I think "design" involves "intention" and "planning." Programming, crafting, and design should be combined in the way that entails prior planning and intentions as opposed to cutting together the semi-random choices, which could be good but I wouldn't call that design. [Lamp participant post 6]"

This comment addresses the concern that the attributes of randomness and generativity do not automatically lead to optimal or good design decisions. Some deciding factor has to play a role in the process, but the designers role in the deciding process is often ambiguous. This criticism touches on a core debate about the role of conscious design and the restriction of intuitive creativity in computational practices overall, however it is particularly relevant to computational design. The emergence of comments like this are encouraging, because they reflect the engagement of the participants, not just with the task at hand, but in a critical evaluation of the creative implications of this form of creation. This comment however also highlights a key restriction of Codeable Objects. While it is ambiguous to the extent at which adjusting the parameters and input values to a system constitutes design, the task of defining the algorithms which shape the system itself are decidedly a form of design. With Codeable Objects however, the user is unable to modify the core algorithms which define the range of forms and patterns that are possible, unless they alter the source code of the library itself. When evaluated as a tool for algorithmic craft, Codeable Objects could have done a better job of supporting some of the deeper components of computational design, in particular, the algorithmic abstraction of personal styles and aesthetics. The stylistic limitations contained in the tool most likely contributed to the high success rate in project completion, and the general attractiveness of the resulting projects, but the experience of the workshop, provided the motivation for future tools to have better balance of stylistic and computational openness and accessibility for new programmers.

One other defining component of the Codeable Objects pilot workshop was the stark contrast

between the nature of the challenges in the computational design and digital fabrication components and the crafting component. The difficulties people experienced while designing and fabricating their projects were often discrete, for example correcting for mathematical error in coordinate placement, or having the incorrect setting on the laser cutter. More complex problems sometimes arose in these contexts as well, such as confusing on the principles behind some of the more complex point generation algorithms, or the programming aspects in general, however they were seemingly aspects that could be addressed through verbal instruction and explanation. The challenges encountered in the crafting session were of a different quality, concerning the best techniques for assembling the parts so that the resulting product maintained an attractive appearance. Most participants were surprised at the amount of time required to complete the physical assembly, and were often frustrated when variations in the crafting process violated the precision and perfection of the digital design, and laser cut parts. Some of the frustrations in the physical construction process were addressed in subsequent workshops by creating a paper variation of the lamp that was faster and easier to assemble and required no gluing (figure: 5-8.) In addition, a feature was added to the software which to report the approximate material size required for a design, so that users could ensure their would fit on the bed of the laser cutter.

Figure 5-8: The revised paper lamps

Although there is often an opportunity to improve a user's experience through improvements in the interface and artifact design, in craft, it is both impossible and undesirable to eliminate the properties of material variation, and the benefits of practice and experience. Many of the difficulties workshop participants experienced in the crafting process therefore do not reflect a failure in the tools, but rather challenges that are intrinsic to crafting, and best addressed through practice and familiarity with the materials. In this way, craft practice differs from computational practice. While approaches benefit from experience and practice, the approaches for solving problems differ significantly in programing and hand crafting. Programming often requires an analytical approach with an emphasis on consistency and regularity, whereas crafting requires a more intuitive process of responding and adjusting one's technique while in direct contact with the materials. An interesting question is

then, how then, can algorithmic tools be presented in a manner that accustom users to operating in both discreet and intuitive modes of problem solving, and furthermore how can these two modes of working inform both craft and computational approaches simultaneously?

Chapter 6

Soft Objects

After an evaluation of the successes and limitations of the Codeable Objects library I made an effort to expand the library in a way that would allow for a broader range of computational design approaches and end products. In particular, I was interested in exploring the domain of algorithmically crafted garments and fashion accessories. To explore computational fashion design in the context of algorithm craft, I expanded the Codeable Objects tool into a more general programing library named SoftObjects and evaluated it over a 10 day workshop with young people.

6.1 Motivation

Fashion is an exiting domain to connect to computation, because it appeals to groups of people who are often under-represented in computer science, particularly women and girls. In addition, because garments and accessories are wearable, computational fashion design requires the programmer to consider questions of comfort, sizing and personal taste and and style; a set of concerns not often associated with most computer programs. With the growth public awareness of digital fabrication, there is particular enthusiasm for the wearable applications of emerging digital fabrication technology. Much of this excitement is directed towards 3D printed wearables and textiles. In July 2010, Iris Van Herpen released her Crystallization collection, which featured her first computationally designed, 3D printed piece, marking the first time a 3d printed garment had appeared on the runway [2]. Herpen and many other fashion designers have continued using 3d printing as a medium for fashion since then. As a result, often in popular culture, computationally designed, digitally fabricated fashion is often synonymous with 3D printing. The 3D printed garments and accessories produced by professional designers like Van Herpen serve as wonderful inspiration for the future of digital fabrication, as they produce garments that would be impossible to fabricate through any other means. For the average individual however, computationally designed and digitally fabricated garments of this nature present significant limitations. Given current technology and material limitations, the majority of 3D printed garments are generally practical for every-day wear and require advanced fabrication techniques that are unavailable to the average designer. Garments like the N.12 bikini, designed by Continuum [**?**] are designed to be more practical, and available to consumers, however they still come at a steep price point ($300 for the bikini top). Most importantly however, the construction of 3D printed garments of this nature, both in terms of materials and techniques, appear to have little in common with methods of traditional garment production, such as sewing, knitting and embroidery, and therefore offers few entry points to individuals with traditional skills to move into this space.

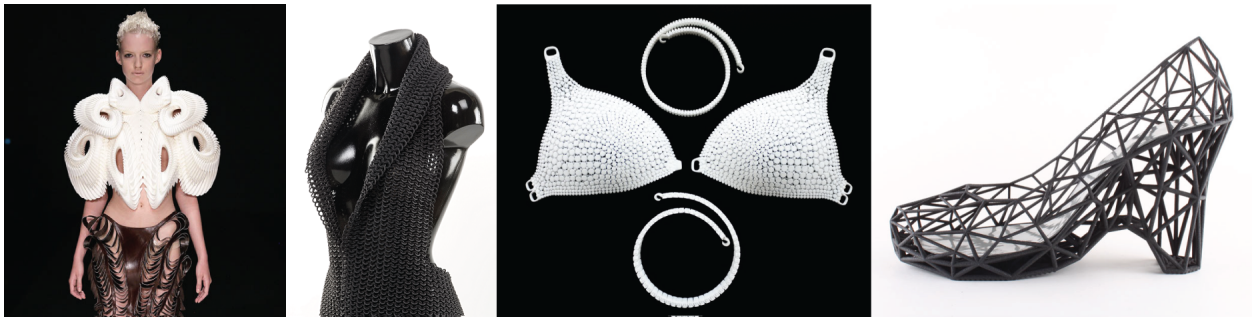


Figure 6-1: 3D printed fashion (from left to right: Crystallization 3d Top by Iris Van Herpen, Drape Dress by Janne Kyttanen, N12 Bikini by Continuum Fashion, Strvct shoe by Continuum Fashion

Although perhaps less publicized than 3D printed fashion, other other designers are merging fashion with computation and digital fabrication in more accessible forms. Diana Eng's Laser Lace tee collection contains laser-cut machine-washable t-shirts with floral-inspired iconography, and her Fibonacci scarf is created through traditional knitting techniques, meshed with a Fibonacci knit pattern. Eunsuk Hur's modular fashion pieces are inspired by tessellations and fractal geometry, but apply these structures for a practical end as well. By creating garments through laser-cut interlocking pieces, Hur's aim was to produce items that were robust and durable, also gave the user the opportunity to use their inner creativity to come up with new and interesting items, by rearranging the individual components (figure:**??**.)

Examples such as these demonstrate a space in computational fashion design that is open to a broader range of participation and compatible a larger set of interests, and skillsets. In general, less novel fabrication machines, like laser cutters and vinyl cutters are dominant in this type of work, because they afford a wider range of materials and can produce garments at a much lower cost than 3D printers. This range of materials also translates to a wider set of possibilities for aesthetics and styles than most forms of 3D printing. It should be noted that accessible forms of 3D printing can produce compelling wearable objects, however they are generally on the scale of jewelry and small accessories. As a whole, garments designed and produced in this fashion fit more so with the parameters algorithmic craft, than the 3D printed high-fashion examples provided in the preceding paragraph. With this distinction in mind, I developed the Soft Objects programming library, and designed and conducted a workshop to explore the creative potential of computation and digital fabrication in the context of accessible and immediate fashion design.

Figure 6-2: "Ready to wear" computational fashion (from left to right: Fibonacci Scarf by Diana Eng, Biomimicry laser-cut bracelet by Stefanie Nieuwenhuyse, Laser Lace All-Over Tee by Diana Eng, Interstice bracelet by Nervous Systems, Modular Fashion by Eunsuk Hur

6.2 Tool description

The Soft Objects library contains a set of methods that allows users to draw shapes and patterns and then export those shapes and patterns in a vector-file format that is compatible with x-y axis digital-fabrication machines. Similar to CodeableObjects, to use the library, a user imports it into the Processing environment and then writes and compiles code using the Processing editor. Soft Objects allows users to define and manipulate basic geometric primitives such as Points, Lines, Curves and Polygons. These primitives can then be collected within Pattern and Shape objectsstructures designed to capture surface decoration and 2D structure, respectivelyto form increasingly complex designs.

Soft Objects is formulated on an Object Oriented Programming (OOP) paradigm, which lets users create and manipulate collections of geometric primitivesPatterns and Shapes. This structure differs from Processings drawing API, which uses a functional programing approach. The structure of Soft Objects enables users to simultaneously apply transformations to all of the elements in a collection that make up a complex pattern or shape. It is also possible to import scalar vector graphics files (SVGs) to incorporate pre-drawn designs as elements within a pattern or as a container for existing patterns.

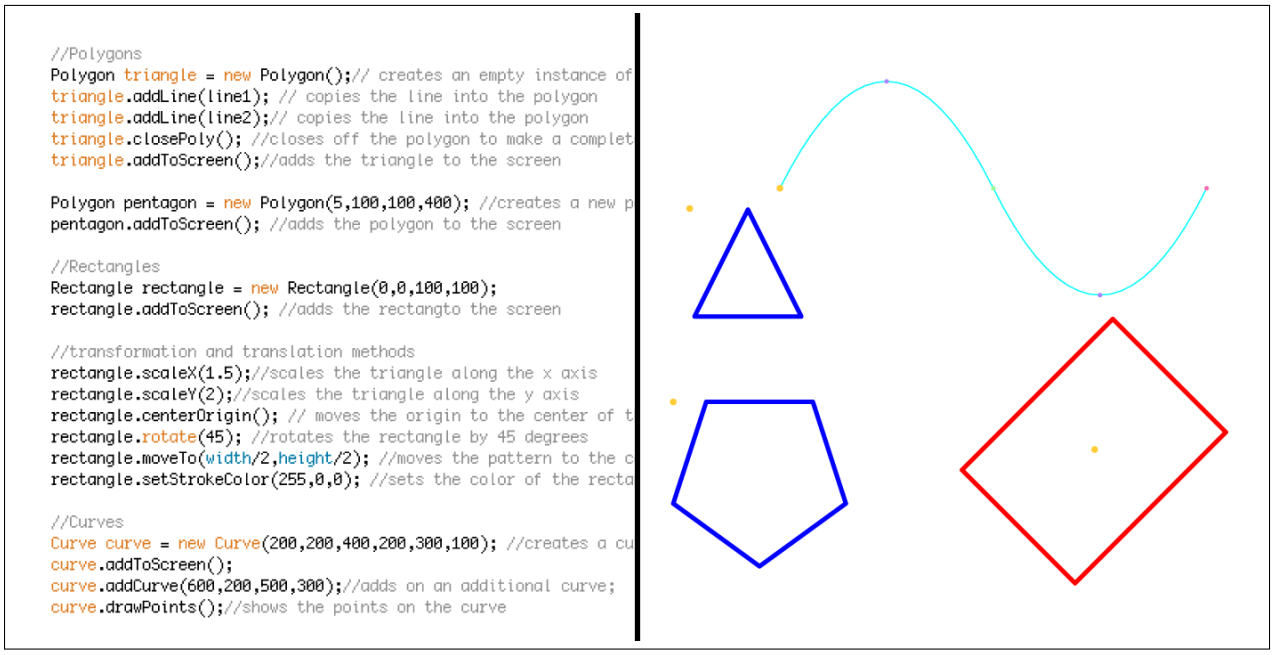


Figure 6-3: Soft Objects primitives

Users are presented with a 2D preview of their designs when they compile their code. Soft Objects supports a variety of digital-fabrication machines by allowing users to save designs to vector

portable document format (PDF) files. PDFs can be used by different production tools, including ink-jet printers, vinyl cutters, laser cutters, and computationally controlled embroidery machines. Output from Soft Objects can be fabricated on essentially any x-y axis tool. 3D structures can be created by assembling fabricated pieces. Figure 6-4 demonstrates the workflow from code to a finished object. The Soft Objects library also contains a collection of pre-defined algorithmic patterns that can be initialized, including Voronoi diagrams, Koch curves, and L-Systems, and an extensive set of example programs that users can modify and combine to produce individual results.

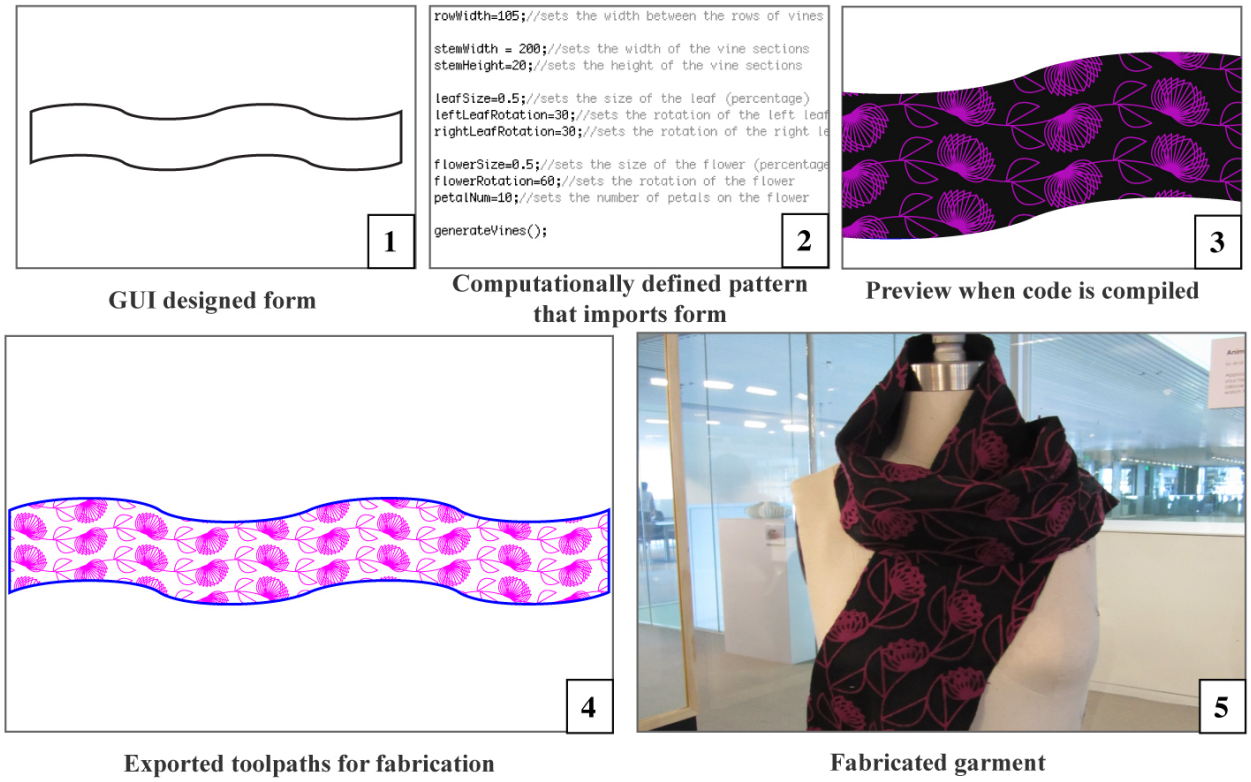


Figure 6-4: Soft Objects workflow

6.3 Workshop

The evaluation of Soft Objects was conducted during a 10-day workshop with a representative group of participantseight young adults, aged 11-17, 75% male and 25% female. A significant majority (88%) stated in pre-surveys that they had little or no prior experience in programming, and only one participant had prior experience in Processing. All of the participants indicated some level of prior experience in art, design, or craft. Most attributed their design or craft experience to art or drawing classes.

The workshop was conducted at the Nuvu Magnet Innovation Center for Young Minds. Participants were given 10 days to conceptualize and construct a garment using a combination of computational design, digital fabrication, and traditional sewing and crafting. The second study was more open than the first; participants could produce any type of garment they wished as long as components of it were computationally designed and digitally fabricated. During the workshop, participants were introduced to Soft Objects and the concept of computational fashion through a multi-step process that engaged participants in different levels of programming through the construction of different garments and accessories. First, participants were provided with a small set of example programs similar to the lamp workshop. This step allowed them to manipulate a core set of parameters to generate the pattern and form of a scarf, which they then cut on the laser cutter (figure: 6-5.)

Figure 6-5: Completed garments (from left to right: octagon dress, flag pants, samurai dress, viral sweatshirt)

Second, participants were instructed in a number of primary programing concepts, including iteration, function definition, and the use of variables and primitive data-types. During this instruction, participants were guided through the process of independently using Soft Objects and generating their own programs from scratch. They used these programs to create a design for a wooden bracelet (Fig. 3), which was then laser cut and assembled. After these two initiation activities, these participants were asked to conceive their own garments and provided with the resources to design, prototype, and craft finished garments.

6.4 Results

Participants in the fashion workshop were successful in using programing and digital fabrication to design and produce finished garments. During the initiation activities, participants independently wrote and compiled programs of their own and produced physical products based on the design generated from that program. Furthermore, with assistance from the instructors, the participants were able to apply more sophisticated programing methods to produce a diverse set of final products

(Fig 4). One pair of students developed an armor dress by writing a program that geometrically described a single scale shape, imported a dress pattern from Illustrator and filled it with rows of scales that corresponded with the dimensions of the dress. Another pair created a geometrically inspired dress with a patterning of different-sized octagons and squares that were laser cut from starched fabric. Another student created American-flag-inspired pants using a program that generated random orderings of red and blue stripes on a white background. One group that was less interested in the process of sewing clothing created a program that generated a recursive virus-like pattern and then screen-printed the pattern on pre-made sweatshirts and t-shirts.

Figure 6-6: Bracelets and scarves from preliminary activities

On the post survey, when asked if they were able to complete a finished project to their satisfaction, 100% of the fashion participants responded yes. The resultant garments were attractive and functional, indicated by the fact that participants from the fashion workshop kept and wore their creations. Direct comparison of the pre-and post-workshop surveys also demonstrated that on average, participants in the fashion workshop indicated their interest in crafting increased after the workshop, as did their enjoyment of the design process. Eighty-eight percent of the participants in the fashion workshop indicated that they felt more comfortable programing after the workshop than before.

6.5 Discussion

The Fashion workshop occurred over a longer time period than the Lamp workshop, and explored a wider range of crafting and computational techniques. As a result, I had an opportunity to spend greater amounts of time observing and talking with the participants. leading to a more comprehensive perspective on the benefits and challenges in conducting algorithmic craft workshops.

In addition, because the participants were nascent programers, their experiences better reflect those of the target demographic of this research. Through this workshop we confirmed that algorithmic craft activities can actively support the expression of personal identity in a positive setting. It can also foster feelings of confidence in programming and support aesthetic and technological literacy. The workshop also promoted a deep understanding of computation as evidenced by critiques of the participants as well as demonstrating the importance of physical prototypes in the design process. Finally, the workshop promoted a sustained engagement in programing. The evidence of these outcomes is discussed in greater detail below

6.5.1 Identification as a programmer

Similar to the participants in the Lamp workshop, the Fashion participants began with vague ideas about the applications of computation. When asked in the pre-workshop surveys how they thought programing, art and craft could be combined, participants responded in writing in the following ways:

"To be honest, I am not sure too sure how it would all be combined because I don't know much about programming." [Fashion Participant Y]

"With programing, we can make programs do things for us. "[Fashion Participant J]

In addition, the participants had almost no prior knowledge of computational design. Those participants who had some form of prior programing experience associated it largely with interactivity and actuation:

"I would love to combine things like T-shirts and speakers or other different types of technologies. " [Fashion Participant J]

Following the completion of their final projects, participants were much more descriptive about the applications of programing:

"I think [programing and fashion] are really interesting but I never thought they could ever be together in one concept, and its awesome that I know that now- that you can design aesthetically pleasing things from coding." [Fashion participant K]

"Ive never thought of programing as physical, I thought it was only in computers. But then when we made the scarves and stuff, I thought that was really fun." [Fashion participant M]

In addition, they expressed a growing confidence in their ability as programmers. During the interviews, several separately stated that that while they did not feel completely comfortable programing independently, the experience made programming feel significantly more accessible:

"For someone who never had any programing explained to them before, when you look at [computational drawing examples] it feels really inaccessible, but now that Ive been taught a little bit of [programing], I can kind of crack at the walls a little bit and understand how it works, and that makes it more accessible to me." [Fashion participant S]

"Even though now Im not really a Processing expert now, Ive just experienced it and its not as scary to me, like the idea of coding, you just kind of have to learn some stuff and practice it more, but I think I definitely understand the concept of it. [Fashion participant K]"

Finally, the programmers in the fashion workshop expressed feelings of pride and a sense of accomplishment in their new-found programing skills:

"Im actually kind of proud. I know whats going on. It feels differentI just thought [programming] was just about these huge programs that you have to piece together, and that you have to be really, really smart to do it, but I can do it." [Fashion participant P]

"I think it was really cool that we used [programing] for fashion, cause I think a lot of people might think people who do fashion arent really smart or something, and then they think that people who design code are like brilliant coders and can do really awesome stuff with it." [Fashion participant K]

The confidence, sense of belonging, and personal agency demonstrated in these comments stand in contrast to popular views of programming as specialized and inaccessible. These sentiments were also present in later workshops I conducted with other nascent programmers (discussed in the section 7), indicating that introducing programing in the context of algorithmic crafting not only has the potential to change peoples understanding of the relevance and applications of programing, but also promote a personal awareness of technological literacy and competence.

6.5.2 Application of computational affordances

Aside from a general understanding of the potential applications of programing, participants were successfully able to leverage the advantages of computational design in their final projects. For example, the octagon dress used parametric design principles as the participants were able to change the size and orientation of the octagons in the dress by modifying several parameters at the start of their program to affect the entire pattern, rather than rotating and adjusting each shape independently. The samurai dress took advantage of the computational properties of precision and automation. With help from an instructor, the creators programmatically generated an individual vector file for each row of scales of the dress, expediting the production process (figure:6-7.)The samurai dress is particularly interesting because it shows a remarkably successful transition from the design expressed in original concept art by the participant, and the resultant garment. In later workshops it was often difficult to reconcile the sketched concepts with the qualities of computational design. Lastly, the viral shirt, demonstrated an application of generativity. The aesthetic of the viral pattern was produced through the use of a weighted random-number generator to determine the number and length of the branches in each recursion of the pattern. Although the implementation of the weighted number generator was facilitated through the help of the instructor, the participants came up with the idea of using it on their own.

Figure 6-7: Progression of samurai dress (from left to right: concept sketch, computationally generated pattern,laser-cut components of final garment)

Aside from conceiving of and applying computational-design approaches, participants demonstrated an understanding of the rationale behind the methods they were using. One of the participants in the armor dress project compared the process of programing the design of the dress to that of manually drawing it:

"With drawing you can achieve everything programing can, but I would prefer to program it. [Programing] can be pretty convenientthe computer is helping me. Like if you want to make pizza, the computer is like a pre-made crust."[Fashion participant E]

Participants also demonstrated an understanding of specific programing functionality. One participant described the point at which she understood the application of parameterization:

"One moment that stuck out was when you helped me make a code with original geometry that could be changed so that when you changed one thing it changed everything and that was cool because I felt like I actually made something that could be changed and then applied." [Fashion participant K]

The ability to understand and describe how computational support one's creative objectives is essential in motivating an individual to spend time learning and implementing these methods. Once the aesthetic possibilities of the recursive viral pattern were apparent to the designers, they became engaged in better understanding the underlying algorithm, so that they could produce a pattern to their exact specifications. Conversely the participants were far less engaged during several of the introductory lessons when we introduced several elementary computational principles, such as defining data types, where it was unclear what the immediate design application was. Algorithmic craft presents the opportunity for participants to tackle complex computational problems with sophis-

ticated approaches, however the problems themselves must be clearly grounded within the design objectives of the individual. One of the challenges in engaging new programmers in Algorithmic craft therefore, is in selecting programing approaches that allow for compelling aesthetic and design possibilities, but are also approachable for first-time coders.

6.5.3 Aesthetics and Identity Expressed through Code

The fashion workshop provided the opportunity for participants to use computational aesthetics as a way to express their visual identity. Fashion can serve as a means of self-expression and for conveying ones identity. Discussions of fashion conducted with the participants of the second workshop indicated that participants were aware of the connection between fashion and identity and were eager for opportunities to create clothes that expressed their style. As a result, the majority of the garments created in the workshop contained an expression of the fashion sense of the participants who created them. For example, the participant who created the flag pants was very explicit that the pants have some form of an American flag motif, but not resemble the traditional, and as he put it tacky flag pants that he commonly saw. He wanted his flag pants to be as he put it something that he would actually want to wear. His programing choices were made in direct consideration of his desire to create a pair of pants that he felt were fashionable.

When asked about the experience of making and designing his pants he said:

"[The workshop] definitely changed my impression of making clothes, I thought it was pretty quick to make clothes, but it actually takes a long time, and its also really fun. I love the fabric I made." [Fashion participant M]

His enthusiasm also was evident in the fact that after the pants were complete, he tried them on and wore them for the remainder of the workshop. This level of enthusiasm was common among participants; they all proudly modeled their creations, and many of them wore them home. This behavior suggests a relationship between the decisions made in a programing context, and the participants desires to express their visual identity. The participants were selective in the code they wrote to design their garments because they intended to wear the garments, and as a result, be represented by them. This powerful affective relationship between computation, design and self-expression provides a natural way to engage people in programing and design by supporting their personal interests.

6.5.4 Physical and Digital connections

One of the challenges of Algorithmic Craft, alluded to in the Codeable Objects discussion, is that the practitioner must work between digital designs with physical materials and processes. Physical prototypes often serve as a key point of transition between these spaces. In the fashion workshop, prototyping played an important role, and demonstrated how computational tools can support and

sometimes hinder the prototyping process. The focus on fashion made it possible to supply the participants with large amounts of inexpensive test fabric. The laser cutter could cut fabric much more quickly than thick materials, which allowed participants to produce numerous prototypes of their projects before creating a final piece. Most groups produced two or three prototypes, with one participant creating six iterations of a single jacket. This rapid production process formed a direct connection between discoveries made in the physical prototyping space and decisions in the programing realm. In the case of the octagon dress, (figure:6-6), the participants first cut test rows of octagons to determine the appropriate scale, then adjusted their design by modifying their program. When they had cut out a second more complete version of the dress, they rotated one of the shoulder straps on the physical prototype and formed an idea for a one-sided shoulder strap. They implemented this design change in the digital version of the dress by making additional changes to the size and rotation of the shapes defined in the code. When asked about this process in the interview one of the participants said:

"I think it was really fun that we got to do a prototype first because then if you dont like it, you dont feel a lot of pressure because you can make it again really fast, and theres no stress because if it doesnt turn out well, then its not your final project. [Fashion Participant K]

The combination of programming, rapid fabrication, and physical construction allowed for a design approach that transitioned from programing to fabrication to programming adjustments based on the fabricated elements, and then back to fabrication. This iterative approach resulted in a closely linked cycle of physical and digital engagement.

Despite these positive results, many participants struggled with prototyping. These struggles were evident in practical aspects, such as participants not saving their programs and digital design files to come back to later(despite repeated reminders from the instructors to do so). Participants also frequently spent too much time on assembling their early prototypes and were frustrated when they realized they would have to make design changes and repeat some of the manual labor. Although we took pains in the workshop to introduce participants to the concept of prototyping, these instructions were not always absorbed. This may present an opportunity for future algorithmic crafting tools to contain specific features to encourage and support the physical prototyping process. One possibility that is often proposed in the HCI community is to include simulation tools that preview the constraints and behaviors of physical materials. The value of working with physical prototypes should also be supported however, by features that allow the designer to scale their files and fabricate first in miniature, and software that allows easy access and management of multiple versions of a design throughout the design process.

6.5.5 Enthusiasm in crafting and coding

One of the most encouraging aspects of following the fashion workshop, was the participant's enthusiasm and desire to continue making. Participants talked extensively about what they would like to make in future with programming, consisting citing that they would like to continue making clothing, or other personal functional items like furniture and things they could use around the house. The experience of both sets of workshop participants also demonstrated the ability of these techniques to produce objects that were designed to complement personal items and living spaces. When asked what she would like to make if she continued to program, one participant responded:

"Things like were making now, things that you would want to keep or use, things that look nice as opposed to like computer games, or input-output devices. I think those are fun, but its not as cool as things that you can hold in your hand. I actually hung up the scarf I made in my room, and now I can be like I made this on Processing and people will be like what? Its cool! "[Fashion participant K]

This enthusiasm, combined with the high potential for individual expression and sense of accomplishment encouraged us to continue exploring fashion and garment production as topic space for algorithmic craft. While the fashion workshop highlighted many positives in this sense, we also encountered several areas for improvement.

6.6 Limitations

The most evident barriers in the Fashion workshop involved the syntactic challenges of programming. Many participants expressed a frustration with the syntax in both surveys and in-person interviews. Although the workshop participants were able to generate their own programs, they required more assistance from an instructor to write some of the commands. In addition, a feeling of needing to memorize programming syntax frequently translated to a sense of frustration. One participant stated in an interview:

"I couldnt memorize things, so it also was frustrating for me to always have to get you to help me write the code." [Fashion participant K]

Many people requested some form of written "cheat sheet" that listed the key methods and how to use them. They also pointed out that you often had to write a lot of code (such as import commands and setup and draw functions), even for simple tasks. Writing code for the first time is always challenging, however, the high levels of frustration registered by the participants often focused on aspects of programming that seemed extraneous to design. Because we wrote Soft Objects as a Processing library, it required that the syntax correspond to Java, which is a difficult language for beginners. Because Java is a general application language, it has many syntactic requirements that are unnecessary for computational design applications. Based on this difficulty I concluded that

future algorithmic crafting tools should explore domain-specific languages that directly applied to design and fabrication, and were better suited for first time programmers.

Along with difficulties with the programing syntax, participants struggled with some of the post-processing techniques. In order to be suitable for digital fabrication, many of the participant's computationally generated designs required some processing in Adobe illustrator. Usually this required using illustrator's shape boolean functionality to merge shapes or expand outlines so that the vector paths would correspond to the desired cut pattern on the laser cutter (figure:6-8.) Although the methods to perform these operations were simple, they needed to be repeated every time the design was modified programatically. This was not only inefficient, but sometimes prevented people from determining if their designs were feasible for fabrication when they were in the programing environment. This difficulty encouraged me to focus on ways of removing the need for illustrator from the process altogether so that all design modifications could be initialized and updated programatically.

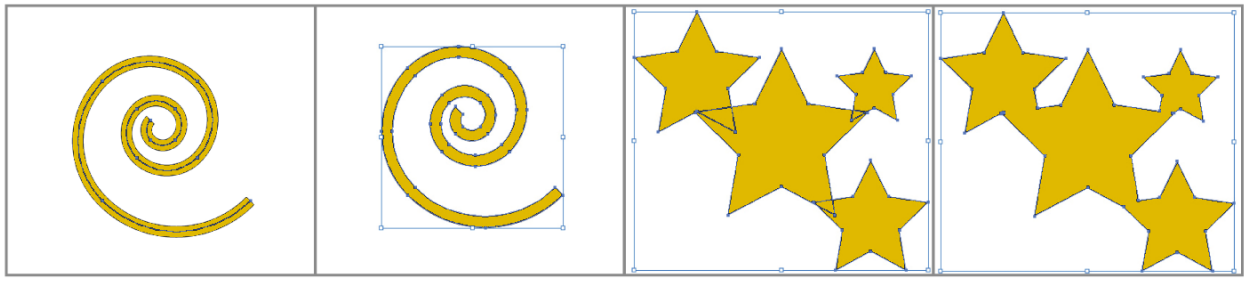


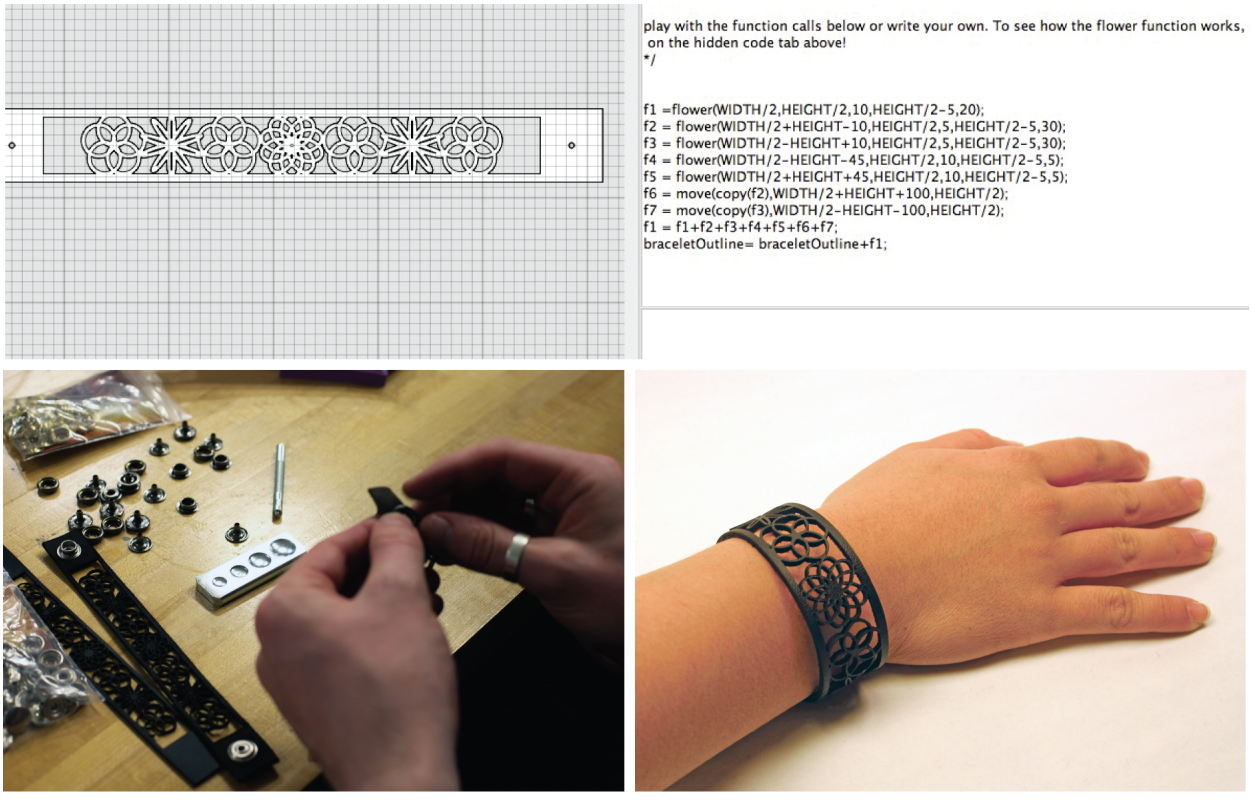
Figure 6-8: Illustrator shape booleans - the laser cutter would cut along the blue line (from left to right: vector path, expanded vector path, group of individual vector polygons, merged group of vector polygons)

Finally, as in the Lamp workshop, participants were frustrated by the delay between adjusting their code and seeing the results. Although, not an absolute solution to the challenges in learning programing syntax, a programing environment with more immediate feedback could assist with the issue of syntactic challenges by providing novice users with improved ways to visualize the effect their syntactic changes have on their design. The implementation of background compiling, the process by which code is automatically compiled and executed as changes are made, has been applied successfully in several tools for novice programmers, including Scratch and Alice, and more recently with Khan Academy [**?**]. Our reliance on the Processing for Soft Objects made the incorporation of real-time compilation infeasible, however it quickly became a goal for future tools.

6.6.1 Critiques of computational design

Chapter 7

DressCode



The preliminary work of Codeable Objects and Soft Objects clearly demonstrated that algorithmic craft offers a compelling opportunity for personal, creative expression through programming. My goal following these projects was to address some the limitations present in these preliminarily projects by developing a stand-alone programing environment and design tool. The resulting software, DressCode is a tool developed expressly to support new programmers in open-ended casual computational design for digital fabrication. To evaluate Dress Code, I conducted two separate day long workshops, one with experienced programmers and designers, and one for young people who

were new to programming. I also worked with FUSE, an out of school STEAM exploration program to develop a set of online activities using DressCode, which is currently in the preliminary stages of evaluation.

7.1 Design principles

Based on the success of the prior fashion workshop, and my continued interest in exploring applications that appeal to women and girls, I decided to focus on computational fashion design and the fabrication of wearable artifacts in the development of DressCode. After reflecting on the potential limitations of this focus however, I decided to develop DressCode as a more open-ended computational design tool that could support the creation of a variety of artifacts, rather than just fashion. To preserve the emphasis on fashion, the majority of the example projects and artifacts I created with DressCode for this thesis were fashion-oriented. The workshops I conducted, as well as the curriculum I helped develop also had an emphasis on fashion or wearable artifacts.

By building my own software, I sought to improve on the preliminary programming libraries in three key areas. First, DressCode contains its own programing language. The DressCode language has a simplified syntax and contains a limited set of textual programming methods, allowing people with little-to-no prior programming experience to working with the language quickly and effectively. Second, the DressCode environment is designed to equally prioritize textual programing and visual design and manipulation. To that end, the software has a two-panel development environment that displays a graphic two-dimensional (2-D) rendering of the users current design in the first panel, and their code in the second. As a user makes changes to their code, the effects on the design are rendered in the graphic panel. Finally, DressCode contains functionality that is specific to digital fabrication and algorithmic craft. The tool allows for a variety of methods to translate one's design to a tool-paths for fabrication machines. In addition, the API contains programing methods that support the creation of forms and patterns that are suitable for fabrication, with minimal effort on the part of the user. The goal of designing a software 2-panel programing and design environment, with a specialized programing language and immediate support for digital fabrication was to assist non-programmers in independently making design decisions with programing. In short, I wanted to make it as easy as possible for people to decide on their own desired style or aesthetic, and then realize it by writing their own code. The following section describes the features of the DressCode software in greater detail.

7.2 Tool description

DressCode is a programing and graphical design environment that enables the creation of 2D designs for 2-axis fabrication machines. DressCode is composed both of the application itself, and a custom programing language designed to correspond with the application.

7.2.1 Interface Design

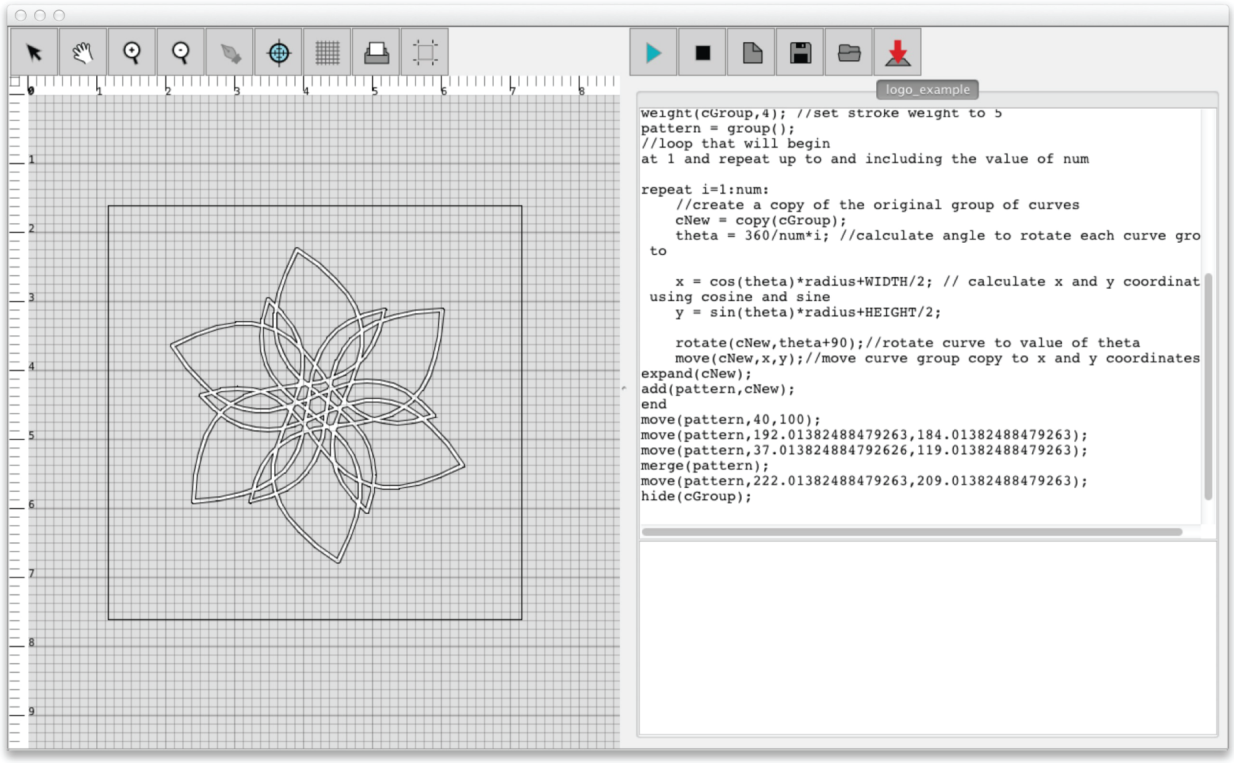


Figure 7-1: The DressCode interface

The interface of DressCode is divided into two sections, a design panel and a coding panel. The divider between the two panels can be resized as needed. The coding panel contains a primary window for entering text, an output console for print output and error reporting and a row of buttons on the top. When the play button is pressed, the program in the code window is run and the resulting design is displayed in the design panel. The first version of DressCode ran the code each time the enter key was pressed or a new statement was completed, however we received many requests from our initial user tests for manual control of the run process. The stop button attempts to terminate programs that are taking too long to run.The following three buttons allow for the creation of new programs and the saving and opening of existing programs. The final button opens

a dialog that enables the user to select an Scalable Vector Graphics (SVG) file to import into their script (the rationale for this is described in 7.2.2 section below.)

The design panel is primarily composed of the drawing canvas. The drawing canvas has a set of rulers and a grid, along with a black rectangle in the center which serves as the drawing board. The drawing board defines a reference for the coordinate system of the canvas, with the upper left hand corner corresponding to (0,0) in cartesian coordinates. Designs can be drawn on any part of the canvas, including outside the drawing board, however the exported designs file-dimensions will always correspond to the size of the drawing board. The right-most button in the design panel allows for the specification of the units of the project, and the resizing of the drawing board. The print button opens a dialog that allows the user to export their current design in a vector format. I experimented with various vector file formats including DXF, SVG and PDF, however for the time being, I settled on PDFs as they were compatible with the specific fabrication machines I intended to use in my workshops.

The grid button allows the user to toggle between a grid in the units that correspond to the current units of the project (either millimeters or inches), pixels, or no grid. The target button allows the user to graphically select a set of coordinates and have them appear in the programming window as text, and can be used like an eyedropper for pixel coordinates. The plus and minus magnifying glasses and the hand tool are used to pan and zoom in and out of the screen. Lastly, the arrow tool allows for design elements to be graphically selected and manipulated. After a design element is moved with the arrow tool, with the corresponding code for the move appears in the programing window.

7.2.2 Programing Language

The DressCode programing language is interpreted with semantic functionality that is simulated through a Java-based library. We relied on the ANTLR framework to generate the necessary lexing and parsing methods for the language, and developed the semantic functionality using java and the java openGL (JOGL). When a program is run in DressCode, the raw script is first tokenized and then parsed to generate an abstract syntax tree (AST). During this phase, all user-generated function definitions are stored in memory. If any parsing errors are encountered, they are output to the console as "compiler errors". Assuming the parse is successful, DressCode then walks the resultant AST and attempts to execute the semantic functionality of the program (figure:7-3.) After being executed, the resultant design is rendered on the display panel. Any runtime errors are displayed in the output console. For most programs, this process is instantaneous, however some programs with complex operations require several seconds to be executed.

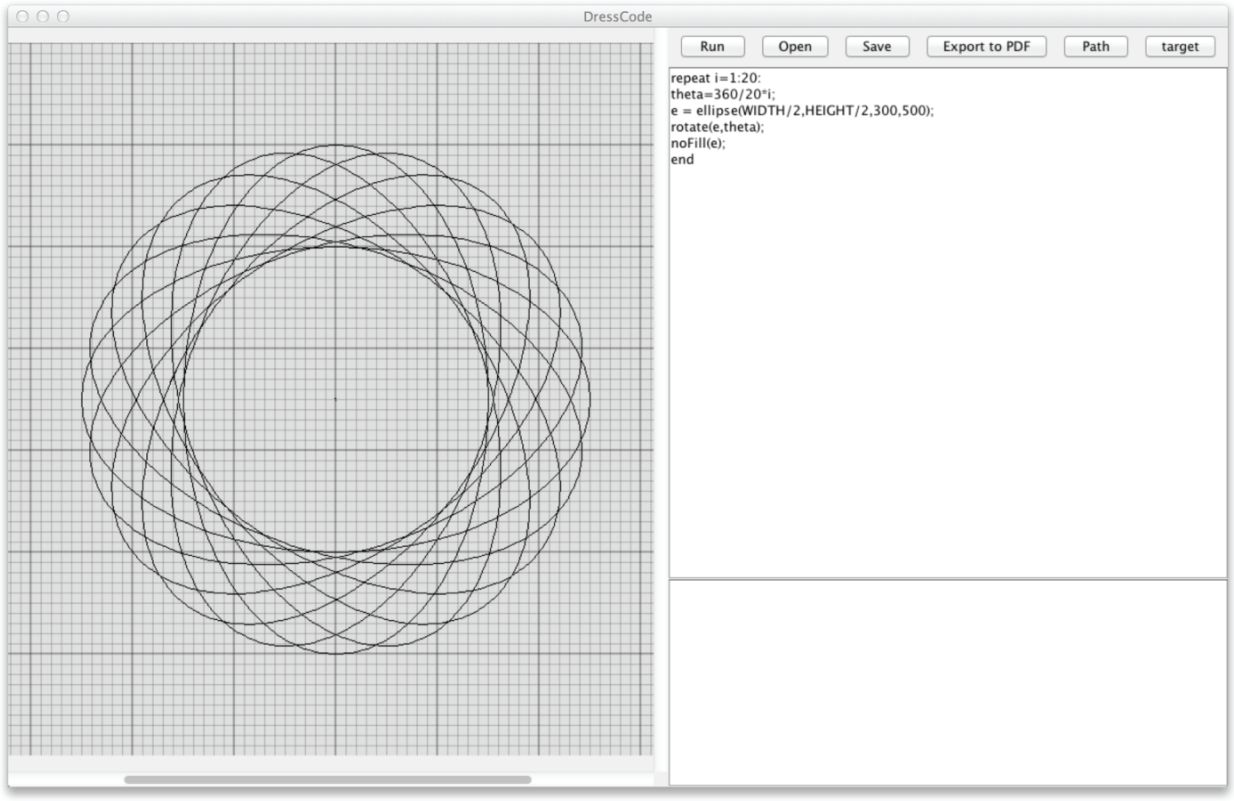


Figure 7-2: The first bare-bones version of DressCode

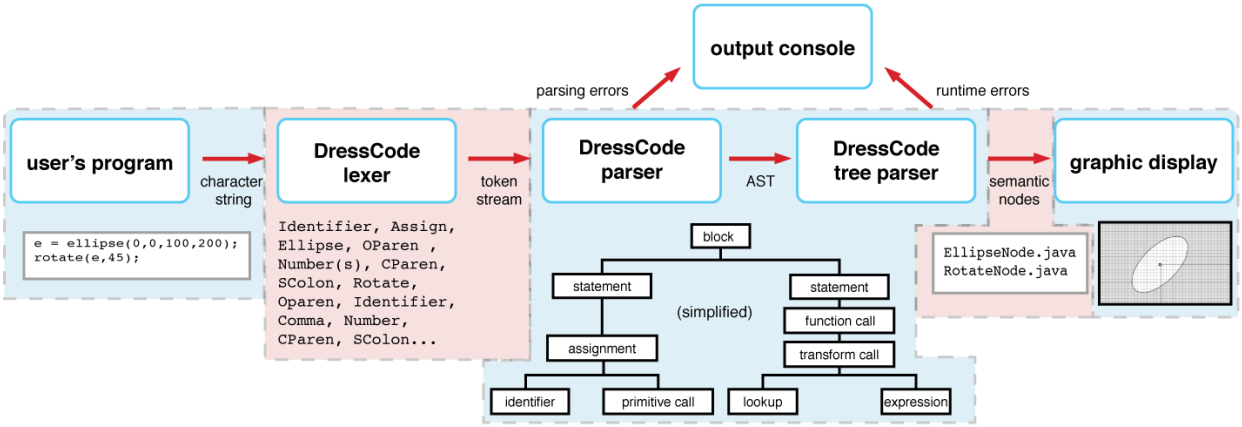


Figure 7-3: Interpreter structure

The language is imperative in nature, with statements being executed in the order in which they are read. The exception to this is user defined functions, which may be defined at any point in the program, and called before they are defined. Each statement in DressCode must be terminated by a semicolon. I spent some time experimenting with terminating statements with newlines, similar to

whitespace sensitive languages like Python, as I thought this might be more conducive to novice use. Unfortunately, the implementation of selective whitespace recognition is a challenging task to define in a grammar, so while we were able to develop a version of DressCode that recognized statements terminated by a line break, it would have taken significant time to develop this functionality to a level of robustness appropriate for user testing. For the short term, we have opted for a syntax requiring semicolons, however we plan to continue to explore newline statement termination in the future.

Language Structure

Here, I define some of the primary elements of the DressCode syntax and drawing API, and the rationale for their structure. A more thorough language reference is available on the DressCode wiki[3].

DressCode supports number, string, boolean, and drawable datatypes (figure **??**.) The first four of these types are relatively standard in comparison to conventional programing languages. Numbers include integers and floating point values. Strings include any sequence of characters enclosed in double quotations. Booleans have two possible values: true, and false. Drawables are a special type, discussed in detail in the API description below. DressCode also contains a list data structure, which can store multiple kinds of datatypes.

```
myString = "hello";  //string
myNum = 10.4;  //number
numbermyBool = true;  //boolean
myList = [10,11,false,"world"];  //list with multiple data types
println(myList[3]); //prints world
```

Variable identifiers in DressCode must begin with a letter which followed by 0 or more letters or digits. Variables can be initialized through assignment, or declared and assigned later in the program. All assignment in DressCode is dynamically typed, and variables can be assigned to datatypes that differ from their original assignment at any point.

```
s1 = "hello";
s2 = "world";
s3; //variable without initial assignment

s3 = s1+" "+s2;
println(s3); //prints hello world

n1 = 2;
n2 = 2.5;
println(n1+n2*10); // returns 27.0
```

The language also contains support for basic expressions, as well as block statements, including conditionals, loops and user-defined functions. For mathematical expressions, standard order of operations is maintained, unless parentheses are introduced. All block statements are signified by a keyword followed by a colon, and terminated with the end keyword which is not followed by a semicolon.Conditionals are defined with the if keyword, and may an optional else clause and zero or more else if clauses.

```
//if statement
if 5<10:
println("true"); //prints true
end

//if statement with else if and else clause
i=10;
if i<10:
println("less than 10");
else if i==10:
println("equals 10"); //prints equals 10
else:
println("greater than 10");
end
```

There are two possible loop statements: repeat statements and while statements. Repeat statements begin with the repeat keyword and are followed by a the initialization: a variable identifier with a numerical assignment, followed by a colon and the test, a number that determines the point at which the repeat statement will terminate. By default, all repeat statements have an update value of 1, however this can be modified by following the test value with "add" and a 3rd value specifying the update condition. While loops are initialized with the while keyword and followed by a test condition.

```
//repeat statement
repeat i=0:10:
ellipse(0,i*10,10,10); //draws a vertical row of 10 ellipses
end

//repeat with modified update condition
repeat i=0:10 add 2:
println(i); //will print 0,2,4,6,8
end

```

```
//while statement
c=0;
while c < WIDTH:
ellipse(c,10,20);
c = c+20;
end
//draws a row of ellipses that span the width of the canvas
```

Finally, custom functions are defined with the def keyword, followed by an identifier and a set of parentheses containing 0 or more arguments separated by commas. The function block is terminated with the end keyword like all other block statements. Just like general assignments, functions arguments are dynamically typed. Functions, loops and conditionals all have their own scope and will prioritize identifiers based on that, however if they cannot locate an identifier within their own scope, they will look for it in the level above.

```
//basic function defintion
def foo(a,b,c):
println(a+b+c);
end

//function call
foo(1,2,3);  //prints 6

//function with return statement
def bar(a,b,c):
return(a*b*c);
end

result = bar(4,5,6));
println(result);//prints 120
```

Drawing API

The API is organized around the creation and transformation of 2D shape primitives. By duplicating and manipulating these primitives in a structured manner, it is possible to generate complex and interesting designs from simple forms. The API is composed of three primary main categories, divided by functionality: shape primitives, shape transforms, shape booleans, math operations and property access. A selection of the primary methods from each of these categories is detailed in figure 7-4.

Shape Primitives	Shape Transforms	Shape Booleans	Shape Properties	Math
ellipse	move	union	getX	sin
rect	rotate	diff	getY	cos
poly	hide	xor	getOrigin	aTan
line	show	clip	getWidth	tan
curve	copy	merge	getHeight	random
point	group	expand	dist	round
import	scale			map
	mirror			

Figure 7-4: A selection of methods from the DressCode API

Shape Primitives: DressCode shape primitives currently serve as the primary means of drawing (figure: 7-5.) The simplest primitive is a point, initialized two numerical parameters that denote the x and y coordinates. All closed-shape primitive methods (ellipses, rectangles and polygons) require an argument of either two x,y coordinates, or a point. This coordinate defines the origin of the shape and determines where the shape will be drawn on the canvas. Ellipses and rectangles have an additional two optional parameters which specify width and height. If only two arguments are provided, the ellipse or rectangle will be drawn with a default width and height; if three arguments are given, the width and height will be the same. Polygons also have two optional parameters following the x and y origin coordinates, however they specify number of sides and length of each side, rather than width and height. Lines can be initialized either as four numerical values, specifying start and end x and y coordinates, two points, or as a vector, with a origin point, a magnitude, and a heading in degrees. Curves are defined by a set of four points or eight coordinates, which determine the start, first control point, second control point and end point of a 4-point bezier curve. One other method of primitive generation was added in at the request of some of our early users: vector paths stored in Scalable Vector Graphics (SVG) format can be imported and drawn in the DressCode environment, and used in conjunction with primitive transformation and boolean methods.

Shape Transformations: There is no required "draw" method for DressCode, as there are in many other graphics APIs, Instead all primitives are automatically drawn in the display panel in the order of their initialization, once the program has been run. In Soft Objects and Codeable Objects, when we required users to manually call draw method for any and all shapes in their program, they often forgot, and then had difficulty determining the reason why parts of their designs were not visible.

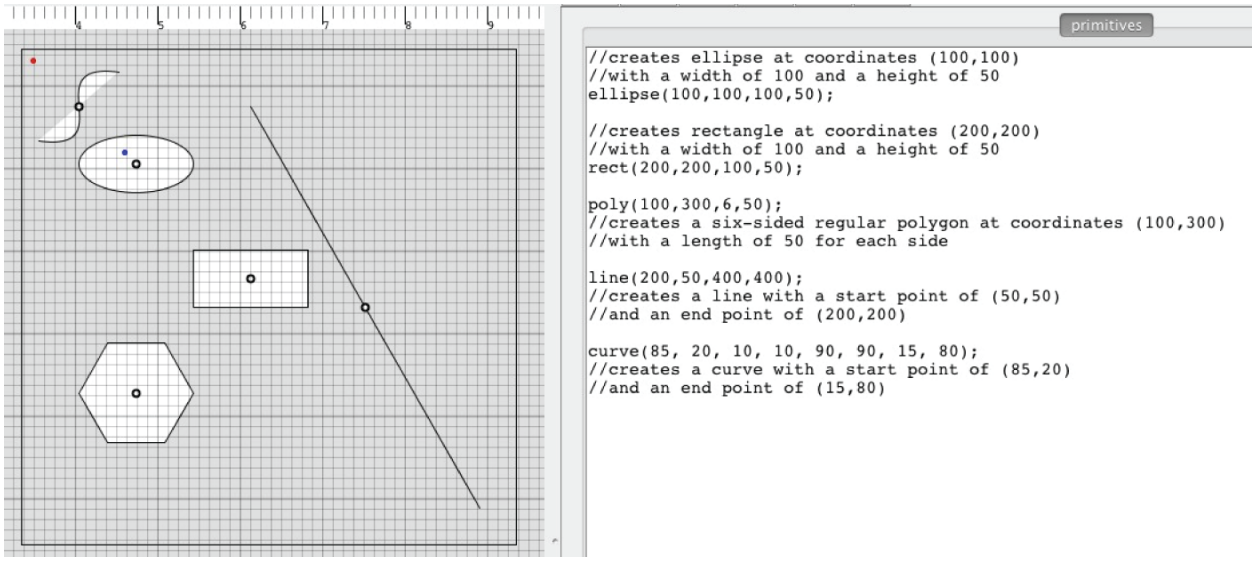


Figure 7-5: DressCode shape primitives with visible origins

Each shape primitive method returns an instance of the primitive it creates. A programmer can either create shapes with no reference as in figure 7-5, or they can assign them to an identifier at initialization, allowing them to reference and modify the shape later in the program. The shape transformation methods allow for a range of modifications to a shape, by passing a reference to the primitive the first argument, followed by different parameters for the modification, depending on the method (figure: 7-6.) We began with a basic set of transformations, including move, rotate, methods that allowed for the modification of the stroke weight and color of the shape, and a hide method that prevented a primitive from being drawn. Based on our early user testing however, we gradually added in additional methods, based on what people wanted to do with their designs. This included a moveBy method, scale and mirroring functionality, a copy method, and an addition to the rotate method which allowed a shape to be rotated around a point other than its origin.

DressCode primitives can also be programmatically "grouped". Grouping provides an organizational structure for more complex configurations of primitives allows for the automatic transformations to multiple shapes. Groups have their own origin, which is used as the reference point for all move, rotation and scale methods applied to the group (figure: 7-7.) For example, a collection of ellipses that is grouped and then moved to the center of the canvas would move the origin of the group to the center, and all the ellipses relative to the new origin. The origin of a group is automatically calculated as the centroid of all of the objects in the group and updated each time a primitive is added or removed. Groups are treated similarly to lists in that individual objects can be accessed and manipulated by via their index. Originally, primitives within a group were transformed relative to the origin of the group, however this caused confusion among many of our initial users. Each

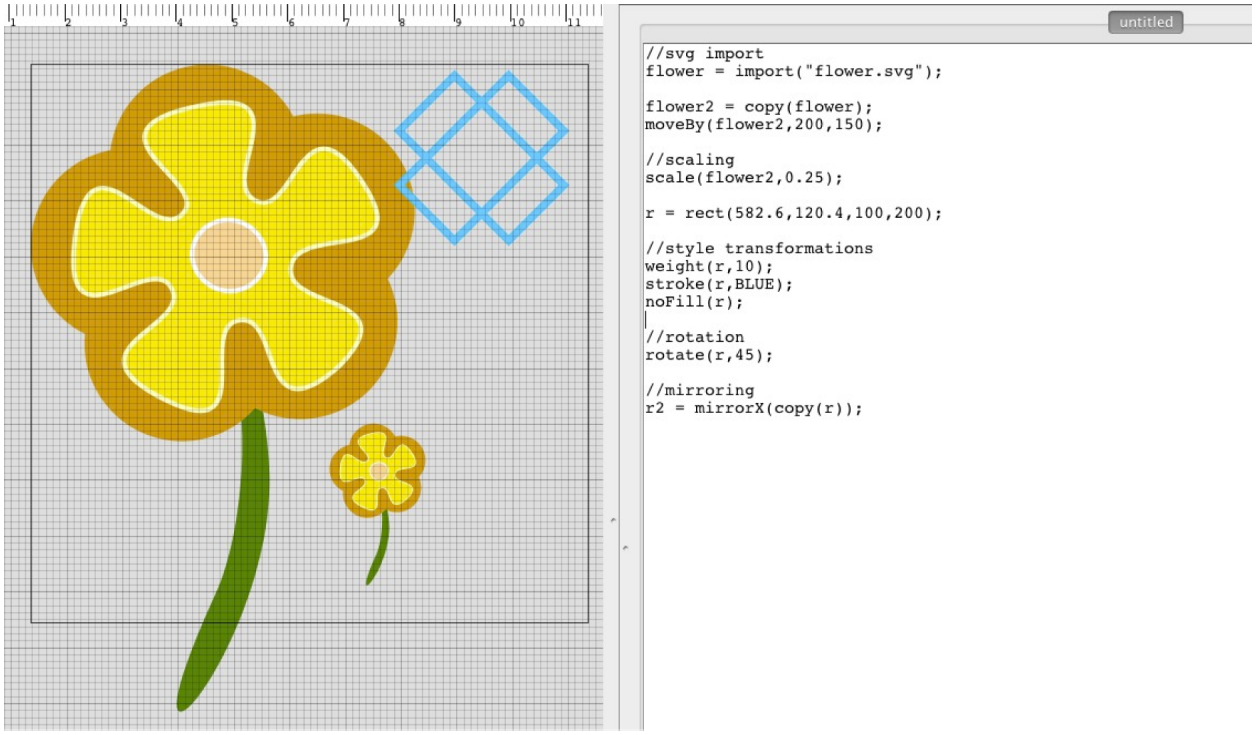


Figure 7-6: SVG import and sample of transformation methods

individual primitive is now translated according to the coordinate system of the canvas, regardless of whether it is in a group.

Polygon Booleans: DressCode also features a subset of transformation methods for polygon boolean operations (figure: 7-8.) The union, difference, exclusive or and intersection methods accept two primitives as arguments and return the corresponding shape, or group of shapes, depending on the operation. In addition, the merge method performs a union on all objects within a group and returns a single primitive. The expand method converts an object's stroke to a filled polygon with dimensions that match the weight of the stroke. The boolean methods were added for multiple reasons. Polygon boolean operations play an important role in CAD design for fabrication, because they allow for the creation of vector paths that appropriate for subtractive manufacturing processes. In particular, the merge method can function as a catch-all method for preparing designs for fabrication, and the expand method is useful for converting line drawings to paths that will retain their appearance when cut. In prior workshops, we relied on illustrator to prepare file paths for manufacturing, which was difficult, time-consuming and a hindrance to the computational design process. By integrating boolean operations as a part of the programming language in DressCode, it not only prevents users from having to rely on multiple software tools to design, but allows booleans

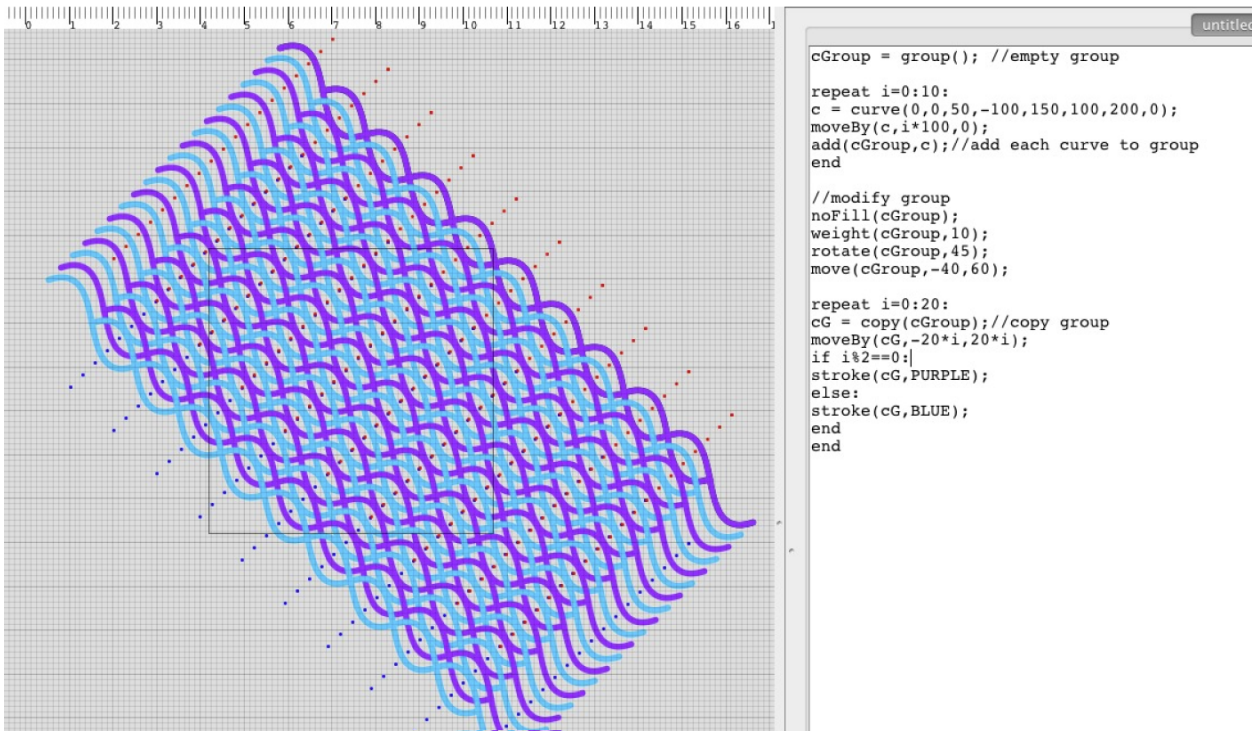


Figure 7-7: Grouping with transformation and repeat statements

to be used in a computational context, facilitating the creation of a whole new range of forms with simple primitives.

Both the syntax and functionality of the boolean operations have been modified significantly since the first version of DressCode. In early versions, the majority of the booleans were expressed through mathematical notation, rather than with built in functions (e.g. ellipse1 + ellipse2 would perform a union). After some of our early user testing however, we depreciated this syntax, because many testers felt that mathematical notation was not intuitive beyond the union and difference operations. In addition, the mathematical notation required that all booleans be set equal to an identifier (e.g. ellipse1 + ellipse2 = e3), in order to evaluate as a valid statement. The new method syntax still returns a resulting boolean, but can also be called without a corresponding assignment statement. Semantically, we originally had the boolean statements operate in a destructive manner as is the case in many graphic drawing programs; union(ellipse1, ellipse2) would destroy the original two ellipses and set the identifier ellipse1 equal to the resulting boolean. This was confusing however, because in a imperative program, the programmer could still reference the destroyed ellipse2. What we opted for instead, was a non-destructive boolean method wherein union(ellipse1,ellipse2) would return a third shape, which would automatically be drawn on the screen, and could be assigned to an identifier as needed. The original ellipse1 and ellipse2 would be hidden, but could both still be referenced without conflict, and could be revealed again on the canvas with the show() method.

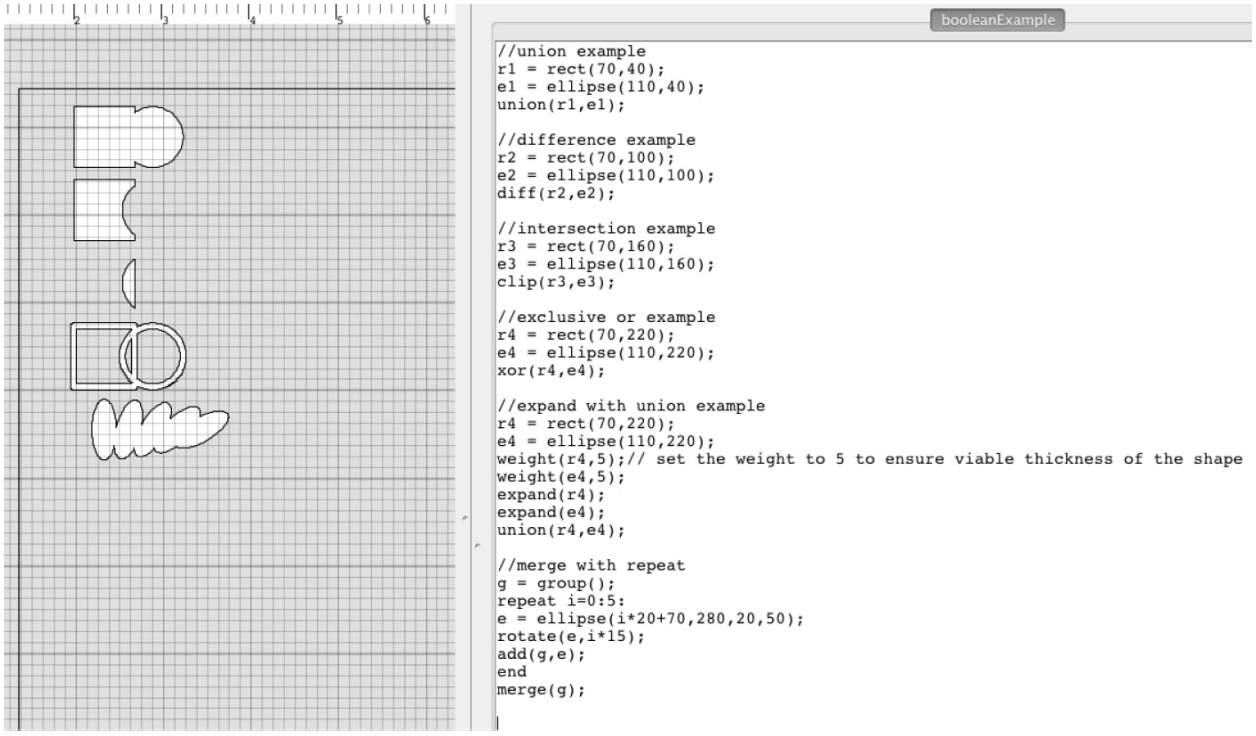


Figure 7-8: Polygon boolean operations

Additional Methods: Aside from methods for shape generation and transformation, DressCode features a set of getter methods that return information about a shape primitive, including its location, origin point, width and height, as well as a method to determine the Euclidean distance between two objects. There are also a set of methods that enable more advanced mathematical operations, including trigonometric functions, mapping, rounding and random number generation. The random method posses an interesting questions for implantation, as it generates different numbers each time a a program is run. This can be a useful design feature, allowing for the generation of numerous patterns with random variations. Figure 7-9 demonstrates variations of an algorithm that produces randomly symmetrical stripe patterns. It can also become problematic however, when the user wishes to keep the same randomly generated number for each interpretation of their program. Finally, we recently added a set of methods that allow the specification of numerical values in inches, centimeters and millimeters, or in the current units of the drawing board. These methods are helpful when sizing and transforming primitives to real world measurements rather than pixel values.

Constants: DressCode has a small set of constants to assist with design. WIDTH and HEIGHT correspond to the width and height of the current drawing board. PI corresponds to the numerical Pi value. Lastly, there are a set of color constants; RED, GREEN, BLUE, PURPLE, PINK, etc.,

that can be used to define the color of objects with fill and stroke methods. Custom colors can also be specified with 0-255 RGB values.

Figure 7-9: Stripe pattern variations using the DressCode random method

7.3 Fabrication and Crafting Process

Similar to the prior tools described in this thesis once a design is exported from DressCode, it can be imported to the control software of any 2-axis fabrication machine and fabricated into a set of parts.Depending on the material and fabrication process, a variety of end products are possible. In the process of protopying DressCode, I experimented several different fabrication machines, crafting techniques and artifacts. I tested DressCode primarily with the laser-cutter, however I also used it with more accessible machines including low-cost craft vinyl cutters and inkjet printers. I also examined sending DressCode files to online fabrication services such as Shapeways and SpoonFlower, to produce 3D-printed jewelry and larger inkjet printed garments. I tested the resultant pieces from these fabrication processes with different crafting techniques that ranged in difficulty. Highly accessible techniques included making artifacts that required basic paper-folding or iron on appliqus for clothing. More difficult techniques included sewing garments, screen printing onto fabric, and

jewelry making. There was frequently correlation between how easy a craft was to create, and how well it corresponded with the DressCode design process, indicating that for more sophisticated crafting processes like sewing, more advanced organizational functionality might be required for the DressCode software.

To provide some support in the crafting process for independent users, my undergraduate research assistants and I created a set of example templates in DressCode that are directed towards specific end projects and fabrication machines (figure: 7-10). We also began documenting some of the crafting techniques required to complete these projects online [**?**]. Mainly though, I relied on in-workshop instruction or curriculum development to assist people in using DressCode to complete finished artifacts.

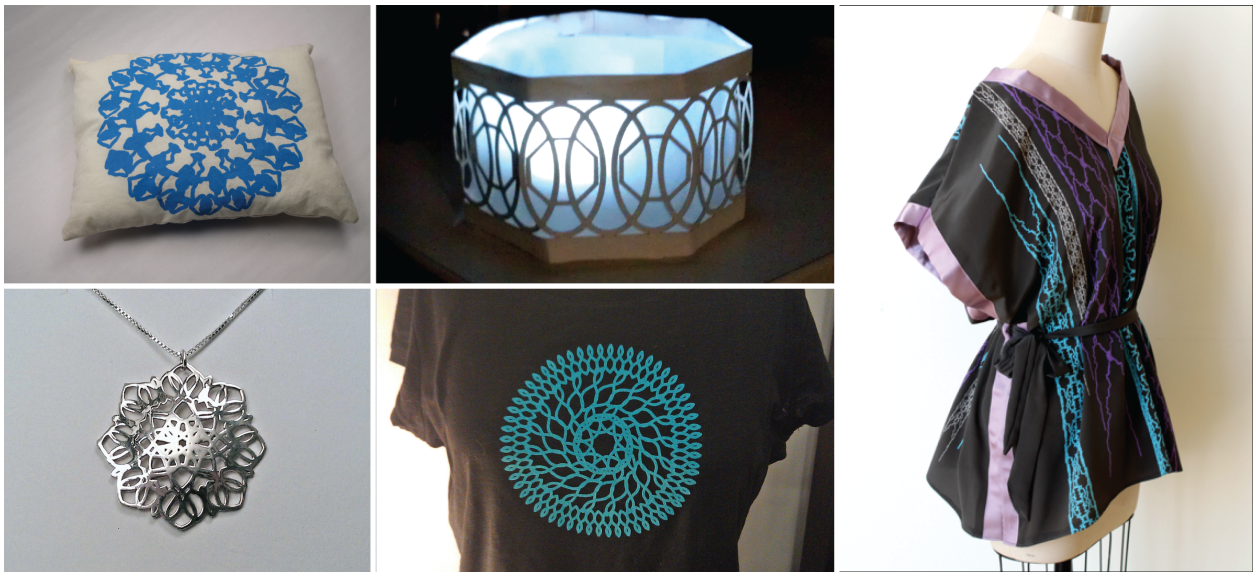


Figure 7-10: Several example artifacts created with DressCode Designs (clockwise from top left: vinyl-cut screen printed pillow, vinyl-cut tea light, ink-jet printed silk caftan, laser-cut iron-on tshirt, 3d printed silver pendant)

7.4 Workshops

DressCode was evaluated in

7.5 Results

7.6 Curriculum description

7.7 Preliminary curriculum results

7.8 Discussion

7.8.1 Starting notions of craft and computation

7.8.2 independent programming with a sense of ownership

7.8.3 Design Process and resultant aesthetics

7.8.4 Pride and Acomplishment in Algorithmic Craft

7.8.5 Design history and selection

7.8.6 Prototyping pt 2

7.8.7 Connection to other applications

Chapter 8

Discussion (rename)

magical qualities- enjoyment, dedication, pleasure, excitement, return

Processes- planning vs experimentation Prototyping The role of craft The affordances of algorithmic fabrication - how best to communicate them? The aesthetics of computational design Critique in computational design

difficulty in reconciling practice and felt experience of craft with discrete knowledge of computation

Chapter 9

Future Directions

Version Control (The loss of design) Better selection mechanisims Longer-term studies Targeted audience (revised)

Bibliography

[1] How to make a laser cut lamp. http://www.instructables.com/id/How-to-make-a-laser-cut-lamp, November 2007. (Accessed: 05/25/2013).

[2] About, iris van herpen. http://www.irisvanherpen.com/about#collections, 2013. (Accessed: 05/29/2013).

[3] Dresscode wiki. http://jacobsj.scripts.mit.edu/dresscode/, 2013. (Accessed: 05/29/2013).

[4] M. De Berg. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin,Germany, 2010. Third Edition.

[5] G. H. Hardy. A mathematician's apology. 1940.

[6] Resnick M. and Silverman B. Some reflections on designing construction kits for kids. In *Proceedings of the 2005 Conference on Interaction Design and Children*, 2005.

[7] LUST Reas N., McWilliams C. *Form and Code: In Design, Art and Architecture, A Guide to Computational Aesthetics*. Princeton Architectural Press, 2010.