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
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

"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

This is the acknowledgements section. You should replace this with your own acknowledgements.

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[4]. 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 [3]

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 computational 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 [5].

- **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 modeling. 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 computational 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.

Over the course of my masters 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. In the following three chapters, I detail the development, feature set and evaluation of each tool in accordance with the above criteria.

Chapter 5

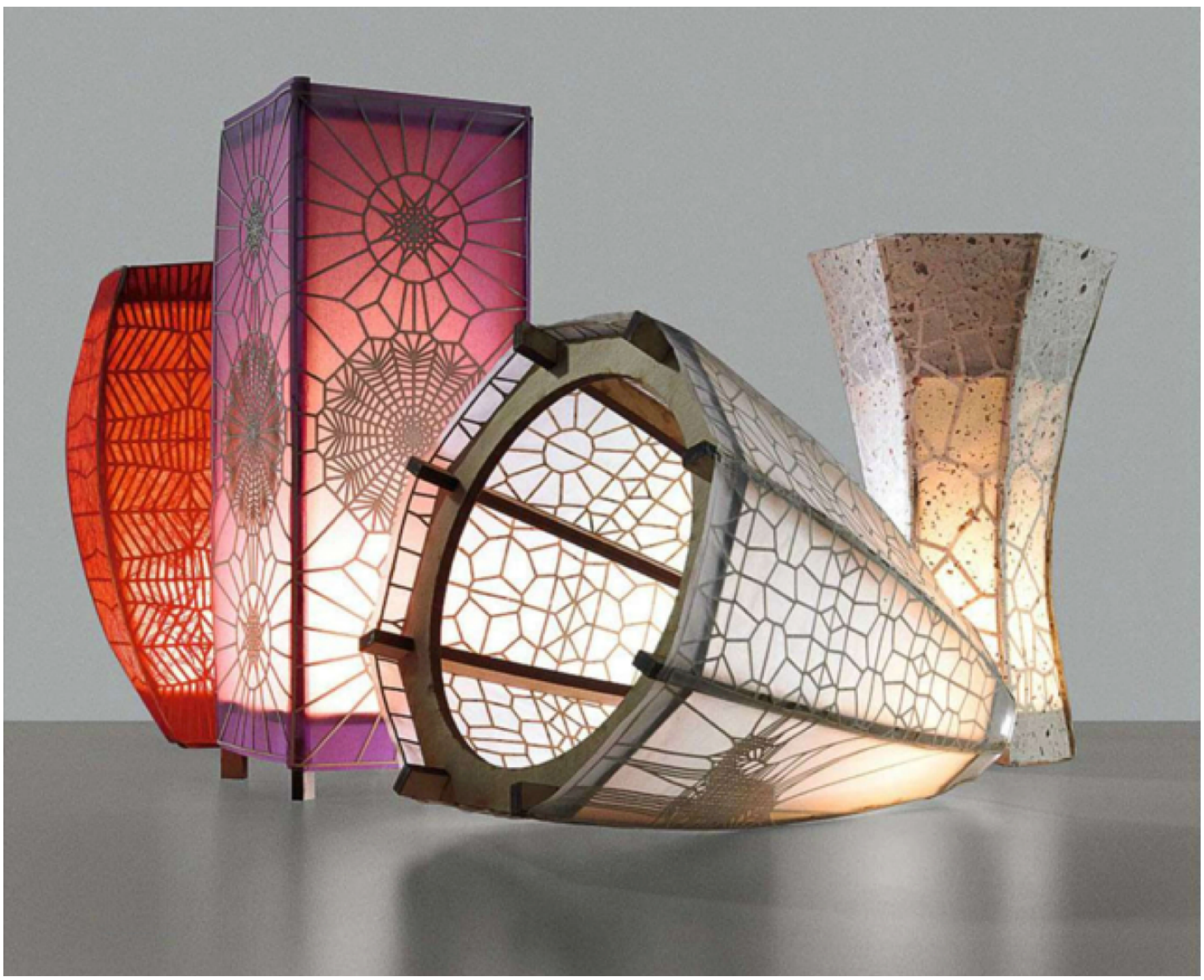
Design Tools

introduction to design tools

5.1 Design Tool Evaluation Methodology

The workshops were evaluated 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.

5.2 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.

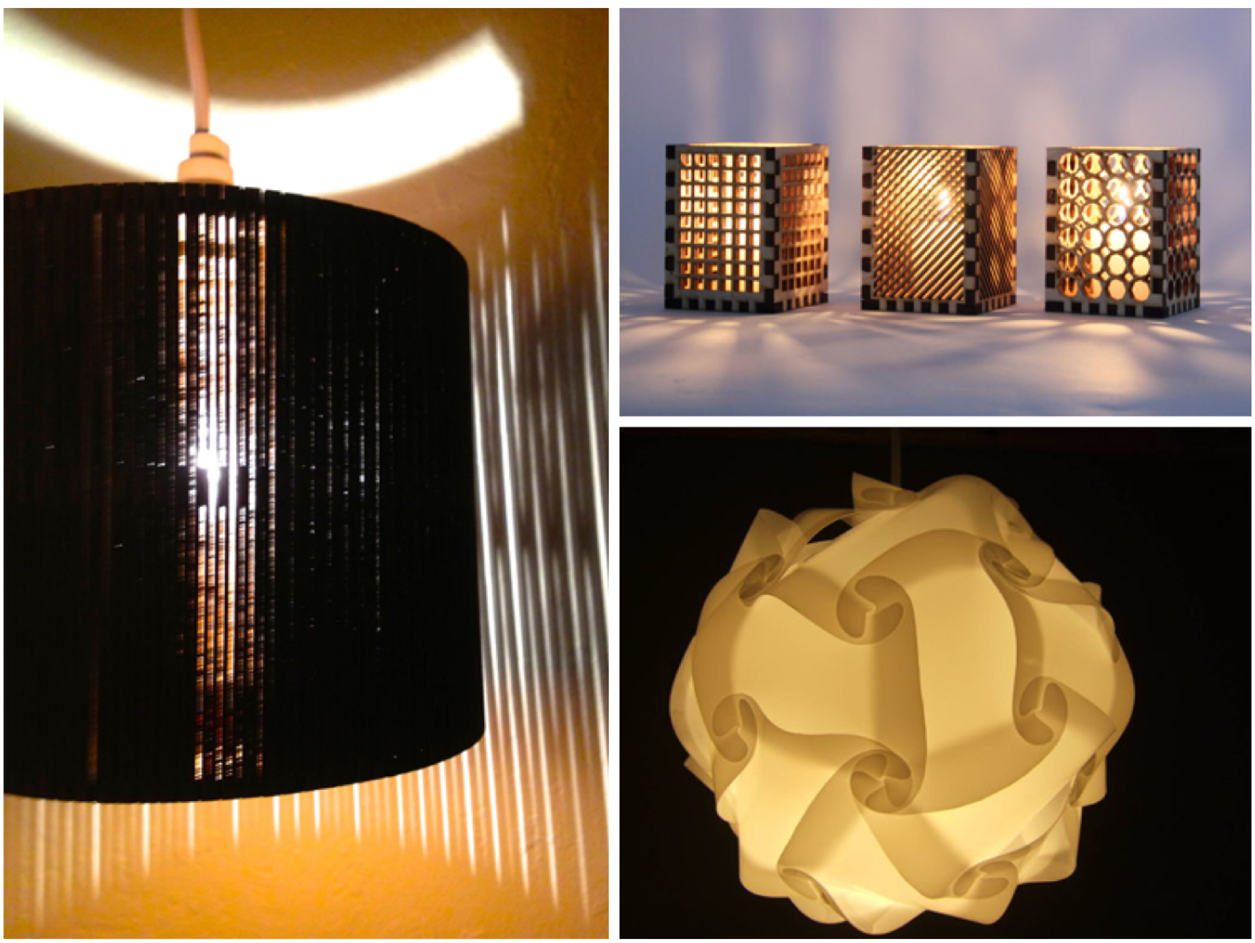


Figure 5-1: a selection of laser cut lamps from Instructables

5.2.1 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.

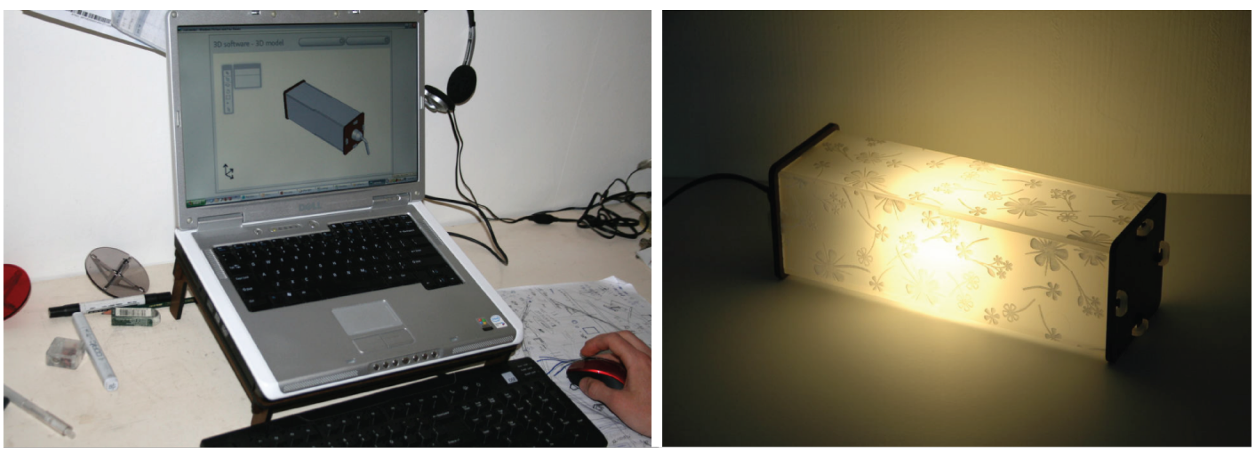


Figure 5-2: Instructables lamp tutorial with SolidWorks design process

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.

5.2.2 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.)

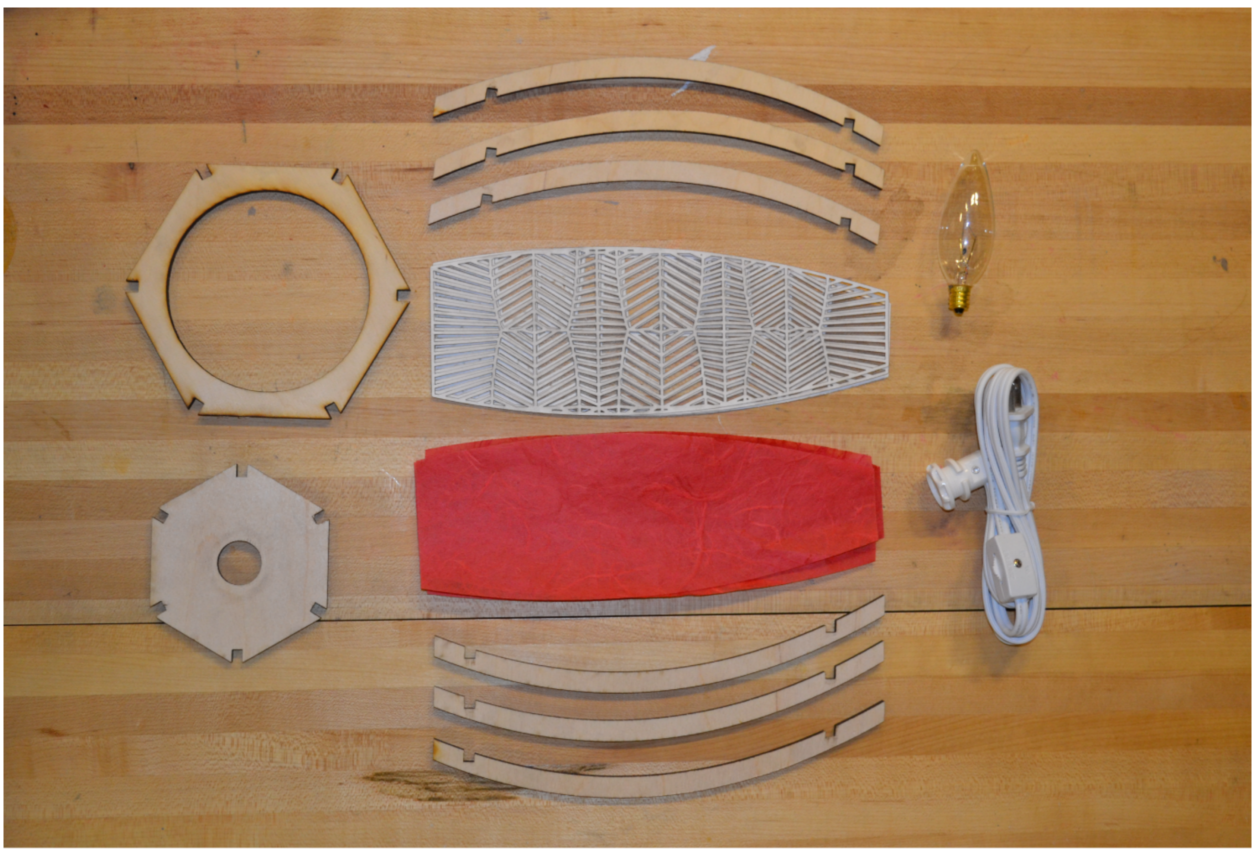


Figure 5-3: the individual parts of a lamp

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 [2]. 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.

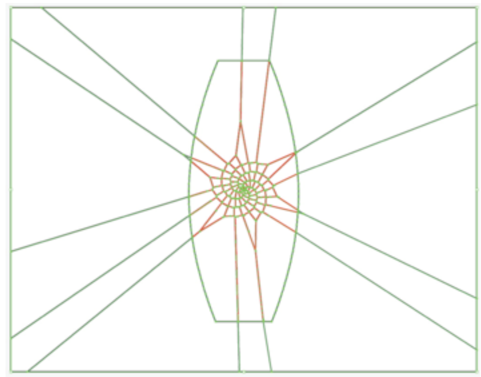


Figure 5-4: Algorithim for constraining the voronoi diagram within the shade

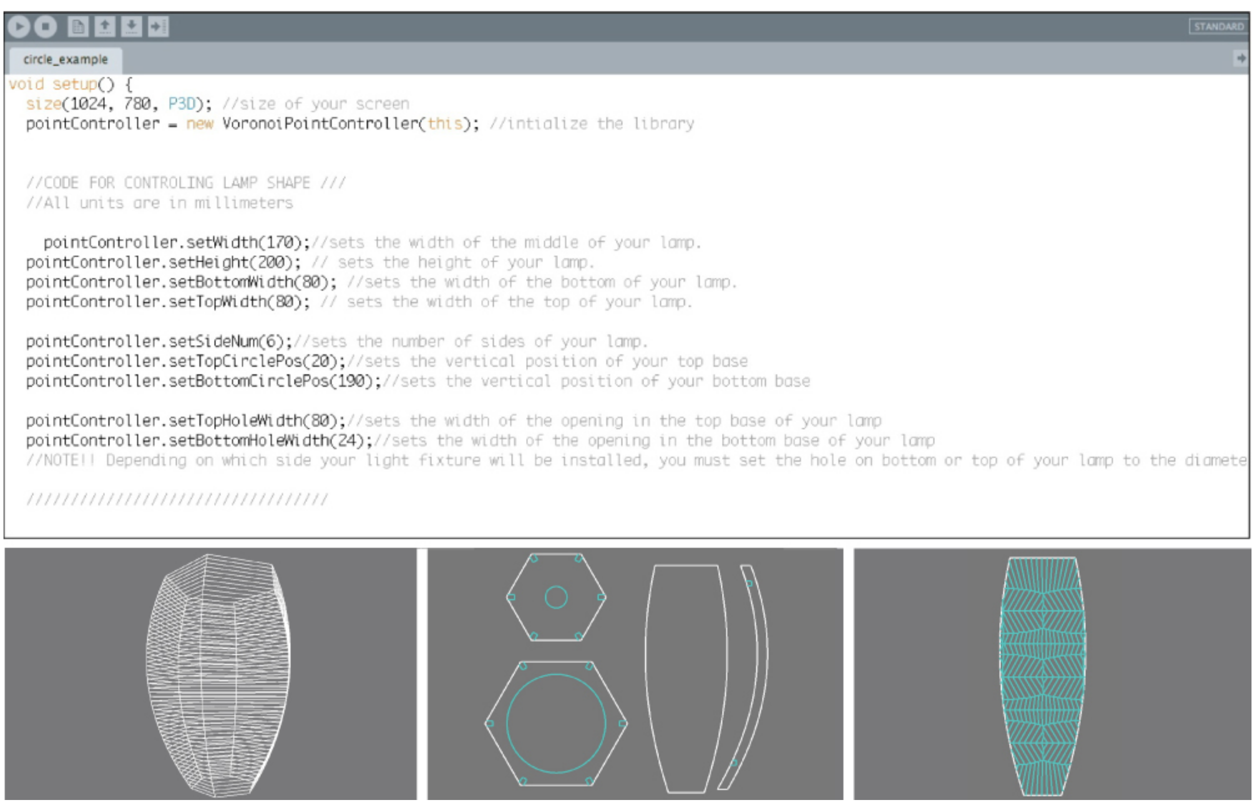


Figure 5-5: The first version of Codeable Objects, with only text-based interaction

5.2.3 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 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.

5.2.4 Workshop 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

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

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 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.)

5.2.5 Discussion

The most evident success of the pilot version of Codeable objects was the high rate of project completion. This success rate was closely connected to the ability of the library to correctly constrain the design parameters of the lamp. Although participants at times had to re-fabricate parts due to incorrect settings on the laser cutter or variations in the physical materials, at no point did a participant have to re-fabricate their piece due to errors in the design itself. Once fabricated, all participants pieces fit together correctly. This success came at the cost however, of significant design limitations. Participants who wished to modify the form to have more than one curve, or create patterns that went beyond the restrictions of the Voronoi diagram had to resort to post-processing their design files with a different CAD tool. In general, the

physical construction difficulties

Dissemination

programing vs graphical interaction

5.3 Soft Objects

Motivation and design principles Tool Description Workflow description Workshop Workshop results

5.4 DressCode

Motivation and design principles Tool Description Workflow description Workshop Workshop results Curriculum building Curriculum results

Chapter 6

Discussion (rename)

Processes- planning vs experimentation Prototyping The role of craft The affordances of algorithmic fabrication The aesthetics of computational design Critique in computational design

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

Chapter 7

Future Directions

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

Chapter 8

Conclusion

Appendix A

Tables

Table A.1: Armadillos

Armadillos	are
our	friends

Appendix B

Figures

Figure B-1: Armadillo slaying lawyer.

Figure B-2: Armadillo eradicating national debt.

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

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

[4] 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.

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