

WYSIWYFab and CSlice: Improved Interfaces for Rapid Prototyping

by Kenneth Shaw Friedman

S.B., EECS, Massachusetts Institute of Technology (2017)

Submitted to the
Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

Massachusetts Institute of Technology

June 2019

© Kenneth Shaw Friedman. All rights reserved.

The author hereby grants to M.I.T. permission to reproduce and to distribute publicly paper and electronic copies of this thesis document in whole and in part in any medium now known or hereafter created.

Author:

Kenneth Shaw Friedman, Department of Electrical Engineering and Computer Science
May 24, 2019

Certified by:

Stefanie Mueller, X-Window Consortium Career Development Assistant Professor
Thesis Supervisor
May 24, 2019

Accepted by:

Katrina LaCurs, Chair, Masters of Engineering Thesis Committee

Abstract

People should be able to create anything they can imagine.

Toward that vision, people need tools to augment their ability to turn an idea into a prototype.

Recently, hardware improvements in fabrication tools, such as 3D printers, have decreased the cost of prototyping. Computational improvements have enabled new ways to model and analyze designs. However, the interfaces that people must interact with to prototype an idea are lacking. Current interfaces are too complex for novice users and too weak for advanced users.

In this thesis, I start by describing a vision of the future to show why people should have the tools to create anything they can imagine.

Then, I provide two novel interaction models, along with their implementations. First, I present WYSIWYFab: a method that unifies the 3D printing pipeline, by merging the steps of modeling a 3D object and slicing a 3D object into a single step. Second, I present CSlice: a method that inverts the traditional slicing workflow, by optimizing slicing based on user constraints, instead of imperative parameters.

Finally, I discuss the results of these methods, and suggest future work for improving the interfaces of rapid prototyping and personal fabrication.

Acknowledgements

I want to sincerely thank my research supervisor — Professor Stefanie Mueller — for the mentorship, guidance, and support over the past two years. Thank you for welcoming me into your lab, teaching me how to do research, and showing me how great work gets done.

I would also like to thank my academic advisor — Professor Patrick Henry Winston — for the advice and support throughout six years at MIT.

Thank you to current and former members of the Human Computer Interaction Engineering group for feedback, brainstorming, and collaboration. I would like to extend a special thanks to Carolyn Lu for work on the WYSIWYFab project. I would also like to thank Faraz Faruqi for the incredible work on the CSlice project.

Thank you to all of my friends at MIT for helping me make it through. And thank you to my roommates of the past two years — Joel Gustafson, Colin McDonnell, and Nikhil Punwaney — for the adventures outside of lab.

Finally, an extra special thank you to my mom — Alice Friedman — for a lifetime of support and encouragement.

Support

This work in this thesis was supported in part by The National Science Foundation (NSF) Awards IIS-1716413.

Contents

1	Introduction	7
1.1	Background	9
1.2	Rapid Prototyping	9
1.3	Outline	10
2	Vision	11
2.1	Personal Fabrication	11
2.2	Climate Change	12
2.3	Agency	14
2.4	Negative Implications	15
3	Related Work	17
3.1	Classic HCI	18
3.2	Fabrication	20
4	WYSIWYFab: Integrating 3D Modeling & Slicing	25
4.1	Motivation	26
4.2	Environment and Walkthrough	28

4.3	Implementation	35
4.4	User Study	39
4.5	Discussion	49
4.6	Conclusion	51
5	CSlice: Constraint Centered Slicing	52
5.1	Motivation	53
5.2	Environment and Walkthrough	65
5.3	Implementation	70
5.4	Discussion	77
5.5	Conclusion	79
6	Future Work	80
6.1	Short Term	80
6.2	Medium Term	81
6.3	Long Term	82
7	Conclusion	83

List of Figures

1	Sutherland Using Sketchpad	18
2	Challenges In Personal Fabrication	20
3	WYSIWYFab Overview	26
4	WYSIWYFab vs. Traditional Workflow	27
5	WYSIWYFab Initial Setup	29
6	WYSIWYFab Printer Selection	29
7	WYSIWYFab Build Volume	30
8	WYSIWYFab Layer Height	31
9	WYSIWYFab Support Material	31
10	WYSIWYFab Support Geometry	32
11	WYSIWYFab Volume Highlighting	33
12	WYSIWYFab Material Color 1	33
13	WYSIWYFab Material Color 2	34
14	WYSIWYFab Model vs. Actual	34
15	WYSIWYFab CuraEngine	36
16	WYSIWYFab GCode	37

17	WYSIWYFab Support Rendering	38
18	WYSIWYFab Vase Model	41
19	WYSIWYFab Study Set Up	44
20	WYSIWYFab Iterations	46
21	WYSIWYFab Participant Models: Support Material	46
22	WYSIWYFab Participant Models: Fine Tuning . .	47
23	WYSIWYFab Participant Models: Build Volume .	47
24	CSlice Interface	53
25	CSlice vs. Traditional Workflow	54
26	Slicing: Layer Height Impact on Time	56
27	Slicing: Material Impact on Time	57
28	Slicing: Size Impact on Time	58
29	Results From Task 1	61
30	Results From Task 2	61
31	Results From Task 3	62
32	Results From Task 4	62
33	Slicing Parameters Used By Participants	65
34	Slicing Parameters Count	66
35	Re-Slices Per Participant	66
36	CSlice Walkthrough: Time Constraint	69
37	CSlice Walkthrough: Material Constraint	70
38	CSlice Walkthrough: Cost Constraint	71

39	CSlice Implementation Overview	72
40	CSlice Lambda Function	74
41	CSlice End-to-End Cloud Diagram	75
42	CSlice GCode Return Method	76
43	CSlice Slicing Cost	78

Chapter 1

Introduction

We make, not just to have,
but to know.

Alan Kay
The Early History of
SmallTalk (1996)

3D printers make rapid prototyping more accessible than it has ever been.

3D printers — and other fabrication tools — have dramatically decreased in price. Recent consumer grade 3D printers have become so accessible that a movement towards personal creation has developed known as the "maker movement," which relies heavily on 3D printing.

Computational tools have also improved people's ability to create physical objects. Better, more accessible software tools have enabled people to design 3D models and share designs around the world.

However, the interfaces for creating physical objects are still lacking. The majority of work in the field of Human Computer Interaction has gone into improving software and interactions that are limited to the digital world. However, as the desire and need for physical creation increases, interfaces will need to be re-thought and imagined for the physical world.

One of the keys to improving the creation of physical objects is to make *rapid prototyping* faster, easier, and a more integrated aspect of the design process. People need to be able to think of an idea, and quickly test it. Currently, it's too difficult to quickly prototype something to determine whether a concept will work in the real world. Interfaces must be designed that let people quickly prototype their ideas.

In this thesis, I attempt to provide two steps towards an improving interfaces that enable rapid prototyping. In the long term, this ability to rapidly prototype will help people be able to create anything they can imagine.

The first step is a method and implementation called WYSIWYFab. *WYSIWYFab* stands for *What You See is What You Fabricate*. This tool combines two common steps in the 3D printing pipeline: 3D modeling and 3D slicing. Combining these steps into a single tool enables novice users to have an easier time preparing a 3D print. It also enables more advanced users to have more immediate feedback on the designs they are creating.

The name *WYSIWYFab* is derived from classic HCI work in graphical user interface editors that were known as *WYSIWYG editors*, which stands for *What You See Is What You Get*.

The second step is a method and implementation called CSlice, or *Constraint-Slice*). This project inverts the traditional workflow of 3D slicing. Traditional workflows require users to input slicing parameters into the slicer, and the slicer will inform the user of the requirements of the print (such as the print time, and amount of material). CSlice enables users to tell the system their constraints, such as a time limit, and CSlice will determine what slicing parameters are needed to print within the constraint.

In the remainder of this chapter, I provide more background information on 3D printing and personal fabrication, as well as describe the organization of the rest of the thesis.

1.1 Background

For those unfamiliar with the basic concepts of 3D printing, this section is a brief primer on the terms and concepts of 3D printing that will be used throughout the thesis. The terms surrounding digital fabrication and 3D printing are relatively new and are therefore imprecise. I will define the terms as I apply them throughout this report, but reasonable people could have different definitions of these terms.

Digital Fabrication is the collection of processes for creating physical objects using computational methods, such as using computer-aided-design (CAD) tools to model an object before producing it. One of the most common methods of digital fabrication is 3D printing.

3D printing is a general term for process of creating three dimensional physical objects by way of additive manufacturing. *Additive manufacturing* is the process of creating physical objects by starting with an empty space and adding material over time to form an object.

One of the most accessible and common methods of 3D printing is *fused deposition modeling* (FDM), also known as *Fused filament fabrication* (FFF)³³. I will use FDM to refer to this type of printer throughout the report. FDM printers work by moving a filament material (usually a thermoplastic) through a tube and heated to a melting point at an extruder, which adds the filament to the existing model in a precise location.

The alternative to additive manufacturing is the more traditional *subtractive manufacturing*, which starts with a block of material (often wood or metal) and removing material from the object until the desired shape is formed.

1.2 Rapid Prototyping

This thesis could have been titled "Improved Interfaces for *3D Printing*" as opposed to "Improved Interfaces for *Rapid Prototyping*". The methods and implementations described in this thesis apply directly to 3D printing.

However, I hope to make it clear that the reason these meth-

ods are implemented for 3D printing is because 3D printing is currently the cheapest and most accessible form of rapid prototyping. I hope the thesis reveals that while these concepts are currently implemented for 3D printing, they apply generally to rapid prototyping and personal fabrication. In chapter 6, I discuss how the concepts described in earlier chapters could apply to other personal fabrication devices.

1.3 Outline

In chapter 2, I describe a vision of the future in which people are able to create anything they can imagine. I explain how that could impact on both global challenges (such as climate change) and personal challenges (such as creative expression).

In chapter 3, I describe related work that stem from two areas of research. First, I review a work related to user interfaces, humane interfaces, and interaction models of Human Computer Interaction (HCI) The second area of research is the HCI research in the subfield of personal fabrication.

This early HCI work, starting as early as the 1960s, is referred to as *classic HCI* throughout this thesis.

In chapter 4, I discuss the WYSIWYFab project. I provide the motivation, a walkthrough of the system, and a user study evaluating the system.

In chapter 5, I discuss the CSlice project. I repeat the format of the previous chapter by again providing the motivation, a walkthrough, and an evaluation of the system.

In chapter 6, I discuss the results of both of these two systems. I also describe a range of future work. I first describe short-term future work: implementation improvements to WYSIWYFab and CSlice. I then describe medium-term future work, including further research that was inspired by these projects. Finally I describe some possible long-term work that will help enable the vision described in chapter 2.

In chapter 7, I conclude the major findings of this thesis and summarize my contributions.

Chapter 2

Vision

The purpose of computers
is human freedom.

Ted Nelson
Computer Lib (1974)

In the future, people must be able to create anything they can imagine. The ability for individuals to create what they need and want will have a positive impact on climate change, self-reliance, and creative expression.

In this chapter, I describe vision of the future in which personal fabrication is accessible and ubiquitous. I also discuss both global and individual impacts that could occur. Finally, I outline how more advanced interfaces are a necessary component of that vision.

2.1 Personal Fabrication

Neil Gershenfeld defines personal fabrication as "the ability to design and produce your own products, in your own home, with a machine that combines consumer electronics with industrial tools."¹¹

In his 2017 book, *Designing Reality*¹², co-written with his broth-

The vision of *creating anything you can imagine* stems directly from Professor Stefanie Mueller's original vision of the HCI Engineering group at MIT CSAIL, *in the future anybody will be able to create anything anywhere anytime.*²⁷

ers, Niel Gernshenfeld describes the opportunities and challenges of what they call the *third digital revolution* — that is, the ability to digitally fabricate physical goods. In the book, they describe the rise of the maker movement and *FabLabs* (Fabrication Laboratories) that are increasing in number and decreasing in cost exponentially. They argue that — in the future — fabrication technology will be as ubiquitous and necessary for modern life as the internet is today.

The key insight in the digital fabrication revolution is that atoms travel very slowly, while bits (literally) travel at the speed of light. That is, the transportation of physical goods is time and energy consuming while the transportation of digital information is nearly free and nearly instantaneous. Therefore, it's much better to only send the information about a physical object, and have the object physically created near its final destination.

This dream of digital fabrication has many applications. However, Gernsheld claims that the ultimate goal of digital fabrication is *personal* fabrication: individuals making things of their own design. In *Designing Reality* he says:

[...] just as the killer app for digital computing was personal computing, the killer app for digital fabrication is personal fabrication.

2.2 Climate Change

Climate change is one of the greatest existential threats to humanity. In 2012, the Bulletin of the Atomic Scientists's "Doomsday Clock" — a metaphor for global threats — moved one minute closer to midnight, in part due to concerns that "the pace of change may not be adequate and that the transformation that seems to be on its way will not take place in time to meet the hardships that larges-scale disruption of the climate portends."⁴ The 2018 report¹⁷ from the Intergovernmental Panel on Climate Change outlined the catastrophic potential of unchecked global warming. As a society, we are not concerned enough

about climate change, even when considering that we are not concerned enough.

While technological solutions are not sufficient to reduce the impact of global climate change, they are necessary. With such an urgent threat, all work should³ take climate change into consideration in thinking about the vision of the work and the effects the work could have.

In the 2015 article "What Can A Technologist Do About Climate Change,"⁴³ Bret Victor considers not only the direct actions that technologists can take towards climate change, but also the indirect⁴⁴ projects that can have indirect but large impacts. In it, Victor describes the need for tools that interact and engage with the physical world:

... software engineers find a fluid, responsive programming experience on the screen, and a crude and clumsy programming experience in the world. It's easier to conjure up virtual fireworks than to blink an LED. So more and more of our engineers have retreated into the screen.

But climate change happens in the physical world. The technology to address it must operate on the physical world. We won't get that technology without good tools for programming beyond the screen.

The sections of "What Can Technologist Do..." involving "Tools for Scientists and Engineers" and "Media for Understanding Situations" directly inspired the WYSIWYFab and CSlice projects.

There is also a secondary motivation for these projects, from a climate perspective. That is, tools that help personal fabrication will enable more people to create the things they need, and in turn, those people will be less reliant on major corporations and mass manufacturing, which impacts the environment. This increase in an individual's *agency* to create what they need is discussed more in the following section.

2.3 Agency

Currently, people are dependent on corporations to be able to obtain the goods they need. Increasingly, companies are making it difficult for consumers to modify the products they buy. In many aspects of society, there is a large divide and large imbalance between consumers and producers.

Personal fabrication presents the chance to increase an individual's agency. The lines between consumers and producers can be blurred so that people can produce the things they need, want, and imagine. In short, personal fabrication "provides the means to make."¹²

Gershenfeld questioned the status quo of relying on corporations to produce what individuals need by describing the concept of a post-profit world. In *Designing Reality*¹², he described the following (emphasis mine):

The loss of jobs to globalization and automation and the damages done by an economic race to the bottom underpin social and political unheavals around the world. Yet implicit in all sides of the debate over competing financial and social models is an assumption about the nature of work.

For many people, it means traveling away from home to get a job they'd rather not be doing, producing a product designed by someone they don't know for someone they'll never see, to make money to buy what they need and want.

What if you could skip all that and *just make it for yourself?*

Personal fabrication enables this vision to become a reality. If people can create what they imagine, the control and power returns to the individual. Gershenfeld, again, says that we will be able to "dream, create, and shape our world with individual and collective action."

There is some fight against this, in a movement that's known as the *Right To Repair*¹⁶, but the movement has not gotten the attention it deserves.

Returning control to the individual from concentrations of power is generally good, but can have unintended consequences. The ability for you to create what you can imagine also means that bad actors have the ability to create weapons and destructive things that they can imagine.

While this challenge is much greater than the projects here are able to tackle, I address this concern briefly in the following section.

2.4 Negative Implications

The vision of creating anything you can imagine comes with a downside. Some people can imagine some very destructive things.

The projects described in this thesis are too small to be effected by this possibility, but since they take steps towards a vision of creating anything imaginable, it's necessary to consider the potential downside of this vision.

Like all technology, fabrication tools are just an amplifier of human motivations. They can be used to benefit the world or to be destructive, depending on the user's intents.

There are two steps, in general, that can be taken to mitigate the negative implications of personal fabrication. Both steps are effectively in the category of "improving education" — the main method for elevating humanity throughout history.

First, we can improve *literacy*; not only in terms of the written word, but a fabrication literacy and cultural literacy. A fabrication literacy can help reduce the chance of dangerous objects being created, and for objects being created in a dangerous manner. For a fabrication literacy to exist, a fabrication literature needs to be developed, which is an under examined area of the field.

Second, — as Vi Hart has said¹³ — "we must ensure that human

However, Gernshenfeld does this discuss this in depth in *Designing Reality*¹² and Stefanie Mueller addresses the issue as one of the core problems of personal fabrication in *Personal Fabrication*².

wisdom exceeds human power." That is, we need to be able to reason and consider possible options more powerfully than the creation tools we have at hand.

Some fabrication research, described in the next chapter, attempts to improve the knowledge of what we can build, but more fundamental work is needed in augmenting human intellect¹⁰ to ensure that we aren't able to build things that we don't understand.

Chapter 3

Related Work

An interface is humane if it is responsive to human needs and considerate of human frailties.

Jef Raskin
*The Humane Interface*³⁵
(2000)

The early computer pioneers envisioned the goal of computers as the ultimate tool to augment human intellect. Meanwhile, advances in physical construction tools have made fabrication cheaper and safer than ever before. Towards the vision of creating anything one can imagine, tools must be developed that combine physical fabrication tools and digital software tools.

In this chapter, I describe two lines of previous research, and their connection towards the projects described in later chapters of this thesis. First, I outline the classic HCI research that envisioned interfaces for people to enhance their creative thought. Then, I draw on more specific and more recent fabrication work, which attempts to address modern challenges in personal fabrication.

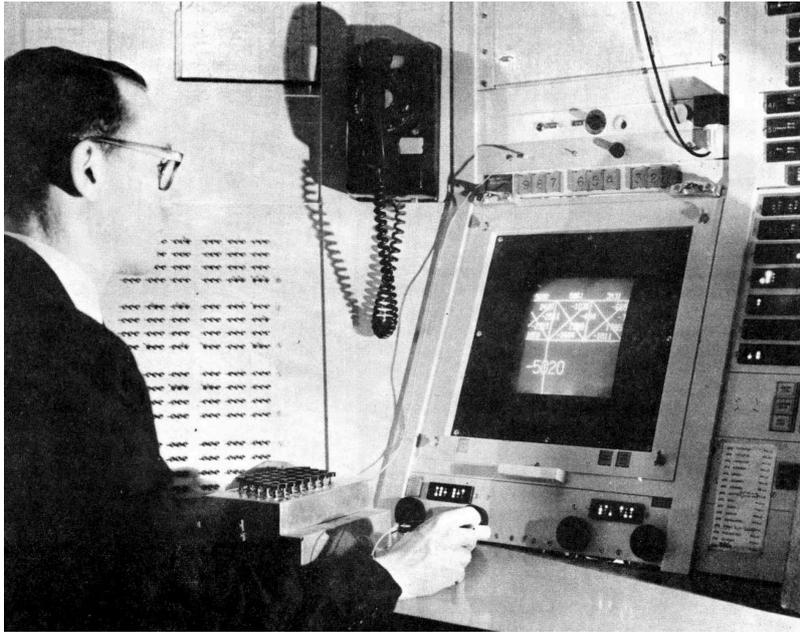


Figure 1: Ivan Sutherland using Sketchpad, a lightpen in his right hand to manipulate the CAD drawings on the screen in front of him. Image from *Sketchpad*.⁴⁰

3.1 Classic HCI

In 1963, Ivan Sutherland wrote what is sometimes called the greatest PhD thesis of all time. In it, he "invented the first interactive graphics program, the first non-procedural programming language, and the first object oriented software system"²².

In it, he described Sketchpad, the first true "Computer Aided Design" (CAD) tool. Not only did it inspire generations of CAD research, but it also began the first close look at human computer interaction.

Many others in this period made major contributions in the then-nascent field of Human Computer Interaction (HCI).

Vannevar Bush's 1945 article *As We May Think*⁵ theorized many modern concepts of HCI. In it, he envisioned:

[...] a future device [...] in which an individual stores all his books, records, and communications, and which is mechanized so that it may be

consulted with exceeding speed and flexibility. It is an enlarged intimate supplement to his memory.

J.C.R. Licklider also provided a very early vision, in a seminal article called *Man-Machine Symbiosis*²⁴, which described how humans and machines can have complementary relationship by each focusing on their strengths. He said:

The main aims are 1) to let computers facilitate formulative thinking as they now facilitate the solution of formulated problems, and 2) to enable [humans] and computers to cooperate in making decisions and controlling complex situations without inflexible dependence on predetermined programs.

And most famously, Douglas Engelbart proposed and demonstrated the *oN-Line System*⁹, with the fundamental goal of augmenting human intellect¹⁰. He explained:

[...] population and gross product are increasing at a considerable rate, but the complexity of his problems grows still faster, and the urgency with which solutions must be found becomes steadily greater in response to the increased rate of activity and the increasingly global nature of that activity. Augmenting man's intellect, in the sense defined above, would warrant full pursuit by an enlightened society if there could be shown a reasonable approach and some plausible benefits.

All three of these visionaries — Vannevar Bush, J.C.R Licklider, and Doug Engelbart — had a dream of computing that involved enhancing society by creating tools to improve one's ability to think and create. This is a vision that seems all but forgotten in the current technology field. However, the work is still aspirational and inspiring, and has motivated — at a very fundamental level — the work in this thesis.

Licklider also provided an early and even more direct motivation to CSlice (see Chapter 5), when he said that "instructions directed to computers specify courses; instructions-directed to human beings specify goals." CSlice aims to enable users to dictate instructions to the computer by *goals*, instead of *courses*.

society	(5) sustainability		(6) intellectual property
software & user	(2) domain knowledge	(3) feedback & interactivity	(4) machine knowledge
hardware	(1) hardware & materials		

Figure 2: A Chart showing the six challenges in the field of personal fabrication. Chart created by Stefanie Mueller and originally appeared in *Personal Fabrication*.¹¹

3.2 Fabrication

In Stefanie Mueller's *Personal Fabrication*, she discusses six of the largest challenges facing the field (seen in Figure 2). Three of the challenges described helped directly motivate the projects in this thesis.

All three of the challenges that this thesis tried to address are in the "software & user" section of the stack: domain knowledge, feedback & interactivity, and machine knowledge.

3.2.1 Modeling Tools with Immediate Physical Output

A large number of tools have been developed that support users in designing physical objects.

Uncertainty in Measurement¹⁹, for instance, compensates for measurement errors beginners tend to make by integrating post-adjustment mechanisms.

RetroFab³⁴ uses 3D scanning and a sketching interface to support novice users in retro-fitting existing interfaces with new functionality.

SketchChair³⁸ allows users to sketch chair designs and provides analysis tools to determine if a chair is stable enough to hold a person.

Mechanical Characters⁷ supports users in designing animated toys with working gear and linkage mechanisms solely based on an input motion path.

3.2.2 Slicing Tools that Optimize Fabrication

A range of tools help users in optimizing designs for fabrication. For instance, for objects that are too large to fit into the build volume, Chopper²⁶ splits the model into smaller parts while ensuring that the parts are easy to assemble and the seams are unobtrusive. Dapper⁶ follows a similar goal but packs parts efficiently with respect to different printing processes. To optimize print speed, Wang et al.⁴⁵ print low-detail regions as thicker layers, which saves up to 40% print time. Researchers have also developed methods for reducing support material using novel types of support structures, such as branching support³⁹⁴² and bridging support⁸, or by laying out support material to reduce visual artifacts⁴⁸. Since objects break more easily along the interface of two layers, researchers also proposed to re-orient objects to increase stability⁴¹¹⁴. To reduce infill material researchers demonstrated how to use custom infill patterns optimized for mechanical load (Build-to-last²⁵) and how to print the infill directly underneath an object's surface (Skin-frame structures⁴⁶). Finally, researchers have also investigated how to optimize the print path to improve fabrication results (Connected Fermat Spirals⁴⁹).

3.2.3 Model Repair Software for 3D Printing

Many existing 3D editors provide plugins to check if a model has broken geometry and thus cannot be sliced and fabricated. Blender's '3D Print Toolbox' plugin and SketchUp's 'MakePrintable' plugin, for instance, help users to identify manifold geometry. Finally, tools, such as MeshMixer, help repair this broken geometry by converting surface meshes into solids for 3D printing.

Blender
<https://www.blender.org/>
SketchUp.
<https://www.sketchup.com/>
MeshMixer, Autodesk.
<http://www.meshmixer.com>

More recently, 3D modeling tools also started to integrate information about the fabrication device into the modeling environment. The 3D editor SketchUps, for instance, now provides a static MakerBot template that displays the build volume in the modeling view. Most closely related to our approach is an open-source effort to create a slicer within the 3D editor Blender called Dicer . The goal of the project is to provide a gcode export button directly in the 3D editor, i.e. rather than exporting an .stl file, the button would export gcode that can be directly loaded into the 3D printer. However, the project never finished and was discontinued.

Dicer, <https://github.com/RyLangrayston/Dicer>

3.2.4 Supporting Users with 3D Printing

In the investigation of this space, we explored previous work related to tools that support users in creating objects for 3D printing, improved slicing tools, and research that helps users to explore speed-fidelity trade-offs. In this section and the sections that follow, we document some of the previous work that inspired CSlice.

Hudson et al.¹⁵ were among the first to provide an in depth analysis of the issues novice users ('casual makers') encounter when using 3D printers. They explored the creation of the 3D model itself and the subsequent 3D printing process.

In recent years, much of HCI research has focused on supporting users in the modeling process:

RetroFab³⁴, supports users in creating 3D models that act as augmentations for existing devices; and Lamello³⁷ enables users to create 3D designs with interactive input components using only a microphone. While there is a large body of work in HCI on improving 3D modeling tools, only a few research projects have focused on facilitating the slicing process.

Existing work on helping users to convert an object into machine instruction, has mainly focused on laser cutting (e.g., VisiCut³¹, PacCam³⁶).

3.2.5 Improving Algorithms Underlying Slicing

Over the last years, several research projects in computer graphics have investigated how to improve the algorithms underlying various slicing parameters. For instance, novel print path algorithms lead to better print quality (Connected Fermat Spirals⁴⁹) and novel support structures generate faster printing supports (e.g., branching support^{39,42}, bridging support⁸). Similarly, novel algorithms to split objects into parts to fit them into the build volume ensure that the parts are easy to assemble (Chopper²⁶, Dapper⁶). Researchers also developed algorithms to optimize slicing for visual quality, i.e., they reduced artifacts by either laying out the support material in those regions least visible to the user⁴⁵ or by stacking high-resolution and low-resolution layers depending on visual features on the model⁴⁸. To improve durability against forces from a specific direction, researchers also proposed rotating the 3D model inside the slicing tool^{41,14} or to use custom infill patterns that optimizes the strength-to-weight ratio (Buildto-last²⁵). Finally, to make an object stand or spin after fabrication, researchers optimized the infill distribution to shift the center of mass (Make it Stand³², Spin-it¹). Rather than inventing new algorithms for individual slicer settings, our method searches for a combination of existing settings to match a user's external constraint.

3.2.6 Exploration of Speed-Fidelity Trade-Offs

Supporting users in finding the best speed-fidelity tradeoffs has a long history in HCI research and has recently also been explored in the context of 3D printing. Low-fidelity fabrication techniques²⁸, such as WirePrint²⁹ and Platener³, for instance, allow users to determine which parts of the prototype should be fabricated in fast low-fidelity and which in slow high-fidelity. Similarly, tools, such as SPATA⁴⁷, allow users to trade-off design and speed by visualizing the support material during 3D modeling. This allows designers to make an informed decision about either modifying the design to avoid supports or spending extra time on printing them. Our work also supports users in

finding the best speed-fidelity trade-off but applies it to the slicing process, in which users' trade-off different quality settings with print time, material, and cost.

Chapter 4

WYSIWYFab: Integrating 3D Modeling & Slicing

The defining application emerging for digital fabrication is personal fabrication.

*Neil Gershenfeld
Designing Reality¹² (2017)*

In this chapter I describe WYSIWYFab, which is a concept for integrating two steps in the 3D printing pipeline into a single step, as well as an implementation of that concept that enables users to model and slice in the same application. I first discuss the environment and walkthrough how the system works. Then, I discuss how the system was implemented. Finally, I discuss the results and describe possible future steps.

This project was designed and implemented during the Fall 2017 and Spring 2018 semesters. The project was implemented by myself and Carolyn Lu, under the supervision of Professor Stefanie Mueller. The implementation was done collaboratively, but I focus on the parts of the implementation that I focused on throughout the project.

The work was submitted as a full paper to the *Proceedings of the*

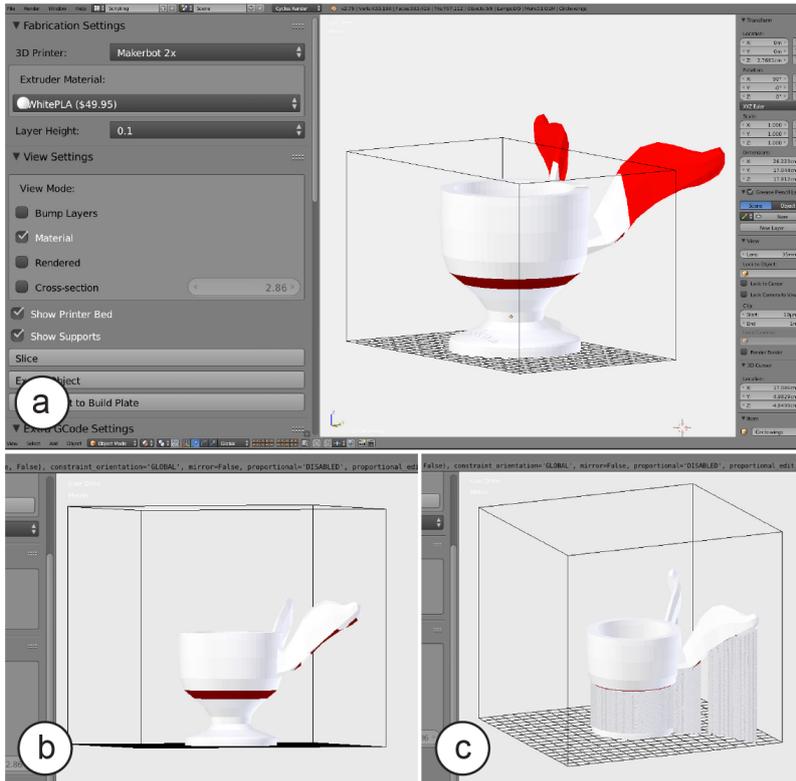


Figure 3: WYSIWYFab Overview: We integrate 3D modeling and slicing into a single editing environment: (a) While modeling, we notice that our model extends beyond the build volume and that support material will be generated at the bottom. (b) We scale down the model to make it fit. (c) To better understand, how much time the support material will require and thus, if it is worth changing the design, we hit the ‘slice’ button in the 3D editor. We notice that the support material adds 2 hours of printing time, thus we go ahead and slightly change the angle at the bottom of the cup to not require support anymore.

31st ACM Symposium on User Interface Software and Technology, abbreviated *UIST 2018*. The paper was not accepted, but a revision of the paper is planned to be submitted to a future conference.

4.1 Motivation

Current 3D editors do not know about the constraints of the fabrication process. Instead, users have to export the model at the end of the design process and load it into a slicing program.

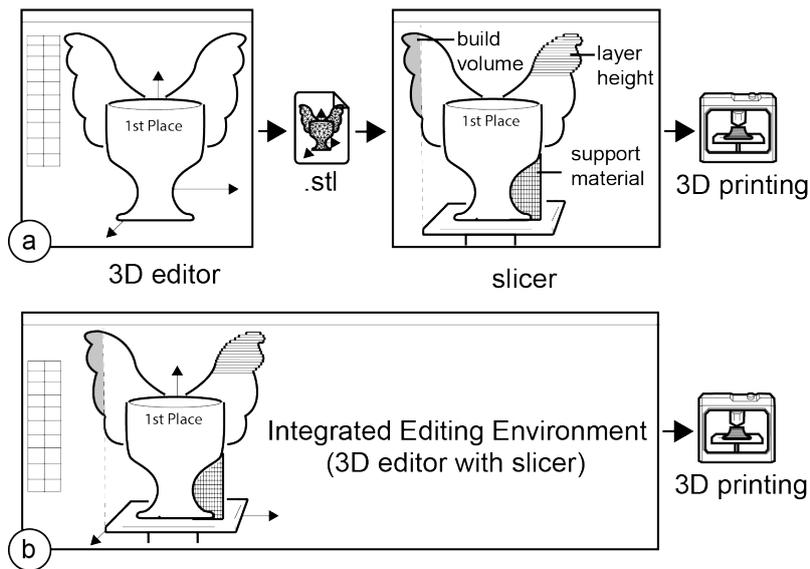


Figure 4: WYSIWYFab vs. Traditional Workflow

Only then can users see if the model fits inside the build volume, where support material is generated, and how much printing time and material their design requires. Users then need to repeatedly switch between the 3D editor and the slicer to fix the issues in the model. This makes the process of optimizing a model cumbersome and time-consuming and potentially leads to suboptimal designs.

In today's workflow, 3D modeling and preparing a 3D model for fabrication are seen as two independent steps (Figure 4a).

If users notice a mistake or need to optimize their design for faster printing and lower cost, they have to go back to the 3D editor, change the model, and then export again. This is not only a cumbersome and time-consuming process, but since 3D modeling and slicing are separate steps, problems tend to get noticed only at the end of the design process when it is increasingly difficult to make changes.

In this project, we argue that by combining 3D editing and slicing into a single integrated editing environment, we can eliminate many of these issues (Figure 4b): With integrated modeling and slicing, users get an accurate preview of the required printing time and material cost early on. They can see which parts of the design cause support material to be generated and

which parts of a model extend beyond the build volume. Thus, users are able to identify problems in the 3D model early on when changes are still easy to make. In addition, integrated editing and slicing allows users to optimize their design for better cost-time-design trade-offs, i.e., when deciding how much of the design and which parts should be adjusted to reduce material consumption and required printing time.

To illustrate our idea, we built WYSIWYFab, What-You-See-Is-What-You-Fab, an integrated editing environment that combines 3D modeling and slicing. With the example of an end-to-end design process, we show how this enables users to create working designs faster while making the designs more cost and time efficient to fabricate. We also tested our integrated editing tool in a user study with 12 participants and show how participants were able to create working designs in less iterations while creating better models.

4.2 Environment and Walkthrough

To integrate 3D modeling with slicing, our WYSIWYFab editing environment is built on top of the 3D editor Blender. When users hit the *slice* button, our Blender plugin queries slicing data from the slicer Cura . It then displays the printing time and amount of material directly in the 3D modeling view, and renders the support material around the 3D model by processing the returned gCode.

Cura, <https://ultimaker.com/en/products/ultimaker-cura-software>

While our current system requires users to hit the *slice* button, future versions will have the slicing and modeling seamlessly integrated. That is, every time the model geometry changes, new slicing data — including printing time and material cost — will be generated in real-time. However, to date, slicing is too slow for real-time processing since even small models take seconds to minutes to slice. We discuss further possible improvements to the system in Chapter 6.

In the following sections, we provide a walkthrough of our sys-

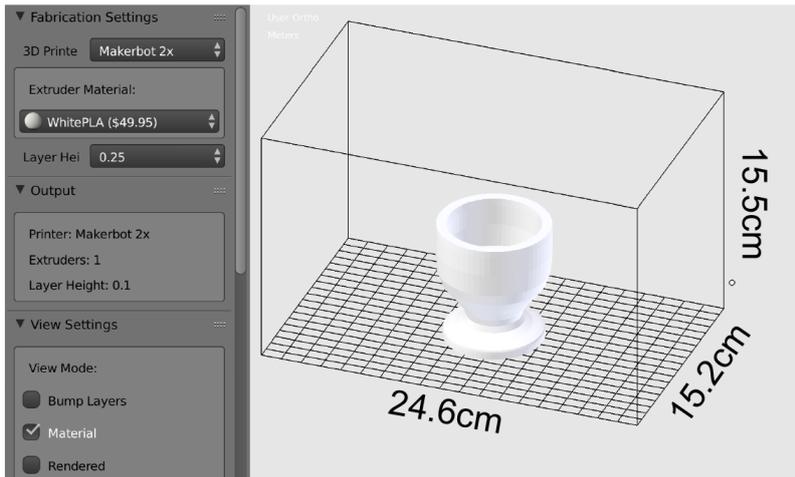


Figure 5: Initial setup when using WYSIWYFab

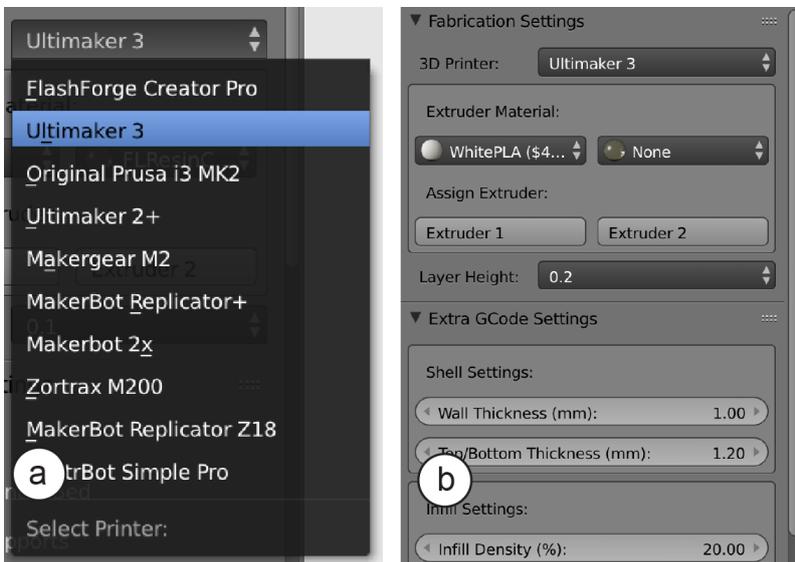


Figure 6: WYSIWYFab Printer Selection

tem at the example of modeling a trophy cup.

4.2.1 Adjust 3D Modeling View Based on Printer Settings

To begin, we load a basic model for the trophy cup that we found on the internet into the modeling view. Before refining and customizing the geometry, we decide to configure our integrated editing environment for the 3D printer we have available. By default — as seen in Figure 5 — the MakerBot 2X is selected from the menu.

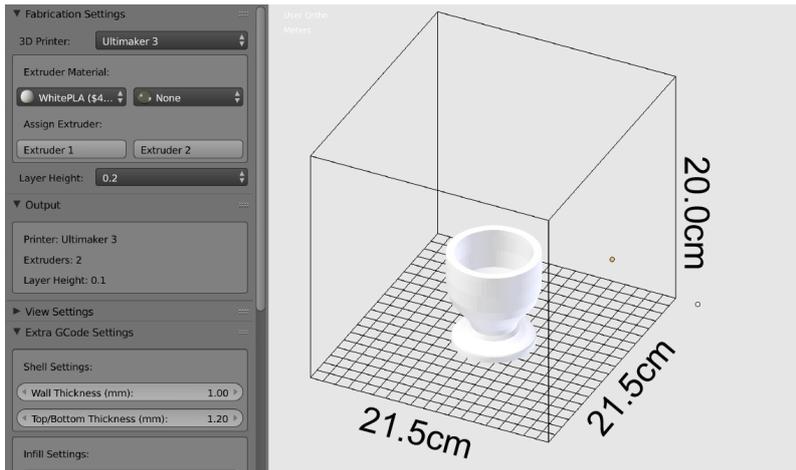


Figure 7: WYSIWYFab Build Volume

However, our lab uses Ultimaker 3s for 3D printing, so we decide to change the fabrication device by selecting our 3D printer from the dropdown menu (see Figure 6a). Selecting the *Ultimaker 3* 3D printer automatically loads all key properties, such as the build volume, layer height, support material angle, infill density, print speed, number of extruders, and available materials, and displays them in a menu right next to the 3D modeling view (Figure 6b).

Loading the properties causes the 3D modeling view to also update the build volume to match the selected 3D printer (Figure 5). In our case, rather than showing a build volume of 24.6cm x 15.2cm x 15.5cm for the MakerBot 2X, it now shows a build volume of 21.5cm x 21.5cm x 20.0cm for the Ultimaker 3 (see Figure 5 vs. Figure 7). In the current version of our system, the measurements seen in the figure are not actually displayed to the user. The dimensions are included in these figures because it is hard to tell the change in print bed size from a screenshot.

Like the build volume, all other settings that are typically available only in the separate slicing program, are integrated into our WYSIWYFab environment. For instance, while the standard layer height we started out with was 0.2mm, we decide to set it to 0.1mm since we would like to print the trophy in high-quality resolution. We can see how the change in layer height results in a slower printing time being displayed in WYSIWYFab's user interface (Figure 8).

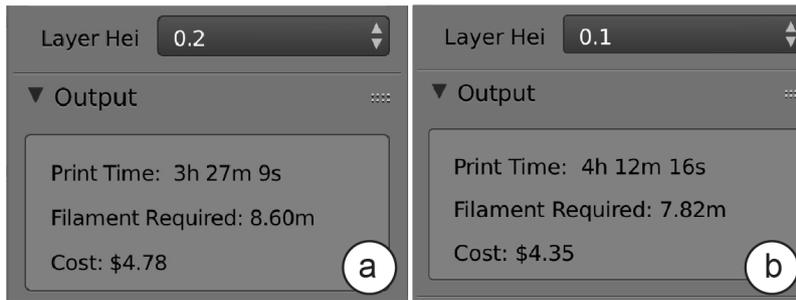


Figure 8: WYSIWYFab Layer Height: Changing the layer height updates the time.

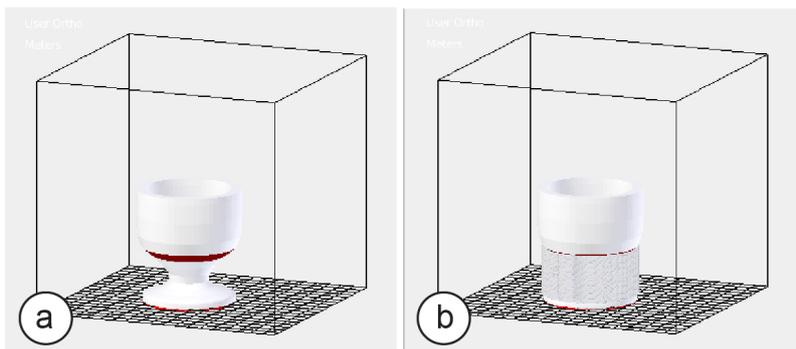


Figure 9: WYSIWYFab Support Material: (a) Red highlights show where support material will be required. (b) After hitting the ‘slice’ button, we can see the support structure and the printing time.

4.2.2 Displaying Support Material

Next, we start refining the trophy cup as illustrated in Figure 9. We start by adjusting the bottom part of the model, but then realize that our change in the geometry causes red highlights on some of the faces on the model. This indicates that these parts of the model would need support structures printed underneath them. We hit the *slice* button to get an estimate for the printing time this extra support would add.

As a result of slicing, we can see that this model would require an additional 1 hour and 7 minutes of printing time (5h 8min vs. 4h 12min) and would increase cost by \$0.95 in material (\$5.37 vs. \$4.35 from the previous version). We also see the support structures rendered in the view. Using the ‘show support’ checkbox, we can turn the support material geometry on/off to have a clear view of the model. We decide that we are satisfied with

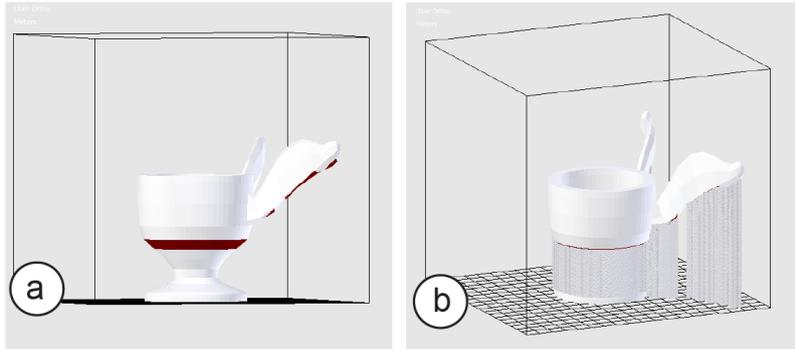


Figure 10: WYSIWYFab Support Geometry: Creating the wing geometry, it needs support. Modifying the wing geometry to not require support.

the printing time/cost including the support material and that it is more important for us to keep the design as is including the steep angles.

We continue our design and create the decorative wings that attach to the cup. While we are working on the wings, we now also see that some faces on the wings turn red as an indicator of support material (Figure 11). Again, we query the printing time to see how much time this extra support would require. The printing time increases from 5h 36 min to 6h and 49 min and the cost increased from \$5.37 to \$5.95. Since this will take too long, we decide to modify the wing geometry until the red highlights are gone. We hit the *slice* button again now fabrication time and cost are significantly reduced, i.e. only 5h and 10 min to print and \$4.71 of material.

4.2.3 Build Volume

Next, we want to integrate the award name on the trophy. While we are working on the text, we notice that there is not enough space for placing the text in sufficient size. To fix this issue, we decide to scale up the trophy cup (Figure 11).

As we scale, we notice that the wings that extend from the cup are being highlighted in red since they are reaching beyond the build volume (Figure 11a). We thus scale the model down until

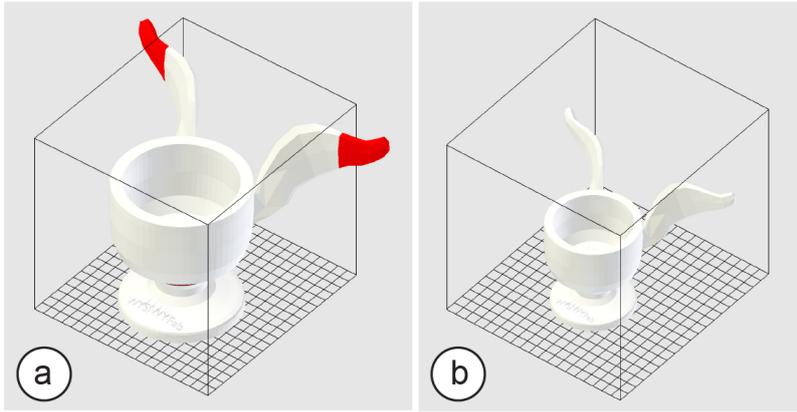


Figure 11: WYSIWYFab Volume Highlighting: (a) Too large for build volume. (b) Now it fits.

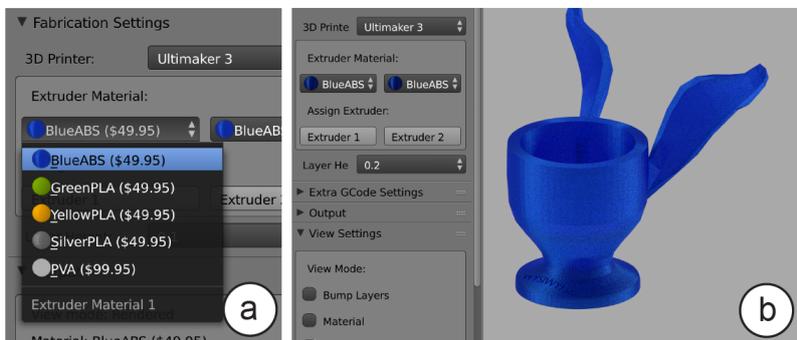


Figure 12: WYSIWYFab Material Color: (a) A dropdown allows the user to select from the available materials for a given printer. (b) Selecting a material renders it in the right color.

it fits (Figure 11b).

4.2.4 Materials

In the last step, we want to try different materials for our trophy cup. Using the ‘materials’ menu, we go through all the materials available on the Ultimaker 3 and then select ‘BlueABS’ for the main body of the sculpture (see Figure 12).

Before exporting our design for 3D printing, we want to do a final check on the model and hit the ‘render in high quality’ button. This provides us with a more realistic preview of how the material will look like when printed. We notice that the text is very hard to read when the model is printed in blue since

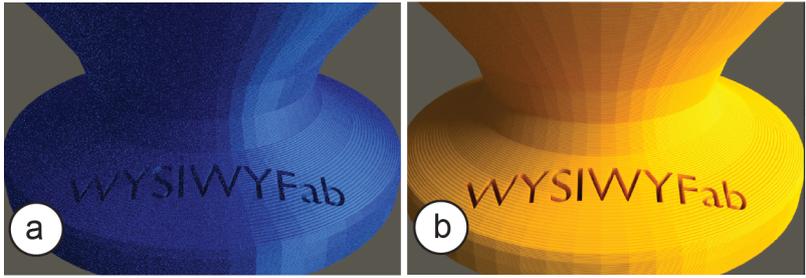


Figure 13: WYSIWYFab Material Color: (a) Blue vs. (b) yellow for readability.

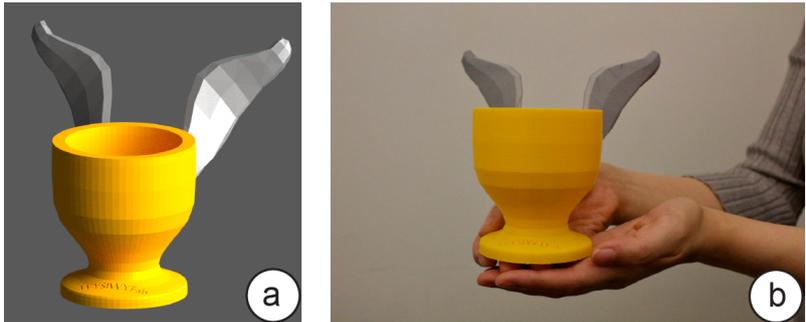


Figure 14: WYSIWYFab Model vs. Actual: (a) Rendering before export, (b) printed.

shadows do not add contrast to the material (Figure 13a). We thus decide to try a brighter color and select yellow, which after rendering turns out to be more readable (Figure 13b).

Finally, we get a request from one of our friends who asks us to use a different color for the wings than for the main cup body. Since two colors plus the support material required for the bottom part of the cup would require more extruders than we have available, we decide to adjust the remaining geometry that needed support until it is no longer required.

We are now done with our design and send it to the 3D printer. Figure 14a shows the final rendering of the object before export, and Figure 14b the final printed object.

4.3 Implementation

Our integrated editing tool is implemented as a plugin for the 3D editor Blender, i.e. it uses the Blender Python Library (BPY library) to interface with Blender's modeling tools and user interface. For slicing, we use the Cura Engine API that processes a 3D model into fabrication instructions (.gcode).

Blender Python Library.

<https://docs.blender.org/api/current/>
Cura Engine.

<https://github.com/Ultimaker/CuraEngine>

4.3.1 Generating a .json file with the correct printer settings

Before we can call Cura Engine's *slice()* function, we need to generate a .json file for each 3D printer. The .json file contains the print settings, such as layer-height, print speed, and support material angle. This only needs to be done once at the beginning when a new 3D printer is added to the WYSIWYFab editor.

To generate the .json file, we proceed in the following way: First, we manually open Cura, select the desired 3D printer (e.g., the Ultimaker 3), and load an .stl file. This automatically starts the slicing process in Cura and generates a log of all printing parameters in the cura.log file. On our operating system (macOS High Sierra), the file is located in the folder: Library Application Support Cura 2.3 cura.log. In the log file, the printing parameters are listed in the following format:

This is an example of a step that could be automated in a future version of the system, discussed further in Chapter 6.

```
support_bottom_height="1" raft_acceleration="4000"  
layer_start_x="213.0" etc.
```

We copy-paste this part of the cura.log file into a text file. Next time we open our main application, it automatically converts this parameter list into the .json file we need for the Cura Engine's *slice()* function.

4.3.2 Slicing with the Cura Engine

When the user hits the 'slice' button in our user interface, our system first exports the current model as an .stl file by using the

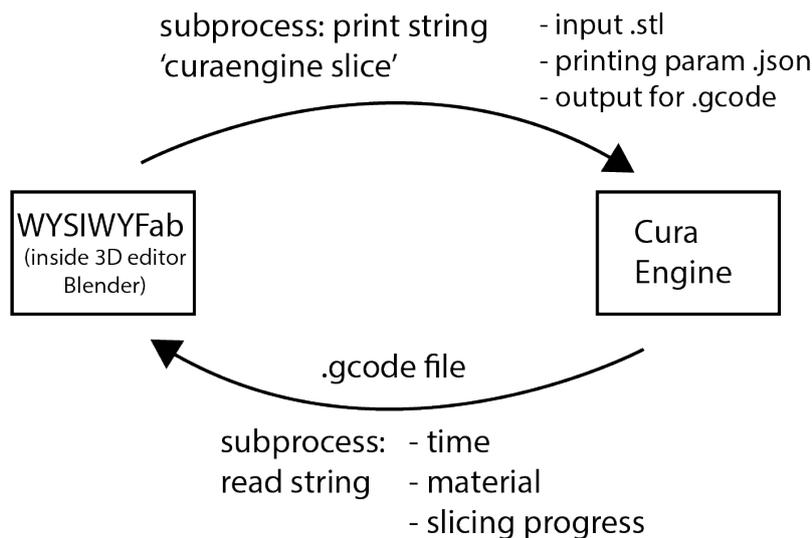


Figure 15: WYSIWYFab CuraEngine: WYSIWYFab queries information from the Cura Engine.

BPY library's functions for exporting meshes, and then calls the Cura Engine for slicing (Figure 15).

To make a call to the Cura Engine, we use the *subprocess* Python library from within Blender. Subprocess allows us to interface with the MacOS terminal.

While the .json file provides our baseline printing parameters for slicing, we can overwrite individual parameters by appending a parameter on the command line. For instance, if the user changes the layer height from fast 0.2mm to fine 0.1mm we can append '-s layer_height='0.1'.

4.3.3 Receiving .gcode + time / material values from slicing

When the Cura Engine is slicing, it prints the results for time and material onto the command line in the following format:

```
;TIME:3056; MATERIAL:200
```

Where time is reported in seconds and material is reported in millimeters of material. Using the subprocess library, we can

Giving subprocess a terminal command as a string, it will be executed in the terminal. For instance, to start the Cura Engine we give it the command 'curaengine slice' and give it the following parameters: location of the 3D model .stl file, location of the printing parameter .json file, and location of the .gcode output file.

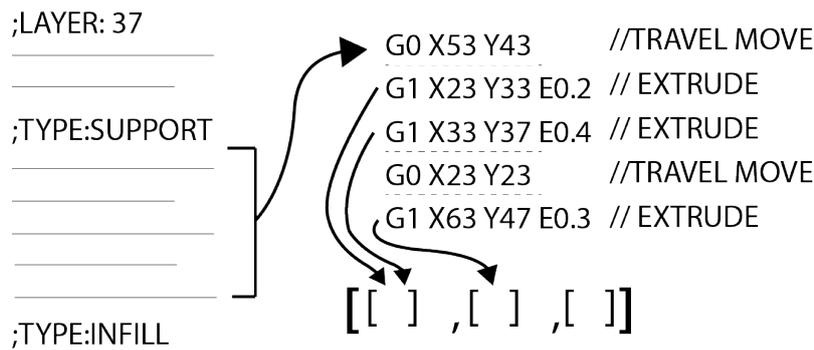


Figure 16: Processing GCode

read these strings from the command line and process them further in our main application.

During slicing, the Cura Engine also prints a range of other strings onto the command line. For instance, it reports which layer it is currently slicing, which we use as a time estimate for how long the slicing is going to take. We display the current slicing progress in the user interface.

Finally, the .gcode that the Cura Engine produces as a result of slicing is written into the output location we specified in the previous step.

4.3.4 Parsing the .gcode for the Support Material

In the next step, we parse the gcode to be able to render the support material in the Blender viewport (Figure 16). Since we only want to consider gcode commands that relate to support material and not to the infill or shells, we search for the ‘;TYPE:SUPPORT’ command in the .gcode file. We extract all gcode commands after this line until the gcode type changes, which is indicated by a line, such ‘;TYPE:INFILL’ or ‘;TYPE:SHELL’.

Now that we have extracted the gcode commands that relate to the support material, we still have to distinguish between G0 commands (travel move to a point, no extrude) and G1 commands (move to a point and extrude on the way).

When we first encounter a G1 command, we create a new point

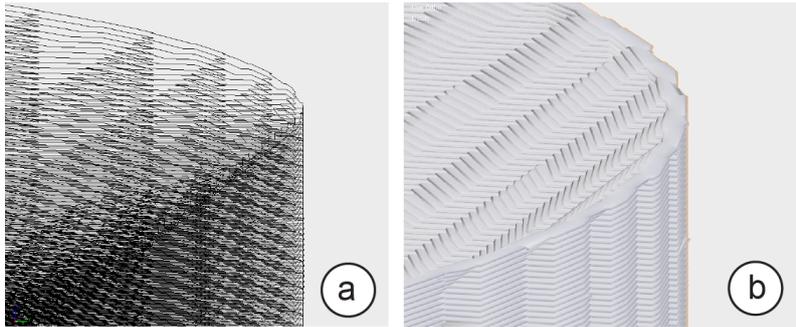


Figure 17: (a) The support material, extracted from processed gcode, is a series of connected points. (b) Adding thickness and texture for a more realistic look.

list and add the first G1 as the first point (see Figure 16). All subsequent G1 commands are then added as additional points to this list until we reach a G0 travel command. We end the current point list, and create a new one for a second path. We then repeat the process as described above with the next G1 commands. At the end of the process, we have a list of all paths that are being extruded.

The G1 command comes in the form G1 X5 Y7 E0.2. The G1 command does not contain a z-coordinate. Instead, the z-height is given via a layer number in the format of ‘;LAYER:37’. Since we know the layer height from our .json file, we can infer the z-coordinate by multiplying the layer number with the current layer height.

4.3.5 Rendering the Support Material

After parsing the .gcode, we still need to render the support material in the view. For this, we generate a ‘curve’ object in Blender and add to it a set of paths, each path representing one of the lists that contain the G1 extrude commands. As a result, the support material is displayed as 2D lines in the Blender view (see Figure 17a). In the next step, we give the 2D paths a thickness to render them as volumetric objects that more closely resemble the support material structure (Figure 17b).

Finally, we still have to translate the curve object that represents

the support material along the x and y-axis since the coordinate system of the Cura Engine does not match with Blender's coordinate system. We manually determined the translation factor by creating a cube at (0,0,0), exporting the cube as .stl file, slicing it with Cura, and then reimporting the .gcode and comparing the two positions.

4.3.6 Early Error Detection

Regions of the model that are outside the valid build volume, or have overhangs that will require supports are marked in red based on the printer's settings. We generate a procedural material based on these settings, so that users can view how their changes introduce or fix problems in real-time, without the need to slice the model.

To color regions of the model in red, all procedural materials are generated with additional node groups. The material is setup in the following way: node group #1 has the regular printing material and is shown by default. Node group #2 has the bright red color and is only shown when a face is outside the bounding box. Finally, node group #3 has the dark red color and is only shown if a face is steeper than the currently setup support material angle, and not at the lowest point on the model, which would not require supports as it is against the build plate.

4.4 User Study

We ran a user study to compare our integrated editing approach to the traditional two-step process of first 3D modeling and then slicing.

In our user study with 12 participants, we found that knowing about fabrication constraints early on helped users avoid mistakes that led to non-working designs and that it enabled them to fine-tune the geometry better than in the traditional approach in which modeling and slicing are separate steps.

In the rest of the section, I outline how the user study was conducted, and I explain the results in more detail.

Hypothesis

Our hypothesis was that with our integrated editing environment:

1. participants would be able to identify problems in the design faster, i.e. if the model is too large for the build volume and which faces cause the support material to be generated.
2. participants would be able to fix the problems faster, i.e. find a working configuration that would stay within size constraints and minimized printing time by removing support.

We collected both quantitative data and qualitative feedback to answer these questions.

Conditions

Our study had two conditions:

1. **3D Editor + Slicer:** In the 3D editor + slicer condition participants used the standard procedure in use today. After changing the 3D model in the 3D editor, participants had to export the model as an .stl file, and then load it into the slicer to see the printing time, where support material is generated, and if the model fits into the build volume. They then had to go back to the 3D editor to fix the issues and then export and slice again.
2. **Condition 2 WYSIWYFab:** In the second condition, participants used our new integrated editing environment,

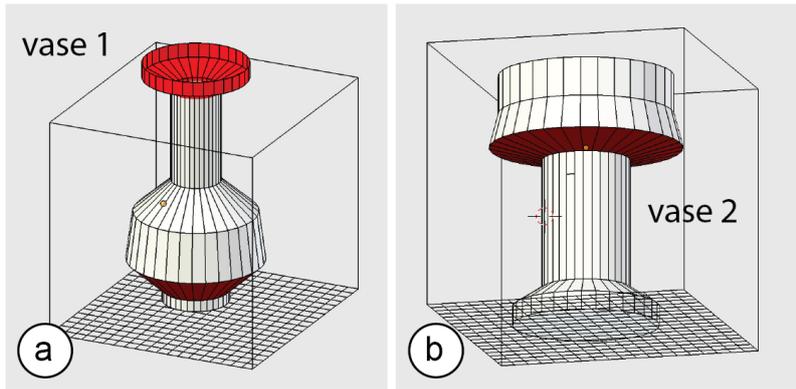


Figure 18: Vase models used for the study

i.e. in the 3D editor they were able to see the build volume and support material angle violations in real-time and they were able to use the 'slice' button to request the current printing time.

We ran a within-subjects experiment, i.e., participants participated in both conditions but were randomly assigned an order.

Task

In the study, participants were given a 3D model of a vase and were asked to make the vase as large as possible while making sure that it would still fit inside the build volume. In addition, we asked participants to optimize the design for fast 3D printing time. We told them that minor adjustments to the 3D models are ok but that the base of the vase should keep its size.

The initial versions of the 3D models that we gave participants at the beginning of the task are shown in Figure 18 with their respective problem areas highlighted in red. Both models had parts that had angles steeper than 45 degrees and thus generated a lot of support material. One vase was too large for the build volume while the other was smaller and thus could have been increased in scale.

They used a different 3D model for each condition, but always used 3D model #1 first and then 3D model #2. The task for

both models was the same.

Procedure

We conducted our study in the following phases:

Phase 1: Pre-test Questionnaire: We first collected basic information about participants' skill level. In particular, we asked if they had any prior experience with 3D modeling and 3D printing and if yes, how much and which tools they had used.

Phase 2: Introduction to 3D Modeling in Blender: After assessing their skill level, we gave participants an introduction to 3D modeling with Blender. In particular, we explained the 'move' and 'scale' tools that allowed participants to change the existing geometry of the 3D models we provided. Since the vase models were setup as a number of extruded circles (see Figure 18), all that participants had to do for the task was to move the circles up/down or scale them along the zaxis to change the steepness of the angle to prevent support material from being generated.

Phase 3: Introduction to 3D Printing: We then gave every participant an introduction to 3D printing. In particular, we explained that 3D printers only have a particular build volume and thus cannot print arbitrarily large. We also explained what support material is, that it takes a lot of extra printing time, and showed a picture that explains why the angle affects the need for support material (angles >45 degrees).

Phase 4: Explanation of Slicing: After that, we explained how to convert a 3D model for fabrication. Participants received the explanation for their corresponding condition:

In the 3D editor + slicer condition, we explained how participants can import an .stl file into the Cura slicer and how they can slice it. We pointed them at the time and material display inside of Cura and also showed that Cura will show a warning as a popup if the model is too large for the build volume. We also told them that they were not allowed to change any print

settings for the purpose of this study.

In the WYSIWYFab condition, we explained to participants how they could request the support material to be displayed by hitting the 'slice' button. We also showed them the time and material display inside of WYSIWYFab. Finally, we demonstrated how WYSIWYFab highlights parts of the model in red as soon as they extend beyond the build volume and also highlights parts that are responsible for causing support material to be generated. Similar to the 3D editor+ slicer condition, we told participants that for the purpose of the study, none of the print settings should be changed.

Phase 5: Modeling Task: After this introduction, participants were given the study task, i.e. we loaded the 3D model of vase #1 and asked them to make it as large as possible while fitting inside the build volume and making sure it requires minimal printing time. Participants had 15 minutes to complete the task.

Phase 6: Post-test Questionnaire Single Condition: After participants had completed their first condition, we gave them a post-test questionnaire. The questionnaire asked them which functionality in the tools they used helped them to get their task done, which functionality held them back, and what functionality they wished they had.

Phase 7: Modeling Task in Other Condition: After this, we introduced participants to the other study condition, i.e. if participants had done 3D editing + slicing first, they used WYSIWYFab and vice versa. We repeated Phase 3, i.e. introduced them to the other slicing tool and then repeated the modeling task, this time with model vase #2. Participants again had 15 minutes to complete the task.

Phase 8: Post-test Questionnaire Both Conditions: We then gave participants another questionnaire asking them to compare both tools they had used. In particular, we asked them, which of the two tools they preferred and why, if they have any further suggestions for improvement, and if they had any other use cases for which they thought the tools could be useful. Figure 19



Figure 19: Study Setup

shows the study setup we used throughout our experiment.

Captured Data

For each participant, we saved the following data: First, we took a screencast of the entire modeling session. Second, we collected .stl files of all intermediate versions of the models. In the WYSIWYFab condition those were generated each time participants hit the 'slice' button, for the 3D editor+ slicer condition we asked participants to save the .stl file under a new version number every time they exported the model (p1-v1.stl, p1-v2.stl etc). Finally, we also collected the data from the different questionnaires.

Participants

We recruited twelve participants (8 female, 4 male) from our institution. Participants' ages ranged between 18-31 years (mean=23.25, s=1.22).

Results

From our pre-test questionnaire, we found that most participants were beginners to 3D modeling and 3D printing. Only 1/12 participants stated that they felt competent in 3D modeling, 6/12 said they had done some 3D modeling by following tutorials but that they cannot model something without instructions, and 5/12 participants had never used any 3D modeling software. Similarly, for 3D printing, only 3/12 participants had seen a 3D printer in action (but somebody else did the slicing for them), 8/12 at least knew what a 3D printer is but had never seen one in action, and 1 participant did not know what a 3D printer is. From the participants who knew some 3D modeling, nobody had used Blender before.

In the next step, we analyzed the screencasts. For this, we coded the sequences by writing down the time stamps of the following events: slicing (either export of .stl to Cura or 'slice' button in WYSIWYFab), viewing slicing result (after slicing, was the build size valid: yes/no, support material removed: yes/no), adjusting angle, and adjusting size (every time the user moved a circle on the model).

Since WYSIWYFab uses highlighting of problem areas even without slicing, we found that when WYSIWYFab was used 7/12 participants immediately realized that the model had several problems. They started fixing the build volume and support material issues right away without querying time consuming slicing results. Participants then used slicing only for the final export.

In contrast, in the 3D editor+ slicer condition only 3/12 participants started modifying the model right away. Rather the majority of participants (9/12) decided to first export an .stl file and import it into the slicer to see what problems it had. From the participants that tried to fix the model without getting slicing results, nobody was successful. They all had to go back to the 3D editor, change the model, and export a second time.

When counting how often participants had to query slicing results to fix the problems, participants in the WYSIWYFab con-

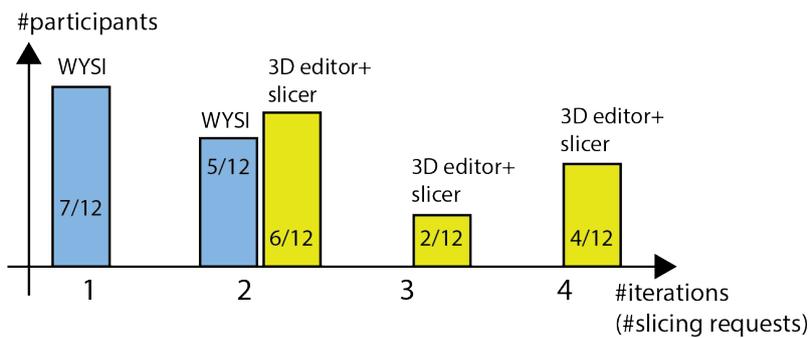


Figure 20: In WYSIWYFab, participants needed less iterations to solve all problems in the model.

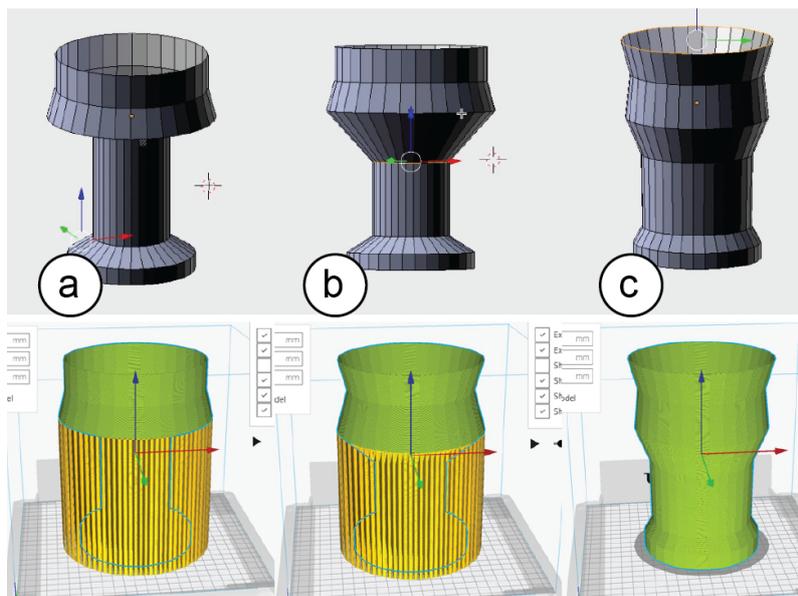


Figure 21: Participant 11 in 3D editor+slicer tries to remove support material: (a) start condition, (b) first model adjustment, (c) second model adjustment.

dition needed significantly fewer iterations (Figure 20). 7/12 participants had fixed all problems in the model before any slicing and thus only needed to export once, another 5/10 participant sliced once mainly to know not only where but also how much support material is generated. In contrast, in the 3D editor+slicer condition, nobody fixed all problems in the first iteration. Rather, in the 3D editor+slicer condition participants often had to export multiple times to fix a problem.

For instance, in the 3D editor+slicer condition, p3, p9, and p12 had to export and modify their model twice to get the size for the model to fit into the build volume (otherwise the Cura slicer

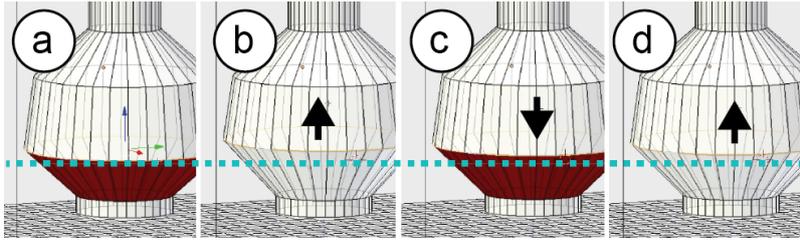


Figure 22: Participant 2 fine-tuning the support material angle.

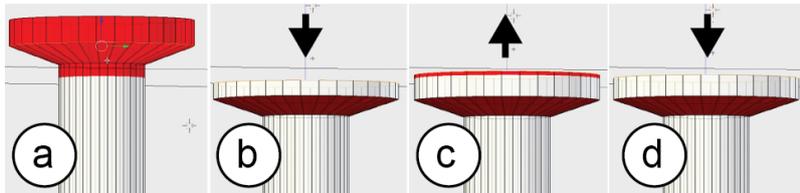


Figure 23: Participant 2 fine-tuning the size of the model to maximize its size while still fitting it inside the build volume.

would not even start to slice). Similarly, p4 and p11 had to adjust the support angles twice to get it right (Figure 21).

As a result of the time pressure and the repeated failure, we observed that several participants got impatient and made large adjustments to their models to make them fit within fabrication constraints. As a result, models looked very different when compared with the original versions we gave participants (see Figure 21 for an example).

In contrast, in the WYSIWYFab condition, participants tended to tweak the geometry to get the red highlighting to just disappear without modifying the shape too much. Figure 22 shows P2’s approach to removing the support material by iteratively approximating the correct angle.

Similarly, adjusting the height of the model to fit into the build volume was done in an iterative approach by repeatedly moving the top part of the model up and down. Figure 23 continues with screenshots from P2’s screencast and shows how the participant gradually adjusts the geometry.

We also analyzed the time participants needed until they were satisfied with their result. For vase #1, participants in the WYSIWYFab condition were faster with 6.33 min on average (slowest

participant 8 min) compared to 10.67 min for 3D editing+slicing (slowest participants 15 min). Similarly, for vase #2, WYSIWYFab participants were faster than 3D editing+slicer participants but since the model was easier to fix the difference was less pronounced (average time 5.83 min for WYSIWYFab vs. 6.2 min for 3D editing+slicing). Based on a t-test, the time required in WYSIWYFab was significantly less than the time required with 3D editing + slicing with p value = 0.028.

Post-test questionnaires: In the post-test questionnaires, we found that 10 out of 12 participants preferred WYSIWYFab over 3D editing+slicing. When asked why, participants stated: 'It's unnecessarily redundant to go back and forth, and since you cannot change the model in Cura, you can't maximize size precisely, you have to guess and check.' (p2) 'It [WYSIWYFab] is much more streamlined and involves less task-switching, making it more convenient and easier to use.' (p3) '[In WYSIWYFab] I could see where support material would be needed before slicing which meant I needed to go through fewer design iterations which saved me a lot of time.' (p4). '[WYSIWYFab] It saves time and effort and much easier to operate.' (p7) 'Cura is not [integrated] in the software Blender, so I need to compare the model to edit in Blender and the layer view result in Cura to get my task done.' (p7) 'The red coloring helped me keep the object in the box and know where problem areas were in regards to needing support material. The "slicer" button functionality was also helpful in evaluating efficiency.' (p10)

The reason one participant preferred 3D editing+slicing was that the slicing time was shorter than in WYSIWYFab, which is due to some of the custom processing we add on top of what we receive from the Cura slicer. The other participant preferred 3D editing+slicing because he/she preferred the more colorful user interface in Cura.

When other participants were asked which features they would like to have in the future, most participants also focused on improving the slicing time: 'The slicing, especially in WYSIWYFab, takes too long.' (p4) '[...] making it faster would be the biggest improvement.' (p7)

One participant had an interesting suggestion, i.e. to display the time/material over multiple versions rather than only for the current version: ‘It’s better to show the differences between two results = how much time of 3D print has been saved after the modification.’ (p9) We will integrate this feature in the next version of our tool.

While this experiment only gives a first indication that integrating 3D editing with slicing allows users to identify problems faster and fix them in fewer iterations, the results are promising as indicated by both our quantitative and qualitative results.

4.5 Discussion

The main contribution of this project is the idea to integrate 3D modeling and slicing into a single integrated editing environment. Since integrated editing and slicing provides information on printing time, cost, and fabrication constraints, such as the build volume, early on, users are able to better trade off cost, time, and geometry changes for a design.

To demonstrate the benefits of integrated modeling and slicing, we implemented a system based on the 3D editor Blender and the slicer Cura. In section user study, we show how 12 participants used our system to optimize a design. Participants in the integrated modeling + slicing condition identified problems in the model faster and were able to fix them with smaller changes to the geometry than participants that used the traditional workflow.

However, our approach is also subject to the following limitations:

1. Slicing time: since slicing is still a time-consuming process, users have to wait for the slicer to finish before they can continue modeling. For the different models shown in the above walkthrough, slicing took between 30 seconds for the small initial model to 2.5 min for the slow-

est model. An interesting area of future work would be to explore different ways to optimize slicing time, for instance, by providing users with a low-fidelity slicing first (e.g., layer resolution 1mm) and then gradually refining the precision. Another approach could be to only re-slice the parts that actually changed and to reuse the information from the rest.

2. **Design Optimization for a Specific Machine:** Since users optimize the 3D model for a specific 3D printer, users have to repeat the process if they change to a different fabrication device later on. One of our future lines of work will thus investigate how to best translate a design from one printer specification to another (e.g., if the user indicated that the model should be as large as possible, we could automatically scale the model to the maximum build volume depending on the selected 3D printer).
3. **Influence on the Design Process:** Knowing about the fabrication constraints early on might restrict the creative process of designers. We thus think our editing environment is most suitable in the intermediate design phase, i.e. when designers already have a rough design in mind but are not set on the details.

4.5.1 Expanding to other fabrication processes

While our work has focused on fabrication constraints related to 3D printing with a hot extruder nozzle (FDM 3D printing), our concept of integrating modeling and slicing goes beyond one fabrication technology. For instance, one can imagine a similar editing environment for powder-based 3D printing (e.g., SLS) or resin-based 3D printing (SLA). Similarly, our work could be expanded to work for subtractive fabrication processes, such as laser cutting or milling.

4.5.2 Switching the 3D Printer

In this project, we focused on optimizing a design for a single 3D printer. Such a scenario is realistic for, e.g., home 3D printing, when users only have one device available. However, in the next step, we want to investigate scenarios in which users switch between multiple devices. For instance, users might want to first use a low-resolution 3D printer with low-cost materials for the early versions of a design, and a high-resolution 3D printer with more expensive material for the final versions. To allow users to switch between different devices, we need to investigate how to transfer a design from one representation to another.

4.6 Conclusion

We presented WYSIWYFab, an integrated editing environment that combines 3D modeling and slicing to provide users with a preview of how their object will be fabricated already while 3D modeling. We demonstrated how knowing about fabrication constraints allows users to identify problems in their designs faster and allows them to find a working solution in less iterations.

Chapter 5

CSlice: Constraint Centered Slicing

What I cannot create, I do not understand.

*Richard Feynman
Machine That Changed The
World (1988)*

In this chapter I describe CSlice, which is both a new slicer implementation and a concept that reverses the traditional slicing workflow. First, I then discuss the user study we designed and carried out. Next, I discuss the interface and implementation of the system. Finally, I discuss how CSlice can be improved and expanded on in the future.

This project was designed and implemented during the Fall 2018 and Spring 2019 semester. The project was implemented by myself and Faraz Faruqi, under the supervision of Professor Stefanie Mueller.

The work was submitted as a full paper to the *Proceedings of the 32nd ACM Symposium on User Interface Software and Technology*, abbreviated UIST 2019. As of the time of writing this, the paper is still under review.

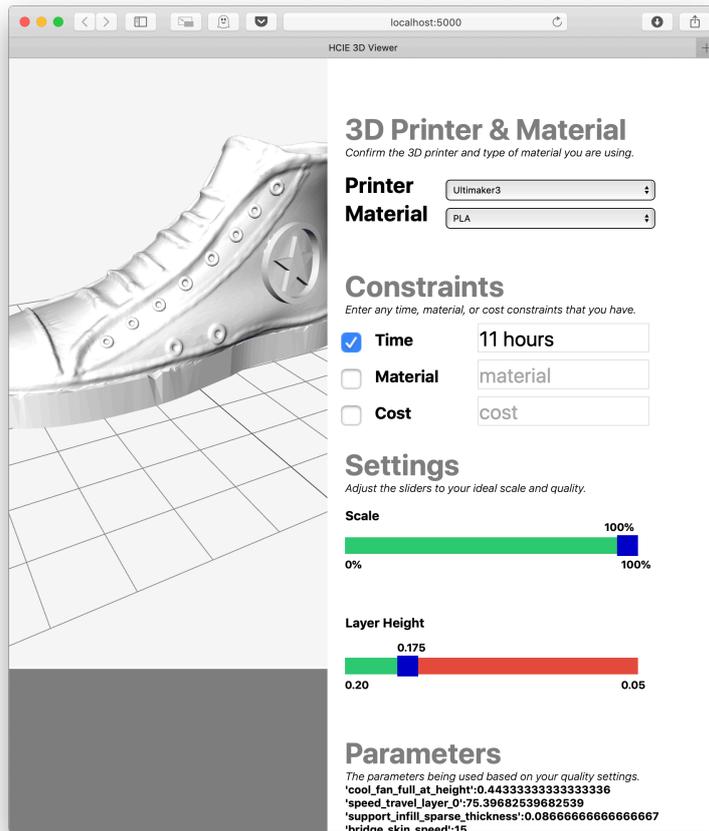


Figure 24: User-Facing CSlice Interface

5.1 Motivation

CSlice is a slicer that automatically selects slicing parameters based on users' constraints on time, material, and cost for a 3D print (see Figure 24). Previously, a user that needed to print within 2 hours would have to manipulate printer parameters by guessing-and-checking. Instead, CSlice enables the user to simply input the "2 hour" constraint, and the printing parameters will automatically be determined which will print the object on time.

In addition to reversing the traditional slicing workflow, CSlice also interpolates between slicing profiles (e.g., layer height of 0.175mm rather than only 0.1mm/0.2mm), which allows it to further fine-tune the results.

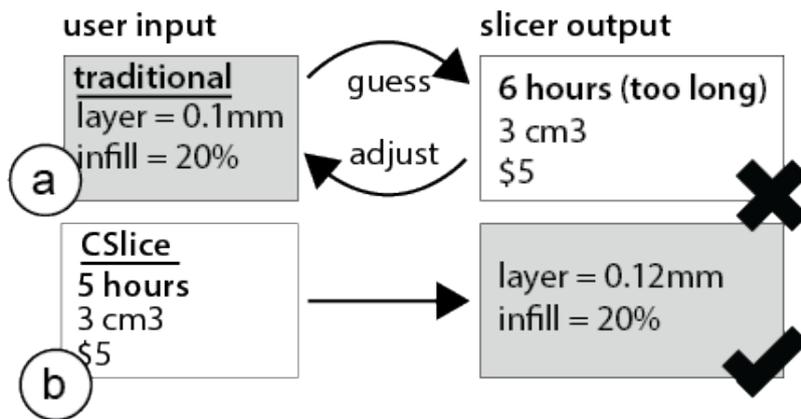


Figure 25: The traditional slicing workflow is shown in Part A. The CSlice workflow is shown in Part B.

CSlice uses parallel slicing in the cloud to compute its results and produces better quality-design trade-offs compared to traditional slicers, while only requiring a few cents of compute time.

When 3D printing an object, users often have to make trade-offs between the quality of the fabricated prototype and the time, material, and budget available². For instance, a designer might have to change a prototype quickly before a client meeting in 3 hours; a maker in a FabLab discovers that only 100g of material are left from a roll of filament the maker wants to use; and a student working on a tight class budget might only have \$3 worth of material left for printing an assignment.

To find slicing settings that produce the object within those time, cost, and material constraints, users have to manually change the print settings in today's slicers in a trial-and-error process in one of two ways. One way is by selecting from a range of pre-defined slicing profiles (*fine*, *normal*, *fast*). The alternative is to change slicing parameters, such as the layer-height and infill, one-by-one. After each change, the object is re-sliced, which can take several minutes since slicing is slow, and only afterwards the print time and required material are displayed. Thus, after guessing an initial set of slicing parameters and seeing the slicing result, users have to iteratively narrow down the search space until they find a working solution.

In this project, we suggest to reverse the workflow. Rather than

having users guess the slicing settings that match their constraint (seen in Figure 25A), our slicer CSlice asks users how much time, material, and cost they have available and then suggests slicer settings that deliver the print within those constraints (seen in Figure 25B). Therefore users no longer have to use a trial-and-error process to find a working solution. Instead, they are presented with a working solution right away.

In this project, we contribute:

- an overview of current challenges when users try to slice an object under time, material, or cost constraints, including a formative user study with 12 participants
- the concept of constraint slicing, i.e. asking users for their time, material, cost constraint first, then finding matching slicing settings
- an implementation of a constraint-based slicer that uses parallel slicing in the cloud
- an evaluation that shows that constraint-based slicing leads to better quality design trade-offs when compared to users using traditional slicers

Our work was initially motivated by our own experiences in using slicing tools to produce 3D prints on time (e.g., for a paper deadline) or within a specific budget (e.g., when using an expensive 3D printing service, such as Shapeways). Below, we first report on issues we found when using traditional slicers to estimate time and cost for a 3D print, and then discuss the results of a formative user study on users' strategies when slicing within a time or cost constraint. <https://www.shapeways.com/>

5.1.1 Challenges When Estimating Time and Cost

When slicing a model for a time or cost constraint, we found several issues in today's slicer:

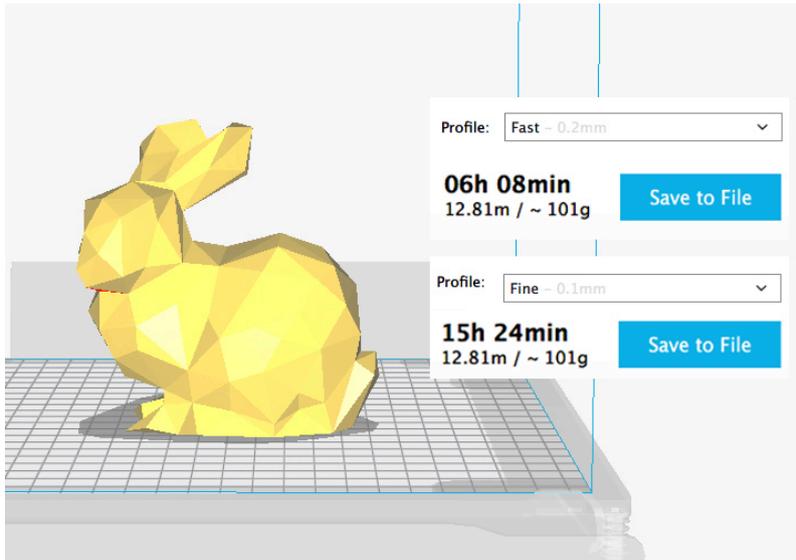


Figure 26: While one may expect that 0.1mm prints twice as slow as 0.2mm, it takes almost 2.5 times as long making it hard for users to predict the time.

Unclear which settings to start with

Since the print time and required material are displayed only after slicing has completed, it is unclear which slicer settings users should start with. Only after the initial slicing process finished, users have enough information to subsequently select a higher or lower resolution profile.

Non-linear Relationship Between Resolution Profiles

Even when users know after the initial slicing that their print requires a lower resolution profile to print within their time constraint, it is unclear, which resolution profile to select next. While one may expect that slicing with fine (0.1mm) takes twice as long as slicing with fast (0.2mm), it in fact takes almost 2.5 times as long (see Figure 26, result from Ultimaker Cura, model from Thingiverse). We found that the reason for this is that for profiles with different layer heights, other settings change as well, i.e., finer layers require a slower print speed. Therefore, the relationship between slicing profiles is non-linear, making it hard for users to predict the print time for their model.

<https://thingiverse.com/thing:151081>

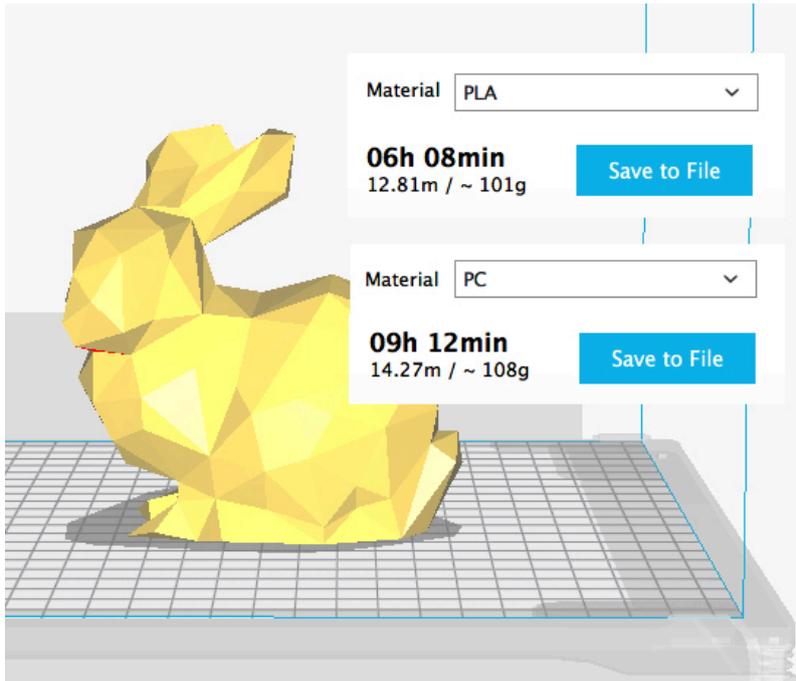


Figure 27: Using the same resolution profile with a different material leads to vastly different print times. Here switching from PLA to PC adds 50% print time.

Only Few Profiles to Choose From

Even when users found a profile that creates their 3D print within the time constraint, it is likely that the profile will not make optimal use of the available print time. The reason for this is that existing slicing profiles provide only few options to choose from (e.g., fast (0.2mm), normal (0.15mm), and fine (0.1mm)). Thus, if users have 10 hours available, ‘normal’ may produce the object in 11 hours (too slow), while ‘fast’ may produce the object in 7 hours (at the expense of a worse printing resolution than necessary).

Switching Material Profiles

Even when users have settled on a resolution profile (e.g., 0.2mm), switching to a different material requires users to start all over (Figure 27). For instance, switching from the standard PLA to the stronger PC material increases print time by 50%; in the case shown in Figure 27 this amounts to three extra hours of

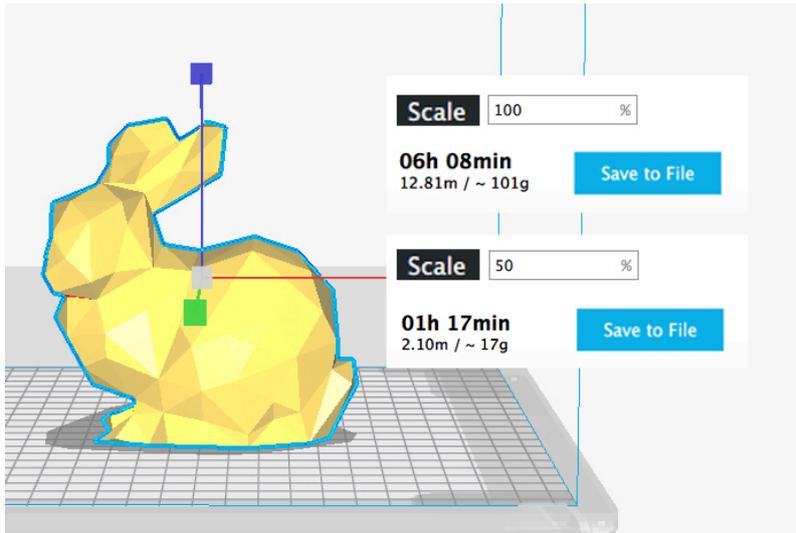


Figure 28: Since models are volumetric, it is difficult to estimate the effect of scaling on time/cost. Here scaling the model by 5% reduces print time by over 75%.

printing.

The reason for this is that each material requires different slicing settings, such as different print speeds and wall thicknesses. Thus, every time users switch the material, they have to restart the entire process of finding matching slicing settings.

Effect of Scaling on Print Time/Material: Finally, as users scale the model to save time and material, it is difficult for users to estimate how much the model should be scaled. For example, scaling a model by 50% may reduce the print time by 75% (see Figure 28). Thus, users have to apply trial-and-error to find the right trade-off between size and print time.

5.1.2 Slicing Strategies Employed by Users

While the above analysis demonstrates some of the issues in today's slicing workflow, we also collected quantitative data on users' slicing strategies to further motivate our work.

Tasks: We gave participants four 3D models and ask them to slice each one within a specific time, material, or cost constraint. The study was within-subjects, i.e. each participant completed

all tasks but tasks were randomly assigned an order.

In task #1, participants were asked to slice the 3D model of a shoe for a customer presentation on the same day. They were asked to print the shoe with PLA material in under 11 hours.

In task #2, participants were asked to switch the material from PLA to Nylon to print the shoe in a softer material. Since some time had passed, we asked participants to slice the model so that it prints in under 9 hours.

In task #3, participants were asked to print a Christmas ornament with only 60g of material left, i.e., the FabLab they were using was running out of the material.

In task #4, participants were asked to print the shade of an interactive lamp with only \$5 left, i.e. they had already spent the rest of the budget for this class project on the electronics and previous iterations.

We asked participants to print the objects as close in size as possible while also ensuring high print quality.

Participants

We recruited 12 participants (5 females, avg. age 23, std. 2.55) with prior exposure to 3D printing and slicing.

Procedure

We first introduced participants to the Ultimaker Cura slicer, i.e. we showed them how to import a model, how to move/scale/rotate a model, and explained where to find the slicing profiles and the individual slicer settings. We then assigned the tasks from the scenarios in a random order. The tasks had no time limit assigned, i.e. participants were allowed to take as much time as they wanted. After they finished all tasks, participants filled out a post-study questionnaire. The study took ca. half an hour

(avg. 31.3min, std. 11.7min) and participants received \$20.

Collected Data

We recorded screencasts and collected the .gcode files participants exported at the end of each task.

Data Analysis

We encoded the screencasts in the following way: Every time a participant changed a setting, we noted the setting name, the previous value for the setting, and the new value for the setting. We also noted the resulting fabrication time and required material for the model after each setting change. Finally, we also noted how long it took participants to find a working solution.

5.1.3 Results

All participants found slicing parameters for each of the models that would print the models within the assigned time, material, or cost constraint.

Overall, participants employed the trial-and-error strategy as one would expect from our previous analysis. We show the results of two representative scenarios in detail in Figure 29 (task #1: time constraint) and Figure 31 (task #3: material constraint).

As shown in the graphs, participants iteratively narrowed down the search space. Across all four scenarios, participants sliced an average of 8.1 times (std: 5.2 times) to check the result of changing one or multiple settings. In total, they spent an average of 7.8 minutes (std: 4.1min) until they had finished the task; the time was spent either waiting for the slicer or deciding which settings to change.

The final slicing result participants generated deviated from the

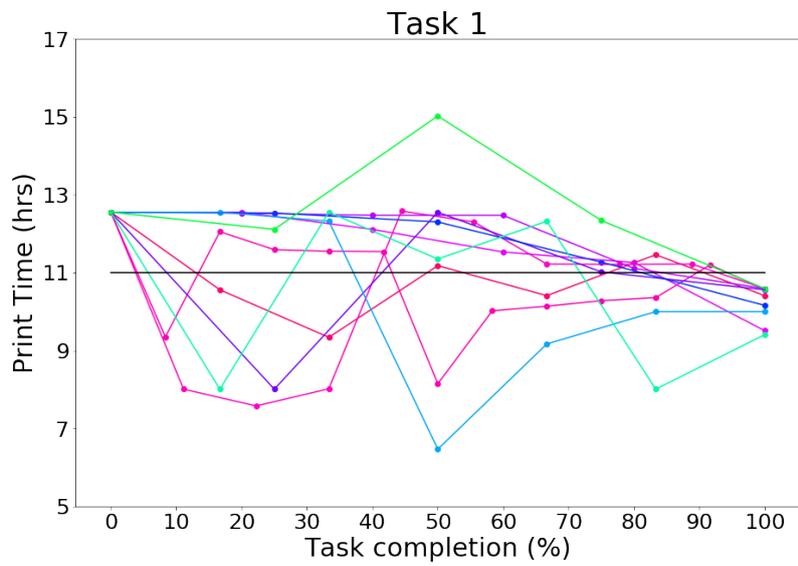


Figure 29: Results From Task 1: Participants iteratively narrowed down the search space. Black line represents the time constraint

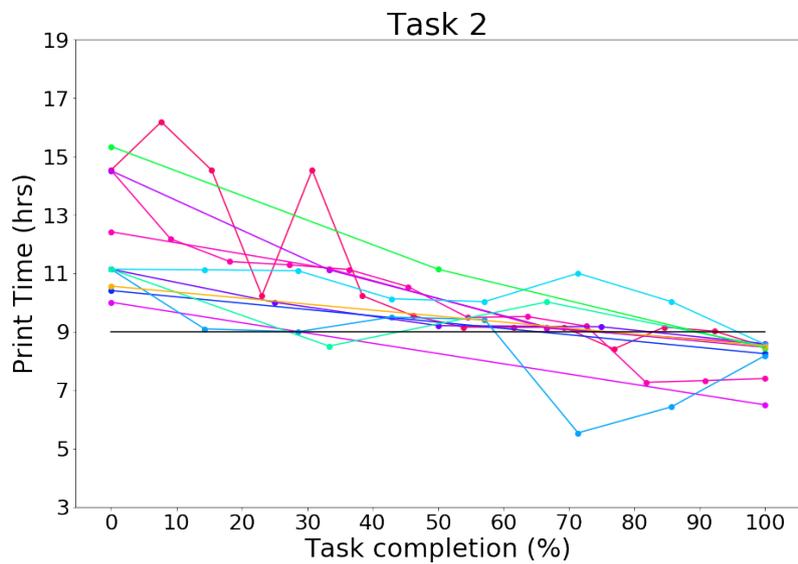


Figure 30: Results From Task 2: Participants iteratively narrowed down the search space, but started at different points based on their task 1 results. Black line represents the time constraint

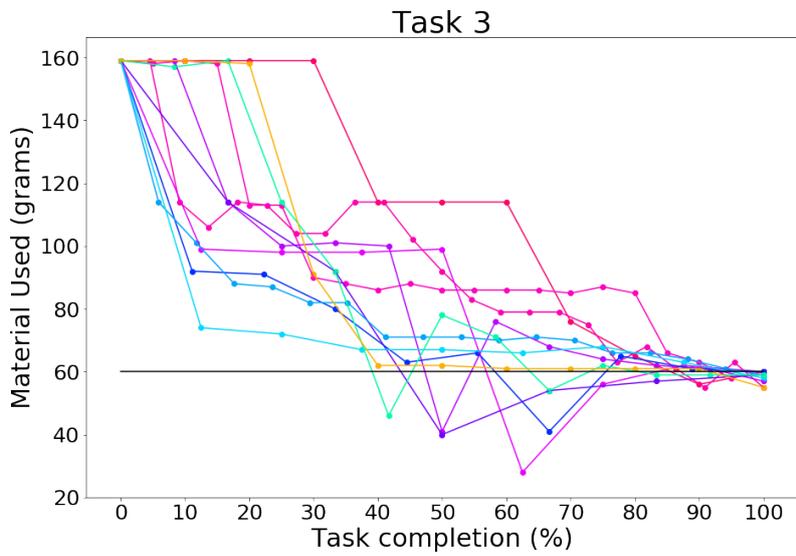


Figure 31: Results From Task 3: Participants iteratively narrowed down the search space. Black line represents the material constraint

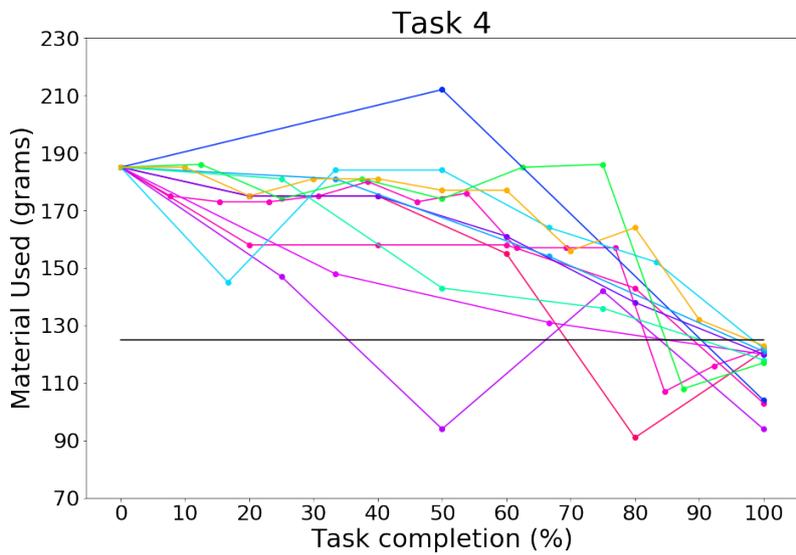


Figure 32: Results From Task 4: Participants iteratively narrowed down the search space. Black line represents the material constraint

target by 6.82% on average (std. 6.73%). All 12 participants used scaling as a means to reduce print time and required material. 7/12 participants changed the slicing profile at least once.

Since we did not restrict participants to only use the slicing profiles, participants modified a wide variety of individual slicer settings to find a working solution. In the individual slicer settings, we found that infill density (12/12) was modified most often. Several participants reduced infill density close to 0%, which provided great time and material savings but likely results in an invalid print. Similarly, 9/12 participants changed the layer height individually without making the necessary changes to correlated settings, such as the print speed, likely resulting in invalid prints as well.

This raises an important issue: While, as discussed, all participants found a working solution, i.e. slicing settings that would produce the object within the time or material constraints, the resulting prints may fail. Participants also expressed uncertainty about their decisions: ‘When deviating from profiles, I wasn't sure how my print would turn out.’ (p10), ‘[it would be good to have] better suggestions on what is recommended not to change’ (p2). In a study, Hudson et al.¹⁵, reported on similar issues, i.e. in an interview with a shop manager, the shop manager said ‘Sometimes we'll look at their file and say It might be a good idea to add supports. But the supports and the raft add time, so often they turn them off without really understanding... that can produce a bad print.’ This was confirmed in our own interview with a local shop manager who said that when users go on to change individual slicer settings, they go to extreme measures: ‘they make the wall thickness so thin to save time, it will definitely break apart after printing’.

We next report on additional qualitative feedback from our post-study questionnaire:

Qualitative Feedback: When asked what in the slicer held them back from finishing the task, participants said: ‘no intuition or indication of what would affect speed/weight and by how much. Had to do ridiculous binary searching to find optimal

values' (p1), 'changing things back and forth when I didn't know what effect they were having.' (p3).

Qualitative Feedback: When asked what in the slicer held them back from finishing the task, participants said: 'no intuition or indication of what would affect speed/weight and by how much. Had to do ridiculous binary searching to find optimal values' (p1), 'changing things back and forth when I didn't know what effect they were having.' (p3).

When asked how they would improve the slicing tool, several participants suggested: 'If I can input a target print time, or filament usage, and it can recommend settings for me, then I have a baseline to start with.' (p11), 'I put in my constraint (be it time or material) and the slicer automatically optimizes the settings to meet my needs.' (p5) 'I wish there was a more efficient way to set specs (such as # of grams to use) and have the slicer compute the settings based on some desired quality heuristics.' (p10).

5.1.4 Granular User Study Results

Beyond the summarized results of the user study in the previous section, we also recorded many granular details during the user study. In this section, we report more fine grained results from our user study.

Figure 33 shows how many times a given user study participants used a given slicing parameter. This shows a wide range of results. Some parameters were used by a single participant, others were used by nearly all participants.

Figure 34 shows how many times a give parameter was used across all participants and all tasks. Here, it's clear that certain parameters, such as infill density were used very heavily.

Figure 35 shows how many times each participant needed to reslice their object to meet the constraint for every task. This shows that task 3 was the most difficult to complete, however

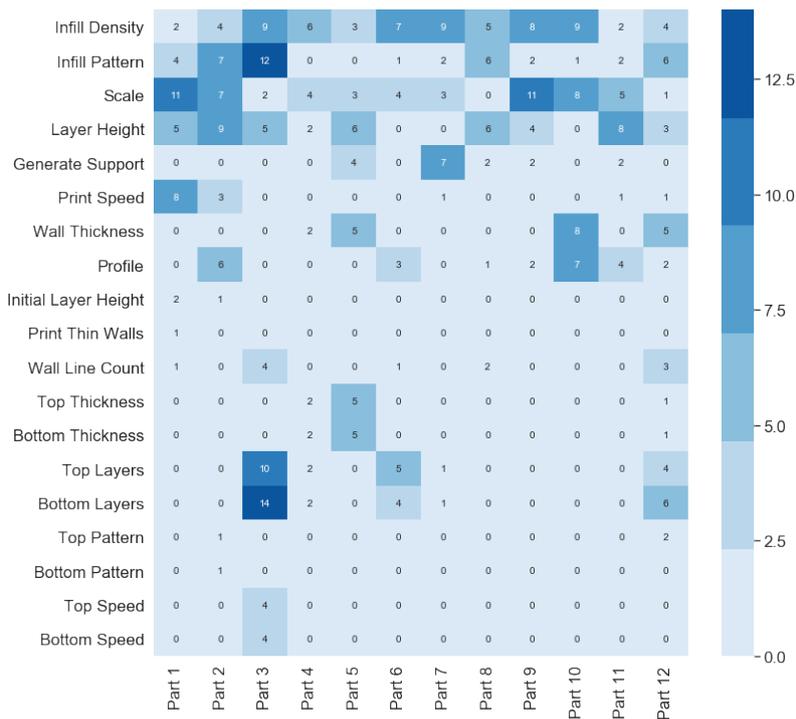


Figure 33: Parameters Used By Participants: This chart shows which parameters were used by which participants during the user study. Darker blue is an increased usage of the parameter.

there was also variation between participants. For example, many participants in task 1 were able to complete the task in 5 attempts, but Participant 12 required 20 re-slices.

5.2 Environment and Walkthrough

To solve the issues mentioned above, we developed CSlice, a constraint-based slicer that asks users to enter their constraint first and then finds matching slicer settings that fabricate the model within the constraint.

In addition to facilitating the process of finding settings that cause a 3D model to print within a time and material constraint, C-Slice also guarantees a valid print since it only interpolates between existing valid slicing profiles.

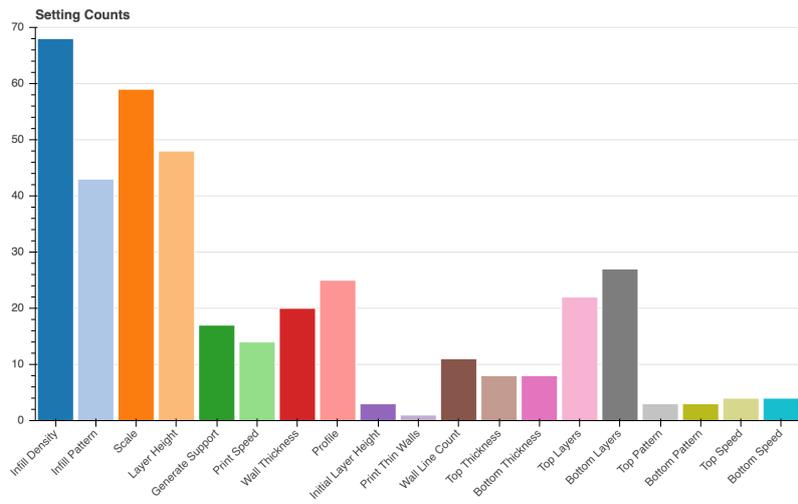


Figure 34: Slicing Parameters Count: Slicing parameters used, displayed by how many times that parameter was used across all participants.

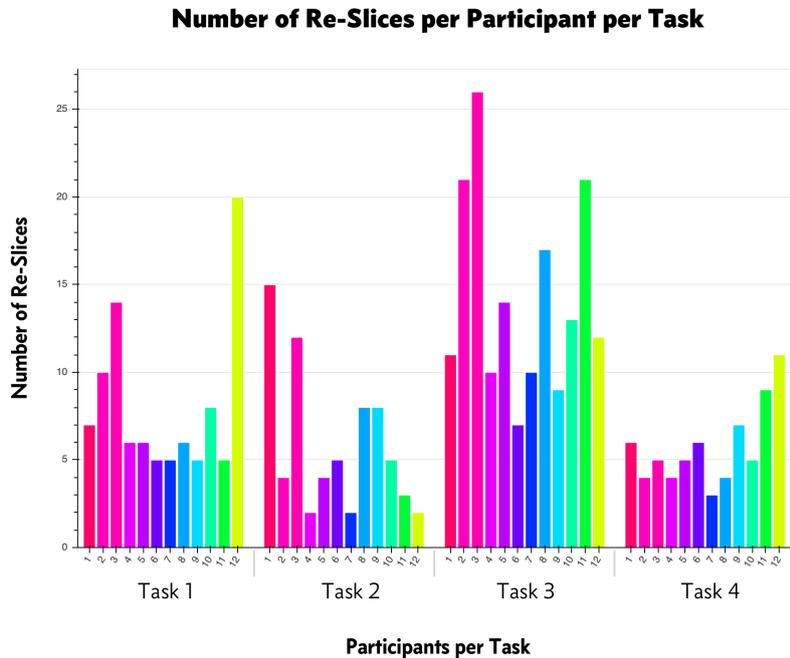


Figure 35: Re-Slices Per Participant: How many times each participant had to re-slice until the constraint was met, for each task.

5.2.1 User Interface

Figure 1 shows the user interface of CSlice: After loading a 3D model in the view, users enter their time, material, or cost constraints in the corresponding text fields. CSlice then starts searching the space of slicing profiles for profiles that have a print time or material cost smaller than the entered constraint. If multiple constraints were entered, CSlice only selects slicing profiles that fulfill all constraints.

By default, CSlice picks the slicing profile that minimizes changes in size at the expense of lower print. The reasoning behind this is that many printed models are tested for ergonomic fit or mechanical function, which requires them to be printed in full size. However, if size is not important, users also have the option to trade-off size and quality using the sliders provided in CSlice's user interface. By dragging the 'scale' slider towards the lower end of the range the model becomes smaller, which frees up additional print time that in turn automatically increases the print quality.

5.2.2 Slicing Time for a Constrained-Centered Slicer

CSlice populates the sliders based on actual slicing results, which gives a more precise preview of the printing time and required material than a prediction-based method could. To provide all values for the sliders, CSlice slices the model several times at different scales and across different profiles (in our examples, we use 20 different scales and 20 different profile resolutions leading to 400 slicing processes).

To keep the slicing time similar to traditional slicers, CSlice uses parallel slicing in the cloud (see section 'Implementation'). The total slicing time therefore equals the time for a single slicing process (i.e., the time required for the profile that needs the most slicing time, which is the model at 100% scale at the finest resolution) plus the time to upload the model in .stl file format to the cloud. As we will show later in this chapter, using parallel

slicing in the cloud only comes at the expense of a few cents of cloud time when using Amazon Web Services and is even free for the first 15 sliced models per month when using Amazon's fastest machines (37 hours free cloud time) or 36 sliced models per month when using Amazon's slowest machines (888 hours free cloud time).

Since all slicing profiles are computed as soon as the user loads a 3D model, no additional slicing is required when users change the sliders to explore different size-quality trade-offs. Thus, users can explore different trade-offs at interactive speeds, which is not the case in today's slicers. Even when users decide to change the constraint, no additional slicing is required since CSlice only has to select from the already computed results for print time and material cost for each profile.

5.2.3 Walkthrough

Task #1: To 3D print the shoe on time, we enter our time constraint, i.e. 11 hours, into the corresponding text box in CSlice and hit the 'slice' button (Figure 1). After 4.25min of slicing, CSlice presents us with a solution that requires 10.85/11h print time. The solution minimizes scaling, i.e. is able to keep the model at 100% while printing at 0.175mm layer resolution. Since keeping the size is the most important factor for us, we are all set.

Task #2: To print the shoe in a more flexible material, we next select Nylon as the material (Figure 36). Since Nylon uses different slicing settings, CSlice recomputes the slicing results in the background and then updates the sliders to still match the 11 hours time constraint. Since some time has passed and we only have 9 hours left, we next update the time constraint in the text box and CSlice updates the slider values for us at no extra time. After 3.75min of slicing, CSlice presents us with a solution that requires 08.72h/9.00h print time. The solution minimizes scaling, i.e. is able to keep the model at 100% while printing at 0.2mm layer resolution. Since keeping the size is still the most important factor for us, we are all set.

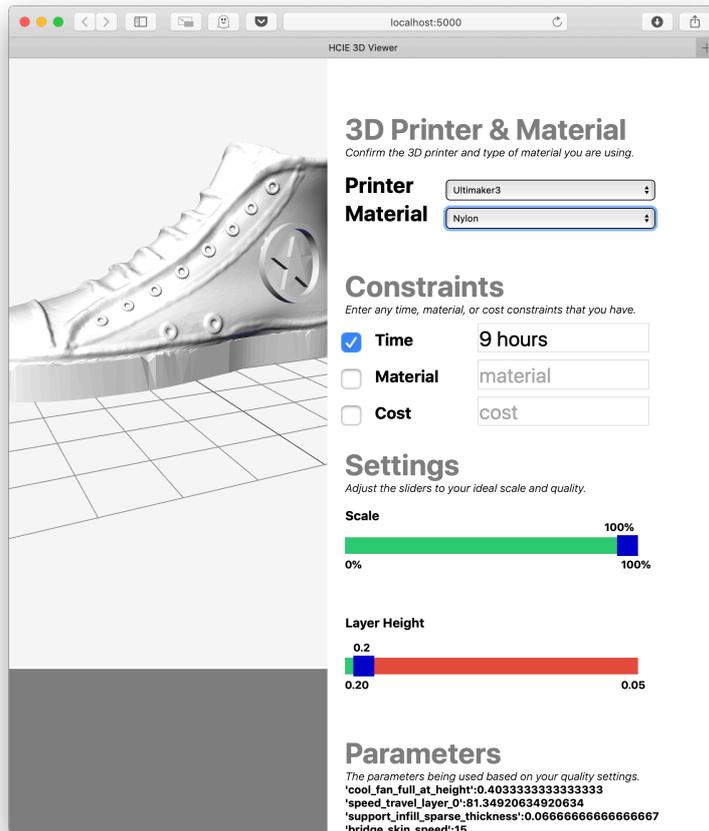


Figure 36: CSlice recomputes the scale and resolution when we switch over to Nylon as the material.

Task #3: To 3D print the Christmas ornament with the material available, we first load the new 3D model and then enter our material constraint, i.e. 60g, into the corresponding text box in CSlice (Figure 37). After 2 minutes of slicing, CSlice presents us with a solution that requires 53.2g/60g material. The solution scaled the model to 65% to make it fit within the available material. Since print time was not a constraint, the solution uses a layer resolution of 0.08mm.

Task #4: To make the lamp shade print for \$5 budget, we enter the cost in the corresponding field and hit the 'slice' button. After 7 min of slicing, CSlice presents us with a solution that requires \$4.98/\$5.00 material. Since cost is a function of material use, the solution scaled the model to 80% to make it fit within the budget. Since print time was not a constraint, the solution uses the highest layer resolution available, i.e. 0.06mm

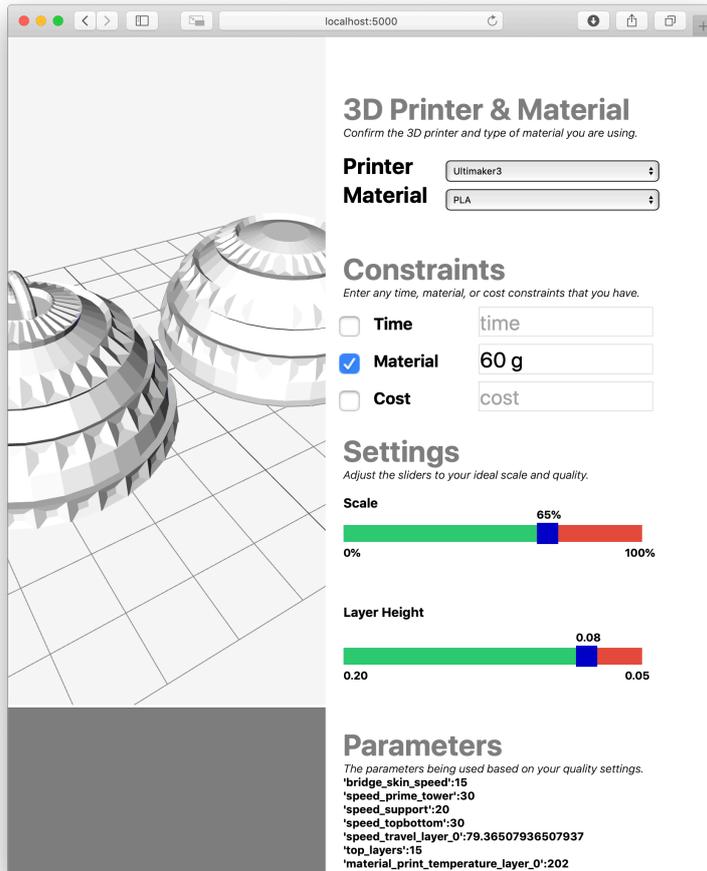


Figure 37: CSlice recomputes the scale and resolution when we switch over to Nylon as the material.

(note that layer resolution has a small effect on amount of material used, thus the resulting layer resolution for this scenario is different from the layer resolution in scenario #3).

For the scenarios illustrated above, we found that CSlice’s results deviate less from the target than participants’ results in the formative study (CSlice 3.95% vs 6.82% for participant results) while also ensuring valid print outcomes.

5.3 Implementation

There are three major components that enable the CSlice slicer:

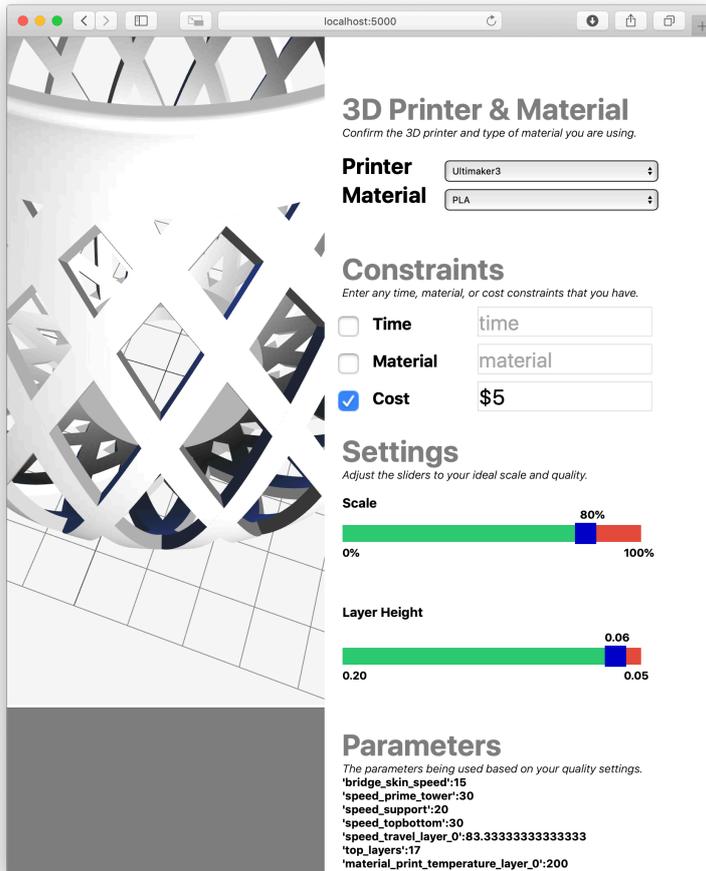


Figure 38: Result for printing the lamp shade with only \$5 worth of material left.

1. a web based, front-end user interface built on top of ThreeJS
2. a backed Flask server written in Python 3 and using the Flask web server
3. a cloud-based CuraEngine, hosted on Amazon Web Services (AWS)

The web based user interface enables users to enter their constraints and select between different size-quality trade-offs. The cloud-based (AWS) CuraEngine computes the slicing results in parallel. The flask backend, finally, connects the user interface to AWS.

In the following sections, I discuss the three subsystems in depth, and how they interact with one another, and why certain tech-

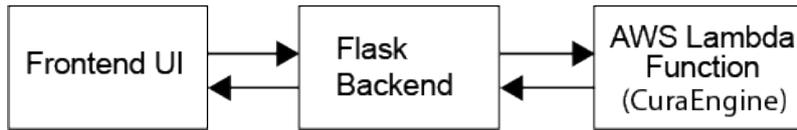


Figure 39: The Three Major Systems in CSlice.

nologies were selected for the implementation.

5.3.1 Web-Based Frontend UI

CSlice's user-facing interface is a web-based UI. The front end stack uses HTML, CSS, and Javascript as base languages. The 3D graphics are built on top of ThreeJS, a 3D graphics library written in JavaScript (JS).

I choose a web-based implementation after considering both the ease of development and ease of use for users of the system.

From a development perspective, the combination of HTML, CSS, and JS make rapid-prototyping and iterating on the system very fast. Because JS is an interpreted language, there is no wait time while the software compiles. JS is also a dynamically and weakly typed language, which decreases the amount of time needed to program data transformations.

From a GUI perspective, web-interfaces allow the specific manipulation of any button, object, or even pixel in the browser window. This provides a great deal of freedom to implement the interface exactly as designed. This feature of web based interfaces should not be overlooked, and future projects should ensure their implementation method will allow them to create the software as close to its ideal form as possible. That is, developers should ensure that they have the optimal amount of control over the systems they are building, and are not limited by abstraction barriers or dependency idiosyncrasies that will impede their vision of the system.

The ability to display any graphic on screen is a major improvement from the original WYSIWYFab interface, which was implemented to a single pane in an existing application.

From a user perspective, web-based interfaces have many advantages. Web interfaces are platform agnostic. As long as a user has access to a standards-compliment browser, the under-

lying operating system and hardware are irrelevant. There are also no dependencies needed to run the software.

There are limitations to a web based interface for developers and users. For developers, the ease of prototyping in JavaScript needs to be compared it's relatively weak debugging tools and increased likelihood of run-time errors from type errors. JavaScript is also (infamously) single threaded, and the modern attempts to allow parallelization are quite poor compared to other languages. Finally for developers, JavaScript is much slower than alternate possibilities, and optimization is much harder.

For users, the biggest cost to web-based interfaces (besides speed, already discussed) is the requirement of a network connection. The only alternative to a network connection access the interface is to locally host the system, which requires users to interact with a command-line interface.

For CSlice, the developer benefits far outweighed the costs — but the costs were carefully considered. For users, an internet connection is necessary to access the cloud-computing slicing, so the network requirement is irrelevant.

The front end system has a number of distinct components that interact with each other. The 2D buttons, sliders, and text-boxes are "plain vanilla" HTML elements, decorated with CSS. The 3D view is a ThreeJS scene which loads data passed to it by an STL/OBJ parser.

5.3.2 Backend Flask Server

The intermediary backend is a Flask server. The Flask server can run locally on the same computer as the user, or on any computer with Python 3 and Flask. The role of the backend is to serve the front end in the browser, and to connect the frontend to the cloud slicing by passing the 3D object to the cloud slicer, and returning cloud slicing results back to the front end.

I choose Flask as the backend server for much the same reason

AWS Lambda Function



Figure 40: Setting up CuraEngine on AWS Lambda in the cloud.

that JS was used for the UI: it makes rapid prototyping very fast. Flask is a minimal micro-server built on Python. It is extremely easy to set up easy, and the server auto-reloads every time a change is made to the code.

5.3.3 Cloud Slicing

For slicing, we used CuraEngine an open sourced C++ library that operates as a command line tool for slicing 3D models. CuraEngine's slice() function takes as parameters (1) an .stl file for the 3D model and (2) a json-string containing the slicing parameters. After slicing, it returns the print time and material volume.

To prepare Cura Engine to run in the cloud on Amazon Web Services (AWS), we used CMake to compile it into a binary that runs on Amazon's operation system (Amazon Linux OS). In addition to the binary, we created a python script with a main() function that calls CuraEngine's startSlicing() function when executed. We pack both the binary and the python script into a .zip file.

On AWS, we create a new Lambda function, which takes as input the .zip file we previously prepared (containing the CuraEngine binary and the python script, see Figure 40).

From our backend, we can then call the Lambda function using the Boto python library's *invokeLambda()* method. Whenever the Lambda function is called, it copies the zip file on

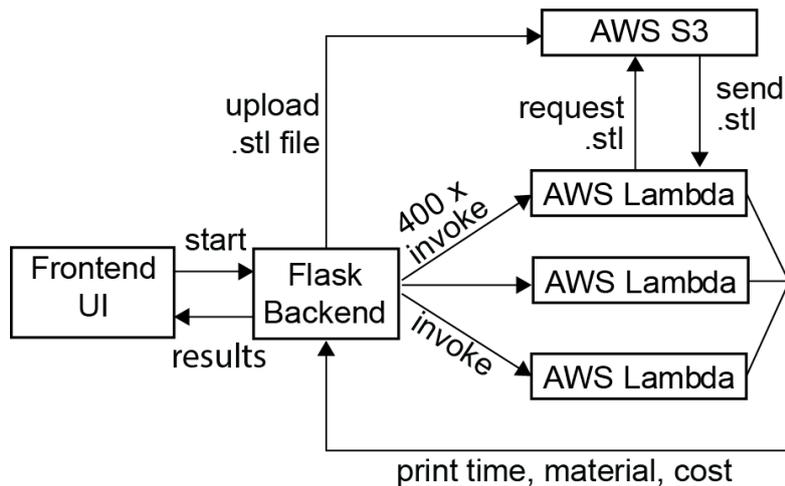


Figure 41: End-to-end slicing process in the cloud.

a new cloud computer on AWS, unzips it, and then executes the contained python script's main() function, which starts CuraEngine's slicing process. Thus, calling invokeLambda() multiple times from the backend allows us to run the same slicing code on multiple computers.

5.3.4 Invoking Parallel Slicing

As mentioned above, calling invokeLambda() multiple times from our backend allows us to execute CuraEngine on as many cloud computers as we need (Figure 41).

To execute the slicing process with different slicing profiles, we provide invokeLambda() with a json-string as input parameter that contains the different slicing parameter. Since Lambda does not allow passing files as arguments, we provide as a second input argument the path to the .stl file of the 3D model and then upload the .stl file separately to AWS's file storage system, called Simple Storage Service, or S3). To upload the file, we again use the Boto python library but this time use its uploadFile() function. After each Lambda machine downloaded the .stl file, it scales the model to the correct size and then slices it.

After slicing has finished, the results, i.e. the print time and required material for the sliced model, are returned from the

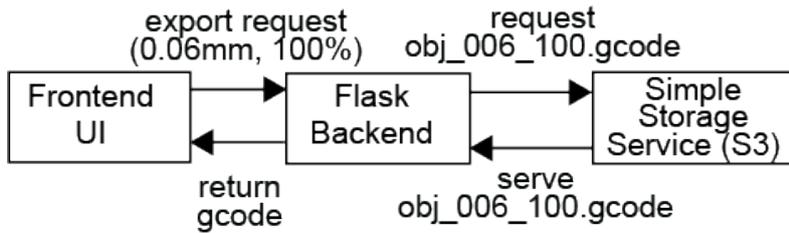


Figure 42: Returning the requested .gcode file.

invokeLambda() function via a json-string. Once the slicing results are returned, CSlice populates the scale and resolution sliders.

5.3.5 Returning the .gcode file

After slicing has finished, the .gcode files of all processes are saved back on the S3 server and named after the convention (obj_res_scale.gcode). Since returning the .gcode files of all slicing processes would take a lot of time for downloading them onto the user's computer (400 .gcode files, each several MB), CSlice waits until the user settled on a size-quality tradeoff and only downloads the matching g.code file from S3 once the user hits the 'export' button (Figure 42).

5.3.6 Generating Intermediate Slicing Profiles

While our approach of constraint slicing would also work using only the four standard profiles: “fast”, “normal”, “fine”, and “extra fine” available on Ultimaker Cura, we wanted to enable finer-grained control by interpolating between the profiles. Although we can generate any number of intermediate profiles, we decided on generating 19 profiles in total (4 standard profiles + 15 interpolated profiles) to have a reasonably high parameter resolution while keeping the cost per slicing process low.

To create the new profiles, we first determined the specific parameters of the existing profiles in Ultimaker Cura. Since not

all slicing parameters are exposed in the UI, we determined the parameters by inspecting the Cura-generated log file (which is stored at `/Library/ApplicationSupport/cura/<CuraVersion>/cura.log`) that is generated at the start of every slicing process. As an alternative, we also considered extracting the parameters from the configuration files for each slicing profile (found in the `/Resources/` directory of Cura). However, we found that the configuration files contain a series of internal dependencies and formulas that are hard to reverse engineer.

Once we obtained the raw values of the parameters for each profile, we created new profiles by linearly interpolating between subsequent profiles on a per-parameter level. The resulting profiles were then saved as JSON files, that CSlice can pass to CuraEngine.

5.4 Discussion

Next, we discuss the cost/accuracy trade-off when slicing in the cloud.

5.4.1 Cost of Slicing in the Cloud

As discussed in the implementation, CSlice uses slicing in the cloud through Amazon Web Services, which charges for its computing time (see Figure 43). We next provide an estimate how much slicing a model with CSlice would cost if CSlice was deployed today.

Figure 15 shows the cost using either the slowest (but cheapest) or fastest (but most expensive) computing services Amazon Web Services offers. Since each combination of slicing profile and scaled model has a different slicing time (and thus AWS cost), we provide both an upper bound (model sliced at 100% scale and 0.06mm resolution) and lower bound (model at 5% scale and 0.2mm resolution), as well as the average of the two.

memory (MB)	cost/100ms	slicing time	cost for one slicing	total cost (400 x slicing)
Upper Bound:	100% scale, 0.06mm reso.			
128 MB (slowest)	0.000000208	433s	\$0.001	\$0.36
3008 MB (fastest)	0.000004897	77.7s	\$0.0038	\$1.52
Lower Bound:	5% scale, 0.2mm reso.			
128 MB (slowest)	0.000000208	5.2s	\$0.000010816	\$0.004
3008 MB (fastest)	0.000004897	2s	\$0.00009794	\$0.036
Average:				
128 MB (slowest)	0.000000208	219s	\$0.00045552	\$0.16
3008 MB (fastest)	0.000004897	39.9s	\$0.001953903	\$0.76

Figure 43: Computing cost for using constraint slice.

To generate the data for Figure 43 we proceeded as following: When using the AWS ecosystem, users are charged N cents per 100ms of computing time, or slicing time. We first had to determine how long an average model needs to slice. To do this, we downloaded 100 of the most popular models from Thingiverse and sliced them on both the slowest AWS machine (avg. slice time 433s, std: 567s) and the fastest AWS machine (avg. slice time 77.7s, std: 120s) using as upper bound 100% scale and the finest resolution profile 0.06mm and lower bound 5% scale and the lowest resolution profile 0.2mm. We then converted the resulting slicing times into ms and multiplied AWS cost per 100m/s to calculate the average cost for slicing a model once.

Depending on the slider resolution, we can now multiply this value by the number of slicing requests required, i.e. to support

20 resolution profiles at 20 different scales, we need to slice 400 times (slicing at only 10x10, i.e. 100 in total, would cut the cost on AWS by 75% at the expense of more deviation from the target constraint).

Therefore, the average cost of using CSlice on the slow machine is \$0.16 and on the fast machine \$0.76. However, since AWS provides 400,000 GBseconds free computing time every month, in practice the first 36 models on the slow machine and the first 15 models on the fast machine are free.

5.4.2 How many values to slice

Depending on the desired deviation from the target, i.e. the granularity of the resolution and scale sliders in the user interface, slicing can come at a lower or higher cost. While we use slicing profiles spaced at 20x20 resolution and scale, the granularity depends on the use case. For instance, while the 10x10 granularity may be sufficient for a Maker in a Fablab, a designer who wants to maximize output for a high-paying customer may be willing to pay for 30x30 to receive less deviation from the target.

5.5 Conclusion

We presented CSlice, a slicer that finds the best slicing parameters based on a user's constraints on time, material, and cost. We demonstrated how such a constraint based slicer can be implemented using parallel slicing in the cloud, which provides the user with slicing results across a large range of profiles and model scales at only little extra time. We discussed the cost associated with slicing in the cloud and showed examples for both upper and lower bounds of slicing time.

Chapter 6

Future Work

Just play it grand: the way to stay with the future as it moves, is to always play your systems more grand than they seem to be right now.

Alan Kay
The Computer Revolution
*Hasn't Happened Yet*²¹
(1997)

6.1 Short Term

In the short term, there are improvements that could be made to both the WYSIWYFab and CSlice projects.

The limitations described in 4 involve two main problems:

1. a limited visual UI for the interface
2. slow speed of slicing

I describe improvements to the UI here as a short term improvement, and increasing the speed of slicing in the following section, as medium term improvement.

WYSIWYFab's current implementation is built on top of Blender, which restricts the options for the user interface in two ways. First, Blender add-ons can only execute code when a user clicks on a button. Therefore, WYSIWYFab is not able to execute background or event driven code. Second, Blender add-ons can only add UI elements to a side pannel — therefore the UI elements can only be in a small section of the window, they can not be placed anywhere.

The biggest UI improvements to WYSIWYFab would be to implement the system in an environment other than Blender. This lesson was learned for the second project (CSlice) which was implemented with web technology. WYSIWYFab could also be implemented in the browser, however the biggest challenge is the limited number and quality of existing web-based modeling tools.

From the lessons learned with WYSIWYFab, we were able to address those problems from the start with CSlice. In the short term, CSlice's future work involves making the interface easier to user and more obvious, without the need for instruction. For example, adjusting the scale of the model with a slider changes the GCode behind the scene, but doesn't visually change the model preview. In a future version, changes to the scale slider will effect the 3D preview, to ensure user's can clearly understand what's happening.

Specifically, online modeling tools that are open sourced. Tools such as *Onshape*³⁰ are fantastic web based modeling tools, but they can't be modified because they are closed-source. The only decent open sourced modeler we could find was *J-Sketcher*¹⁸, but it's still in development and serverly under-documented.

6.2 Medium Term

As previously mentioned, one of WYSIWYFab's biggest problems is the speed of slicing — which forces users to wait while the model slices. The solution to this problem is to slice in *real time* or use *just-in-time* methods to slice. That is, new slicers need to be developed that only slice the changes made to a model, instead of re-slicing the entire model. This will significantly speed up the slicing times, with the goal of working fast enough that the user never has to wait for computation to finish. Research into this topic should be done, but will take significant resources

and time to implement.

In the future, CSlice can be improved in two ways. First, there needs to be more granularity in the number of options presented to the user. Currently, the user has 400 options (20 quality levels and 20 scale levels). As compute costs decrease, it should be possible to increase the number of options for a more detailed level of control. Second, CSlice should enable users to input more types of constraints. Currently, there are three constraints available: time, material, and cost. It is easy to imagine (but harder to implement) many other constraints, such as visual constraints (a minimum feature size or minimum wall thickness) or physical constraints (an object that needs to be able to width-stand a certain amount of stress and strain).

6.3 Long Term

In the long term, there are many steps to take towards the vision of seamlessly prototyping ideas.

Many of the challenges described in *Personal Fabrication*² can be considered interface challenges. Towards that end, research can be done on the full stack of digital fabrication, from operating system design to hardware machine design, with an HCI perspective. Many lines of research already exist to improving the technology stack, but they often evaluate their results quantitatively.

The biggest technique to improving personal fabrication in the long term, is to simply consider the user as the core component of the system, and to build tools and methods that fundamentally augment people's ability to create. A deeper focus on the human part of Human-Computer Interaction will enable a future of interfaces that truly augment people's creative abilities.

Chapter 7

Conclusion

As always, the strongest
weapon we have to explore
this new world is the one
between our ears —
providing it's loaded.

Alan Kay

*User Interface: A Personal
View (1989)*

In this thesis, I claimed that people should be able to create anything they can imagine. I envisioned the impact of that vision on both personal and global scales.

Next, I outlined work that stems from two fields: a classical investigation of human interface, and a more recent investigation of fabrication research.

I then described the two concrete steps taken towards that vision. The first step is the WYSIWYFab project. WYSIWYFab unifies the 3D printing pipeline, by merging the steps of modeling a 3D object and slicing a 3D object into a single step.

The second step is CSlice, a method that inverts the traditional slicing workflow. CSlice was developed with lessons learned from WYSIWIFab, and a need for a deeper understanding of the constraints of 3D printing.

These contributions are small but necessary steps towards the vision described in chapter 2. Future work should build on these concepts, and expand the range of interfaces and techniques used in personal fabrication to enable a future where people can create anything they can imagine.

Bibliography

- [1] Moritz Bächer, Emily Whiting, Bernd Bickel, and Olga Sorkine-Hornung. Spin-it: Optimizing Moment of Inertia for Spinnable Objects. In *ACM Transactions on Graphics* 33, 4, Article 96, 2014.
- [2] Patrick Baudisch and Stefanie Mueller. Personal Fabrication. *Foundations and Trends® in Human-Computer Interaction* 10, 3–4, 165-293, 2017.
- [3] Dustin Beyer, Serafima Gurevich, Stefanie Mueller, Hsiang-Ting Chen, and Patrick Baudisch. Platener: Low-Fidelity Fabrication of 3D Objects by Substituting 3D Print with Laser-Cut Plates. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15)*, 1799-1806.
- [4] Bulletin of the Atomic Scientists. January 2012 Doomsday Clock Annual Statement. <https://thebulletin.org/sites/default/files/2012%20Clock%20Statement.pdf>
- [5] Vannevar Bush (July 1945). "As We May Think". *The Atlantic Monthly*. 176 (1): 101–108.
- [6] Xuelin Chen, Hao Zhang, Jinjie Lin, Ruizhen Hu, Lin Lu, Qixing Huang, Bedrich Benes, Daniel Cohen-Or, and Baoquan Chen. Dapper: Decompose-and-pack for 3D Printing. In *ACM Transactions on Graphics* 34, 6, Article 213, 2015.
- [7] Stelian Coros, Bernhard Thomaszewski, Gioacchino Noris, Shinjiro Sueda, Moira Forberg, Robert W. Sumner, Wojciech Matusik, and Bernd Bickel. Computational

- Design of Mechanical Characters. In ACM Transactions on Graphics 32, 4, Article 83, 2013.
- [8] Jeremie Dumas, Jean Hergel, and Sylvain Lefebvre. Bridging the Gap: Automated Steady Scaffoldings for 3D Printing. In ACM Transactions on Graphics 33, 4, Article 98, 2014.
- [9] Douglas Engelbart. A Research Center for Augmenting Human Intellect. 1968
- [10] Douglas C. Engelbart. Augmenting Human Intellect: A Conceptual Framework. 1962
- [11] N. A. Gershenfeld. Fab: The Coming Revolution on Your Desktop — from Personal Computers to Personal Fabrication. Basic Books, 2005.
- [12] Neil Gershenfeld, Alan Gershenfeld, and Joel Cutcher-Gershenfeld. 2017. Designing Reality: How to Survive and Thrive in the Third Digital Revolution. Basic Books, Inc., New York, NY, USA.
- [13] Vi Hart. HARC Mission Statement <https://harc.ycr.org>
- [14] Kristian Hildebrand, Bernd Bickel, and Marc Alexa. Orthogonal Slicing for Additive Manufacturing. In Computer Graphics 37, 6, 669-675, 2013.
- [15] Nathaniel Hudson, Celena Alcock, and Parmit K. Chilana. Understanding Newcomers to 3D Printing: Motivations, Workflows, and Barriers of Casual Makers. In Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16), 384-396.
- [16] iFixit. Right to Repair. <https://ifixit.org/right>
- [17] IPCC, 2018: Global Warming of 1.5°C. An IPCC Special Report on the impacts of global warming of 1.5°C above pre-industrial levels and related global greenhouse gas emission pathways, in the context of strengthening the global response to the threat of climate change, sustainable development, and efforts to eradicate poverty [V. Masson-Delmotte, P. Zhai, H. O. Pörtner, D. Roberts,

- J. Skea, P.R. Shukla, A. Pirani, W. Moufouma-Okia, C. Péan, R. Pidcock, S. Connors, J. B. R. Matthews, Y. Chen, X. Zhou, M. I. Gomis, E. Lonnoy, T. Maycock, M. Tignor, T. Waterfield (eds.)].
- [18] JSketcher. <https://github.com/xibyte/jsketcher>
- [19] Jeeun Kim, Anhong Guo, Tom Yeh, Scott E. Hudson, and Jennifer Mankoff. Understanding Uncertainty in Measurement and Accommodating its Impact in 3D Modeling and Printing. In Proceedings of the 2017 Conference on Designing Interactive Systems (DIS '17), 1067-1078.
- [20] Alan C. Kay. 1996. The early history of Smalltalk. In History of programming languages---II, Thomas J. Bergin, Jr. and Richard G. Gibson, Jr. (Eds.). ACM, New York, NY, USA 511-598. DOI: <https://doi.org/10.1145/234286.1057828>
- [21] Alan C. Kay. 1997. The Computer Revolution Hasn't Happened Yet. The 12th Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications. <https://youtu.be/oKg1hTOQXoY>
- [22] Alan C. Kay. Doing With Imagines Makes Symbols. <https://archive.org/details/AlanKeyD1987>
- [23] Machine That Changed The World, The. Interview with Alan Kay, 1990. Boston, MA: WGBH Media Library & Archives. Retrieved from http://openvault.wgbh.org/catalog/V_D9DC82D997454711A71B586E17D23119
- [24] J. C. R. Licklider. Man-Computer Symbiosis, IRE Transactions on Human Factors in Electronics, volume HFE-1, pages 4-11, March 1960
- [25] Lin Lu, Andrei Sharf, Haisen Zhao, Yuan Wei, Qingnan Fan, Xuelin Chen, Yann Savoye, Changhe Tu, Daniel Cohen-Or, and Baoquan Chen. Build-to-last: Strength to Weight 3D Printed Objects. In ACM Transactions on Graphics 33, 4, Article 97, 2014.

- [26] Linjie Luo, Ilya Baran, Szymon Rusinkiewicz, and Wojciech Matusik. Chopper: Partitioning Models into 3D-Printable Parts. In *ACM Transactions on Graphics* 31, 6, Article 129, 2012.
- [27] Stefanie Mueller. MIT CSAIL's Human Computer Interaction Group. <https://web.archive.org/web/20170222052721/http://hcie.csail.mit.edu/>
- [28] Stefanie Mueller, Dustin Beyer, Tobias Mohr, Serafima Gurevich, Alexander Teibrich, Lisa Pfistere, Kerstin Guenther, Johannes Frohnhofen, Hsiang-Ting Chen, Patrick Baudisch, Sangha Im, and Francois Guimbretiere. Low-Fidelity Fabrication: Speeding up Design Iteration of 3D Objects. In *Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems (CHI EA '15)*, 327-330.
- [29] Stefanie Mueller, Sangha Im, Serafima Gurevich, Alexander Teibrich, Lisa Pfisterer, Francois Guimbretiere, and Patrick Baudisch. WirePrint: 3D printed previews for fast prototyping. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology (UIST '14)*, 273-280.
- [30] Onshape. <https://www.onshape.com>
- [31] Thomas Oster, Rene Bohne, Jan Borchers. VisiCut - A Userfriendly Tool for Laser-cutting. Bachelor Thesis, 2012.
- [32] Romain Prévost, Emily Whiting, Sylvain Lefebvre, and Olga Sorkine-Hornung. Make It Stand: Balancing Shapes for 3D Fabrication. In *ACM Transactions on Graphics* 32, 4, Article 81, 2013.
- [33] Jones, R.; Haufe, P.; Sells, E.; Iravani, P.; Olliver, V.; Palmer, C.; Bowyer, A. (2011). "Reprap-- the replicating rapid prototyper". *Robotica*. 29 (1): 177–191. doi:10.1017/S026357471000069X.
- [34] Raf Ramakers, Fraser Anderson, Tovi Grossman, and George Fitzmaurice. RetroFab: A Design Tool for

- Retrofitting Physical Interfaces using Actuators, Sensors and 3D Printing. In Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16), 409-419.
- [35] Jef Raskin. 2000. The Humane Interface: New Directions for Designing Interactive Systems. ACM Press/Addison-Wesley Publ. Co., New York, NY, USA.
- [36] Daniel Saakes, Thomas Cambazard, Jun Mitani, and Takeo Igarashi. PacCAM: Material Capture and Interactive 2D Packing for Efficient Material Usage on CNC Cutting Machines. In Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology (UIST '13), 441-446.
- [37] Valkyrie Savage, Andrew Head, Björn Hartmann, Dan B. Goldman, Gautham Mysore, and Wilmot Li. Lamello: Passive Acoustic Sensing for Tangible Input Components. In Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15), 1277-1280.
- [38] Greg Saul, Manfred Lau, Jun Mitani, and Takeo Igarashi. SketchChair: An All-in-One Chair Design System for End Users. In Proceedings of the fifth international conference on Tangible, embedded, and embodied interaction (TEI '11), 73-80.
- [39] Ryan Schmidt and Nobuyuki Umetani. Branching Support Structures for 3D Printing. In ACM SIGGRAPH 2014 Studio (SIGGRAPH '14), Article 9.
- [40] Ivan Sutherland. Sketchpad: A Man-Machine Graphical Communication System
- [41] Nobuyuki Umetani and Ryan Schmidt. Cross-Sectional Structural Analysis for 3D Printing Optimization. In SIGGRAPH Asia 2013 Technical Briefs (SA '13), Article 5.
- [42] Juraj Vanek, Jorge A Garcia Galicia, Bedrich Benes. Clever Support: Efficient Support Structure Generation for Digital Fabrication. In Computer Graphics Forum 33, 5, 117-125, 2014.

- [43] Bret Victor. What Can A Technologist Do About Climate Change? (A Personal View). <http://worrydream.com/ClimateChange> 2015
- [44] Bret Victor, Paul Graham. Twitter Conversation. <https://kennethfriedman.org/more/refs/2016-victor-graham-twitter/>
- [45] Weiming Wang, Haiyuan Chao, Jing Tong, Zhouwang Yang, Xin Tong, Hang Li, Xiuping Liu, and Ligang Liu. Saliency-Preserving Slicing Optimization for Effective 3D Printing. In *Computer Graphics Forum* 34, 6, 148-160, 2015.
- [46] Weiming Wang, Tuanfeng Y. Wang, Zhouwang Yang, Ligang Liu, Xin Tong, Weihua Tong, Jiansong Deng, Falai Chen, and Xiuping Liu. Cost-Effective Printing of 3D Objects with Skin-Frame Structures. In *ACM Transactions on Graphics* 32, 6, Article 177, 2013.
- [47] Christian Weichel, Jason Alexander, Abhijit Karnik, and Hans Gellersen. SPATA: Spatio-Tangible Tools for Fabrication-Aware Design. In *Proceedings of the Ninth International Conference on Tangible, Embedded, and Embodied Interaction (TEI '15)*, 189-196.
- [48] Xiaoting Zhang, Xinyi Le, Athina Panotopoulou, Emily Whiting, and Charlie C. L. Wang. Perceptual Models of Preference in 3D Printing Direction. In *ACM Transactions on Graphics* 34, 6, Article 215, 2015.
- [49] Haisen Zhao, Fanglin Gu, Qi-Xing Huang, Jorge Garcia, Yong Chen, Changhe Tu, Bedrich Benes, Hao Zhang, Daniel Cohen-Or, and Baoquan Chen. Connected Fermat Spirals for Layered Fabrication. In *ACM Transactions on Graphics* 35, 4, Article 100, 2016.