INTERACTING WITH PERSONAL FABRICATION DEVICES

STEFANIE MUELLER

Dissertation zur Erlangung des Grades eines Doktors der Naturwissenschaften

- Dr. rer. nat -

Human-Computer Interaction Group
Hasso Plattner Institute
University of Potsdam

September 2016

Stefanie Mueller:
Interacting with Personal Fabrication Devices
September 2016

ADVISOR:
Prof. Dr. Patrick Baudisch

Published online at the
Institutional Repository of the University of Potsdam:
URN urn:nbn:de:kobv:517-opus4-100908
http://nbn-resolving.de/urn:nbn:de:kobv:517-opus4-100908

'Get used to operating outside your comfort zone. This is how you will truly learn, and grow. Don't shy away from challenges just because they seem too difficult. Acknowledge that fear is a very legitimate sensation and then take the plunge!'
David Salesin

ABSTRACT

Personal fabrication tools, such as 3D printers, are on the way of enabling a future in which non-technical users will be able to create custom objects. However, while the hardware is there, the current interaction model behind existing design tools is not suitable for non-technical users. Today, 3D printers are operated by fabricating the object in one go, which tends to take overnight due to the slow 3D printing technology. Consequently, the current interaction model requires users to think carefully before printing as every mistake may imply another overnight print. Planning every step ahead, however, is not feasible for non-technical users as they lack the experience to reason about the consequences of their design decisions.

In this dissertation, we propose changing the interaction model around personal fabrication tools to better serve this user group. We draw inspiration from personal computing and argue that the evolution of personal fabrication may resemble the evolution of personal computing: Computing started with machines that executed a program in one go before returning the result to the user. By decreasing the interaction unit to single requests, turn-taking systems such as the command line evolved, which provided users with feedback after every input. Finally, with the introduction of direct-manipulation interfaces, users continuously interacted with a program receiving feedback about every action in real-time. In this dissertation, we explore whether these interaction concepts can be applied to personal fabrication as well.

We start with fabricating an object in one go and investigate how to tighten the feedback-cycle on an *object-level*: We contribute a method called *low-fidelity fabrication,* which saves up to 90% fabrication time by creating objects as fast low-fidelity previews, which are sufficient to evaluate key design aspects. Depending on what is currently being tested, we propose different conversions that enable users to focus on different parts: *faBrickator* allows for a modular design in the early stages of prototyping; when users move on *WirePrint* allows quickly testing an object's shape, while *Platener* allows testing an object's technical function. We present an interactive editor for each technique and explain the underlying conversion algorithms.

By interacting on smaller units, such as a single *element* of an object, we explore what it means to transition from systems that fabricate objects in one go to turn-taking systems. We start with a 2D system called *constructable:* Users draw with a laser pointer onto the

workpiece inside a laser cutter. The drawing is captured with an overhead camera. As soon as the the user finishes drawing an element, such as a line, the constructable system beautifies the path and cuts it–resulting in physical output after every editing step. We extend constructable towards 3D editing by developing a novel laser-cutting technique for 3D objects called *LaserOrigami* that works by heating up the workpiece with the defocused laser until the material becomes compliant and bends down under gravity. While constructable and LaserOrigami allow for fast physical feedback, the interaction is still best described as turn-taking since it consists of two discrete steps: users first create an input and afterwards the system provides physical output.

By decreasing the interaction unit even further to a single *feature,* we can achieve real-time physical feedback: Input by the user and output by the fabrication device are so tightly coupled that no visible lag exists. This allows us to explore what it means to transition from turn-taking interfaces, which only allow exploring one option at a time, to direct manipulation interfaces with real-time physical feedback, which allow users to explore the entire space of options continuously with a single interaction. We present a system called *FormFab,* which allows for such direct control. FormFab is based on the same principle as LaserOrigami: It uses a workpiece that when warmed up becomes compliant and can be reshaped. However, FormFab achieves the reshaping not based on gravity, but through a pneumatic system that users can control interactively. As users interact, they see the shape change in real-time.

We conclude this dissertation by extrapolating the current evolution into a future in which large numbers of people use the new technology to create objects. We see two additional challenges on the horizon: sustainability and intellectual property. We investigate sustainability by demonstrating how to print less and instead *patch* physical objects. We explore questions around intellectual property with a system called *Scotty* that transfers objects without creating duplicates, thereby preserving the designer's copyright.

ZUSAMMENFASSUNG

Personal Fabrication Geräte, wie zum Beispiel 3D Drucker, sind dabei eine Zukunft zu ermöglichen in der selbst Benutzer ohne technisches Fachwissen eigene Objekte erstellen können. Obwohl die Hardware nun verfügbar ist, gibt es derzeit kein geeignetes Interaktionsmodel für Benutzer ohne Fachwissen. Heutzutage werden Objekte mit dem 3D Drucker in einem Stück hergestellt. Da der 3D Druck noch ein sehr langsames Verfahren ist und häufig so lange dauert, dass das Objekt über Nacht hergestellt werden muss, müssen Benutzer sorgfältig alles überprüfen bevor sie den Druckauftrag abschicken, da jeder Fehler einen weiteren Tag Wartezeit bedeuten kann. Benutzer ohne technischen Hintergrund haben jedoch nicht das notwendige Fachwissen um alle Faktoren vorhersagen zu können.

In dieser Dissertation schlagen wir vor das Interaktionsmodel von Personal Fabrication Geräten zu ändern, um diese Benutzer besser zu unterstützen. Wir argumentieren, dass die Entwicklung von Personal Fabrication Geräten der Entwicklung von Personal Computern gleicht. Die ersten Computer arbeiteten ein Programm vollständig ab, bevor sie ein Ergebnis an den Benutzer zurückgaben. Durch die Verkleinerung der Interaktionseinheit von ganzen Programmen zu einzelnen Anfragen wurden turn-taking Systeme wie die Kommandozeile möglich. Mit der Einführung von direkter Manipulation konnten Benutzer schließlich kontinuierlich mit dem Program arbeiten: sie erhielten Feedback über jede einzelne Interaktion in Echtzeit. Wir untersuchen in dieser Arbeit ob die gleichen Interaktionskonzepte auf Personal Fabrication Geräte angewendet werden können.

Wir beginnen diese Arbeit damit zu untersuchen wie man die Feedbackzeit bei der Interaktion mit ganzen *Objekten* verkürzen kann. Wir präsentieren eine Methode mit dem Namen *Low-fidelity Fabrication*, die bis zu 90% Druckzeit spart. Low-fidelity fabrication ist schnell, weil es 3D Modelle als grobe Vorschauobjekte druckt, die aber ausreichen um die Aspekte zu testen, die gerade wichtig sind. Abhängig vom aktuellen Testfokus schlagen wir vor verschiedene Konvertierungen vorzunehmen: Unser System *faBrickator* ist besonders für die ersten Testläufe geeignet, wenn ein modulares Design wichtig ist. Unser System *WirePrint* ist besonders nützlich im nächsten Schritt, wenn die Form des Objektes erhalten bleiben soll. Am Ende erlaubt unser System *Platener* ein Objekt so zu konvertieren, dass die technische Funktion des Objektes bewahrt wird. Wir erklären das Design unserer interaktiven Editoren und die zugrunde liegenden Konvertierungsalgorithmen.

Durch die Verkleinerung der Interaktionseinheit auf ein einzelnes *Element*, wie zum Beispiel einer Linie, untersuchen wir wie man Objekt-basierte Fabrikationssysteme in turn-taking Systeme umwandeln kann. Wir zeigen unser 2D System *constructable*, das auf einem Laser-Cutter basiert. Benutzer von constructable verwenden einen Laserpointer um auf das Werkstück im Laser-Cutter zu zeichnen. Die Zeichnung wird mit einer Kamera aufgenommen, korrigiert, und anschließend direkt mit dem Laser-Cutter ausgeschnitten. Wir erweitern constructable zu 3D mit unserer neuen Laser-Cutter Technologie *Laser-Origami*. LaserOrigami erzeugt 3D Objekte, indem es mit dem defokussierten Laser das Werkstück erhitzt bis es verformbar wird, die Schwerkraft biegt das Werkstück anschließend in seine 3D Form. Obwohl constructable und LaserOrigami physisches Feedback schnell erzeugen, ist die Interaktion dennoch am besten als turn-taking zu beschreiben: Benutzer editieren zuerst und sehen danach das Ergebnis.

Indem wir die Interaktionseinheit noch einmal verkleinern, nun auf ein einziges *Feature*, können wir Echtzeitfabrikation erreichen: Benutzereingabe und physisches Feedback sind so eng miteinander verbunden, dass es keine sichtbare Verzögerung mehr gibt. Damit können wir untersuchen, was es bedeutet von turn-taking Systemen zu direkter Manipulation überzugehen. Wir zeigen ein System mit dem Namen *FormFab*, das solch eine direkte interaktive Kontrolle ermöglicht. FormFab basiert auf dem gleichen Prinzip wie LaserOrigami: Ein Werkstück wird erhitzt bis es verformbar wird. Allerdings verwendet FormFab nicht die Schwerkraft zum verformen, sondern ein pneumatisches System, das Benutzer interaktiv steuern können. Wenn Benutzer den Luftdruck ändern, sehen sie wie sich die Größe der Form in Echtzeit ändert. Dies erlaubt ihnen die beste Entscheidung zu treffen während sie verschiedene Optionen evaluieren.

Im letzten Kapitel dieser Dissertation extrapolieren wir die aktuelle Entwicklung in eine Zukunft in der eine große Anzahl von Personen eigene Objekte herstellen werden. Dabei entstehen zwei neue Herausforderungen: Nachhaltigkeit und das Bewahren von intellektuellem Eigentum. Wir untersuchen Nachhaltigkeit mit einem System, das es erlaubt weniger zu Drucken und stattdessen Objekte *anzupassen.* Wir untersuchen Fragen zur Bewahrung von geistigem Eigentum mit unserem System *Scotty,* das Objekte transferiert ohne dabei Duplikate herzustellen und damit das Copyright des Designers erhält.

PUBLICATIONS

Some ideas have appeared previously in the following publications:

Chapter 3: Low-fidelity Fabrication
[1] Stefanie Mueller, Tobias Mohr, Kerstin Guenther, Johannes Frohnhofen, and Patrick Baudisch. faBrickation: Fast 3D Printing of Functional Objects by Integrating Construction Kit Building Blocks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI 2014)*, 3827-3834.

[2] Stefanie Mueller, Sangha Im, Serafima Gurevich, Alexander Teibrich, Lisa Pfisterer, François V. Guimbretière, 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 2014)*, 273-280.

[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 2015)*, 1799-1806.

Chapter 4: Turn-Taking Systems
[4] Stefanie Mueller, Pedro Lopes, and Patrick Baudisch. Interactive Construction: Interactive Fabrication of Functional Mechanical Devices. In *Proceedings of the 25th annual ACM symposium on User Interface Software and Technology (UIST 2012)*, 599-606.

[5] Stefanie Mueller, Bastian Kruck, and Patrick Baudisch. LaserOrigami: Laser-Cutting 3D Objects. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI 2013)*, 2585-2592.

Chapter 6: Sustainability and Intellectual Property
[6] Stefanie Mueller, Martin Fritzsche, Jan Kossmann, Maximilian Schneider, Jonathan Striebel, and Patrick Baudisch. Scotty: Relocating Physical Objects Across Distances Using Destructive Scanning, Encryption, and 3D Printing. In *Proceedings of the Ninth International Conference on Tangible, Embedded, and Embodied Interaction (TEI 2015)*, 233-240.

[7] Alexander Teibrich, Stefanie Mueller, François Guimbretière, Robert Kovacs, Stefan Neubert, and Patrick Baudisch. Patching Physical Objects. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software and Technology (UIST 2015)*, 83-91.

ACKNOWLEDGMENTS

I owe my deepest gratitude to my advisor Patrick Baudisch from whom I learned a tremendous amount of knowledge on this journey of getting a Ph.D. and becoming a faculty member myself. Among everything I learned from you, the one thing that I value most is that you encouraged me to always think big and to extend my definition of big at every step along the way. Thank you for all your help.

Without the support of my fellow PhD students and the Postdocs in our lab who helped in countless brainstormings and last minute hacking activities, the experience would not have been the same. I would like to thank Anne Roudaut, Liwei Chan, Dominik Schmidt, Christian Holz, Sean Gustafson, and Pedro Lopes for their support when I was a new member in the lab, and Alexandra Ion, Robert Kovacs, and Thijs Roumen for being such a great team during the last years.

I also want to thank all the bachelor and master students who collaborated with me on projects either through a project seminar or their thesis. In particular I would like to thank my master thesis students who were of great help on my research projects: Anna Seufert, Alexander Teibrich, Dustin Beyer, Bernhard Rabe, Tobias Mohr, David Eickhoff, and Konstantin Kaefer.

I would like to thank my parents Jürgen and Siglinde Müller for all their support. Thank you so much for always having an open ear and encouraging me to keep moving along my path. Thank you for always providing a quiet place to recharge from this energy-consuming life style. I would also like to thank my sister Kristin Müller who also works hard for her goals–it helps tremendously to have a like-minded sister.

Finally, I want to thank my boyfriend Philipp Berger who always had more confidence in me than I had myself. You forecasted every single step of my career in the last five years and although I had trouble believing you back then I must admit you were right all the time. Whenever I have doubts again, I will recall your voice and move on. Thank you for all your emotional support, it means a lot to me.

CONTENTS

LIST OF FIGURES

1 INTRODUCTION

Personal fabrication tools, such as 3D printers, are on the way of enabling a future in which non-technical users will be able to create custom objects. With the recent drop in price for 3D printing hardware, these tools are about to enter the mass market (Figure 1): While in 2007, the average consumer 3D printer was priced at $14,000, today's hardware costs only $500 on average [103]. Given the decreasing price, it is not surprising that the number of sold consumer 3D printers has doubled every year [104].

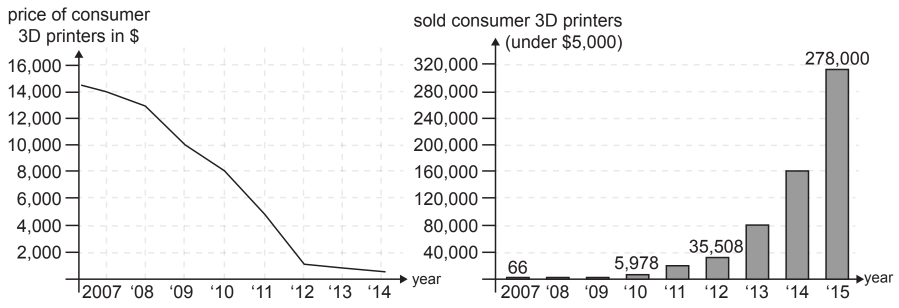

Figure 1: (a) With dropping prices for consumer 3D printers [103], (b) the number of sold devices has doubled every year [104].

While the hardware is now affordable and the number of people who own a 3D printer is increasing, only few create new 3D models. Most users download models from a platform, such as *Thingiverse* [85], and after downloading fabricate them on their 3D printers. At most, users adjust a few parameters of the model, such as changing its color or browsing between predetermined shape options [70].

We believe that personal fabrication has the potential for more: Instead of only consuming existing content, we envision a future in which 3D printers will allow non-technical users to create objects that only trained experts can create today. While there are many open challenges for human-computer interaction, such as abstracting away the necessary domain knowledge and machine knowledge, we focus on improving the interaction model underlying current personal fabrication devices.

1.1 INTERACTION MODEL WITH 3D PRINTERS TODAY

Figure 2 illustrates the current interaction model. Users sit at a computer and use a digital 3D editor to create a digital 3D model. Only at the end of the design process, users send the file to the 3D printer, which creates the object in one go. Since 3D printing is slow, this tends to take hours of printing time for small objects or even requires overnight printing.

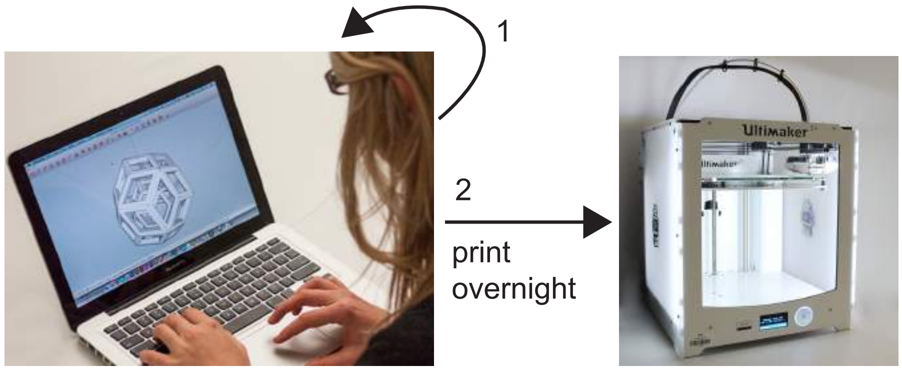

Figure 2: Current interaction model: (1) Users first create a digital model, iterate, and (2) only at the end produce a physical version.

Consequently, the current interaction model requires users to think carefully before printing as every mistake may imply another overnight print. This is not feasible for non-technical users as they lack the experience to reason about the consequences of their decisions.

1.2 DRAWING A PARALLEL TO PERSONAL COMPUTING

Looking back in history, this interaction model with the delayed feedback was also common with early computers [20]. In the early '60s, computers were so slow that the average program had to be executed overnight. Feedback was delayed until the next morning and if a program failed, users had to repeat the entire process, potentially waiting another night. Similar to 3D printers today, early computers were limited to expert users because when programs were executed in one go overnight, users had to know what they were doing in order to succeed.

1.3 TOWARDS TURN-TAKING AND DIRECT MANIPULATION

However, today we are at a point at which even non-technical users can use personal computers. Beside many technical developments, two advances in the interaction model enabled this (Figure 3): (1) the move from executing in one go to turn-taking, and (2) the move from turn-taking to direct manipulation [34].

(1) Turn-taking interaction model: By decreasing the interaction unit to single requests, turn-taking systems such as the command line evolved, which provided users with feedback after every input. This enabled the trial and error process non-technical users tend to employ: quickly iterating through potential solutions and building each step onto the results of previous ones [27]. As a side effect, this new exploratory interaction model also allowed for more unconventional solutions as the potential cost of a step not working out was reduced. However, while the turn-taking interaction model provided a great step forward to making the technology available for non-technical users, the feedback-cycle was still limited in that it consisted of two discrete steps: users first had to create an input and only afterwards received feedback.

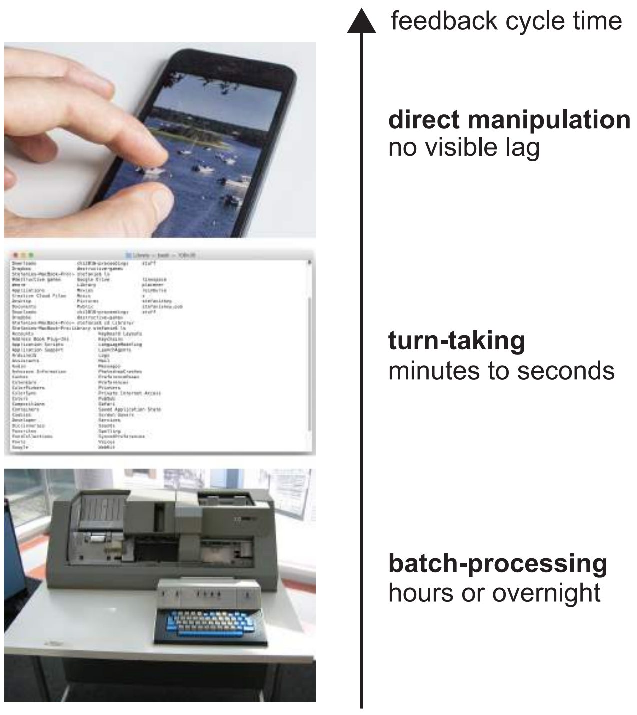

Figure 3: With advances in the interaction model that tightened the feedback-cycle, we are today at a point at which even non-technical users can use a computer.[1]

(2) Direct manipulation interaction model: With the invention of direct manipulation [76] that further decreased the interaction unit to a single feature, users finally received real-time feedback: Input by the user and output by the system are so tightly coupled that no visible lag exists. This tightened feedback cycle has many benefits, among others that ’novices can learn basic functionality quickly’ and ’retain operational concepts’ [75].

As described above, the current interaction model of 3D printers requires objects to be fabricated in one go. Thus, from a human-computer interaction standpoint, we are today at the point at which we were with personal computers in the 1960s: Only few users are able to use the technology and even for experts it is a cumbersome process due to the delayed feedback.

1 Image credit: "IBM card punch station” by waelder is licensed under CC 3.0

Recently, Willis et al. [101] argued that by repeating the evolution of the interaction model from personal computing, we will see the same benefits for personal fabrication. With a concept called *Interactive Fabrication* [101], they proposed that by bringing direct manipulation principles to personal fabrication tools, non-technical users will be able to create physical objects as easily as they manipulate digital data with today's personal computers.

Interactive fabrication systems have four main characteristics: (1) the physical environment is the workspace, not a digital editor; (2) users work hands-on on the physical workpiece through physical tools as known from traditional crafting; (3) each physical action results in immediate physical change, which can also be reversed; (4) in contrast to traditional crafting, users receive support from a computer system that helps to achieve precision.

While a few prototype systems, such as *Shaper* [101] and *CopyCAD* [31] have been built, no systematic exploration of how to achieve interactive fabrication systems with real-time physical feedback exists.

1.5 CONTRIBUTION

In this dissertation, we contribute a systematic exploration of how to implement an interactive fabrication system. In particular, we look at how by decreasing the interaction unit from entire objects, to single elements, to features, we can achieve real-time physical feedback, thereby enabling continuous interactive fabrication based on the principles of direct manipulation.

(1) Object-level: We start with fabricating an object in one go and investigate how to tighten the feedback-cycle on an *object-level*: We contribute a method called *low-fidelity fabrication* (Figure 4), which saves up to 90% fabrication time by creating objects as fast low-fidelity previews, which are sufficient to evaluate key design aspects. The concept draws inspiration from low-fidelity rendering, a method from the early days of computing that was used to give users a fast preview when processing was slow [25].

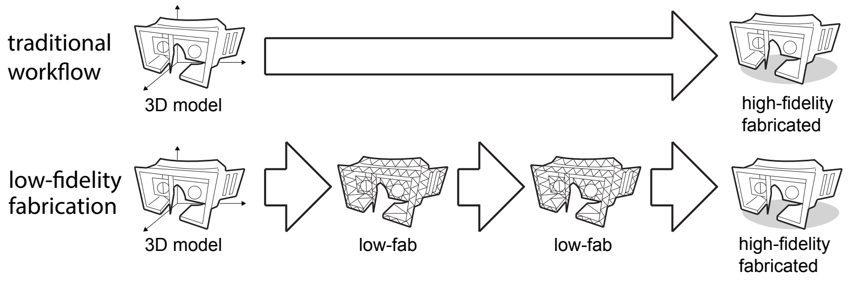

Figure 4: Low-fidelity fabrication saves up to 90% printing time.

Depending on what is currently being tested, we propose different conversions that enable users to focus on different parts (Figure 5): *faBrickator* allows for a modular design in the early stages of prototyping; when users move on *WirePrint* allows quickly testing an object's shape, while *Platener* allows testing an object's technical function. We present an interactive editor for each technique and explain the underlying conversion algorithms.

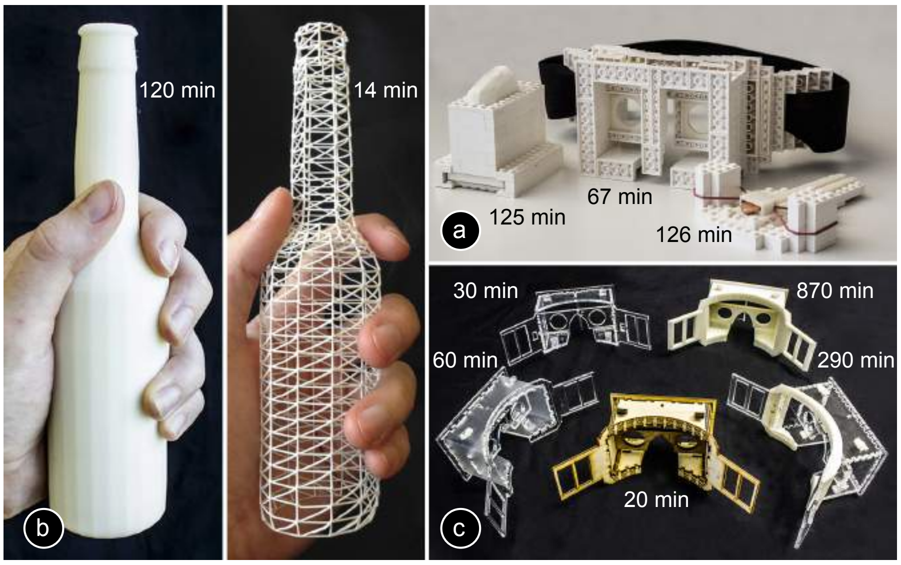

Figure 5: Low-fidelity techniques: (a) faBrickator for modularity, (b) WirePrint for shape, (c) Platener for functionality.

(2) Element-level: By interacting on smaller units, such as a single *element* of an object, we explore what it means to transition from systems that fabricate objects in one go to turn-taking systems. We start with a 2D system called *constructable* (Figure 6a): Users draw with a laser pointer onto the workpiece inside a laser cutter. The drawing is captured with a camera and as soon as the the user finishes drawing an element, such as a line, the constructable system beautifies the path and cuts it–resulting in physical output after every editing step.

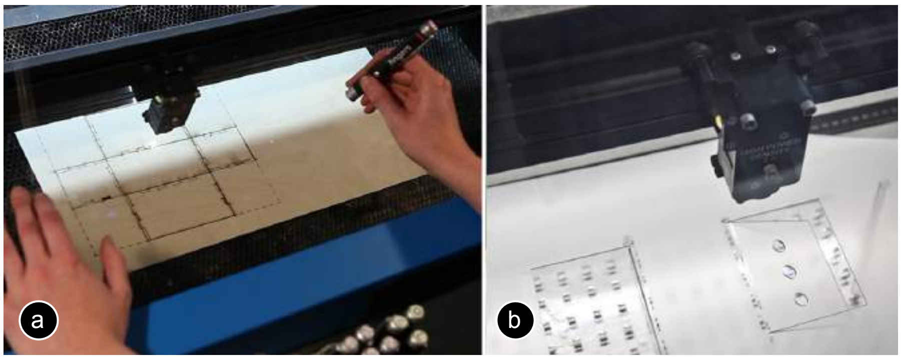

Figure 6: Turn-taking interfaces: (a) 2D editing system constructable, (b) extended for 3D editing with LaserOrigami.

We extend constructable towards 3D editing by developing a novel laser-cutting technique for 3D objects called *LaserOrigami* that works by heating up the workpiece with the defocused laser until the material becomes compliant and bends down under gravity (Figure 6b). While constructable and LaserOrigami allow for fast physical feedback, the interaction is still best described as turn-taking since it consists of two discrete steps: users first create an input and afterwards the system provides physical output.

(3) Feature-level: By decreasing the interaction unit even further to a single *feature,* we can achieve real-time physical feedback: Input by the user and output by the fabrication device are so tightly coupled that no visible lag exists. This allows us to explore the transition from turn-taking interfaces to direct manipulation. We present a system called *FormFab,* which allows for such direct control (Figure 7).

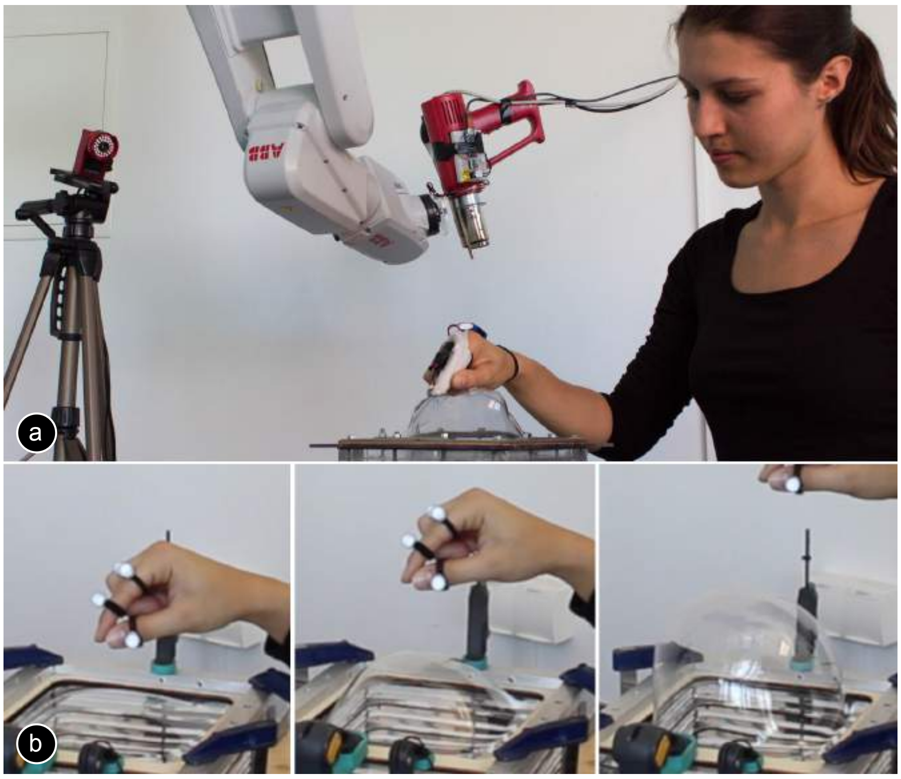

Figure 7: Direct manipulation interface FormFab: (a) Users first select an area, then (b) interactively control the amount of pressure and vacuum, thereby seeing the workpiece change in real-time.

FormFab is based on the same principle as LaserOrigami: It uses a workpiece that when warmed up becomes compliant and can be reshaped. However, FormFab achieves changing the shape not based on gravity, but through a pneumatic system that users can control interactively. As users interact, they see the scale of the shape change in real-time, allowing them to make key design decisions along the way.

Future outlook: We conclude this dissertation by extrapolation the current evolution into a future in which large numbers of people use the new technology to create objects. We see two additional challenges on the horizon: sustainability and intellectual property. We investigate sustainability by demonstrating how to print less and instead patch objects. We explore questions around intellectual property with a system that transfers objects without creating duplicates, thereby preserving the designer's copyright.

1.6 STRUCTURE

We begin this thesis by reviewing the related research that helps non-technical users design for fabrication devices (Chapter 2). Afterwards, we dedicate one chapter to each of the three steps for tightening the feedback cycle: faster feedback on an object-level (Chapter 3), element-level (Chapter 4), and feature-level (Chapter 5). We conclude this dissertation with a discussion of the benefits and limitations of direct manipulation interfaces and provide our view on the future of fabrication devices and the resulting challenges (Chapter 6).

2 RELATED WORK

Research that helps non-technical users design for fabrication devices can be classified into three areas:

(1) Domain knowledge: First, users need to solve the mechanical and structural challenges in their design, especially when designing functional objects. Researchers developed systems that support non-technical users by embodying the relevant domain knowledge, such as optimizing lift and drag forces when designing kites.

(2) Machine knowledge: Second, users need to work with the fabrication equipment. Experts optimize the fabrication process with many different goals in mind, such as optimizing fabrication time and structural strength, while taking machine constraints, such as the build volume, into account. Consequently, researchers have developed tools that help non-technical users overcome these hurdles by providing the necessary machine knowledge.

(3) Interaction model: Third, as outlined in the introduction, today's interaction model is not suitable for non-technical users. Early research on changing the interaction model proposed to make it more intuitive by using spatial gestures as known from the physical task and spatial output through augmented reality. More recently, researchers proposed to not only display the final output, but to fabricate physical output along the user's design process.

2.1 DOMAIN KNOWLEDGE

Existing design tools, such as *Autodesk Inventor* [5] and *SolidWorks* [79] require years of engineering training to gain the necessary expertise as they provide fine control over every parameter in the design. HCI and computer graphics researchers have looked at how to create design tools that abstract away the necessary domain knowledge by letting users specify the shape and motion of the desired object, the system then simulates the mechanical behavior and either critiques the user's design or automatically adjusts the user's design to make it comply with respect to forces.

2.1.1 *Creating matching shapes*

In its simplest form, parts have to fit with respect to other parts. *Enclosed* [97], for example, computes precise enclosures for electronic prototypes. Users only specify the components of their circuit, the system then automatically generates a matching enclosure from laser-cutable parts. Similarly, Igarashi et al. [40] provide a tool for cover design with compliant materials: users only specify where the opening should be, the software then constructs the matching geometry. Finally, when creating an object from several parts, the shape of the parts affects if they can be assembled. To create a working assembly, Lau et al. [45] automatically decompose 3D models into 2D parts while ensuring the result can be put together.

In addition, when designing objects with moving parts, such as the doors of a shelf [43], software needs to ensure that the parts do not collide with each-other and that they fulfill their functional relationships, for instance, ensuring that the doors cover the shelve. In addition, the joints attached to the moving parts need to fit the parts: since they have a minimum diameter, they cannot be positioned on weak geometries as the forces that act on them would otherwise break them. Software can automatically find the optimal placement of joints [7] and can generate joints that hold a part in a desired position [17].

2.1.2 *Kinematics*

Several tools help users design objects that can move. Users simply specify the desired motion of the 3D model, the system then fills in the mechanism that implements this motion. To specify the motion, users can either keyframe different poses of the model [110], sketch the motion path [26], or use data from a motion capture system [21].

Since mechanisms made from gears take up large amounts of space, Thomaszewski et al. [86] and Megaro et al. [56] propose to create the mechanism entirely from linkages. This allows designing mechanisms that form part of the shape of a object rather than a hidden internal structure. *LinkEdit* [8] provides a tool that facilitates the linkage editing process: users can make edits to the shape of selected parts while the system preserves the functional aspects.

2.1.3 *Statics*

The systems shown so far tackled shape and motion, they did not handle forces as part of the optimization. However, even simple objects only work when forces are being taken into account. For instance, a simple aspect of statics is balance: Unlike objects in the digital world, physical objects need to be balanced to not tip over. With *Make it Stand* [63] researchers propose software that helps balance objects by either hollowing the inside of the object to redistribute the weight, by using materials of different weights, or by slightly changing the object's geometry. This approach can also be extended to floating objects that should swim upright [93] and spin-able objects that should spin around a specific axis [9].

The problem of balance is amplified when objects are subject to external influences, such as a chair on which people sit. *SketchChair* [65] solves this problem by using a virtual character that represents the user's body proportions and weight. The software then displays if a chair falls over when the user sits on it or if it performs well.

Similarly, researchers have created systems that allow users to test if an object is going to break. Stava et al.'s software [81], for example, analyzes the weak points in a model and automatically strengthens the part by thickening the weak geometry. Other work has looked at how to minimize the required printing material while still maintaining durability either by printing wireframe structures instead of the honeycomb infill [95] or by calculating an optimal interior tessellation [50].

2.1.4 *Dynamics*

While statics is sufficient for objects that are not in motion, researchers have also proposed tools for designing dynamic objects. For instance, extending Coros et al.'s work on kinematic characters [26], Bharaj et al. [11] provide a tool for designing characters that are not walking on the spot but can move around freely. Megaro et al. [57] add to this work by allowing characters to also take turns without falling over.

With *Pteronyms* [88], Umetani et al. provide a design tool for flying objects. Depending on the orientation of the glider, it will be subject to drag forces that make the glider resist the airflow and lift forces that move the glider upward. All forces do not only depend on the shape of the glider, but also on its velocity and orientation at a certain moment in time, thus they are constantly changing as the glider moves through the air. This creates a large parameter space that is unfeasible for the user to tackle manually. To abstract away the domain knowledge, Umetani et al. provide a design tool that enables users to define the shape of a glider, which after every editing step gets optimized by the system for optimal flight performance. While Umetani et al.'s tool was limited to flat symmetrical two-dimensional gliders, *OmniAD* [53] extends the concept towards three-dimensional kites.

Other researchers support users in handling oscillation: For instance, when designing metallophones, the shape of the individual plates directly influences the acoustic frequency spectrum. Umetani et al. [89] developed a tool that while users design the shape of a plate, the tool provides real-time feedback about its sound. Bharaj et al. [12] take the inverse approach: they let users first specify a desired input shape and a desired sound, then slightly change the shape so as to produce the desired sound.

The systems mentioned above are domain-specific, i.e. they tackle one physical property, such as making an object fly, stand, or sound as desired. A few recent research projects explore more high-level algorithms that can automatically generate solutions for specific domains. One example of such work is *Design by Example* [72], a method that can automatically generate fabricatable assemblies from a database of arbitrary parts and a collection of design examples that use these parts. Similarly, *FabForms* [77] provides a generic approach for browsing only valid physical designs from a collection.

Defining such constraints already during the 3D modeling phase requires appropriate user interfaces for the designer. *MetaMorphe* [87] takes a first step with an interface based on the web-programming metaphor: *html* is used for the shape of the object, *css* for its appearance, and *javascript* for defining its function. However, more work is required as current design tools typically allow creating a single object rather than an abstract meta-design that can be customized by the user.

2.2 MACHINE KNOWLEDGE

Expert users, such as mechanical engineers, optimize the fabrication process with many different goals in mind, such as optimizing the fabrication time and structural strength, while taking machine constraints, such as the build volume, into account. Researchers have proposed software tools for automating the process to make it accessible to non-technical users.

2.2.1 *Material*

When fabricating an object, one parameter that influences the object's performance is the material being used. For instance, a chair fabricated from metal will be sturdy even if it consists of thin geometries. In contrast, a chair fabricated from soft wood will need thicker legs in order to not break. *Mesh2Fab* [107] solves this problem by adapting a model's geometry based on the available material by either thickening parts or correcting the contact angles between them.

2.2.2 *Machine hardware*

Beside working with the available material, machine specific constraints play an important role. For instance, the build volume of a fabrication device is limited, thus large objects have to be split into smaller parts. *Chopper* [51] is a tool that helps users with this task by splitting models in a way that makes them fit the build chamber while ensuring they are easy to assemble and the seams are unobtrusive.

2.2.3 *Fabrication process*

Zooming out from a specific device, the next layer of optimization is the fabrication process itself. For instance, in layer-based 3D printing that works by glueing one layer onto the next one, the object is more likely to break along the interface between two layers. Umetani et al. [90] provide a tool that given a set of forces from a specific direction calculates the optimal print direction.

Similarly, researchers have proposed algorithms that minimize the printing time: Wang et al. [94], for instance, save 30-40% printing time by slicing the 3D model with different layer heights; thinner layers are only used in high-detail regions.

Both faster printing speeds and material savings can also be accomplished when minimizing the required support material. While traditional support encloses the bottom of the object in a dense structure, researchers have proposed to use thin hierarchical structures that connect to the surface of the object with just a few points [71]. In addition, Zhang et al. [109] show how to layout the support in a way that least interferes with the dominant visual features on the model as removing the support often leaves visible artifacts.

2.2.4 *Generating process-specific models*

With current modeling tools, models intended for fabrication tend to be created and edited at a fairly low and machine-specific level. Models for additive fabrication, for example, are often described as a large collection of triangles (*.stl* format). Similarly, models for subtractive fabrication are often described as lines in the 2D plane (*.svg* format).

Designing on such a low-level representation often comes with additional challenges for the user: Laser cutters, for example, require models that consist of individual plates, as well as instructions for how to assemble these. Designing for this class of devices thus requires users to not only think in 3D, but also to imagine how to break down the 3D object they have in mind into 2D workpieces, which is challenging.

Researchers have proposed tools to circumvent this challenge: *FlatFitFab* [55], for example, constrains 3D modeling to only 2D plates that can be arranged in 3D space, allowing users to design a specific subset of laser-cutable models. Other approaches let users design any 3D shape and afterwards convert them into a 2D representation, for instance, by using intersecting planes (*Fabrication-aware Design* [73]), stacks of slices (*Autodesk 123 Make* [6]), and foldable strips (Chen et al. [23]). None of these approaches, however, uses a high-level *functional* specification as the input, they are all solely based on geometry.

2.2.5 *Geometric vs. functional specifications*

Several researchers have proposed expressing models by instead describing the desired properties of the object, not solely its geometry. This allows software to generate low-level descriptions for different machines and processes that achieve these properties.

For example, researchers proposed *function-driven* material specifications [13]. The key idea is to describe an object's properties on a functional level, which is later realized by computing a working combination of available printing materials.

For instance, Bickel et al. [13] show how to replicate deformation behavior of existing objects using the limited set of materials available on a 3D printer. As an example, they use a shoe sole: After measuring the deformation behavior of the existing sole, their algorithm automatically computes a material composition (i.e. a stack of layers from different base materials) that matches this behavior. Similarly, in *Deformable Characters* [78] users only specify an input shape and a set of target poses, the system then automatically computes the internal material composition and the location of actuation points.

While the solutions introduced above work for specific instances, *Spec2Fab* [22] provides a generalized framework for such a translation pipeline. *Spec2Fab* works for a wide variety of different high-level functions, such as mechanical and optical properties, making it easy for users to convert a model for different fabrication devices.

Finally, with *OpenFab* [92], Vidimce et al. provide an optimized rendering pipeline that integrates the printing material into shaders (so called *fablets)* that can be assigned to different object geometries.

One of the key benefits of function-driven specifications is that it makes object descriptions reusable. The more traditional low-level descriptions are geometry specific, thus cannot be transferred across different objects and not even across different 3D printers as, for instance, the 3D printing resolution is subject to changes and thus voxels with no assigned material exist.

2.2.6 *Functional specifications for electronics*

Savage et al. extend the concept of converting high-level specifications to electronics. Since it is not yet possible to print electronic circuits including sensors and actuators, Savage et al. developed a set of techniques for converting such models into fabricatable representations with the same functionality. For instance, in *Lamello* [67] users drag high-level functional components, such as sliders and dials, from a library into their design. On export, these components are enhanced with a set of tines that create different sounds when being struck, which is easily detectable with a single clip-on-microphone.

While *Lamello* used an acoustic approach, *Sauron* [66] uses optical tracking: Users mark the high-level functional components such as the control elements on a game controller in the 3D model. The system then automatically extends the inner geometry of the components to be within the view cone of a camera attached to the controller, making them easily detectable with computer vision. *A Series of Tubes* [68] extends this concept to IR sensing and pneumatics by automatically carving internal pipes into the object geometry. Finally, *Midas* [69] uses conductive tape and a vinyl cutter to generate touch layouts for user defined applications.

In summary, high-level functional representations will ensure that an object looks and performs the same independent of the specific hardware it was created with. This is especially important since fabrication is done locally and a variety of fabrication devices exists.

2.3 INTERACTION MODEL

Beside tools that facilitate the design and fabrication of physical objects, the underlying interaction model plays a major role. In the current interaction model, users use mouse and keyboard to create a digital design and receive visual feedback on a digital 2D display, only at the end of the design process they send the model to the 3D printer to fabricate the physical version.

2.3.1 *Spatial input and augmented reality output*

In a first step towards improving the interaction, researchers proposed to let users perform the actions they would do when manipulating a physical workpiece (a concept called *Spatial Modeling* [100]): For instance, in *Virtual Pottery* [24] users shape a virtual piece of clay by moving their hands as if they were physically shaping clay. Similarly, *Dress-up* [99] allows users to sketch dresses directly around the body of a physical mannequin.

Instead of using gestures, *ToolDevice* [3] provides users with physical props: a knife prop cuts objects and a hammer joins them. Similarly, *SPATA* [96] is a set of tools that can transfer measurements from the physical world to the digital and back.

Extending this approach, researchers made output 3D as well, essentially resulting in augmented reality systems. In *Situated Modeling* [44], for instance, users sketch the design of new furniture in their apartment in order to visually evaluate it in place. *MixFab* [98] follows the same approach, but does not require a head mounted display. Instead, *MixFab* uses a beam splitter and a display mounted at a 45° angle to overlay physical and virtual content.

Visual feedback is beneficial for all physical design tasks. However, it is not sufficient when, for instance, validating the haptics and ergonomics of an object or when validating whether the scale of an object is appropriate for its intended use case. In these cases, only the actual physical object allows users to judge whether it fulfills the requirements. Thus, researchers have proposed to create physical output along the user's design process for every part the user wants to test.

2.3.2 *Physical output with interactive fabrication*

Producing physical feedback along the user's design process has been referred to as *Interactive Fabrication* [101]. *ModelCraft* [80] is a pre-form of an interactive fabrication system: It allows users to edit a 3D model by first inspecting a folded paper version of the object, determining what requires a change, and then drawing the resulting change requests directly onto the paper model using an Anoto pen. By means of the pen, the *ModelCraft* system tracks the change requests, updates the 3D model, and 2D prints a new paper model with folding instructions, which users assemble by hand. While every turn takes way longer than what one would consider an interactive rate and required more manual effort than an automated fabrication system should require, *ModelCraft* was one of the first systems that allowed users to critique the physical object directly.

Another early prototype, *Shaper* [101], in contrast, explored how to provide physical feedback in an automated way, albeit at the expense of not offering on-object input. By touching a touch screen, users instructed the system where to extrude a drop of foam onto a 2D build-plate several feet below the screen.

CopyCAD [31] was one of the first systems that combined on-object interaction with automated fabrication, here a computer-controlled milling machine. *CopyCAD* users sketched onto the physical workpiece with a pen, a camera captured these annotations, and then operated the mill accordingly. The system also allowed capturing the shape of other objects using a camera, which helped users remix designs.

2.3.3 *Hand-held fabrication based on virtual models*

Other interactive fabrication systems were systems that explored how to allow users to replicate an existing digital model using hand-held tools with built-in force feedback. These systems allowed users to change the texture of objects; the overall shape, in contrast, was determined prior to the interaction in a separate 3D editor.

The first of these systems was *Haptic Intelligentsia* [46], a force-feedback device with an attached hot glue gun that only extrudes material when the user is on the correct path. Since users can determine where to start on the path and in which direction to extrude, the texture of every object comes out differently. Similarly, *Position-Correcting Router* [64] only routes when the user is on the path, *Enchanted Scissors* [106] only cuts when the user cuts along the path that was previously marked with conductive ink, and *Augmented Airbrush* [74] only sprays when the user holds it into the correct location.

More recent extensions to this line of work allow users to not only modify the texture of the object, but also parts of the shape. *FreeD* [111], for instance, is a hand-held milling tool that stops its spindle when users are close to hurting the pre-defined model. However, users can locally override the haptic constraint and change the path. *Hybrid Carving* [112] is an extension of this work: when users override the guidance of the system by removing material past the suggested point, the underlying 3D model changes on the fly and adjusts to the next valid shape. This allows users to be guided in the process while being free to change different design aspects [113].

In contrast to this line of work, our goal is to allow users to design objects in the physical workspace without the need to predefine a digital model or to go back to a digital editor.

3 OBJECT-LEVEL: LOW-FIDELITY FABRICATION

As pointed out in the introduction, today's 3D printers are so slow that fabricating slightly larger objects requires overnight printing. The head mounted display body in Figure 8, for instance, took 14:30 hours printing time on our 3D printer *(Dimension SST 1200es).* When designing a new object, which typically requires going through a series of iterations, this slows down the feedback cycle to one iteration per day.

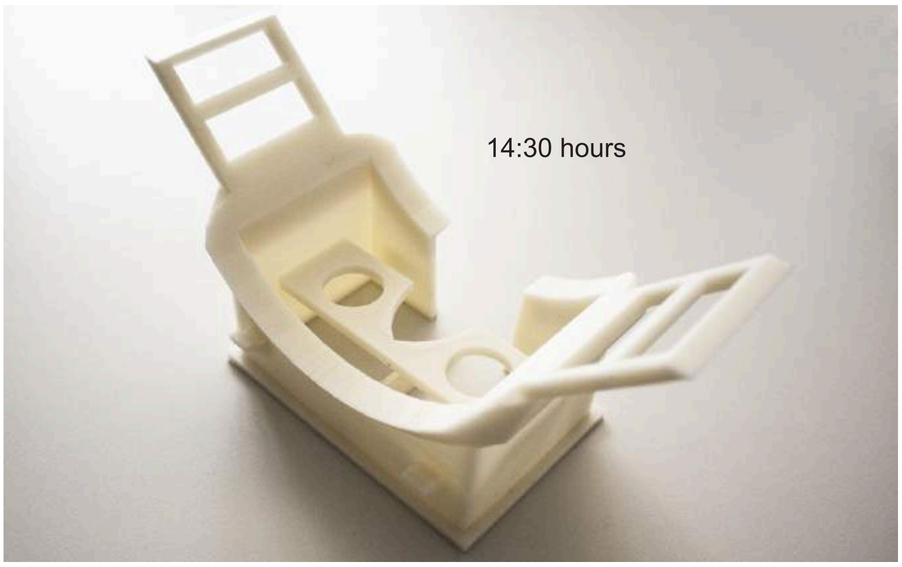

Figure 8: Fabricating this head-mounted display body requires overnight printing, slowing down the feedback cycle to only one iteration per day.

Different approaches try to reduce printing time by either massively parallelizing the printing process using multiple heads [35] or by assembling objects layer-wise from prefabricated voxels of equal size [37]. However, as printing resolution increases fabrication gets slower since more voxels or layers need to be fabricated.

To provide faster feedback, we take a different approach inspired by early computer graphics: When computing was slow, researchers proposed a method for rendering content with different levels of detail to give users a fast preview [25]. The key idea was to render only the important parts of a scene as slow high-fidelity and to render everything else in fast low-fidelity [15]. This approach scales well as users tend to focus on only one part at a time.

Similarly, in fabrication users typically focus on one aspect of the design at a time. For instance, users might want to get the overall look and feel right, then move on to verify that each part of the object functions properly.

By applying the principle of different levels of detail to fabrication, we can fabricate the parts that are currently not being tested as fast low-fidelity, and reduce high-quality printing to only the parts that need detailed evaluation. This allows us to get a testable version faster, thereby tightening the feedback-cycle during design iteration. We call the concept of printing intermediate versions as low-fidelity prototypes *low-fidelity fabrication*, or short *low-fab* (Figure 9).

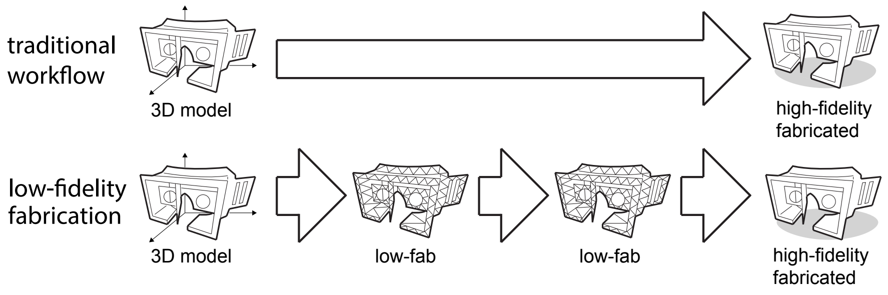

Figure 9: Low-fidelity fabrication prints intermediate versions as fast low-fidelity previews.

Figure 10 shows a conversion with one of our low-fidelity fabrication techniques called *faBrickator*. By printing only the lens mounts and assembling the rest from building blocks, we can get a first testable version within roughly one hour instead of printing overnight.

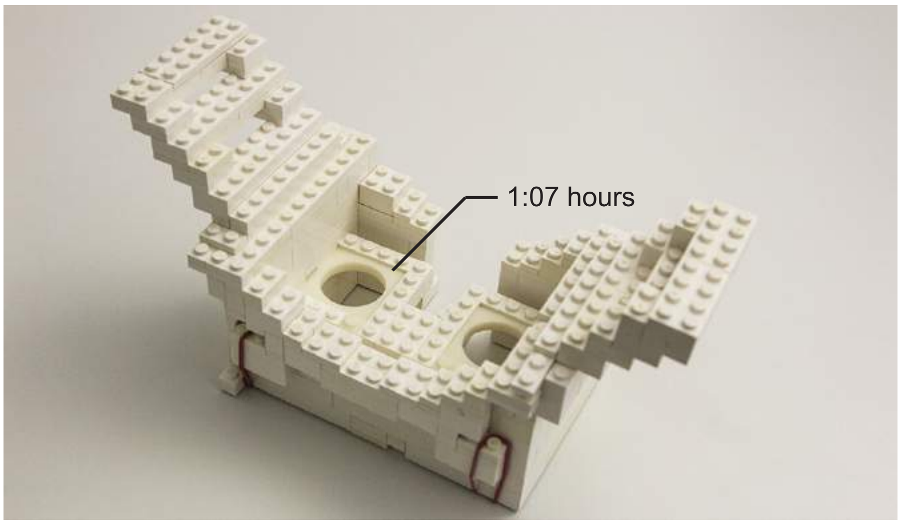

Figure 10: A low-fidelity fabricated version of the head mounted display body allows for a first test within one hour instead of the next day.

The main benefit of our approach is that it changes the fabrication process from overnight printing to something that can be carried out multiple times a day. This allows users to perform several design iterations in a day that would otherwise have stretched out over the course of a week.

Beside *faBrickator,* which focuses on modular designs in the early stages of prototyping, we developed two additional low-fidelity fabrication techniques to complement the space of possible design aspects: *WirePrint* preserves the object's shape thereby allowing users to test, for instance, the ergonomics of a design. *Platener* is a low-fidelity technique that preserves an object's technical function by converting an object into plates of the same size and thickness. In the following sections, we provide an example walkthrough for each of our low-fidelity techniques and explain the details behind the algorithms.

3.1 FABRICKATOR: LOW-FAB FOR MODULARITY

faBrickator is a low-fidelity fabrication technique that saves 3D printing time by automatically substituting sub-volumes with standard building blocks, such as Lego bricks. Users simply mark up specific regions as high-resolution to indicate that these should later be 3D printed. faBrickator then 3D prints the parts and generates instructions that show users how to create everything else from Lego bricks.

3.1.1 *Converting a 3D model with faBrickator*

In the following example session, we revisit the example of the head-mounted display body to illustrate how faBrickator achieves its time savings. The user starts by modeling the body of the head-mounted display in a 3D editor, here *Blender* (Figure 11). To make the head mounted display body work, the user first has to get the optical properties of the display right. Thus, the exact shape and position of the lens mounts is crucial to prevent the image from being blurry. The user decides to use faBrickator to 3D print the lens mounts while fabricating the rest of the design in Lego.

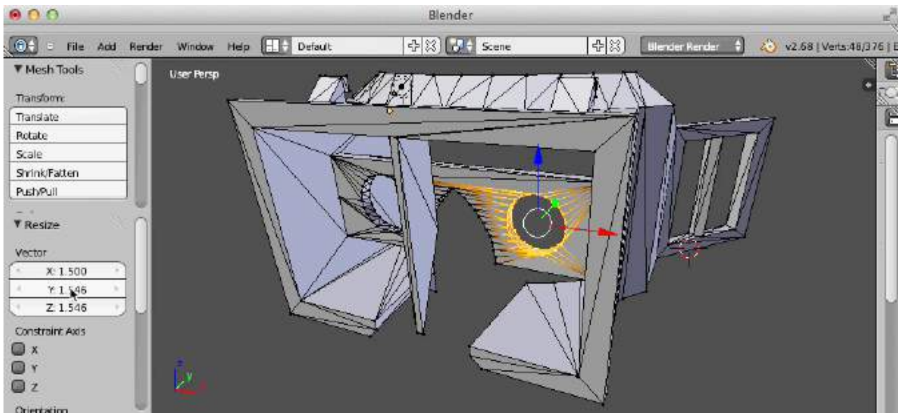

Figure 11: The user starts by creating a model of a head-mounted display body in the 3D editor *Blender.*

#1 Loading and legofication: When the user exits the edit mode in *Blender*, the 3D model is automatically loaded into faBrickator, initiated by the *Blender* plug-in we provide. In faBrickator (Figure 12a), the user hits the legofy-button, which causes faBrickator to display a naive legofication of the model, i.e., one that consists only of Lego's smallest building blocks: the 1x1 plate. The user now hits the layout-button (Figure 12b), which groups the 1x1 plates into larger plates and bricks so as to minimize the overall number of bricks. faBrickator also assures a stable design in this step by iterating over the brick layout until all components are connected.

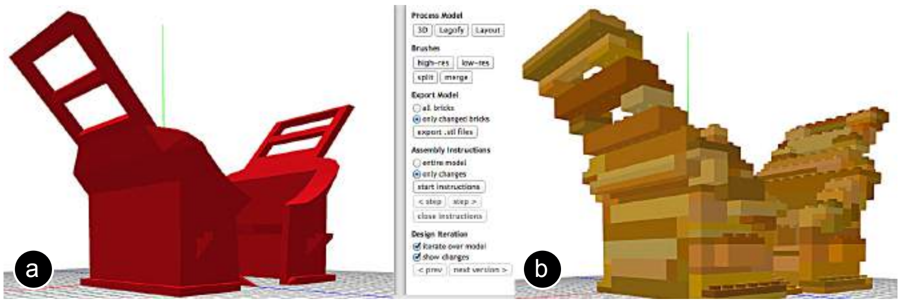

Figure 12: (a) The model is loaded into faBrickator. (b) When the user hits the layout-button, the model is converted into bricks and plates.

#2 Marking the high-res areas of the model: Figure 13 demonstrates how the user defines the regions that need to be 3D printed. (a) The user brushes the region around the lens mounts with the high-res brush. (b) This tells faBrickator to execute this area in full detail. The system responds by changing the preview accordingly. By default, faBrickator merges the marked area into a single 3D-printed block. The user repeats the process for the other lens opening.

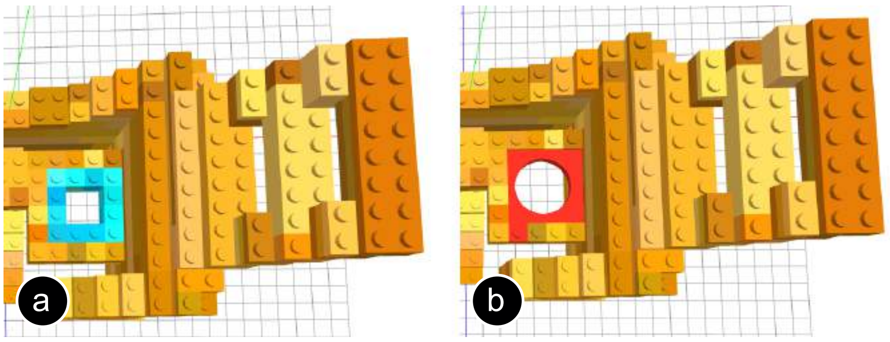

Figure 13: (a) Using the high-res brush the user defines the region that (b) will be 3D printed in full detail.

#3 Printing and assembling: To print, the user exports the model. This creates a 3D printable *.stl* file for each of the two lens mounts. The user then sends the files to the printer using the 3D printer software (in our case *CatalystEX*). As soon as the 3D model has been sent to the printer, the user starts assembling the Lego part of the model. As

shown in Figure 14a, faBrickator provides animated assembly instructions that show how to assemble bottom-up, layer-by-layer. This helps to minimize the time needed for assembly. 67 minutes after the lens mounts were sent to the printer, the 3D printer is done fabricating. Each brick bears a unique ID embossed into one of its sides (Figure 14b). This helps the user to quickly place them into the partially assembled Lego model.

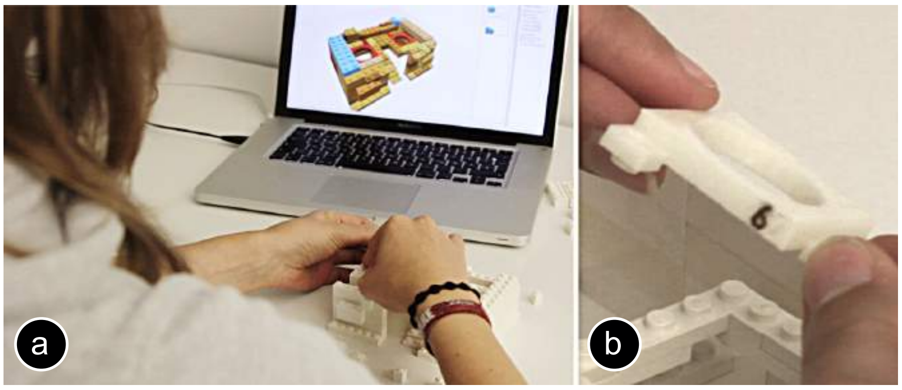

Figure 14: (a) Assembly instructions highlight the next brick in blue. (b) Each 3D printed part has a unique ID.

Figure 15 shows the assembled body of the head-mounted display with the accurately fitted lenses, which took 25 minutes to assemble. The overall fabrication time was thus still bound by the 3D printer, so that the 67 min is the final time for the entire fabrication. In this case, a speed-up of a factor of 12.99.

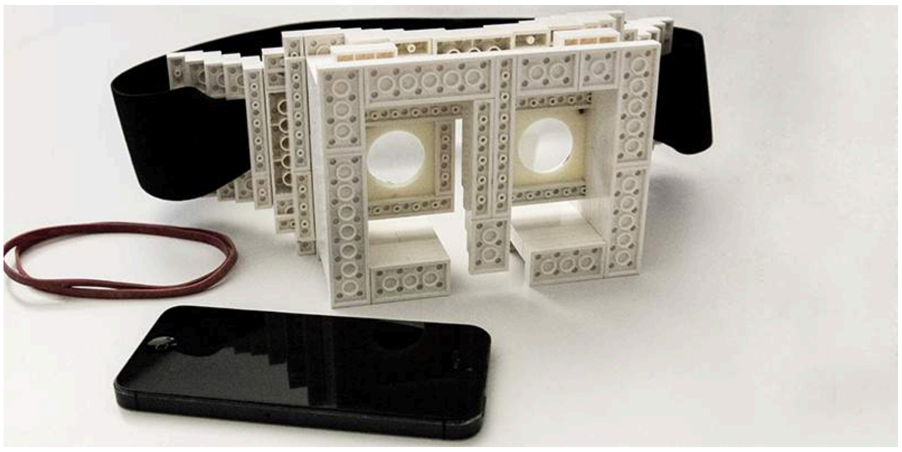

Figure 15: The final faBrickated model.

3.1.2 *Switching from low-res to high-res parts later on*

faBrickator allows users to change their mind later on: it allows replacing additional Lego bricks with 3D printed parts. Figure 16 illustrates this, continuing our walkthrough: The optics of the head-mounted display work great, but after wearing the device for a while, the user notices that it does not sit comfortably on nose and forehead. The user goes back into faBrickator to create more ergonomic forehead and nose pieces. As shown in Figure 16, the user marks the forehead and nose regions as high-res, then exports and 3D prints them.

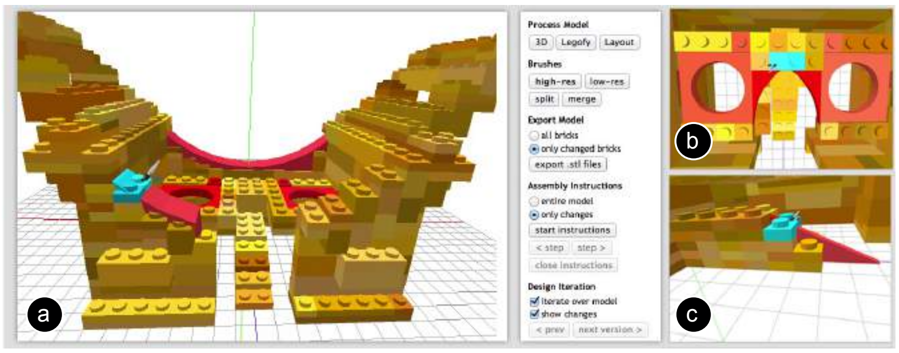

Figure 16: (a) Marking the forehead and (b,c) nosepieces as high-resolution.

Figure 17 shows how the user removes the corresponding Lego bricks from the head-mounted display body and replaces them with the printed parts. This iteration took 3:55 hours including assembly.

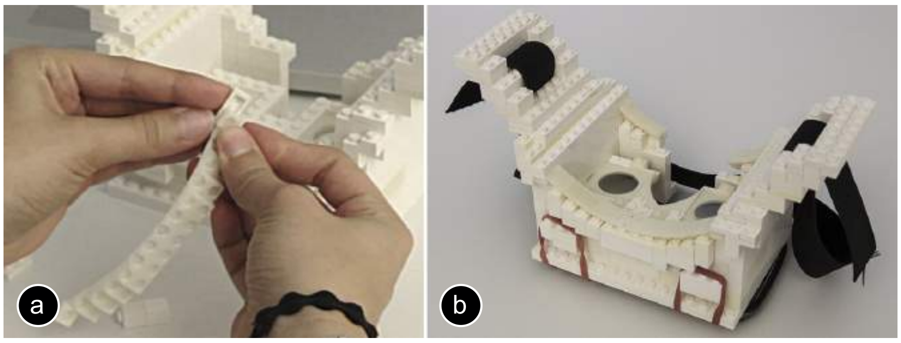

Figure 17: (a) Replacing bricks with the 3D printed parts for the head. (b) The assembled model.

3.1.3 *Faster iteration through localized changes*

While faBrickator already speeds up the fabrication of the initial prototype, it saves even more time when subsequently iterating over segments since users only have to reprint the parts that actually changed rather than reprinting the entire model. Back in the 3D editor *Blender,* the user extends the nosepiece so as to perfectly fit the user's nose. After the user exits the edit mode in *Blender,* faBrickator aligns the current version of the 3D model with the previously printed one to minimize the number of bricks that need to be re-fabricated. Figure 18a shows how faBrickator highlights all bricks that changed compared to the last version. (b) The user now rebuilds the nose piece and (c) after printing exchanges it with the one that didn't fit perfectly.

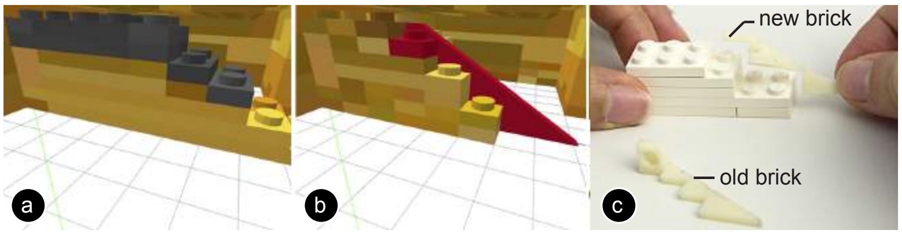

Figure 18: (a,b) faBrickator highlights what changed between two model versions. (c) The user then reprints only the part that changed.

3.1.4 *Evaluating the speed-up*

Beside the head-mounted display body, we faBrickated two additional example objects that we downloaded from *Thingiverse.* faBrickator fabricates the three objects 3.11 times, 2.75 times, and 1.45 times faster (average: 2.44 times) than traditional 3D printing while requiring only 14 minutes of manual assembly on average. How much faBrickator speeds up the feedback cycle depends on how much volume can be substituted with bricks.

Figure 19 shows the first model: a soap bar dispenser that dispenses small flakes of soap when the sled is moved back and forth. It consists of two pieces: the body that holds the soap and that the user moves with his dominant hand in the sled, and the mount for the razor blades and the sled. Producing this object with traditional 3D printing takes 6:30h compared to only 2:05h printing and 5 minutes assembly when using faBrickator (casing: 0:51h printing and 3 minutes assembly, razor blade: 1:14h printing and 2 minutes assembly). This is an improvement of a factor 3.11 for 3D printing time.

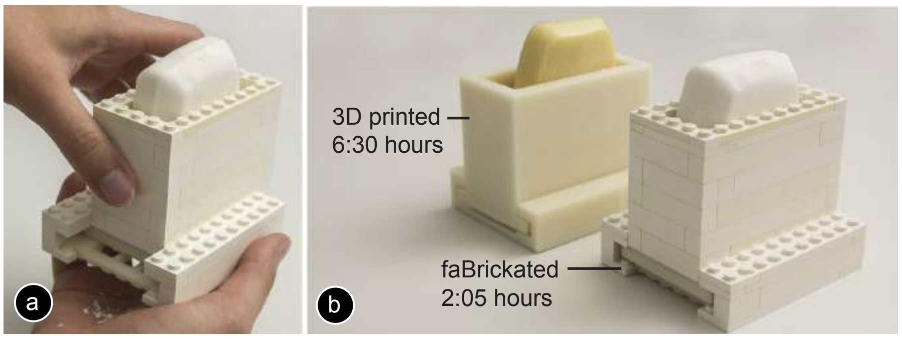

Figure 19: (a) This soap dispenser only takes (b) 2:05h printing time and 5 min of assembly time compared to the 6:30h with traditional 3D printing.

Figure 20 shows the second model: a penny ballista that allows ejecting a penny with a rubber band. The ballista consists of a mount for the penny and the ballista body with a sled. Using 3D printing, this model takes 3:03h compared to 2:06h printing and 11 minutes assembly when using faBrickator (penny mount: 0:59h printing, ballista body: 1:07h printing and 11 minutes for assembly). This is an improvement of a factor 1.45 for 3D printing time.

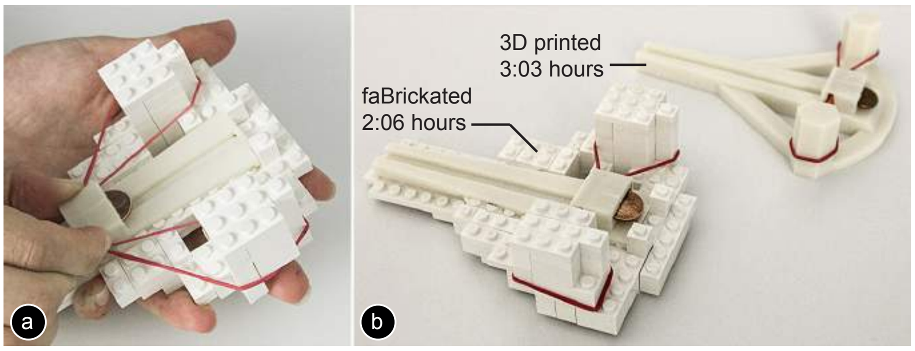

Figure 20: (a) This penny ballista only takes (b) 2:06h printing time and 11 min of assembly time compared to 3:03h with traditional 3D printing.

The third model is the already presented head-mounted display body with the lens mounts and the head and nose parts. When using faBrickator, it took 5:17 hours for 3D printing and only 25 minutes for assembly (lens mounts: 0:33h and 0:34h, forehead piece top: 1:26h, forehead pieces bottom: 0:41h and 0:44h, nose bridge: 0:25h, nose: 0:54h). This is an improvement of a factor 2.75 for 3D printing time, which for the completely 3D printed model is 14:30h. While these examples give first insights into potential time-savings when using faBrickator, it potentially provides an even greater speed-up when creating very large objects.

3.1.5 *Algorithms for automatic faBrickator conversion*

faBrickator is built in *CoffeeScript* using the *constructive solid geometry* library for its geometry operations. We also provide a *Blender* plugin written in *Python* for easier file exchange between *Blender* and faBrickator.

#1 File exchange between Blender and faBrickator
Our *Blender* plugin automatically exports the 3D model as an *.stl* file into a predefined folder as soon as the user exits the edit mode. faBrickator automatically detects if a new .stl file appears in the folder and imports it accordingly. The *Blender* plugin runs in a separate thread to prevent interference with *Blender's* editing functionality.

#2 Legofying: converting the 3D model into Lego
Figure 21 shows how faBrickator converts the 3D model into Lego's smallest building blocks: the 1x1 Lego plates. faBrickator first determines the bounding box of the model and extends it to the full multitude of a 1x1 Lego plate (8x8x3.2mm). Afterwards, faBrickator fills the bounding box with Lego plates in those positions that are either inside the model or share a portion of the volume with it. The remaining locations are left empty.

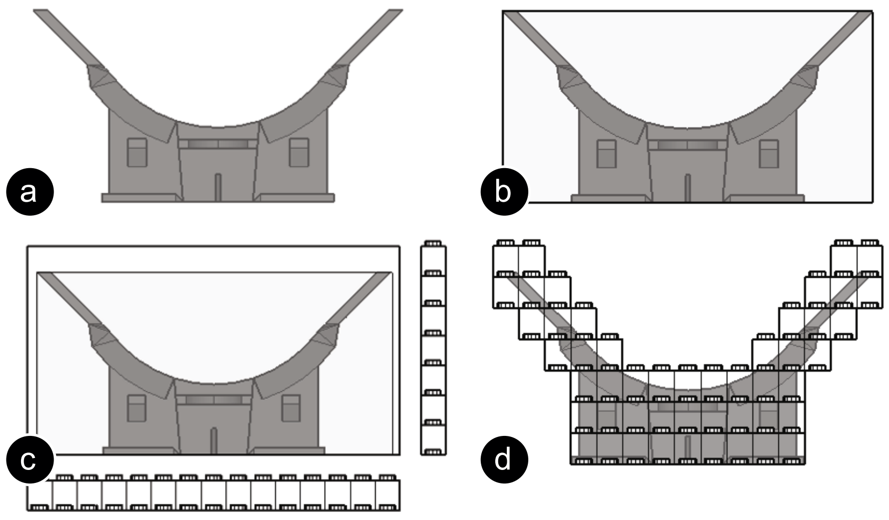

Figure 21: Legofying: (a) model, (b) bounding box, (c) extended bounding box, (d) legofied model.

#3 Layouting Lego bricks
faBrickator's layout algorithm is inspired by existing layout algorithms for generating working brick layouts. Researchers proposed different algorithms, such as using a cost function for simulated annealing [33], evolutionary algorithms [62], beam search [102], and cellular automata [91]. faBrickator builds upon the algorithm proposed by Testuz et al. [83] but extends it with 3D printed bricks, which have irregular shapes and therefore additional constraints that need to be taken into account. faBrickator's layout algorithm optimizes for two things: it minimizes the number of bricks used for assembly while maximizing stability. The algorithm works as following: (1) it merges the 1x1 plates into larger Lego bricks, (2) it creates a connectivity graph representing the brick layout, (3) it identifies weak points in the graph (unstable connections), and (4) tries to relayout the bricks at these positions to erase the weak points.

#4 Converting Lego bricks into 3D printable bricks
When users select a Lego brick with the high-res brush, it is converted into a 3D printable brick. In order to generate the 3D printable brick, we use a Boolean intersection of the Lego brick with the original 3D model geometry (see Figure 22).

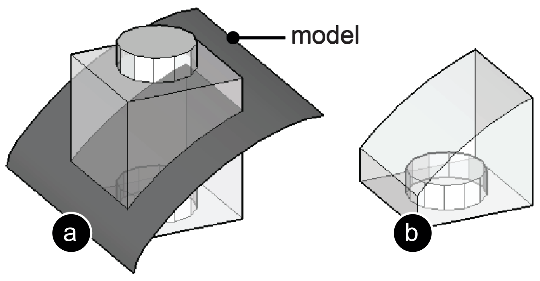

Figure 22: The 3D printable brick is a result of a Boolean intersection of the model and the Lego brick.

#5 Merging and splitting Lego bricks
When users select several bricks using the merge brush, they are merged into a single compound 3D printable brick. In our initial approach, we used the Boolean *union* operation to merge the bricks, which turned out to be computationally too expensive. The reason is that the side faces of the two Lego bricks that should be merged are always co-planar, which is the most expensive case for a *union* operation. Instead faBrickator uses a tagging approach for merging. As can be seen in Figure 23a, faBrickator tags each side of a Lego brick according to its orientation in space (e.g., side-y, side+y). In order to merge two bricks, our algorithm simply identifies the matching neighbor sides by their tag and removes the sides since they would be inside the compound brick (Figure 23b). If later on users apply the split brush, we just enable the sides again.

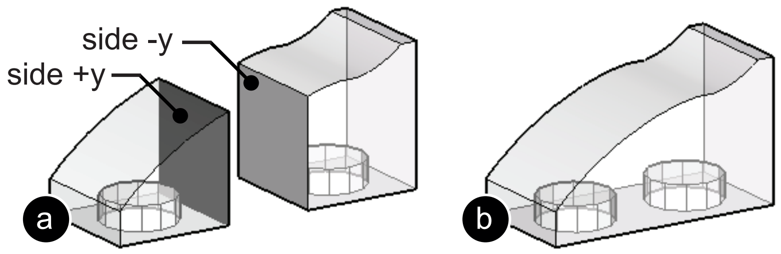

Figure 23: Merging: (a) The sides of each brick are tagged according to their orientation. (b) We use this to identify, which sides need to be removed for merging.

#6 Detecting the completeness of knobs and tubes
In order to ensure a stable brick layout, faBrickator needs to detect working knobs on a 3D printable brick. Only complete knobs can be used to make a connection with another brick. A knob is only complete if all of the knob's faces were inside the model before the intersection with the model (see Figure 24). If a part of the knob lays outside the model, it is cut off during the intersection and the knob is no longer functional. A regular Boolean *intersect* determines inside and outside faces in the process of calculating the intersection. However, as the result the intersect operation only returns a geometrical

volume from which it is difficult to tell if the knob is complete or not. faBrickator therefore has its own intersect operation that stores the meta-information about inside and outside faces in tags. Before the intersection, all the faces of a knob are tagged with the knob-tag (Figure 24a). During the *intersect,* faBrickator tags which faces are inside/outside of the model (Figure 24b). Afterwards, our algorithm checks if there is any face with the knob-tag that also has the outside-tag. If there is none the knob is complete. We use the same mechanism for the tubes on the bottom side of the bricks.

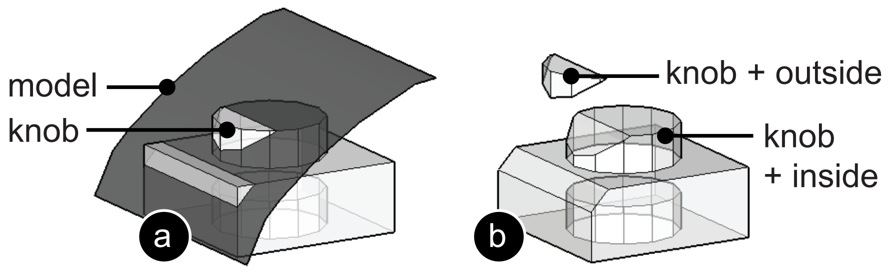

Figure 24: Detecting the completeness of knobs: (a) During intersection, we (b) tag if a part of the knob lies outside the model.

#7 *Calculating changes between design iterations*
When users iterate over a model, faBrickator displays which part of the model changed. For this, we first align the new 3D model with the old 3D model because faBrickator allows users to rotate the 3D model in *Blender* and to move it around freely. In order to align the two models, faBrickator calculates a transformation matrix based on the distance of specific model features to a reference point (we use the point of origin). The triangles of a model are not suited as model features because even small changes of the model result in a completely different tessellation during the export from a 3D modeling program, such as *Blender*.

In order to create a representation of the model that is comparable, we merge the triangles into larger sides based on their orientation. First we merge neighbor triangles that have the same normal into larger convex polygons. In a second step, we merge all convex polygons with the same normal into a side. We use these sides for comparison (i.e. their outline) and after finding a match, calculate the transformation matrix and align the models. Now that the models are aligned, we legofy the new model. Afterwards, we determine which parts changed by calculating the differences based on the 1x1 Lego plates. If a plate was added or removed, the part changed. For bricks that were marked as high-res 3D printed bricks in the old model, we use a volume comparison to determine changes. For bricks that did not change, we copy the brick layout of the old model to minimize the number of bricks that need to be reassembled.

3.1.6 *Conclusion*

On average faBrickator is 2.44 times faster than traditional 3D printing, thereby providing fast feedback and allowing users to iterate multiple times a day. faBrickator achieves this by (1) printing in low-detail where possible, i.e., using Lego bricks, (2) limiting reprinting to those regions that have actually changed, and (3) minimizing the effort for manual assembly by using different sized Lego bricks. We demonstrated that faBrickator is especially useful in the early stages of design as its modular approach allows reprinting only the parts that changed and not the entire model. However, since faBrickator is based on building blocks that have a fixed coarse resolution, it is not suitable when testing an object's shape, for instance, to confirm the ergonomics. For this, our low-fidelity fabrication technique WirePrint is best suited.

3.2 WIREPRINT: LOW-FAB OF SHAPE

WirePrint is a low-fidelity fabrication technique in which surfaces have been replaced with a wireframe mesh. Since wireframe previews are to scale and represent the overall shape of the object, they allow users to quickly verify shape aspects of their design, such as the ergonomic fit (Figure 25).

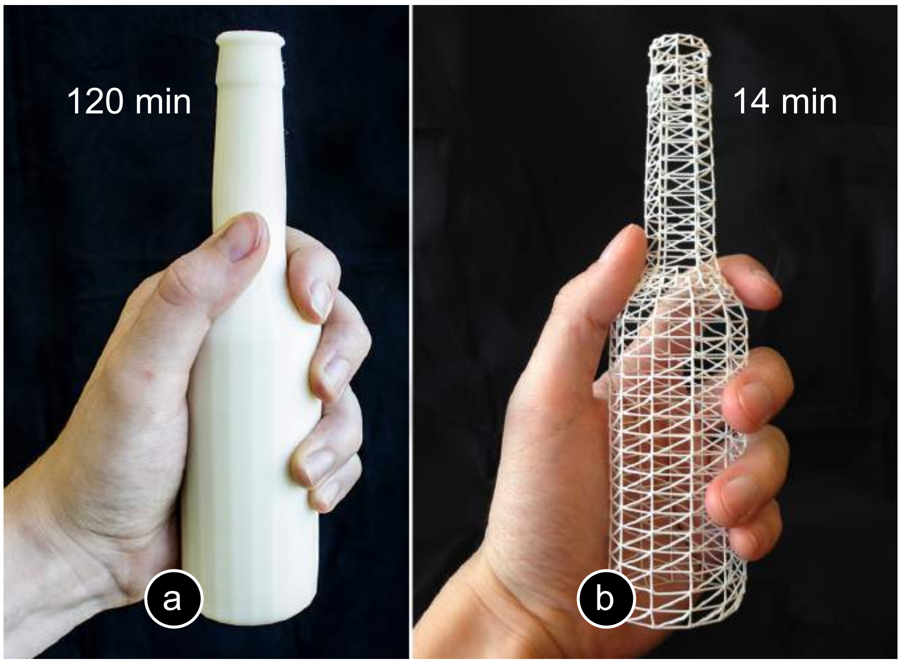

Figure 25: (a) With traditional printing this bottle takes 120 min. (b) WirePrint only takes 14 min while preserving the shape.

3.2.1 *Benefits of vector-printing over layer-printing*

WirePrint saves up to 90% printing time by using a novel 3D printing approach: While traditionally 3D printers print layer-wise, WirePrint extrudes filament directly into 3D space.

We derived the benefits of extruding in 3D space through a set of experiments. Our initial hypothesis was that layer-wise 3D printing can be sped up substantially by leaving out the solid surfaces since this significantly reduces the amount of extruded material. To validate this hypothesis, we implemented layer-wise wireframe printing as follows: (1) print only the edges of a model, and (2) minimize the use of support material as this adds to the printing volume (support material is required for all overhangs larger than 45°). We wrote a piece of software that converts a 3D model into a wireframe mesh and adds additional 45° edges as support structures where needed. For instance, a cube processed with our software has support edges for the horizontal edges at the top (see Figure 26b). We compared the converted wireframe cube (edges have 10% infill) with a solid cube (solid with 10% infill). Both had a size of 28mm.

Surprisingly, the printed wireframe model with substantially reduced volume only provided a 2x speed up compared to the regular model with faces and more infill material. We found that the main reason for this is that the path for the print head remains almost the same in both conditions: Since the material is printed layer-wise, the print head still has to traverse the entire cube outline as it moves from corner to corner, bottom-up.

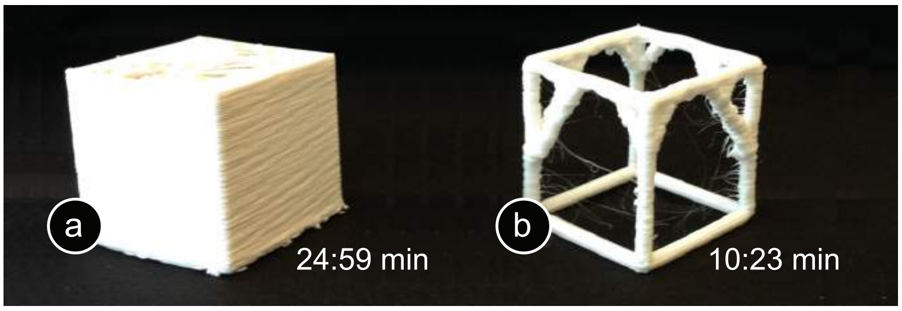

Figure 26: Printing times: (a) layer-wise 3D printing with 10% infill, (b) layer-wise 3D printing of a wireframe (reduced volume, 10% infill).

Based on this insight, our second hypothesis was that to substantially reduce the printing time, we need to optimize the printhead path itself. To validate the hypothesis, we printed another cube of the same size for which we extruded filament not layer-by-layer, but directly in 3D-space, thereby creating the edges of the wireframe model directly one stroke at a time. Figure 27 shows the result: by optimizing the printhead path we saved more than 90% of the printing time.

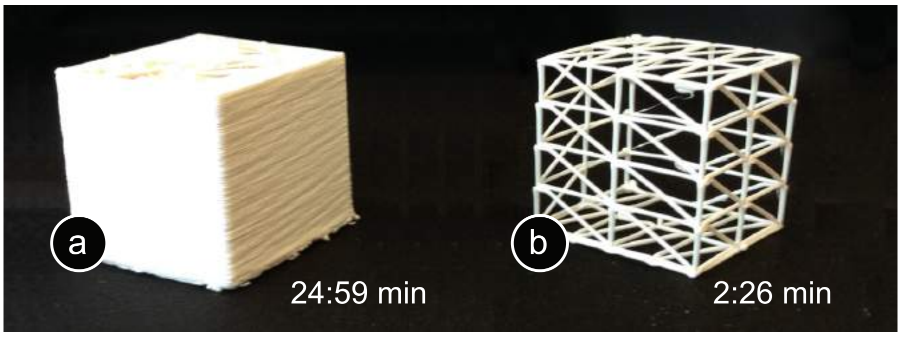

Figure 27: Printing times: (a) layer-wise 3D printing with 10% infill, (b) WirePrint extruding edges in 3D space.

Using the insight from our basic experiment, we implemented a software for converting 3D models into WirePrint meshes that are extruded in 3D space. Since our conversion software only changes the underlying printhead instructions, it does not require changing the hardware of the 3D printer. This makes our approach applicable to many 3D printers already in use today. Figure 28, for instance, shows objects being printed on two consumer 3D printers: the *PrintrBot* and the *Kossel mini*.

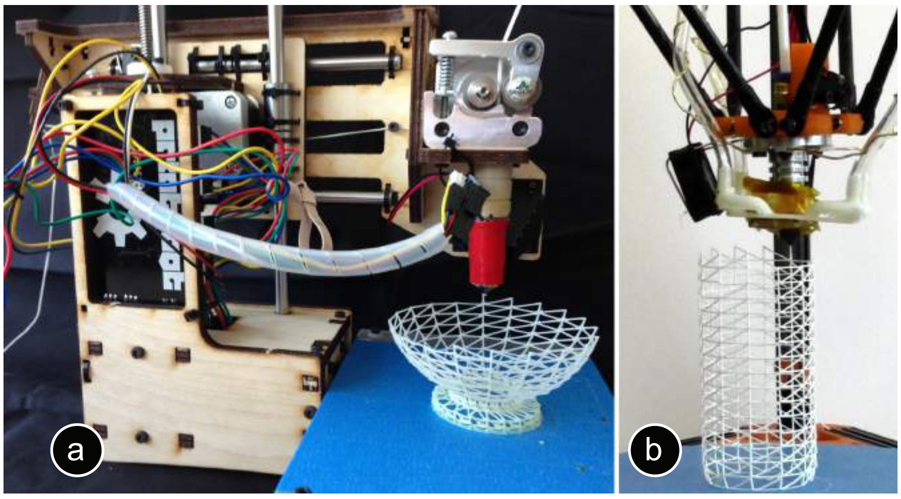

Figure 28: WirePrint works on (a) traditional cartesian-based printers, but is fastest on (b) 3D printers based on the delta design.

3.2.2 *Converting a 3D model with WirePrint*

After converting and printing the bottle shown in Figure 25, the user notices that the bottle does not yet rest comfortably in the hand. The user thus decides to change the model and reprint it. Figure 29 illustrates the typical workflow for converting a model with WirePrint.

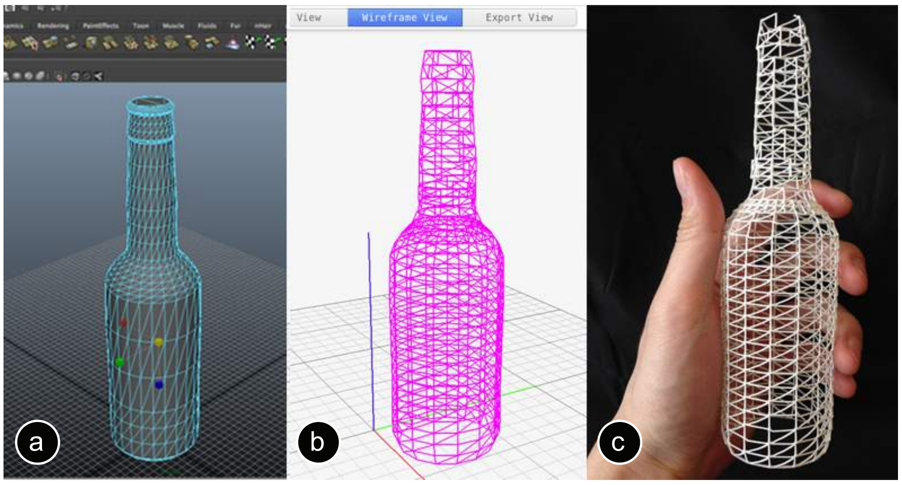

Figure 29: WirePrint workflow: (a) adjusting the model in a 3D editor, (b) converting it in the WirePrint software, (c) reprinting and testing.

The user adjusts the thickness of the bottle in a 3D modeling program, converts the model in the WirePrint software, and reprints it to test its fit again. The user may repeat the process until the bottle fits well. At this point, the user moves on to the details of the design, until finally 3D printing the bottle in full detail (1:59 hours). WirePrint allows this design process, including its iterations, to be completed within only a couple of hours.

The WirePrint printing technique
WirePrint converts a 3D object into a wireframe representation by (1) slicing the 3D model along its vertical axis into horizontal slices and (2) extracting the contours. It then (3) fills the space between slices with a zigzag pattern.

As illustrated by Figure 30, WirePrint fabricates objects by alternating between printing a contour and creating one layer of the zigzag pattern on top of the contour. While printing, WirePrint creates its layers by moving the print head up and down repeatedly.

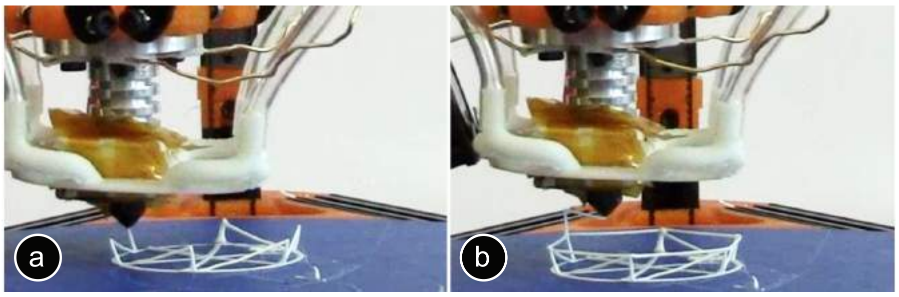

Figure 30: WirePrint's layers consist of an alternation between contour and zigzag.

If a slice contains multiple disconnected contours, such as the telephone receiver shown in Figure 31, WirePrint prints all contours located on the same slice first (i.e. the contour of the left ear cap, then the contour of the right ear cap), before moving up to the next slice.

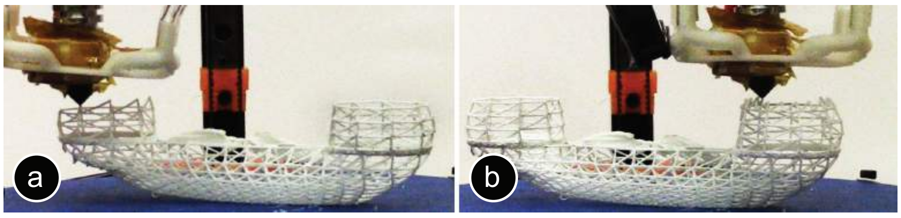

Figure 31: Multiple contours are printed one at a time: (a) first the left ear cap is printed, then (b) the right ear cap.

Figure 32 illustrates why WirePrint uses this particular contour-plus-zigzag approach. The main challenge in printing wireframes is that the print head has to respect already printed material in order to prevent collisions. Each bit of printed material results in additional volume becoming inaccessible.

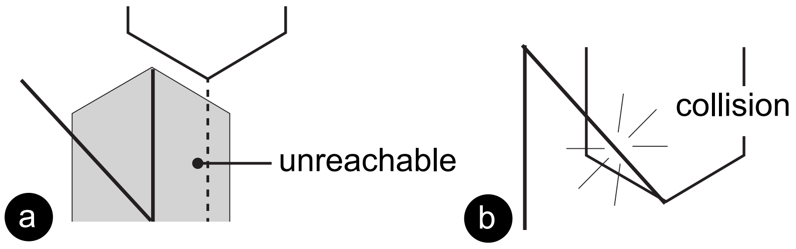

Figure 32: Preventing collisions: (a) the print head can neither print next to already printed material, (b) nor descend steeper than the threshold angle.

Figure 32 also shows the consequences that result from volume becoming inaccessible: (a) two vertical edges, for example, need to be spaced at least one print head diameter apart. (b) While it is always possible to print upwards, we cannot print downwards steeper than the slant of the print head itself, as steeper edges can cause the slanted tip of the print head to collide with what is just being printed. In the case of our 3D printer (a *Kossel Mini*), for example, this threshold angle is 32°.

These constraints still allow for several different approaches to convert a 3D model into a printable wireframe. Figure 33c shows another approach that leaves out the contour and uses a different pattern. While this approach respects the constraints stated above, it leads to less sturdier results.

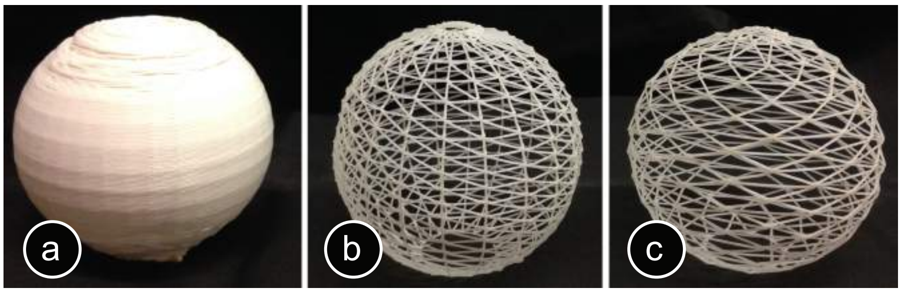

Figure 33: (a) Solid sphere. (b,c) Different wireframe patterns.

3.2.3 *Extending Wireprint to different levels of fidelity*

Several extensions allow WirePrint to handle additional scenarios, such as printing with multiple levels of detail and creating solid surfaces in a post-processing step.

#1 Additional detail by mixing in layer-wise printing
WirePrint also allows users to use different levels of detail in a single print. For instance, in the case of the bunny head shown in Figure 34, the user wants to preview the details of the face, such as the eyes and the nose, in the context of the face. Those detailed parts are printed with slow layer-wise printing, while the rest is printed as a fast WirePrint. This hybrid mix of both techniques allows for quick iteration while ensuring enough detail in those regions where it is required.

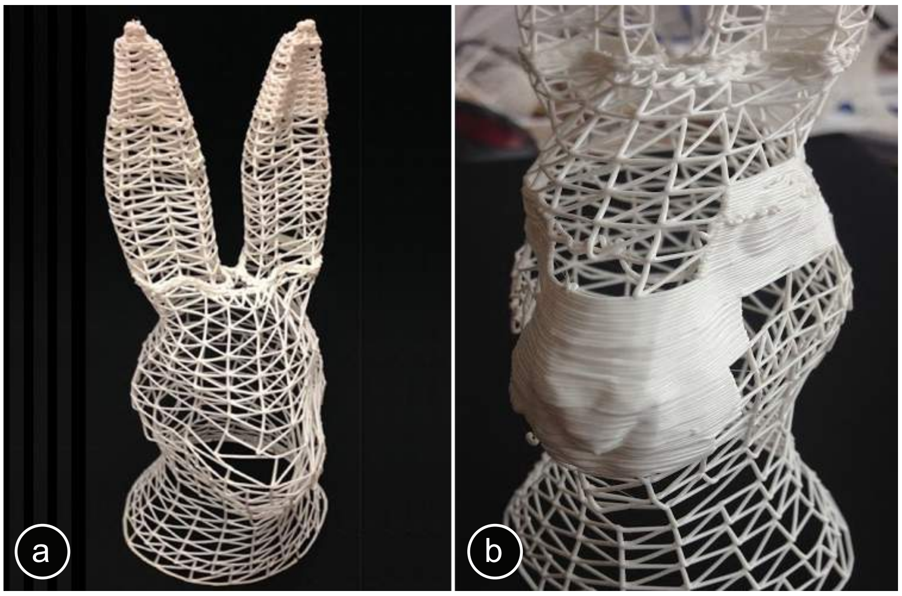

Figure 34: Mixing in layer-wise printing for the nose and the eyes.

To mark a region for layer-wise printing, the user simply uses the *fill* brush in the WirePrint software and brushes the sections of the zigzag pattern that should be printed in additional detail.

As shown in Figure 35, WirePrint prints hybrid models slice-by-slice. (a) It starts each slice by printing the contour, which is shared between the traditionally printed part and the wireframe printed part. It then prints the wireframe zigzag, while it is still able to place the slanted starting point. It finally prints the layer-wise printing part. (b) Afterwards, WirePrint continues with the procedure on the next slice.

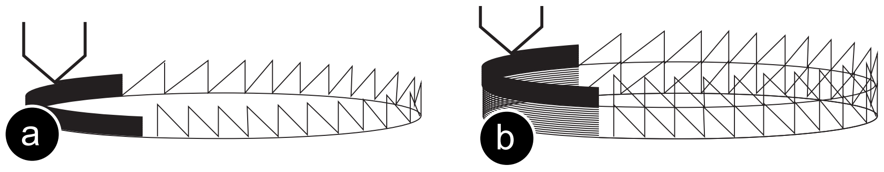

Figure 35: Order of printing edges for hybrid printing when wireframe and traditional printing are mixed.

#2 Objects with filled surfaces
Some 3D models, including the bottle mentioned in the previous section, require closed surfaces. To close the surface, we dip the wireframe print into glue (see Figure 36). An added advantage of this approach is that it strengthens the model.

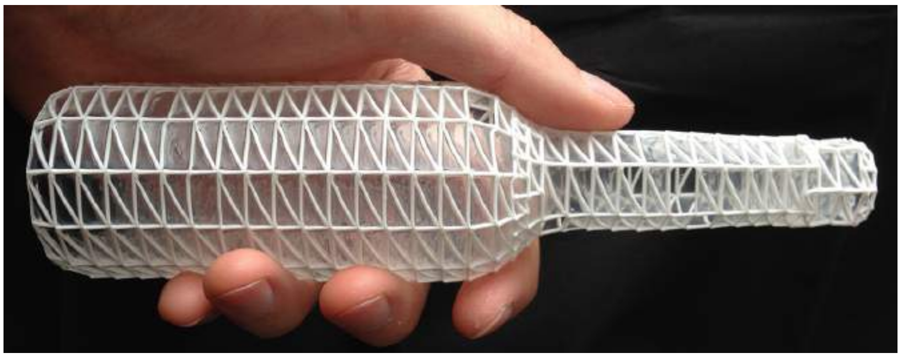

Figure 36: Filling surfaces, here the walls of the bottle, by dipping the wireframe into glue (e.g. *Mod Podge).*

3.2.4 *The mechanical aspects behind WirePrint*

In this section, we show which types of 3D printers are particularly suitable for WirePrint. In addition, we explain how changes to the hardware lead to an additional speed up and increase the stability of WirePrint objects. These hardware changes are optional and WirePrint also works without any modifications but at reduced speed.

#1 Delta printer
All of the shown wireframe models were printed on a *Kossel mini* 3D printer (Figure 37), i.e., a printer that moves the print head using six vertically actuated arms (a so-called delta design).

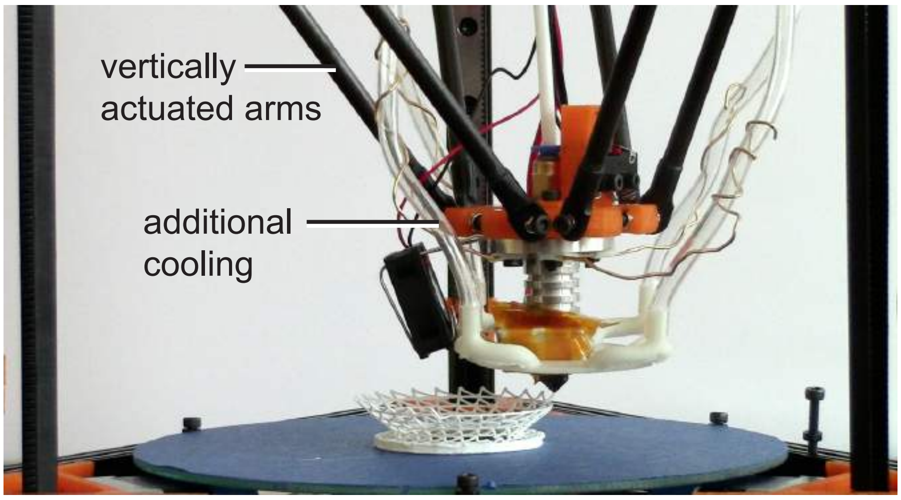

Figure 37: We use a *Kossel mini* delta printer. For additional speed we improved the fan cooling system.

WirePrint is particularly fast on the delta design, because delta printers allow the print head to move up and down quickly. 3D printers following the more traditional cartesian design tend to be slower along the vertical axis since layer-wise printing does not require high speeds in this direction. However, since the axis speed in the z-direction is simply a design decision of the manufacturers, WirePrint can be made equally fast on cartesian printers with the proper hardware design, i.e. by changing both the movement/turn ratio and the stepper motor speed in the z-direction.

#2 *Optimizing the filament for fast wireframe printing*
Since WirePrint requires frequent transitions between compliant and solid, we found materials that have a quick transition time to work best. From the two currently most common 3D printing materials *PLA* and *ABS*, the latter one works best. The reason is that *ABS* has a smaller temperature range, in which it changes its viscosity from compliant to solid (230-250°C) than *PLA* (180-250°C).

#3 *Extrusion thickness*
A larger opening in the extrusion nozzle leads to thicker and thus sturdier edges and thus to sturdier objects. On the flipside, thicker edges require more time to cool down, which slows down the printing process. For our purposes, we found a 0.7mm extrusion nozzle to lead to the best results, i.e. sturdy and fast to print.

#4 *Cooling*
We attached two air jets that are controlled by a solenoid valve to our print head (Figure 37). WirePrint controls the airflow by opening and closing the valve using *g-code* (command M42). The additional cooling causes the filament to solidify faster after extrusion, which allows WirePrint to move on even faster. In cases where WirePrint needs

the filament to stick to another part of the model, it turns the cooling off. When no additional cooling can be added to the 3D printer, WirePrint can either make use of the weaker built-in fan or can add an additional pause to wait for material to solidify.

#5 Support structures
One interesting aspect of our technique is that WirePrint requires less support material than regular layer-wise 3D printing because it can print overhangs of up to 90°. An example of a small 90° overhang can be seen in Figure 28a at the bottom part of the model (the maximum length we tested was 6.5cm length). WirePrint can print these overhangs because the string of extruded material is put under tension until it is completely sturdy. Yet other geometry, such as a human figure extending the arms downwards, can only be printed with support. As in the layer-wise approach, the need of support will reduce the overall print speed. However, one can think of using WirePrint to print support structures again saving time compared to the currently existing approach.

3.2.5 *Algorithms for automatic WirePrint conversion*

To help readers replicate our results, we use the following two sections to explain the details of our software implementation. WirePrint is written in *CoffeeScript* and uses the *constructive solid geometry* library for its geometry operations.

The WirePrint system
Our system WirePrint loads 3D models in *.stl* format and generates custom *g-code* (i.e., the instruction language used by 3D printers). The user can then export the g-code and load it into the standard 3D printing software for printing (we use *Repetier Host*). The custom *g-code* moves the print head along the desired path and controls how much material is extruded at which points.

#1 Slicing the model
After loading the 3D model, WirePrint slices the model into a set of slices. The locations of the slices is determined by (1) important features on the model geometry, and (2) the minimum and maximum height of the zigzag between two subsequent slices (we use a 0.7mm extrusion nozzle, which leads to 1.4mm minimal height, and our print head is 6mm high which leads to 6mm maximum height). To generate a slice, WirePrint cuts the 3D model against a horizontal slice (width and length of the object's bounding box) at a specific height using a Boolean *intersect* operation.

#2 Extracting the contour of a slice
From each slice, WirePrint extracts the top contour by converting it to a high-resolution bitmap and then applying *OpenCV's* findContour() algorithm. If there are multiple contours on one slice, *OpenCV's* findContour() algorithm also returns their relationship to each other, i.e. whether they are located next to each other or contained in each other. We use this information to determine the printing order.

#3 Generating the zigzag pattern
When generating the zigzag pattern, WirePrint maximizes the object's physical stability by aligning all vertical lines across slices. Simply using the points from the slice below does not work, because subsequent slices might: (1) have a different contour length, which can lead to insufficient space between two subsequent points (print head collision), and (2) slices might have different heights, which can lead to invalid printing angles. We therefore use a mixed approach: First we calculate the optimal even spacing of points for each contour. Then we calculate the minimum distance from a point on the bottom slice to the top slice. We then use the average of both, which leads to good stability and a comparably homogeneous spacing. In the case that two vertical lines are still too close to each other, we ignore one of them.

#4 Compensating for mechanical aspects
After generating the wireframe according to the model geometry, WirePrint applies all geometrical modifications that are required due to the mechanical properties of filament and print head, such as removing the last diagonal edge of the zigzag from the list of edges to avoid print head collision.

#5 Exporting g-code
In the last step, WirePrint converts the geometry information into *g-code.* For this, it traverses the list of edges (all contours and zigzag patterns) to export them in the right order for printing. It uses the start and end point coordinates of each edge to generate the movement commands for the print head and the length of the edge to determine how much extrusion is required. For instance, *G1 X10 Y 10 Z10 E5* means: move to those coordinates and extrude 5 units of filament on the way. Our *g-code* exporter also generates the commands for turning the fan on and off to properly cool the wireframe edges. The *g-code* exporter writes these *g-code* commands into a *.gcode* file that the user can then execute on the 3D printer.

#6 Detecting geometry splits in the geometry
In cases where a slice has a single contour and its subsequent slice has two (and vice versa), we print them on top of each other without the intermediate zigzag pattern. We use this particular approach to ensure that the zigzag of each new slice has filament printed underneath it.

#7 Combining with layer-wise printing
For mixing in layer-wise 3D printing, WirePrint generates additional slices between two subsequent wireframe slices. The number of additional slices depends on the printing resolution. For instance, if two wireframe slices are 3.5mm apart and the extrusion nozzle is 0.7mm, WirePrint will generate 3.5mm/0.7mm = 5 additional layers. After generating the slices and extracting their contours, WirePrint analyzes which part has been selected for layer-wise printing. It then cuts the contours at the start and end point of the selected part. The remaining part is filled with the zigzag pattern.

3.2.6 *Optimizing printing speed during solidification*

To create accurate and sturdy wireframes, WirePrint needs to take into account the edge deformation that appears when filament is not yet solidified (see Figure 38).

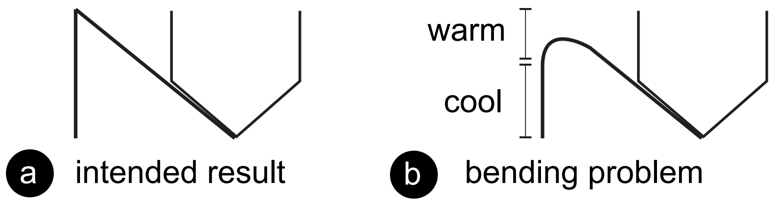

Figure 38: Deformation problem when edges are not yet solidified.

We identified three approaches to improve the print quality:

(1) reducing the overall speed with which the print head moves, allowing the filament to solidify as its being printed.
(2) moving the print head quickly, but pausing at the end of each vertical edge to let the edge solidify.
(3) printing full speed, but anticipating the deformation by extending the vertical movement of the print head.

Through experimentation (Figure 39), we discovered that the fastest way to print is to add a delay at the end of each floating edge, i.e., to move with maximum extrusion speed of the printer (30mm/s) combined with a 1s pause at the end of each vertical line. Although a pause at the end of each vertical edge initially seemed unattractive, it leads to the most accurate results because the edges solidify under tension. Anticipating the deformation during printing was attractive at first, but the extra traveling time of the print head quickly undermined the benefit of a faster printing speed, and in general the results where less appealing.

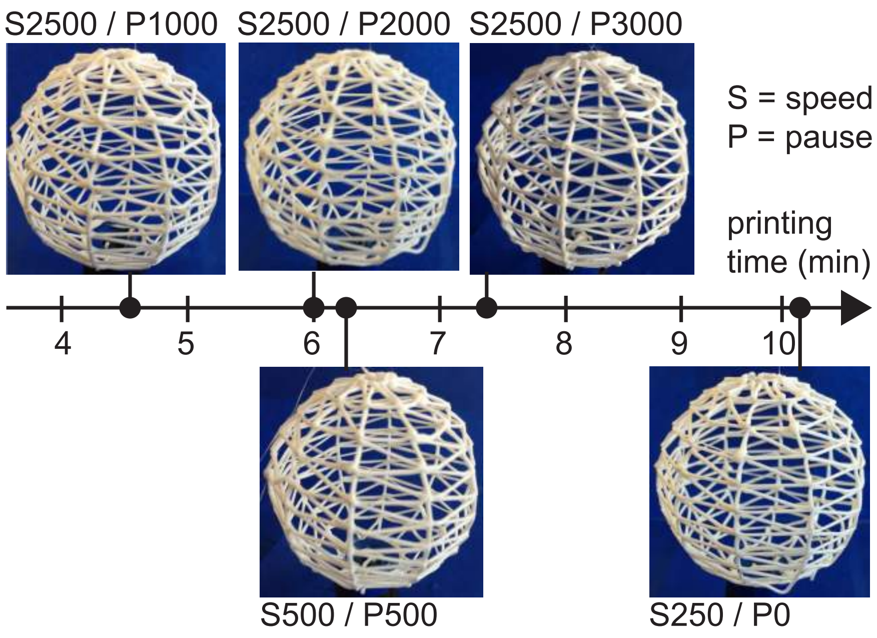

Figure 39: Print head speed/pause trade-off. Sphere radius = 3.9cm.

3.2.7 *Conclusion*

With WirePrint we made four contributions: First, we proposed 3D printing wireframe representations of 3D models as an approach to faster iteration when evaluating the shape of an object. Second, to maximize the speed-up, we printed edges directly into 3D space, i.e., we instructed the 3D printer to extrude filament not layer-by-layer, but along actual strokes in 3D-space. This approach allowed us to provide feedback to the user 90% faster compared to traditional layer-wise 3D printing (factors range from 2.5-10 depending on the model geometry). Third, we provide a software to automatically convert 3D models into WirePrints: Figure 40 shows example objects converted with our algorithm. Forth, to make our approach applicable to a wide range of users, we demonstrated how to create wireframe previews with existing 3D printers, users only need to install the WirePrint software. This is in contrast to special purpose devices, such as *Material* [54] that uses a custom two-component material that immediately sets after extrusion, and the *3Doodler* [1], which is a handheld 3D pen that has to be operated manually. Given that for a WirePrint object only a fraction of material is extruded, our approach is also substantially cheaper, making it even more affordable for users to iterate.

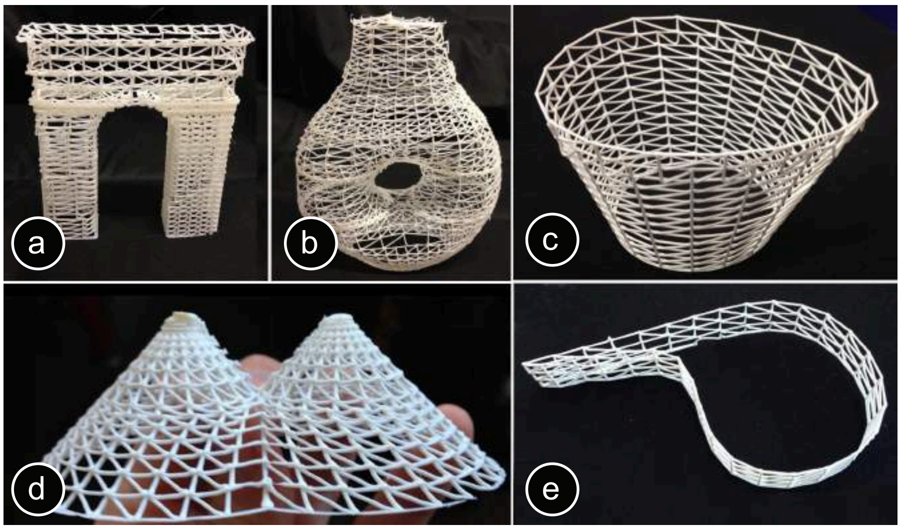

Figure 40: A selection of objects we have 3D printed using WirePrint.

On the flip-side, WirePrint is limited to model geometries that take the print head constraints into account. In addition, WirePrint is especially designed to speed up fused deposition modeling (FDM) 3D printing (e.g., *PrintrBot, Kossel mini, MakerBot*). It does not speed up 3D printers that print layers as raster images (e.g., *Polyjet, ZCorp)* and only small speed-ups might be achieved on selective laser sintering or stereo lithography hardware.

Since the original development of WirePrint in 2014 as described in this thesis, several follow up works have extended our algorithm for new use cases. For instance, in *On-the-Fly Print* [61] Peng et al. extend WirePrint towards 5-axis 3D printing and use it to print incremental changes directly on the physical object. In addition, a paper by Wu et al. [105] builds onto our work and extends the algorithm to not only use triangle patterns but also other mesh types, such as quad meshes.

While WirePrint is useful for quickly testing shape, it is less suitable for the later stages of design when users want to test the actual physical strength of their design that is required to perform mechanical functions. To address this problem, we developed a third low-fidelity fabrication method called Platener.

3.3 PLATENER: LOW-FAB OF FUNCTION

Platener is a low-fidelity fabrication method designed specifically to evaluate the functional aspects of an object. Platener achieves fast feedback by extracting straight and curved plates from the 3D model and substituting them with laser cut parts of the same size and thickness. Only the regions that are of relevance to the current design iteration are executed as full-detail 3D prints. Platener connects the parts it has created by automatically inserting joints. It also engraves instructions to help fast assembly. Platener allows users to customize substitution results by specifying fidelity-speed trade-offs, choosing whether or not to convert curved surfaces to plates bent using heat, and specifying the conversion of individual plates and joints interactively (see Figure 41).

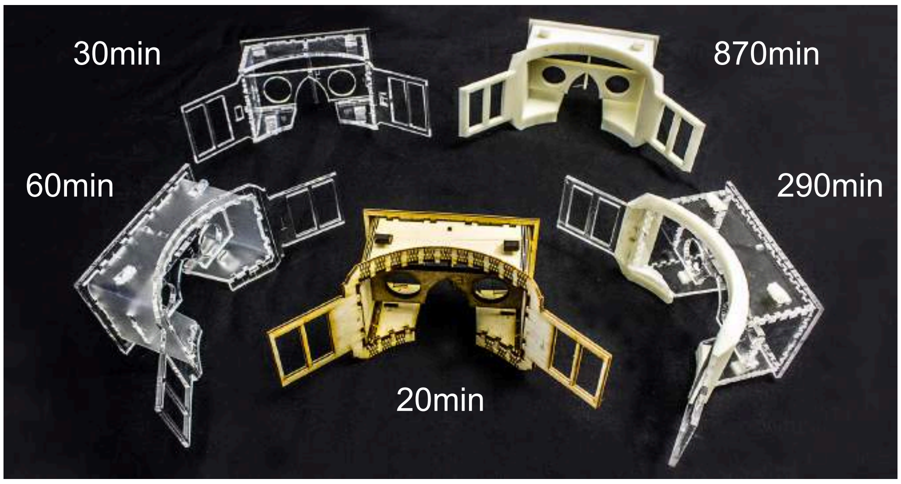

Figure 41: Platener substitutes parts of a 3D model with straight and curved plates that can be fabricated on a fast laser cutter.

Platener is designed to best preserve the fidelity of functional objects, which typically contain a large percentage of straight and rectilinear elements that can be fabricated on a laser cutter. Other approaches that convert 3D models into laser cuttable parts, are not suitable for this goal: Approaches that stack 2D plates (e.g. *123D Make [6]*, *Crdbrd* [36]), ensure that the volume of a 3D object is well approximated, but they are a less accurate surface representation. A different approach uses intersecting planar pieces: The resulting objects require less material and consist of fewer parts (McCrae et al. [55] and Schwartzburg et al. [73], but intersecting planar pieces do not preserve the overall function of the object. Finally, researchers have suggested to fold 2D sheets into 3D objects (e.g. Mitani et al. [58], *Curved Folding* [42]). However, while foldable sheets result in an accurate surface representation, they have no infill and are thus not sturdy enough to perform mechanical functions. Platener's approach of using laser cut parts of the same size and thickness bypasses this problem, thereby preserving the stability and functionality of objects.

3.3.1 *Converting a model with Platener*

For this walkthrough, we revisit the head mounted display body shown in the faBrickator section. While faBrickator allowed to test the position and shape of individual parts due to its high modularity, the resulting assembly was not sturdy enough to be worn on the head for a longer test period as the individual bricks tend to loosen up and subsequently fall apart. With Platener, we gain the ability to actually produce a functionally working version.

With its fastest conversion, Platener allows producing the head-mounted display body in 20 min (15 min cutting plus 5 min manual assembly), which is 44 times faster than the 3D print (14:30h). Platener returns the 3D model in the form of one or more 3D printable *.stl* files and laser cuttable *.svg* files, which can be sent directly to the 3D printer and laser cutter. Figure 42 illustrates this at the example of the hybrid head-mounted display body from Figure 41.

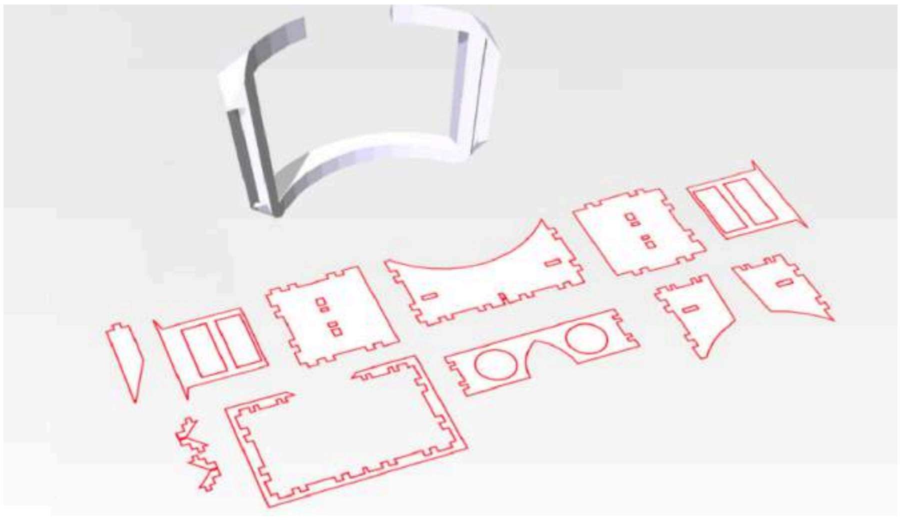
Figure 42: Exported 3D printable and laser-cut parts as *.stl* and *.svg* files.

#1 Adapting the conversion to the current design phase
To best suit the needs of the current design phase, Platener allows users to convert 3D models under different settings. For an early test, for example, users may convert their models with speed in mind; in later phases they may gradually shift the emphasis to fidelity. As an example, Figure 41 shows four different versions of the head-mounted display. They were generated from the same 3D model, but with different fidelity-speed settings. Platener allows users to specify these trade-off settings using a global slider.

Speed-fidelity: Setting this slider all the way to *fidelity* produces an object that is all 3D printed, which trivially preserves fidelity, but at the expense of having no speed-up. Moving the slider towards *speed* causes Platener to initially replace very large regions with individual laser-cut plates (Figure 43a); then more and more regions get replaced, resulting in increasingly higher speed-ups (Figure 43b). Platener performs this in interactive speed, i.e., as the user is dragging the slider, Platener continuously updates the model. This allows users to see the changes right away and thus to quickly find the conversion that best suits their needs.

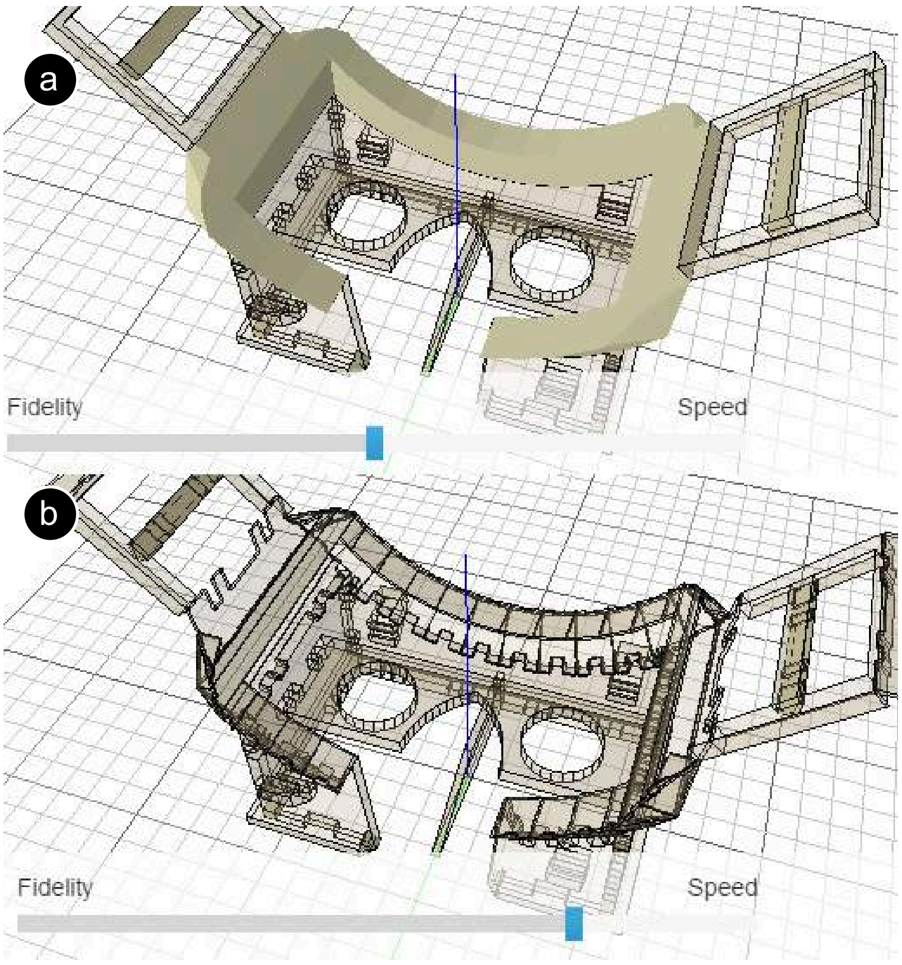

Figure 43: Platener's user interface provides users with a global slider to define the fidelity-speed trade-off.

Curved surfaces: Platener also provides options for handling curved plates. When the *curved plate* option is deactivated, Platener approximates curved plates using individual plates connected by finger joints (Figure 43b). Activating the curved plate option causes Platener to convert cylindrically curved regions in the 3D model to one plate that can be laser-cut and bent. Curved plates come in two styles: The first option *bend acrylic* fabricates a flattened version of the respective surface using the laser cutter that can then be bent using a heat gun. As illustrated by Figure 44, users (a) cut the pieces, (b) heat them up using a heat gun or strip heater [82], (c) pre-shape them along the dashed instruction lines Platener provided, and (d) press them into the corresponding positions in the model, which gives them the desired shape.

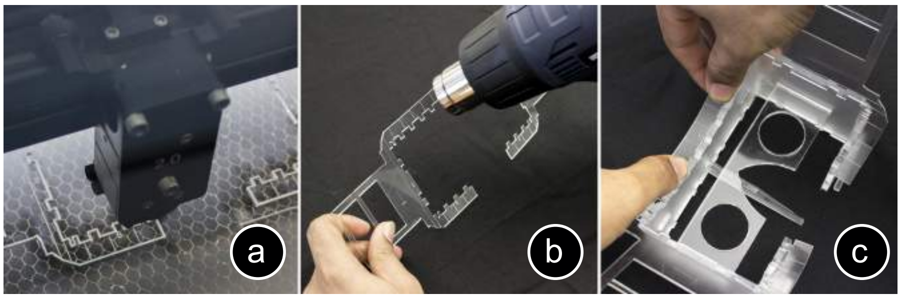

Figure 44: Bending: (a) cut plate, (b) heat it using a heat gun, (c) press it into position to give it the right shape.

The second curved plate option uses a wooden living hinge. As illustrated by Figure 45, for this to work, Platener cuts a dense pattern of living hinges into the plate. This results in a plate that is flexible along the intended dimension . In this case, users only have to (a) cut the curved plate and then (b) mount it using the provided finger joints.

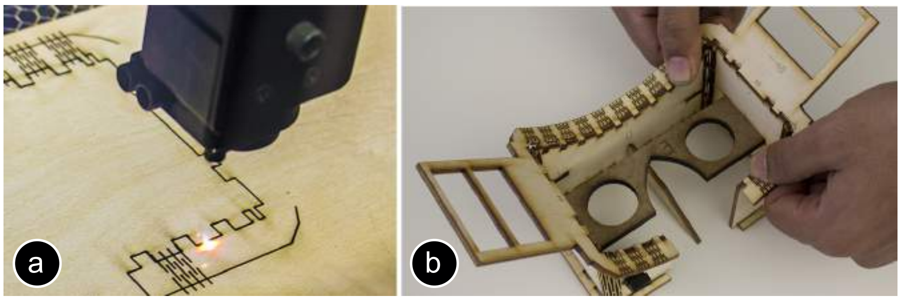

Figure 45: Creating a curved plate from wood. (a) Cut the object on the laser cutter, (b) mount it into position.

Finally, as shown in Figure 46, Platener helps users assemble parts by embedding matching pairs of labels.

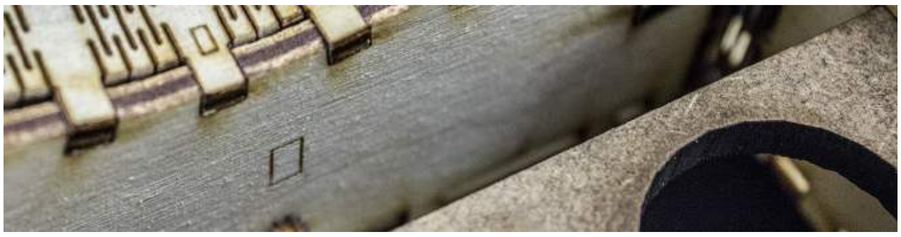

Figure 46: Platener 3D prints/engraves instructions.

#2 *Specifying the focus of the current design iteration*
On top of these global settings, different design iterations tend to put the focus on different parts. Platener allows users to define such a focus by assigning a specific fabrication technique using brushes. For the head mounted display, the global threshold defines that the

forehead piece should be 3D printed. Within this context, the user can now choose the focus of the current design iteration:

Case 1: The current design iteration is about some other part, such as the optical path. The forehead piece thus does not matter. In this case, the user can use the *do not print* brush on the forehead piece and the device will be fabricated without this part (Figure 47).

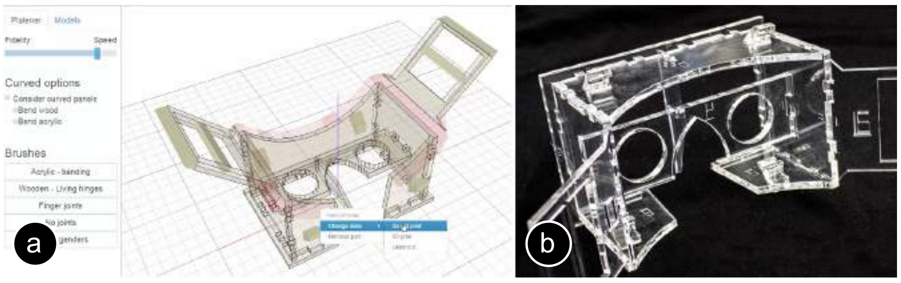

Figure 47: The design iteration does not require the head piece. The user thus uses the *do not print* brush.

Case 2: The current design iteration is about the ergonomics of the forehead piece, thus the contact area of the forehead piece matters. The user has a heat gun and decides to use the *curve acrylic* brush on the forehead piece (Figure 48).

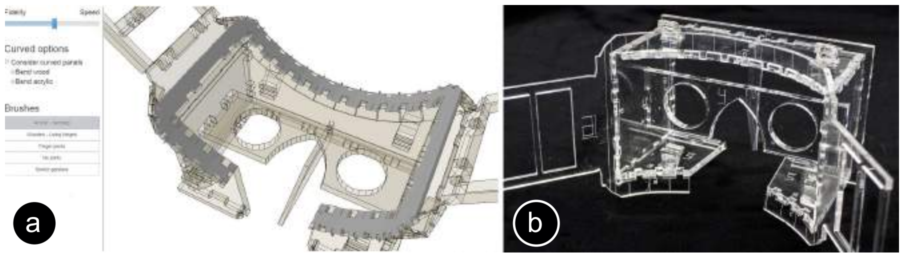

Figure 48: Overwriting the default with the *curve acrylic* brush to obtain a curved version for ergonomic testing.

Case 3: In this round of design, not just the ergonomics, but also the industrial design of the forehead piece is in focus. The user thus overrides the setting of the forehead piece in the last iteration by using the 3D print brush, resulting in the hybrid object shown in Figure 41.

Case 4: In yet another round of design, the mechanics and the stability of the object are in focus. Per default, Platener has chosen to represent the four vertical edges of the casing as finger joints. These are easy to assemble, but brittle and thus not a good approximation for the stability of a 3D print. The user handles this by customizing these edges/connectors, similar to how one customizes plates. As shown in Figure 49a, the user uses the *bend acrylic* brush and brushes across the connectors. This causes Platener to merge the four wall segments into a single long strip (Figure 49b). The assembled result is shown in Figure 49c: a version of the head mounted display that is sturdier than any of the converted versions we showed earlier as its main body consists of only three pieces.

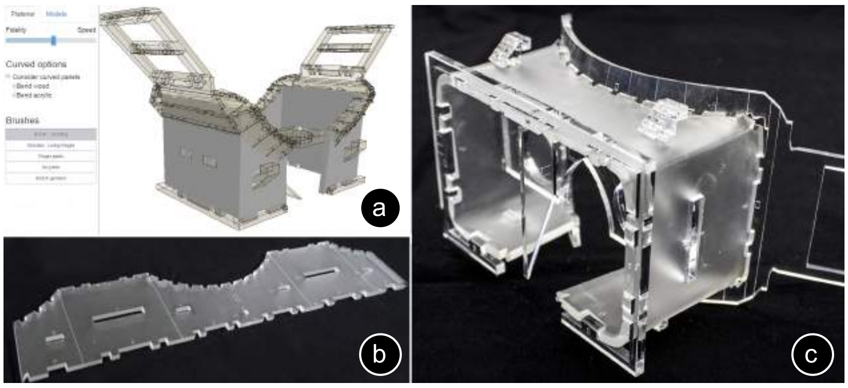

Figure 49: Using the *bend acrylic* brush, the user defines the casing as being made from one piece, which makes it sturdier.

3.3.2 *Algorithms for automatic Platener conversion*

This section describes Plateners core data structure and the implementation of the processing pipeline.

#1 Mesh segmentation
The mesh segmentation algorithm takes a 3D printable manifold mesh as input and outputs laser-cuttable plates and 3D printable parts. Platener then generates the plate graph of the 3D model according to the adjacency relationship of these segments. The segmentation algorithm first extracts all flat planes from the 3D mesh, then identifies plates by iteratively grouping plane pairs that are parallel, have an opposite normal, and are within distance d. The variable d represents the thickness of the plate and users manually configure it according to the laser cutting material. Each plate consists of two planes. The size and shape of the plate are determined by projecting both planes into 2D and intersecting them. We repeat the algorithm with all planes that remain after intersection until no more planes can be converted to plates. After extracting all plates, the algorithm labels the remaining parts of the 3D mesh as to 3D print.

#2 Plate graph generation
After the mesh segmentation, Platener constructs a plate graph that records the fabrication method for each segment and joints between them. As illustrated by Figure 50, each node in the plate graph represents one part of the 3D model and each edge represents a connector between two plates. Each node has one of the three states: *3D print*, *laser-cut*, or *ignore*. Each edge, i.e., connector, has one of the five states:

no connector, bending connector, finger joint, interlocking joint, bending connector, or *glue connector.*

Initially, Platener labels all nodes as *3D print* and all edges as *no connector.* As users move the *fidelity-speed* slider towards *speed,* Platener gradually labels the largest laser-cuttable segments as *laser-cut* and their adjacent edges as *finger joints.* Similarly, as users manually specify a fabrication method for each part, Platener interactively updates the corresponding labels in the graph.

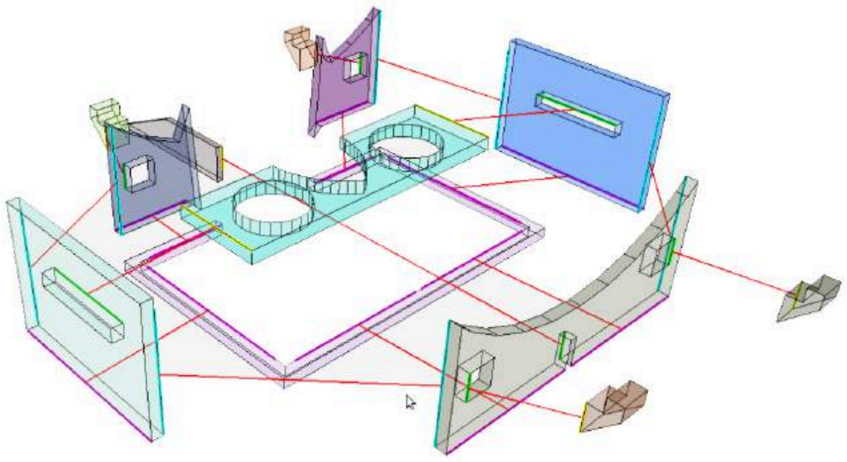

Figure 50: The plate graph represents the 3D model as a graph of nodes (plates) with edges (connectors).

#3 Mesh processing
At the end of the process pipeline, Platener processes the mesh segments according to the plate graph.

Unfolding curved parts: When the *curved plate* option is activated, Platener gradually unfolds the curved segment onto a 2D plane. Plate- ner also adds the bending lines or the living hinge pattern perpendicular to the unfolding direction. If the 3D segment cannot be unfolded without overlapping itself [28], Platener converts it to 3D printing as a fall back method.

Joint generation: According to the labels on the edges, Platener places joints between segments or merges segments into one piece. When the edge is labeled as *no connector,* e.g., both nodes are 3D printed, or *bending connector,* e.g., users applied the *bend acrylic* brush on the joints, Platener merges them into a single 3D mesh or plate. When using *finger joints,* Platener extends the boundaries of the segments and generates finger joints on the overlapping volume. The *interlocking joint* label is used when there is a cross-section between two segments. In such cases, Platener creates a slit on the laser-cut piece, allowing another plate to slide into and interlock with. Finally, in case the overlapping volume is too small, Platener switches to the *glue connector,* i.e., creates flat surfaces on both segments that afford gluing.

3.3.3 *Evaluating the speed up*

We evaluated the effectiveness of Platener by analyzing the speed-up after conversion for a range of different models. Given that Platener was designed to evaluate the functional aspects of an object, such as the stability of a casing and the performance of a mechanical tool, we focused on 3D models that their creators described in such terms.

3D meshes from Thingiverse
In order to obtain an objective sample, we downloaded 3D models from the online 3D model repository *Thingiverse.* At the day of the download, *Thingiverse* contained more than a hundred thousand 3D models in total. We collected the 3D models by searching the website using the sites built-in search box and downloaded the top 200 results. In order to obtain an objective sample, we did not perform any further selection or removal of mislabeled objects at this stage, so our sample represents the noise level currently contained in these sites. Also note that many 3D models on *Thingiverse* consist of multiple 3D meshes. We processed them separately without merging them into one single 3D model. The 10 search terms we used were: *box, camera, controller, gadget, gear, household, mechanical, phone, speaker,* and *tool.* We obtained an overall number of 2,250 3D meshes.

Measurements
We evaluated the effectiveness of Platener based on two measurements: (1) fabrication time before and after conversion, and (2) number of pieces as a metric indicating the manual assembly effort.

Results
Figure 51 shows the time savings achieved by Platener: 39.9% of the 3D models achieve a speed up of more than 10x, which is faster than any of the systems in the related work. Another 13.4% achieve a speed up of 3-10x. 22.8% of models result in no speed up.

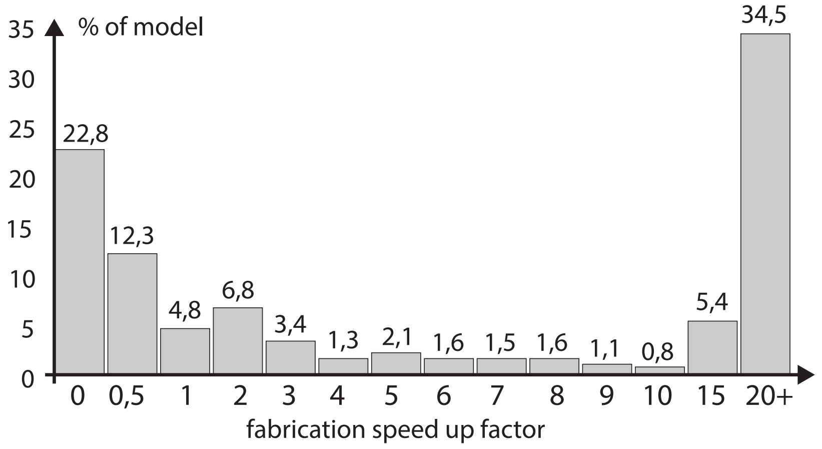

Figure 51: Fabrication time speed up.

Platener processed all 3D models and generated 3D printable *.stl* files and laser-cuttable *.svg* files. We estimated the 3D printing time with the *slic3r* g-code simulator, the laser-cut time by the length of the path contained in the *.svg* files, and we approximated the assembly time via the simple formula of number of pieces multiplied by 15 seconds. Converted objects consisted on average of 12 pieces, which resulted in approximately 3 minutes of assembly time.

Figure 52 shows some example objects that resulted in high speed ups (>10x). These objects consist mainly of flat pieces and contain few parts that cannot be laser cut.

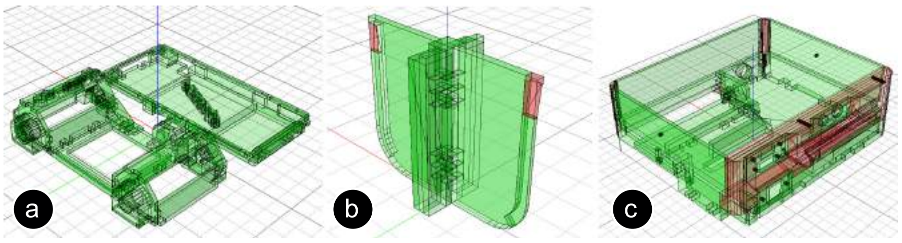

Figure 52: Some of the models that Platener fabricated 10x+ faster: (a) camera body, (b) hardware divider, (c) *Raspberry Pi* casing.

In contrast, Figure 53 shows two examples for which Platener was unable to produce any time savings. As expected, these objects are characterized by spherical curvature shapes that lie outside of what Platener was designed to handle. For these objects the WirePrint low-fidelity fabrication technique is a better option.

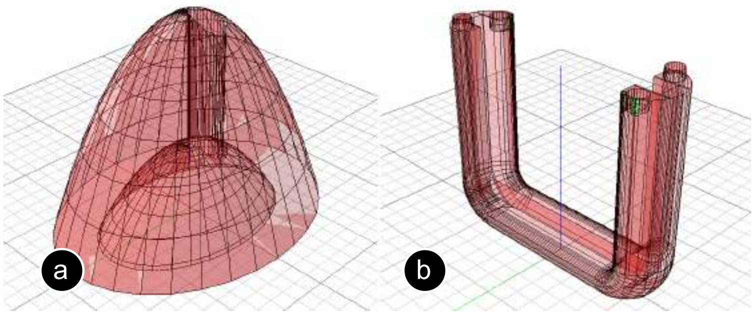

Figure 53: We would not use Platener for these objects (no time-savings): (a) cone shaped mini heater part, (b) *Kindle Fire* comfort grip.

3.3.4 *Comparison with faBrickator and WirePrint*

Compared to faBrickator, Platener generally better preserves a models geometry while also preserving its function. We illustrate this at the example of the three objects used in faBrickator, i.e., the penny ballista (Figure 54a), the soap dispenser (Figure 54b), and the head mounted display body (Figure 54c).

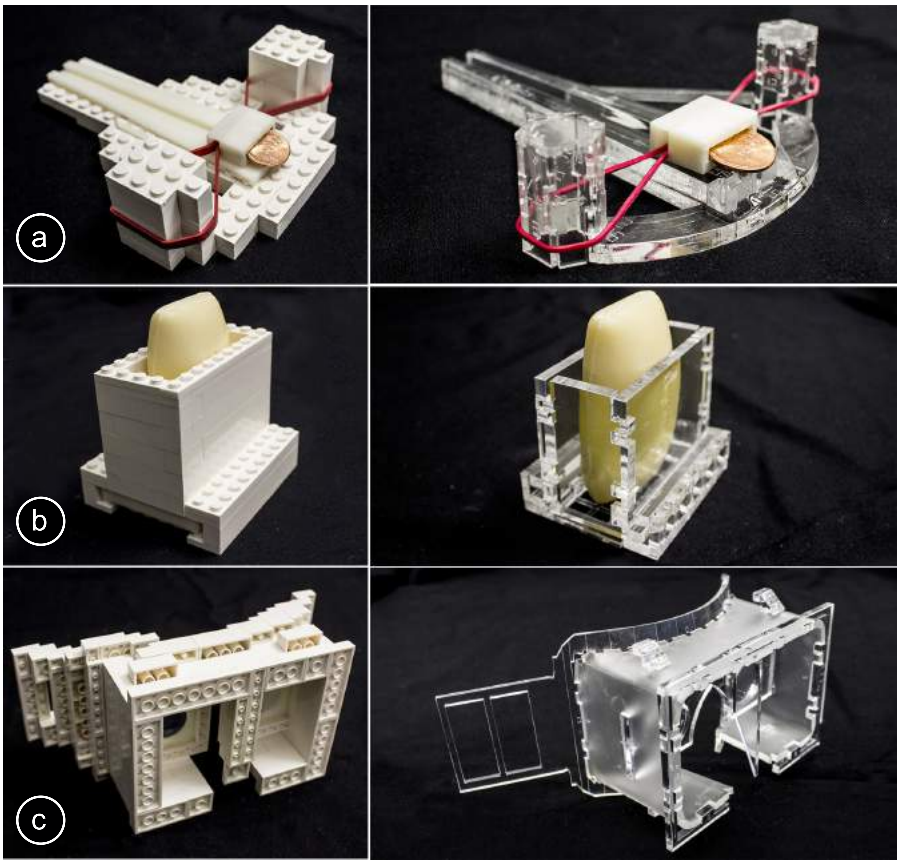

Figure 54: Left side shows conversion with faBrickator, right side Platener: (a) penny ballista, (b) soap dispenser, (c) head mounted display body.

Platener objects are generally sturdier as they are made from fewer pieces than when using building blocks (penny ballista: 17 plates vs 99 Lego, soap dispenser: 21 plates vs 53 Lego, head mounted display body: 9 plates vs 160 Lego). The curved surface, despite requiring more manual effort, results in particularly sturdy results.

3.3.5 *Conclusion*

Our results show that Platener achieves faster fabrication time for many objects with functional aspects, making it a useful tool for substantially speeding up the prototyping workflow and providing faster feedback. While Platener is intended to be a low-fidelity fabrication technique, some objects produced by Platener are of sufficient fidelity that users may choose to use the Platener conversion to fabricate also the final output. On the flipside, Platener is subject to the following limitations: The thickness of the extracted plates is limited to the capabilities of the laser cutter (e.g., 10 mm maximum plate thickness on our *ULS PLS6.150D* laser cutter). While Platener speeds up design iteration of functional objects, it is less suitable for objects that are solely defined by their shape, such the objects shown in Figure 53.

3.4 CONCLUSION FOR LOW-FIDELITY FABRICATION

Low-fidelity fabrication tightens the feedback-cycle from the hours range to the minutes range, allowing users to evaluate the current version of their prototype faster. By providing faster feedback, users achieve more design iterations in the same amount of time, which ultimately leads to a working solution earlier.

We showed how different conversion techniques allow users to focus on different key aspects of an object, such as first testing the position of parts using the modular approach of faBrickator, then moving on to test the shape of objects with WirePrint, and finally testing the technical function of objects with Platener.

On the flipside, low-fidelity fabrication trades in detail for speed. While suitable for iteration, the converted objects do require a full 3D print eventually at the end. Some low-fidelity fabrication techniques also require a modest amount of manual effort since users need to assemble the parts that make up the final object.

For future work, we plan to evaluate data on how users use our tools for their prototyping process. For this, we have already deployed faBrickator as a webservice *(www.brickify.it)* and log data about which models users convert and how much time they save. We are currently in the process of deploying a similar platform for the Platener low-fidelity fabrication technique. WirePrint is also publicly accessible as part of the *Cura Slicer* implemented by the *Ultimaker* 3D printing community based on our research paper.

4 ELEMENT-LEVEL: TURN-TAKING SYSTEMS

While low-fidelity fabrication allows users to create, test, and redo objects quickly, redoing an object in its entirety is not necessarily the most effective approach. Arguably, feedback is most beneficial when making key design decisions along the way. Turn-taking interfaces offer this affordance by providing fast request and response transactions: users first create an input and receive an answer from the system within seconds. This tightened feedback loop allows users to iterate towards a final solution through trial-and-error by building subsequent steps onto the results of previous ones.

To transition from the low-fidelity fabrication systems that fabricate an object in one go to turn-taking system with a request and response interaction, we decrease the interaction unit to a single *element* of an object, such as one line.

To illustrate turn-taking for personal fabrication, we start with a 2D system called constructable and then extend this system towards 3D editing with our novel laser-cutting technique LaserOrigami.

4.1 CONSTRUCTABLE: 2D INTERACTIVE LASER CUTTING

constructable is a turn-taking system based on a laser cutter that produces precise physical output after every editing step. As illustrated by Figure 55, all interaction in constructable takes place on the workpiece inside the laser cutter, mediated through low-power hand-held laser pointers, which we call proxy lasers or simply tools. In the example in Figure 55b, the user uses the *finger joint* tool to add finger joints between two pieces by crossing the two involved edges.

Proxy lasers are too weak to affect the workpiece. To make the interaction 'real', constructable tracks proxy laser interactions using a camera mounted above, reconstructs the tool's path, transforms it using a constraint set defined by the current tool, and implements the effect using its high-powered cutting laser (Figure 55c). Since all key elements were constructed in the context of constraints, constructable allows creating fully functional devices, such as the simple motorized vehicle shown in Figure 55d.

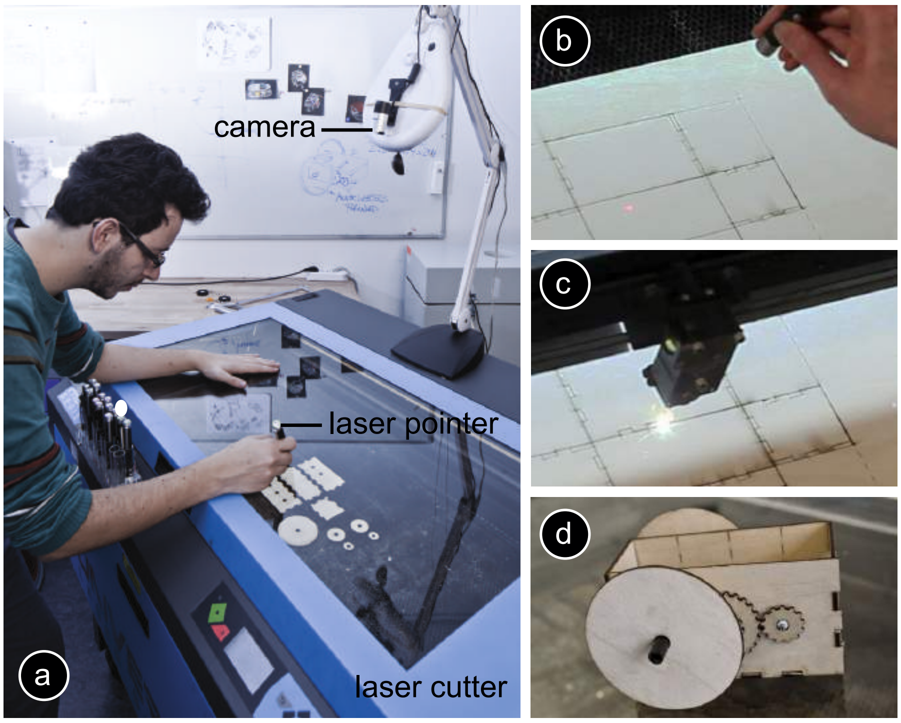

Figure 55: (a,b) Constructable users first draw an element, (c) the laser then cuts it, providing physical feedback. (d) The final object.

4.1.1 *Proxy lasers for direct control and precision*

Figure 56 provides a closer look at the proxy lasers. Each proxy laser features three barrel buttons (Figure 56b). While held depressed, the middle button activates the beam, allowing the system and the user to see where the tool is pointed [59]. The visual feedback allows users to determine a starting point with precision before starting to cut. It thereby implements the tracking state of its three-state model [16].

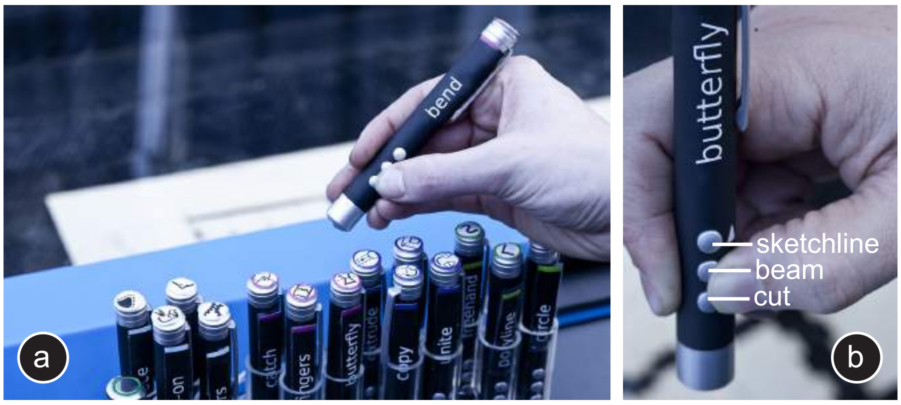

Figure 56: (a) constructable tools. (b) Each laser offers three buttons.

The other two buttons trigger the tool's two modes of operation. The cut button allows cutting a tool-specific shape, such as a circle for the *circle* tool. The sketch line button creates the same shape, but etches it as a shallow dashed line into the surface of the material. Sketch lines have no direct impact on the mechanics of the workpiece, but instead serve as alignment aids that magnetically attract subsequent cuts *(alignment lines* [14]).

All tools explain themselves exclusively through the cut or sketch line they produce and there is no further visual feedback, i.e., no screen or projector.

4.1.2 *Creating, selecting, and copying with proxy lasers*

constructable achieves precision by means of sketch lines and by implementing constraints into every proxy laser. Constraints differ between tools.

Creating: Polyline, circle, and *freehand* are constructable's tools for creating objects from scratch. These tools are only moderately constrained. The *circle* tool, for example, always produces a perfect circle, but diameter and location remain freehand. The *freehand* tool is not subject to any constraints. Most of constructable's tools connect to or extend an existing object and this spatial relationship adds constraints. Users establish these constraints by selecting one or more existing objects. The *finger joint* tool, for example, snaps to existing lines.

Selecting: As illustrated by Figure 57, users select (a) a surface by clicking into it, (b) an edge by crossing it (*Crossy* [2]), and (c) a point by drawing a pigtail close to it (*Scriboli* [38]). We designed this selection mechanism so as to extend seamlessly to multiple objects. Users select (d) multiple surfaces by drawing a path across, (e) multiple edges by crossing multiple edges, and (f) multiple points as a sequence of multiple pigtails.

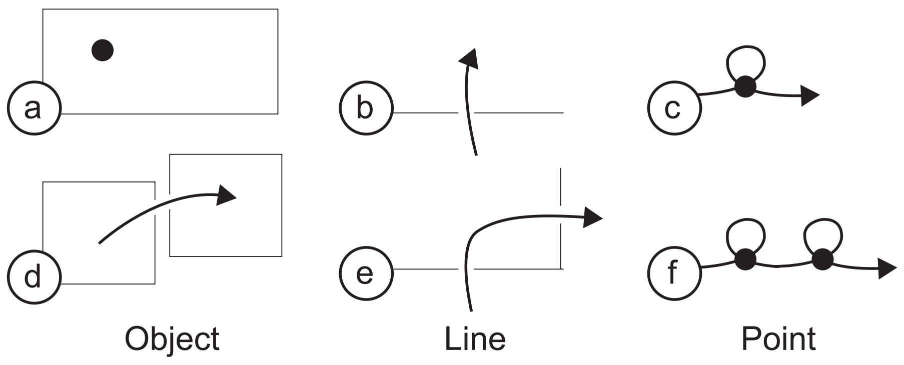

Figure 57: constructable allows users to select (a) objects by pointing, (b) lines by crossing, and (c) points with pigtails. (d-f) Selecting multiple objects, lines, and points.

Pasting: A range of tools, such as the *copy* tool, result in the creation of new objects. The size and shape of a new object is determined implicitly, e.g., by the object being copied and does not require or allow for user input. However, to allow users to optimize material usage, we let users show constructable where to create the new object. As illustrated by Figure 58, users point constructable to available material by drawing a directional cropmark (*Papiercraft* [49]). The orientation of the cropmark specifies the orientation of the pasted object, allowing users to optimize for material use.

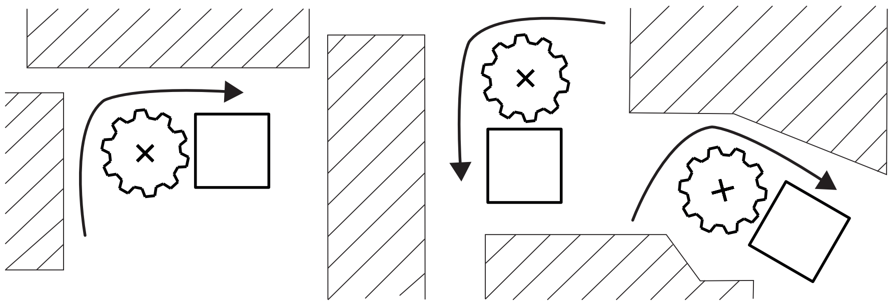
Figure 58: Users paste an object by drawing a directional cropmark.

4.1.3 *Walkthrough: constructing a device*

In the following, we illustrate constructable's tools at the example of the simple motorized vehicle shown earlier. Figure 59 shows the final outcome and the pieces required to produce it.

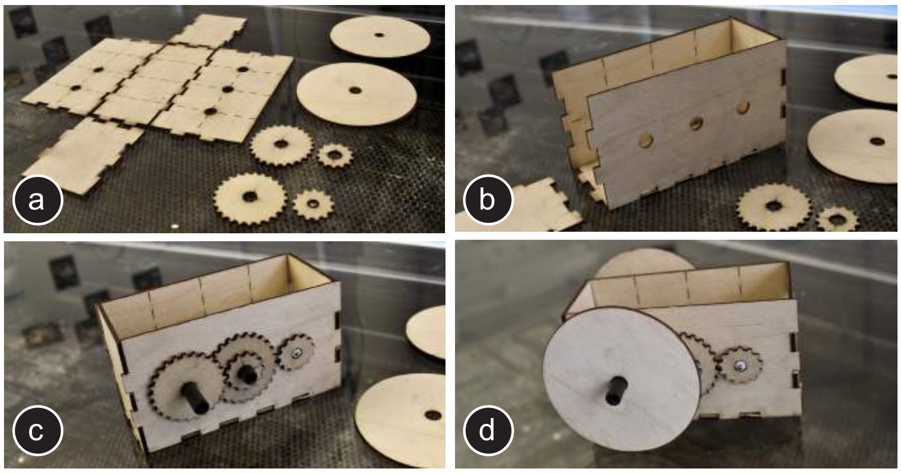

Figure 59: (a-d) The motorized vehicle in different states of assembly.

We start by creating the housing (Figure 60): (a) We use the *polyline* tool to sketch the rectangular base. (b) Using the sketch line button on the *scale* tool we create a sketch line rectangle around the base—this sets the height of the walls. (c) By crossing the north edge of the base with the *extrude* tool, we create the first wall segment. (d) For efficiency, we create the remaining three walls using a single long stroke that extrudes the base east, south, and west. There is no limit on concatenating, so we could have also extruded all four walls in a single stroke. (e,f) To allow us to assemble the housing later, we add finger joints. We connect the walls by crossing pairs of respective edges using the *finger joint* tool. (g) Finally, we assemble the box by connecting the finger joints.

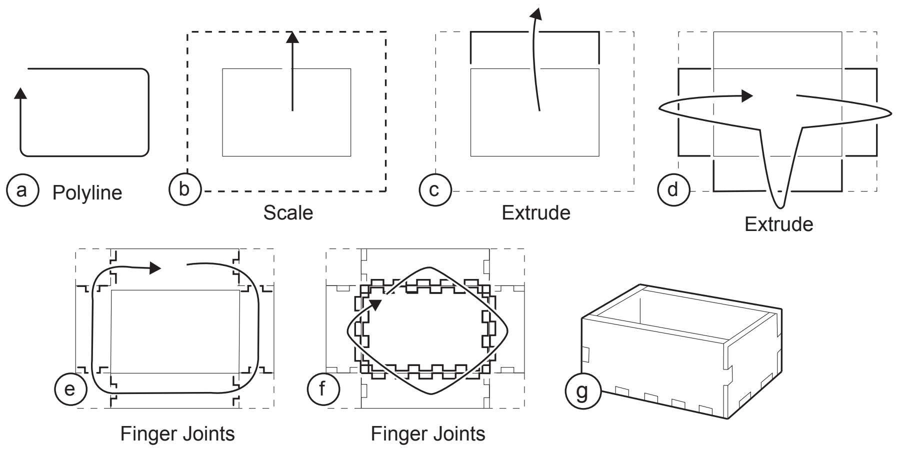

Figure 60: Interactively constructing the housing of the motorized vehicle.

Figure 61 shows how we add gearbox and wheels. (a) To make sure that we end up with straight axles we draw five sketch lines using the *polyline* tool. (b) We create the first axle hole using the *circle tool*, the location of which snaps to the intersection of the two sketch lines located close by. (c) We draw all remaining axle holes using a single stroke concatenating multiple pigtails.

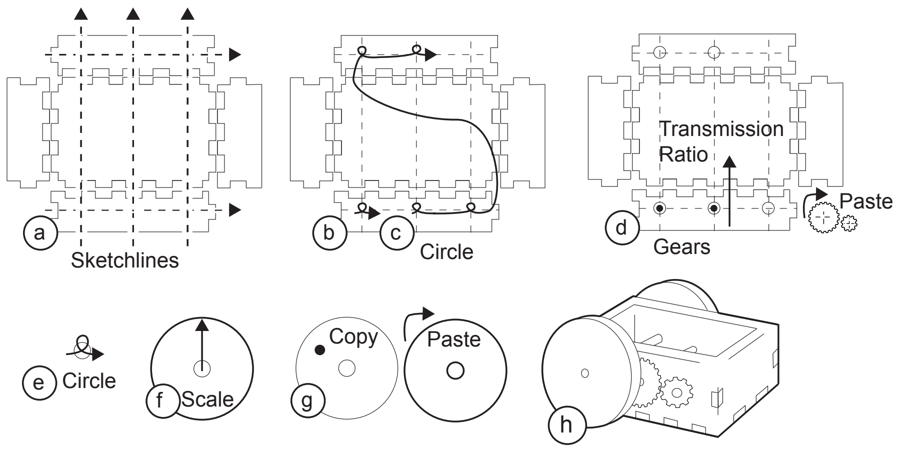

Figure 61: Continuing the previous example, we add axles, a two-stage gearbox, and wheels.

(d) We create the first pair of gears by selecting two axle holes with the *gear* tool, by defining the transmission ratio through marking the point where we want the two gears to meet, and by showing constructable where to create the gears using a cropmark. We create the second set of gears accordingly. (f) To create a wheel, we first create an axle hole using the *circle* tool. We then create a wheel around this axle hole using the *scale* tool. (g) We create a second wheel by copying the first one using the *copy* tool. We are now done creating our parts. We remove them from the machine and assemble them, resulting in (h) the final vehicle. The device is functional because every mechanical connection was created using tools with appropriate constraints.

Decorative functionality
constructable also allows integrating form-giving and decorative functionality. Figure 62 shows how we create a booklet that has round corners and a flexible bend in the middle. We start by (a) drawing the cover with the *polyline* tool, then (b) smoothing the corners with the *round* tool, and (c) making the wood flexible using the *bend* tool.

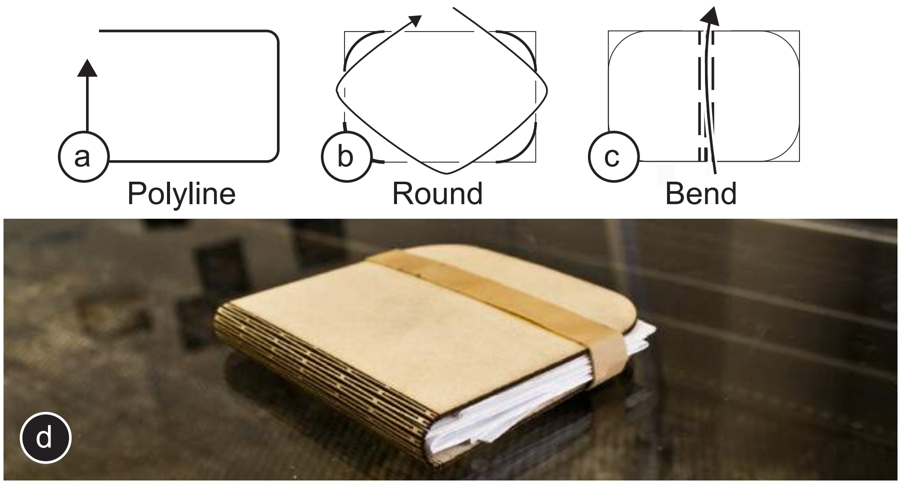

Figure 62: Creating a wooden booklet sleeve.

Figure 63 illustrates how we apply a picture to the housing of our motorized vehicle. (a) We place the housing back into the machine and position the picture on top of it. We wave the *rub-on* tool across the areas of the picture we want to transfer. (b) We take the picture out, and as we close the lid, constructable engraves those areas.

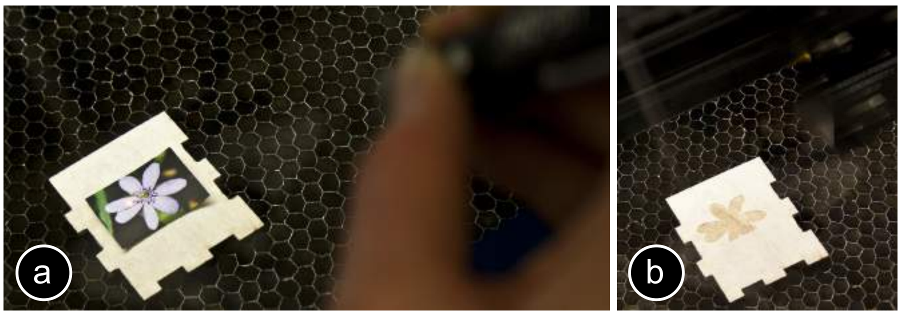

Figure 63: Adding a decorative logo using the *rub-on* tool.

Similarly, the *trace* tool cuts the contours of a physical object into the workpiece. Figure 64 shows how we use this to create a holder for two paint jars by (a) selecting physical objects inside the cutter using the *trace* tool. (b) After removing the physical objects constructable cuts, resulting in (c) the final jar holder.

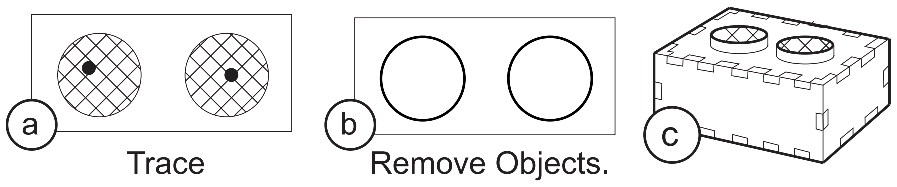

Figure 64: Creating a holder for two paint jars.

Finally, we can use the *freehand* tool to create unconstrained freehand lines and cuts (Figure 65), such as the heart drawing on the other side of the vehicle's housing.

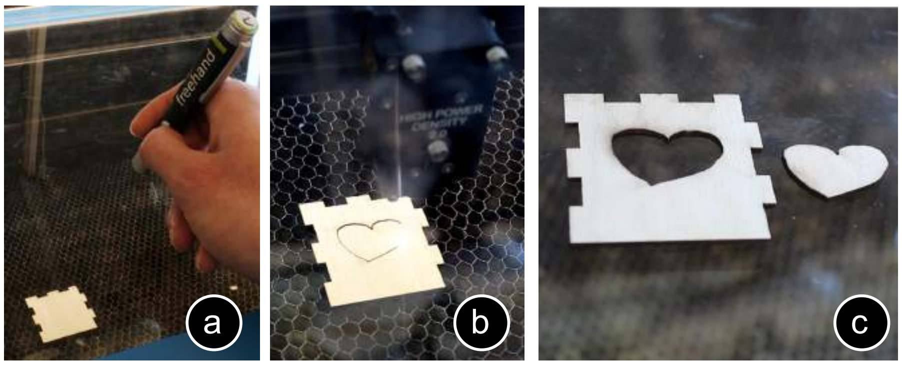

Figure 65: Sketching using the freehand tool.

Trial-and-error support using "undo" tools
Finally, constructable offers basic support for trial-and-error by providing an approximation of "undo" tools. Since physical cuts cannot be undone, constructable's "undo" tools instead refabricate the object — they create a copy that does not have the cut. Users apply the tool by crossing the cuts they want removed; they then paste the newly restored object (Figure 66).

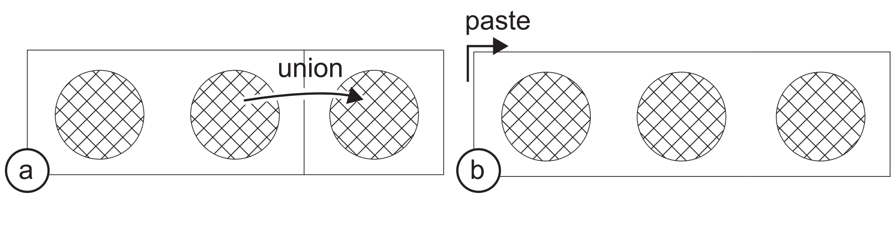

Figure 66: Undoing a cut by re-uniting the pieces using the *union* tool.

While the primary purpose of the tool is to repair and undo, the way it achieves this is by uniting two objects and copying the result. Since this functionality is useful beyond undo, we ended up giving the tool the name *union* tool. In practice, any tool that unites two objects can be used as an undo tool. Figure 67 shows the *butterfly joint* tool, which connects two objects using a butterfly connector. This tool produces a butterfly-shaped hole across the cut and lets users paste a matching butterfly-shaped connector. Users repair the cut on assembly by placing the connector into the hole—it sits tight enough to create a lasting connection. While the *union* tool obviously creates the stronger connection, a butterfly joint consumes less material.

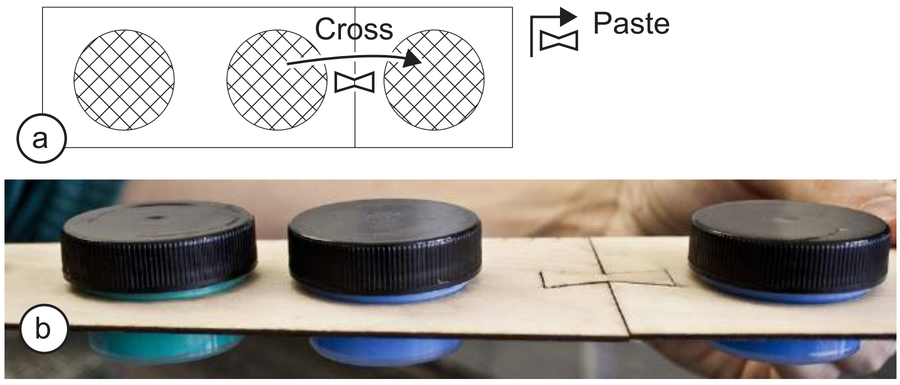

Figure 67: Undoing a cut by joining the pieces using the *butterfly* tool.

4.1.4 *Ergonomics: the drafting table form factor*

While we initially perceived it mostly as a design hurdle, the laser cutter's glass cover turned out to become one of the key elements creating the affordance of our system. (Figure 68a). By allowing users to rest their body weight on the glass, users get even closer to the workpiece without worrying about interfering with it. Furthermore, we found ourselves resting proxy lasers on the glass while drawing (Figure 68b), which adds substantial stability, making the interaction more precise. To invite this interpretation and posture, we positioned the proxy laser tools as shown in Figure 68c. This allows users to reach tools without lifting their arm, but instead pivoting around their elbows similar to the Lagoon in *Alias Sketchbook* [30]. We found all of the above to invite the interpretation of constructable as a drafting table, the "drawing" on which is the actual physical object itself.

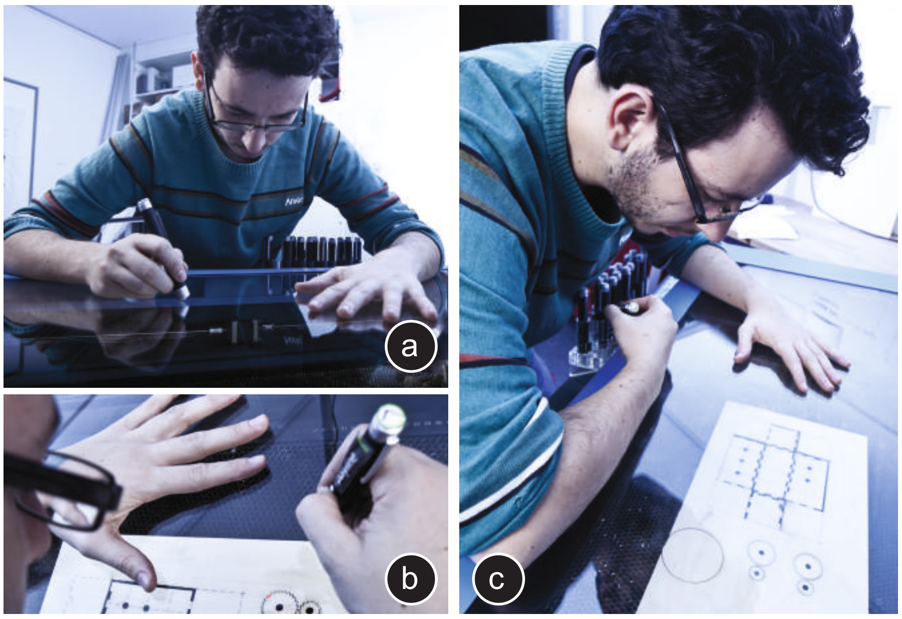

Figure 68: (a) The glass cover supports users' weight, allowing them to get close to the workpiece. (b) Resting proxy lasers on the glass surface allows for precise interaction. (c) Pivoting the elbow to switch the tool.

4.1.5 *Constructable's processing pipeline*

constructable has two main components: the first covers the data processing and is written in C++, the second covers the laser cutter instructions and is written in Java. constructable uses the following libraries to achieve its functionality: (1) *OpenCV* for computer vision, (2) *PaleoSketch* for shape recognition, (3) *OpenOffice* for creating the laser cutter document, (4) *OSC* for the communication between its components, and (5) *QT* for its graphical user interface, which is only used for debugging purposes and to adjust the tracking settings.

Figure 69 shows how constructable processes proxy laser input in order to generate cutting laser output.

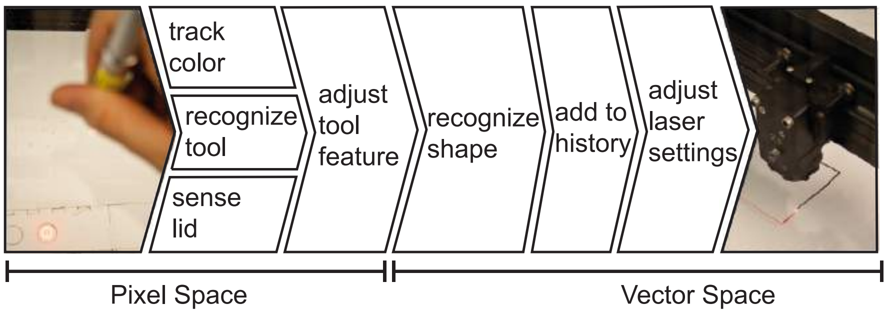

Figure 69: Data processing flow in constructable.

#1 Constructable's tracking pipeline:
constructable observes the workpiece using a camera *(MS Lifecam,* 844x448px, 30 fps) mounted above the laser cutter. Since the camera image needs to be aligned with the cutting bed of the laser cutter, constructable calculates a perspective transformation. We do this by first manually indicating where the four corners of the laser cutter are located in the captured image by clicking on them. Once the transformation matrix is calculated, we apply it to each frame before further processing it.

constructable extracts the red dot produced by the laser-pointer using color tracking (Figure 70a). constructable first converts the image into the *HSV* color space. It then provides the user with a user interface to filter for the bright red laser dot using sliders to set minimum and maximum values for the *hue, saturation,* and *value* channels. When the user is done, constructable writes the current settings into a log file, which is automatically loaded when constructable is started the next time.

constructable only starts tracking the red laser dot, when the *cut* or *sketch line* button on the laser pointer is pressed (Figure 70b). To extract the red laser dot from the image, we first threshold the *HSV* image using the minimum and maximum values for each channel, which results in a binary image. Afterwards, we run the *connected component* algorithm and identify the largest component as the laser dot. We now fit an ellipse around the laser dot and use the center of the ellipse as the tracking coordinate. While the user keeps the cut or sketch line button pressed, we repeat the process on every frame and write the coordinates into a list.

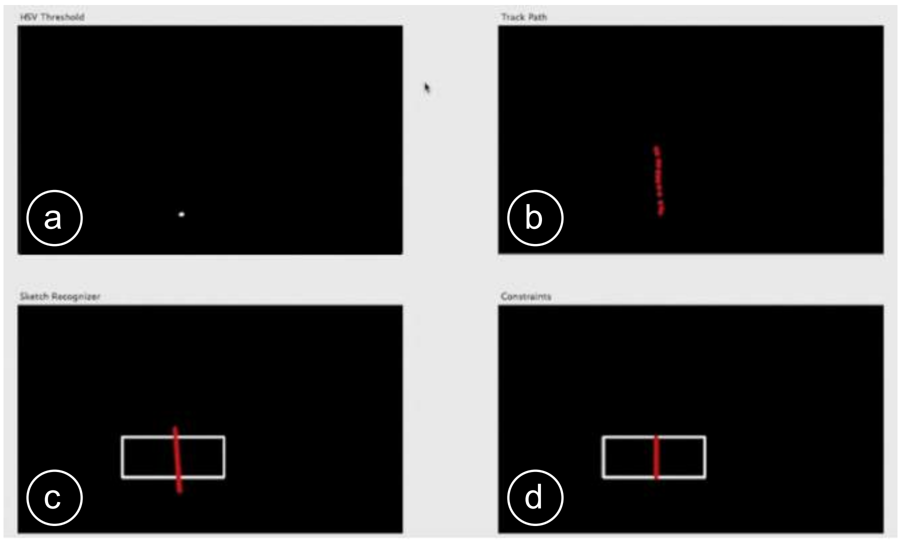

Figure 70: constructable processes input by: (a) thresholding the camera image to extract the laser dot, (b) tracking the laser dot over time, (c) performing shape recognition on the list of points, and (d) applying tool specific operations, such as calculating intersections.

When the user releases the *cut* or *sketch line* button, constructable smoothes the path using the *Kalman Filter.* It then performs shape recognition on the path using the *PaleoSketch* shape recognizer [60] (Figure 70c). Since the sketch recognizer is only available in *Java,* constructable sends the path data via *OSC* to *PaleoSketch* and *PaleoSketch* returns the shape after processing is done. Since all constructable input is based on lines (i.e. a pigtail is also a set of short lines), we only activate the *LineTest* and the *PolylineTest* in *PaleoSketch* and deactivate all other possible shapes, which increases recognition accuracy.

After the drawn path is converted into a geometrical primitive, constructable applies tool specific geometric operations (Figure 70d). For instance, when the user crosses an edge with the *round* tool, we calculate the intersections of the drawn line with the existing geometry. If two connected lines are crossed, the user crossed a corner. We then calculate the corresponding bezier-curve points at the correct locations. Another example is the *extrude* tool. If the user extrudes an edge, we take the endpoint of the path, calculate the shortest distance from the endpoint to the edge, and then use this information to construct the new line parallel to the existing line.

At this point, constructable records the interaction history, which it uses to support undo and selective repairs using the *union* tool.

#2 Output to the laser cutter

constructable now outputs the shape to the laser cutter, currently a *Universal PLS6.150D.* Using the *OpenDraw* API, constructable draws the shape into an *OpenDraw* document that has exactly the size of the cutting table. For this, constructable first initializes a connection to the *OpenDraw* program, it then loads the *OpenDraw* document, and afterwards calls the different shape factories for creating and drawing the shapes. According to the available tools, constructable currently supports circles (for the *circle* tool), beziers (for the *round* tool), point lists for any polygonal shape (for, e.g., the *polyline* or *trace* tool) and images (for the *rub-on* tool). Before the shapes are placed into the *OpenDraw* document, constructable transforms their position from the camera space (844x448px) to the document space (80x45cm) using a constant scale factor.

After constructable placed all shapes into the *OpenDraw* document, it sends the document to the laser cutter using the regular printer interface. As required by our laser cutter, constructable encodes the necessary meta-information into the color of the respective line, i.e., *cutting-depth*(mm), *laser power* (percentage), and *speed* (percentage). A red line, for example, causes the laser to cut, while green is used to create a shallow, low-power sketch line. constructable sends all communication using *OSC,* which makes it easy to adapt the system to other hardware components, such as a different laser cutter model. When the document is successfully sent to the laser cutter, we use a mouse robot to automatically press the start button in the laser-cutter software (Figure 71). While this is the only working approach to control the laser cutting process via code, it also has the drawback that the laser cutter user interface always needs to be the active window and that the window needs to be full-screen as the mouse click position is in absolute coordinates.

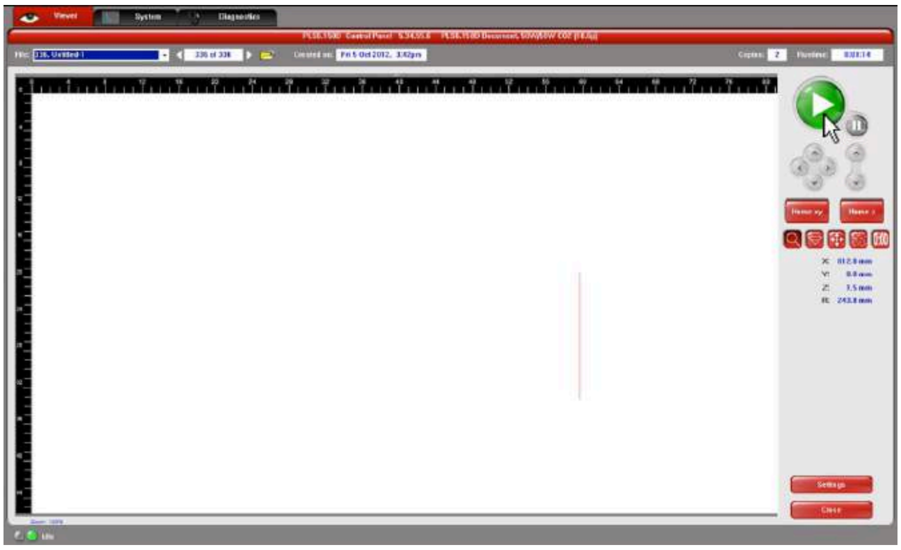

Figure 71: constructable uses a mouse robot to start the cutting process.

#3 Proxy lasers
In order to retrieve the constraint set represented by the current tool, constructable determines which tool is in use using mechanical switches, one of which is installed in each tool holder (similar to pen "recognition" in *SmartBoards*). This allows us to implement all proxy lasers using a single type of off-the-shelf laser pointer. The switches in the toolbox are connected to an *Arduino Uno,* which runs a program to determine which switch was activated and if the tool was taken out or put back in. The current state of the toolbox is then communicated to the main constructable application.

The buttons on all proxy lasers trigger an infrared signal, which constructable monitors using an infrared receiver placed next to the camera. By default, the commands sent by the laser pointers are mapped to the *left* and *right* arrow key, as they are normally used in presentations to forward to the next slide or to go one slide back. constructable, in contrast, maps these key commands to the *cut* and *sketch line* mode using *QT's key* API.

The *cut* and *sketch line* buttons are spring-loaded in order to eliminate mode errors. Users operate them without letting go of the tracking button. For optimized ergonomics during prolonged use, constructable offers a dual-footswitch pedal, which is fulfilling the same purpose as the cut and sketch line barrel buttons. When the footswitch is pressed, a program written in *PureData* notices the change and sends the current state to the main constructable application.

#4 Remixing: using physical objects as a reference
constructable captures objects using the same camera that tracks the laser dot. A bar of fluorescent light mounted inside the cutter supports this by providing homogeneous, reflection-free illumination of the workpiece (Figure 72). We mounted the bar of light inside the laser cutter by attaching magnets to its backside. We also guided the cable in a way that it would not interfere with the laser beam that exits the laser module at the top left corner of the laser cutter.

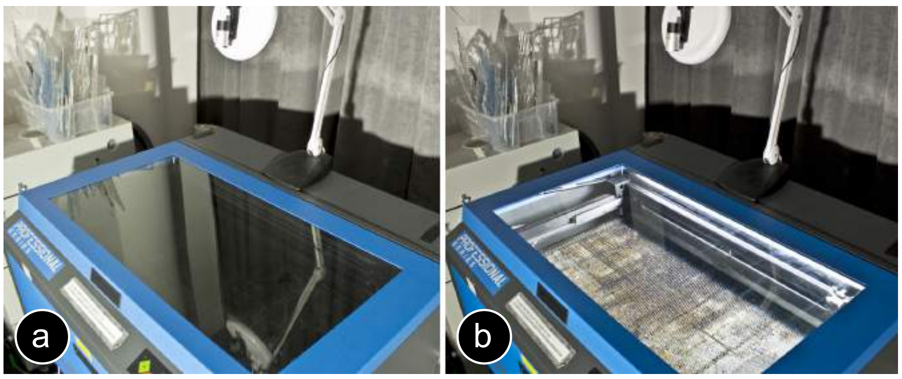

Figure 72: (a) To minimize reflections on the glass surface, we (b) mounted a bar of fluorescent light into the laser cutter.

To detect if the user inserted an object for remixing, we track if the lid of the laser cutter was opened or closed. The laser cutter already provides a mechanism for detecting the lid state for safety reasons, i.e. it stops cutting as soon as the lid is not completely closed. However, without a library to access this built-in feature we were not able to integrate it into constructable.

Instead, we placed an infrared rangefinder sensor (Figure 73) behind the lid that informs the system when the lid is open. When the lid opens, the distance to the range finder gets smaller – when the lid closes again it gets larger. We connected the range finder to the same *Arduino Uno* that is used by the toolbox.

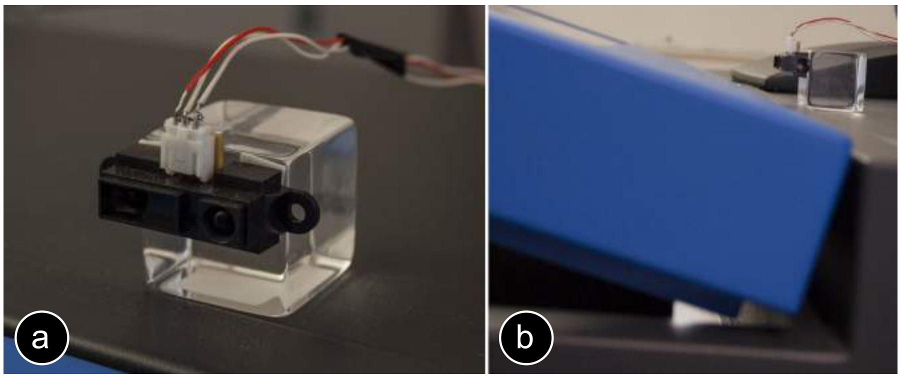

Figure 73: (a) We use this infrared range finder to (b) detect if the lid of the laser cutter is open or closed.

In the following section, we describe how we use *OpenCV* for copying object contours. After the user inserted objects, closed the lid, and used the *trace* tool to indicate that the user wants to copy the objects, we start the processing pipeline. First, we subtract the background: we use the current frame as the input and a frame when no object was in the laser cutter as the background. We then extract all contours using the *connected components* algorithm and remove contours that have an area smaller than a certain threshold as noise. When the user opens the lid and removes some objects, we repeat the process, but at the end check if any contour disappeared. The disappeared contours are the removed objects and these contours are laser cut as soon as the user closes the lid again. The remaining contours are on hold as we do not want the laser cutter to cut in places where objects are still present.

4.1.6 *Conclusion*

We showed how constructable tightens the feedback-cycle by providing physical feedback after every editing step–thereby enabling a turn-taking interaction style: Users draw an element of their design onto the workpiece inside the laser cutter, the system then responds by cutting the desired shape. However, since constructable is based on a fast 2D laser cutter instead of a slow 3D printer, it is limited to two-dimensional parts, which users have to assemble at the end. This makes the design process difficult as users have to mentally map a 2D part layout to a 3D object to imagine the shape and function of their final design. To tackle this problem, we developed a new laser cutting technique that creates 3D objects with a laser cutter via cutting and folding a single piece of acrylic, thereby allowing users to see the 3D shape as a direct result of their input to the system.

The key idea behind LaserOrigami is that it achieves three-dimensionality by bending the workpiece rather than by placing joints, thereby eliminating the need for manual assembly. Figure 74 shows an example object created using LaserOrigami — a mobile phone screen cam holder. This example was fabricated using five *bends*, one of the basic design elements of LaserOrigami.

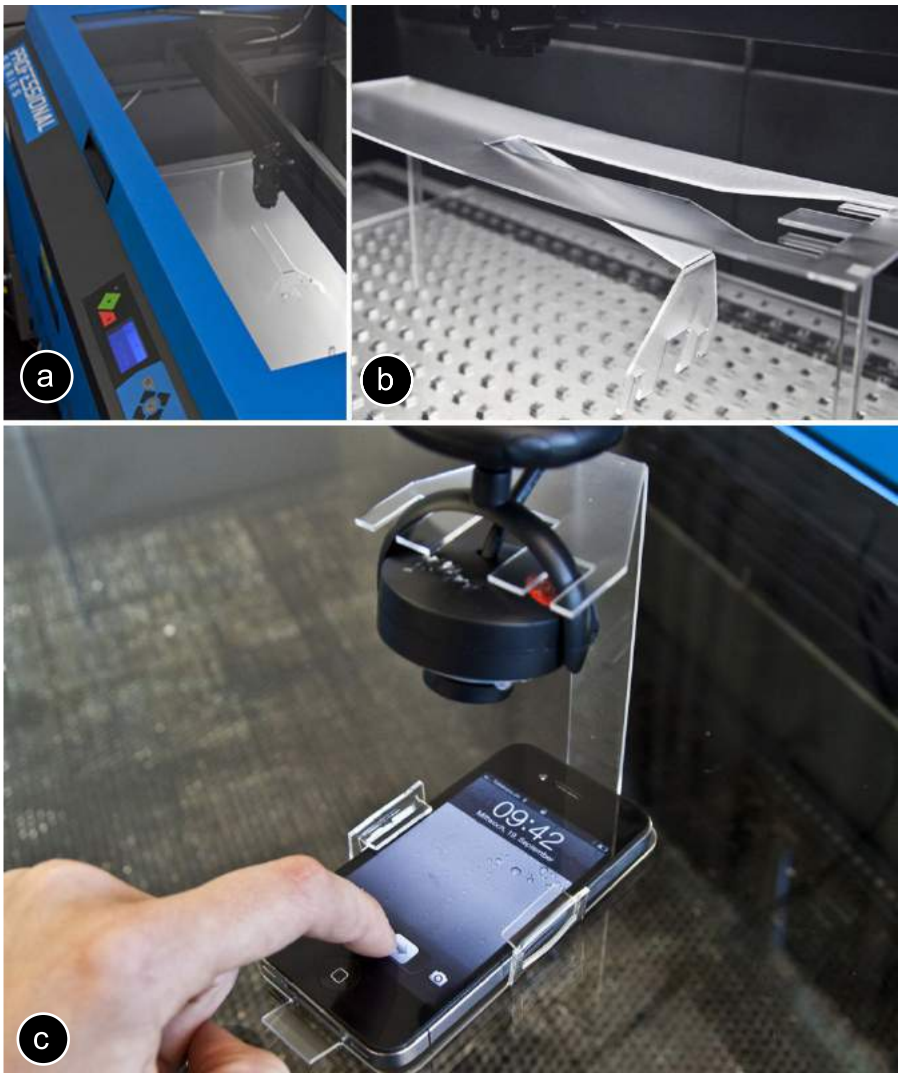

Figure 74: LaserOrigami fabricates 3D structure by bending rather than using joints, thereby eliminating the need for manual assembly.

4.2.1 *Bending with a defocused laser*

Traditionally, users manually bend the workpiece using *heat guns* (emit a stream of hot air) or *strip heaters* (consist of a line of heating elements). LaserOrigami's approach in contrast automates the process.

LaserOrigami is inspired by research that shows how to automatically deform metal using a laser [32]. When heating up metal, the material locally expands and contracts, which the sheet metal evades by bending. Researchers showed how to build concave [10] and convex shapes [48], and how to reshape existing 3D objects [47]. However, while this work relied on internal *tension*, LaserOrigami uses *compliance;* this allows it to work with acrylic and is an order of magnitude faster – fast enough to allow for a turn-taking.

Figure 75 shows how LaserOrigami implements cutting and bending. When cutting, the laser is normally focused on the workpiece, which causes the material to turn so hot that it evaporates (Figure 75a). To bend, LaserOrigami moves the workpiece away from the laser, which defocuses it (Figure 75b). This distributes the laser's heat over a larger region. LaserOrigami distributes the laser's heat further by repeatedly running the laser back and forth over the region to be bent. As a result, the workpiece heats up to the point where it turns compliant; it then bends under the influence of gravity. The result is a precise 90° bend. LaserOrigami modifies focus by moving the cutting table up and down; the cutter we used *(ULS PLS6.150D)* allows doing this under computer control. This allows it to implement cutting and bending in a single integrated process.

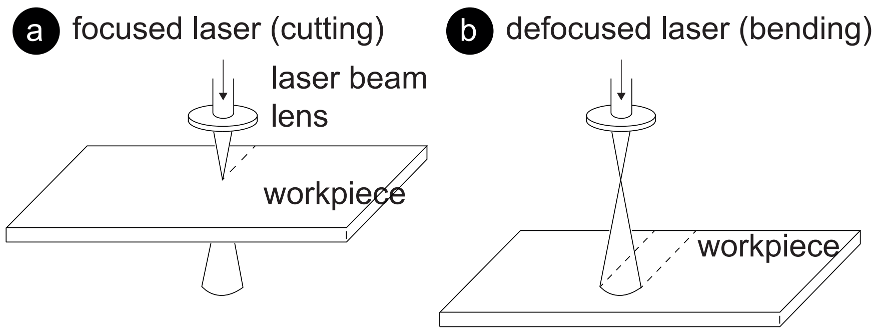

Figure 75: (a) LaserOrigami cuts the workpiece by focusing the laser on the workpiece, (b) it bends by defocusing the laser.

In order to allow parts of the workpiece to drop, we prop up the workpiece inside of the cutter (for our laser cutter, this is possible after removing the honeycomb grid). We created a simple configurable support grid for this purpose (Figure 76).

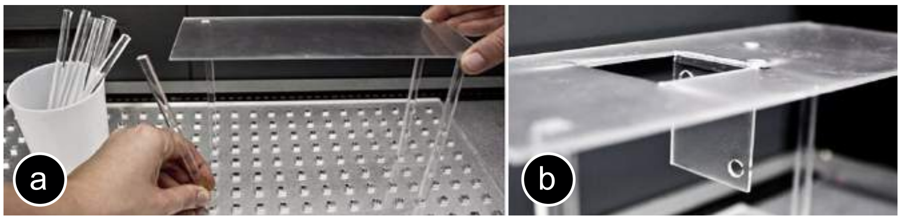

Figure 76: (a) This support grid (b) creates empty space below the workpiece that parts can fold and drop into.

4.2.2 *The design elements of LaserOrigami*

Figure 77 shows LaserOrigami's three design elements: (1) the bend, (2) the suspender, and (3) the stretch.

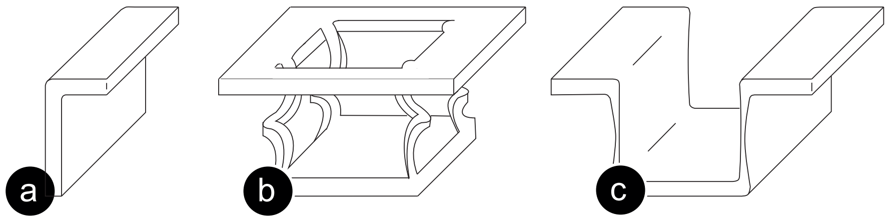

Figure 77: Design elements: (a) bend, (b) suspender, (c) stretch.

#1 Bends
We already demonstrated bends in Figure 74. Bends also allow fabricating decorative elements, such as those shown in Figure 78.

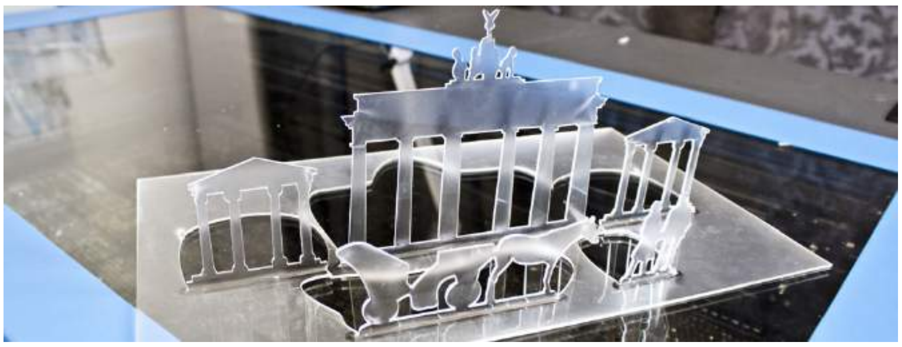

Figure 78: With bending, we can create this decorative city outline.

This bend element only allows bending up to 90°, which limits our designs to 2.5D. We can bend past the vertical axis by holding the workpiece slanted against the desired bend direction (Figure 79).

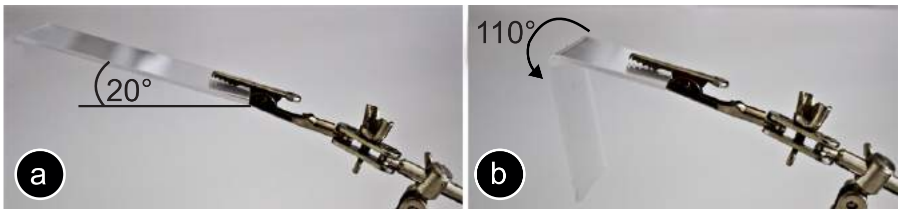

Figure 79: By slanting this workpiece 20° against the direction of gravitational force, we achieve a bending angle of 110°.

A more flexible solution is to bend successively (Figure 80). The purpose of the outer patch is to serve as a lever—once it has done its job we will typically cut it off. The lever approach thus obviously comes at the expense of material.

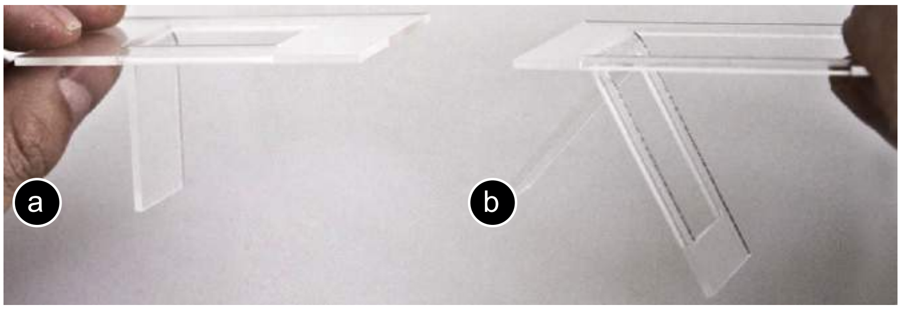

Figure 80: Successive bending allows the inner patch to bend beyond 90°.

A servomotor, finally, allows for any angle by rotating the workpiece repeatedly (Figure 81). A small tab that is part of the workpiece locks into the servomotor. When done, LaserOrigami cuts off the tab, causing the assembled workpiece to drop.

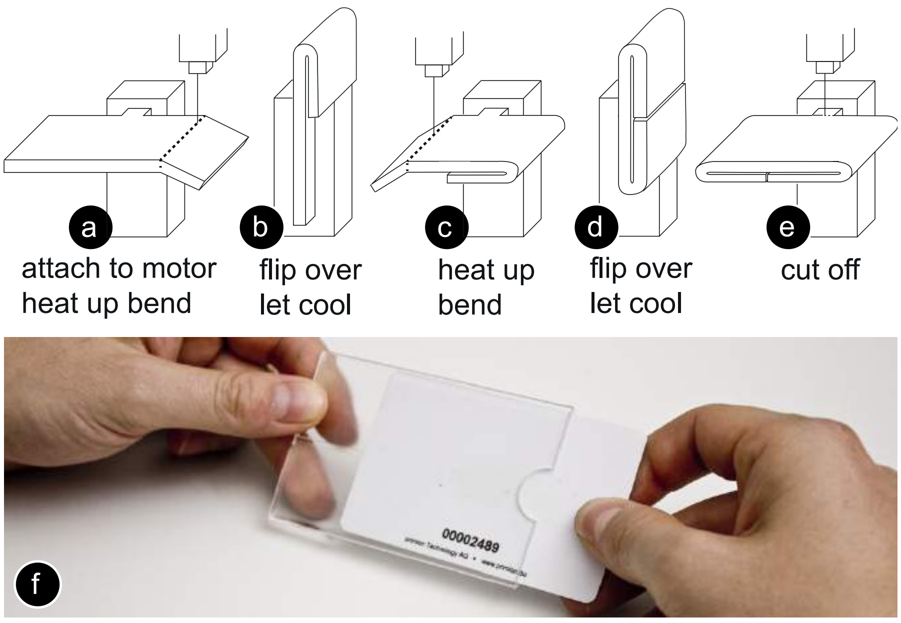

Figure 81: Fabricating a card holder using a servomotor.

#2 Suspenders
Suspenders allow suspending material in a controlled way. They are designed to unfold when heated up with the laser (Figure 82).

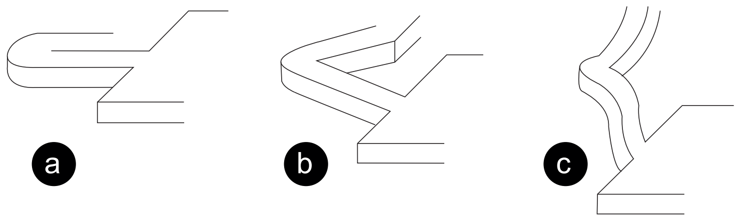

Figure 82: Suspenders: (a) the raw path that is cut. (b) When the laser heats up the suspender, it unfolds until it (c) reaches its final straight shape.

The length of the suspenders defines how deeply the patch will be suspended (Figure 83). The use of three or more suspenders of identical length creates a horizontal patch. While we will most commonly create horizontal patches using suspenders of equal lengths, suspenders of unequal lengths allow us to create ramps.

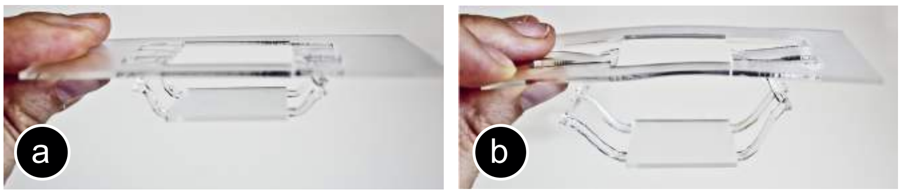

Figure 83: (a) The same patch suspended with a set of short and (b) long suspenders.

Suspending recursively allows us to create 2.5D landscapes from elevation lines (Figure 84). If an upper level gets suspended, it moves all lower levels with it. To keep the next suspension in the plane of the laser cutter, we suspend the inner ring first, working outwards.

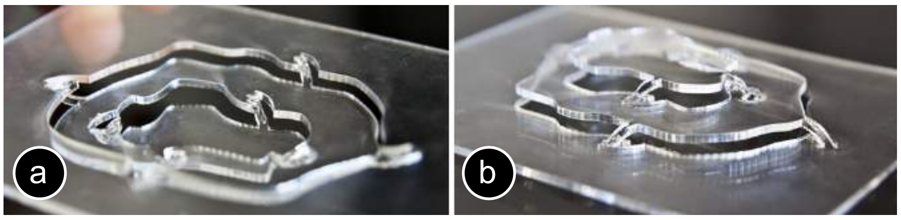

Figure 84: Suspending recursively to create a multi-level terrain.

Creating a suspender creates a hole in the workpiece—one of the limitations of our approach. However, the approach offers freedom in "routing" the suspenders, which allows us to place them so as to minimize interference with the remaining workpiece (Figure 85).

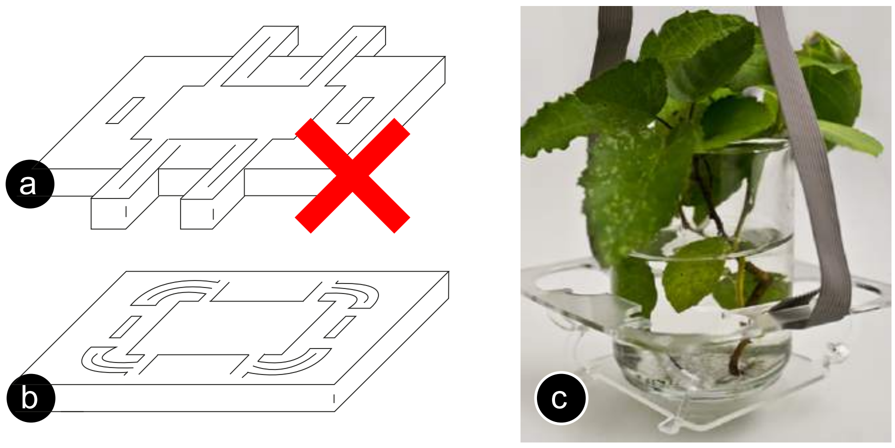

Figure 85: (a) This design does not work because suspenders break the workpiece apart. (b, c) Rerouting the suspenders to minimize interference.

#3 Stretching
Suspenders are our general mechanism for suspending a patch of material. However, in some cases, the material for the suspenders is required, such as for the paint holder shown in Figure 86. In this case, we can suspend by stretching. To suspend a patch by stretching, we heat up its outline until it gets compliant and stretches due to the weight of the suspended patch. If a patch is too light, we can add weights to it before suspending (Figure 86a).

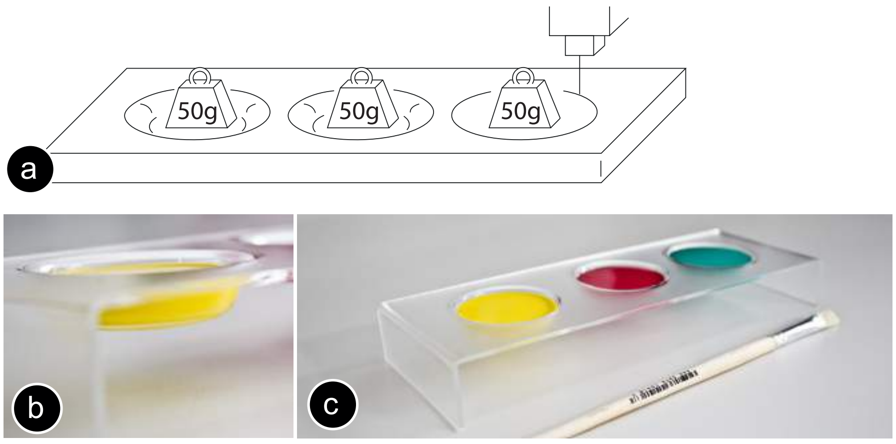

Figure 86: Suspending a patch using stretching creates a container that can hold a liquid.

Stretching causes the walls of the suspended patch to get thinner, which limits the maximum suspension depth. How deep a patch can be suspended by stretching depends on the material thickness as well as the width of the stretched region.

4.2.3 *Interactive laser-cutting with LaserOrigami*

We integrated LaserOrigami into the constructable interactive laser-cutting system (Figure 87). To extend the constructable tool set with the LaserOrigami design elements we turned each of the design elements into a separate laser tool, such as the *bend*, the *suspend*, and the *stretch* tool. To set parameters for these tools, such as the size of the bending angle or the length of the suspenders, we added a numpad that allows users to enter a single global parameter that is passed to the current tool.

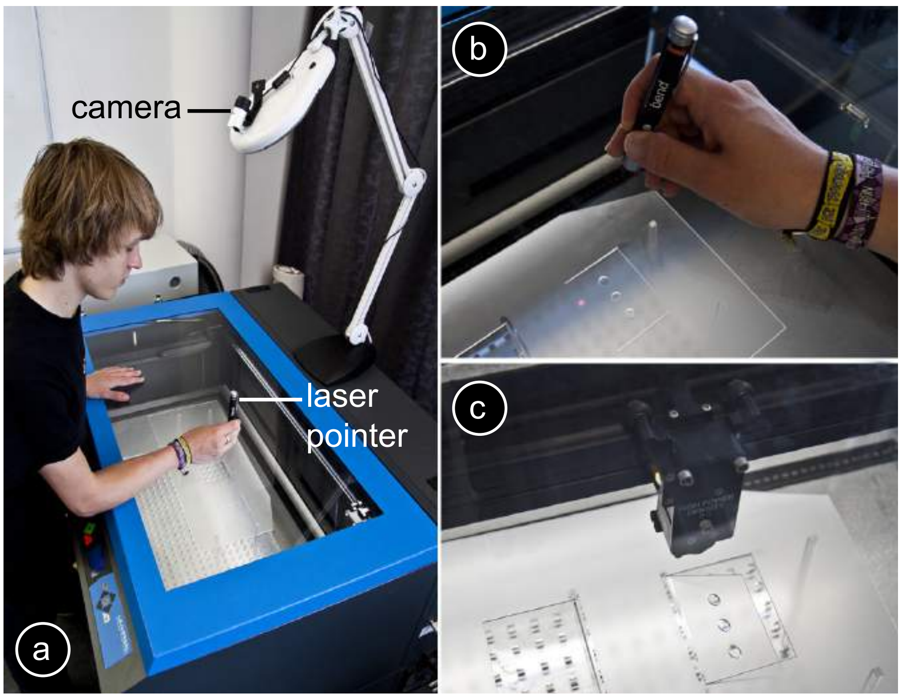

Figure 87: The LaserOrigami extension for constructable.

Figure 88 illustrates how to bend interactively by drawing a stroke across a part of the workpiece using the *bend* tool. A bend can only occur between two cuts, which allows LaserOrigami to compute the bend at the intersection of the user's draw path and the existing cuts in the workpiece. Users can create multiple bends efficiently by crossing using a single long stroke (Figure 88b).

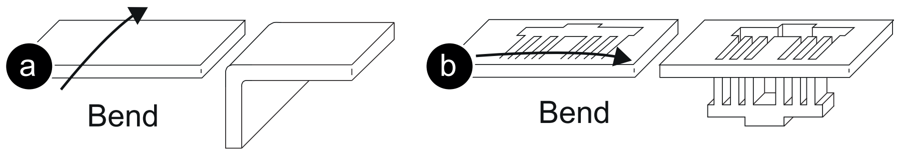

Figure 88: (a) Bending the workpiece by drawing a bend path across. (b) Users can bend multiple parts at once by crossing them all at once.

Interactive LaserOrigami also allows controlling the servomotor by entering the desired angle into the numpad (Figure 89). (a) The user first attaches the workpiece to the motor, (b) then enters the bend angle into numpad and draws the bend path with the *bend* tool to achieve the result shown in (c).

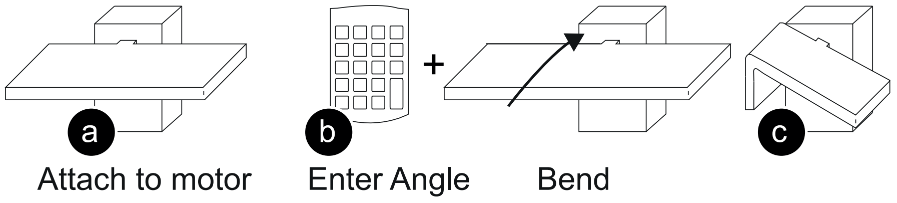

Figure 89: The servomotor allows creating precise bending angles interactively.

Figure 90 demonstrates how to interactively construct a plant holder using the *suspend* tool by (a) drawing the base plate using the *polyline* tool as *sketchline*, (b) creating the top-level surface using constructable's *scale* tool with depth as a numpad parameter, and (c) suspending the base plate using the *suspend* tool.

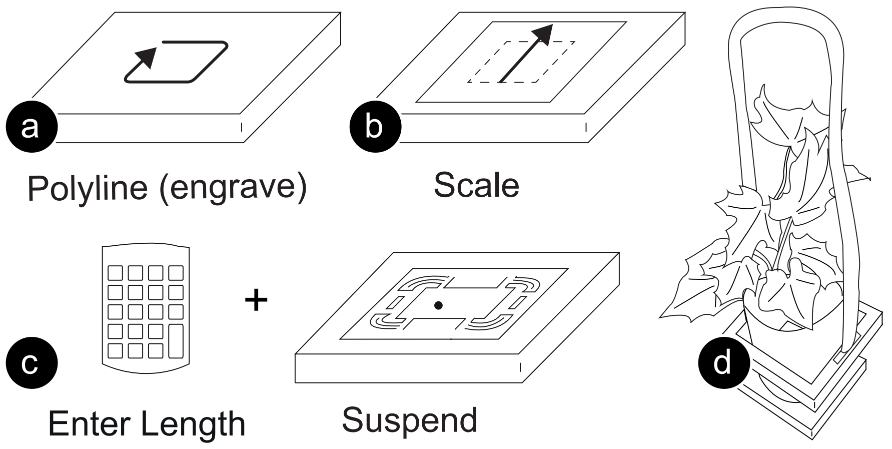

Figure 90: Creating a plant holder interactively.

4.2.4 *Implementation*

The LaserOrigami tools encode all the "instructions" that the laser cutter requires in order to fabricate the respective shape, i.e., the lines that cut and the lines that implement the back-and-forth motion of the defocused laser for bending (Figure 91)

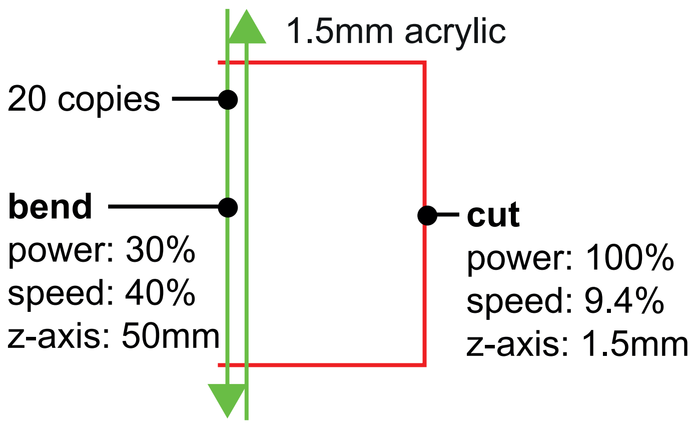

Figure 91: These lines encode a 10cm bend for 1.5mm acrylic.

#1 *Switching between cutting and bending*
Moving the table up and down, is encoded in the line colors. As an example, Figure 91 shows the lines that implement a simple bend. In the configuration dialog of our cutter, we configured red lines to mean cutting, i.e., whenever the laser encounters a red line the table will move the workpiece into focus. In contrast, we configured green lines so as to move the table down, causing the laser to go into defocused mode and heating up the material for bending. The property we manipulate here is called *z-axis* for our *ULS PLS6.150D* laser cutter. It is normally used to move materials of different thicknesses into focus; with LaserOrigami we instead use it to defocus.

#2 *Execution order*
To make sure all features are executed in proper order, the tools define the stacking order of all lines before sending the drawing document to the laser cutter. Disabling the *Vector Optimizer* feature in our laser cutter makes sure that line order is maintained during cutting. Also, since this cutter model always executes all lines of one color before moving on to the next color, we use a new line color for each group of cuts or bends.

#3 *Power and speed settings*
We use the laser cutter's configuration file to encapsulate the power and speed settings for cutting and bending. For our *PLS6.150D* laser cutter with a 2.0" lens and 1.5mm thick acrylic, for example, we defocus the laser by 50 mm and use 40% speed and 30% power and we move the laser 40 times across each bend and 6 times across each inflection point of each suspender.

#4 *Heating path*
To maximize the time the laser is actually running and minimize the time elements cool off, we create pairs of bend lines of opposite orientation. For the same reason, we perform multiple bend lines on a

single suspender before heating up the other suspenders in the same group. To compensate for heat loss along edges, we make bend lines protrude past object outlines.

#5 Inflection points
To make sure all elements unfold properly, we continue to heat all inflection points past the moment when an object is starting to bend. In the case of suspenders, the inflection point in the center moves not only vertically, but also horizontally while dropping (see Figure 82). We compensate for this horizontal movement by creating additional bend lines located across the different horizontal positions of the dropping inflection point.

#6 Calibrating laser and cutting bed
The *PLS6.150D* laser points down at an angle, which introduces a horizontal offset during defocusing. We compensate for the offset by shifting bend lines accordingly.

#7 Keeping the workpiece stable
Cutting the entire outline of the workpiece causes it to wiggle, which renders subsequent steps imprecise. To keep the workpiece stable, we keep it attached to the main sheet until all bending is done (Figure 92).

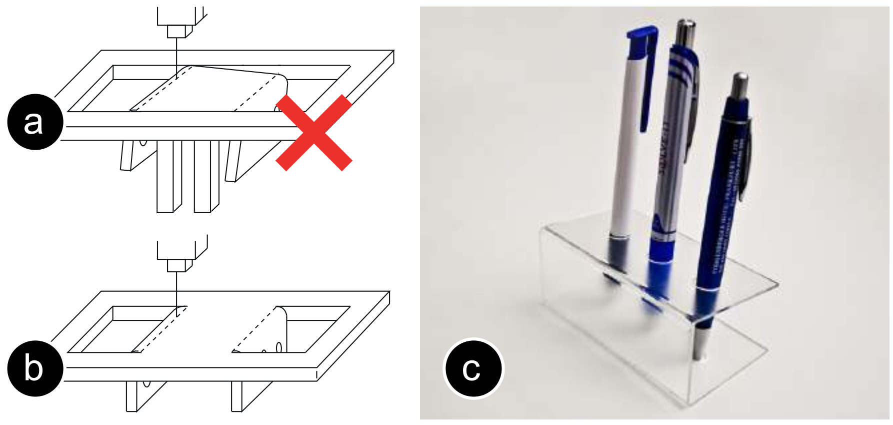

Figure 92: (a) Wiggling. (b) To keep the workpiece stable, we keep it attached to the main sheet until all bending is done. (c) The result.

#8 Preventing the workpiece from getting stuck
During bending, the diagonal of the bent piece has to pass the opening. As shown in Figure 93a, this can cause the bent piece to get stuck in the adjacent material, especially if the bent patch is small and thus light or if it has an irregular outline. We address this by first cutting out an extended outline, which creates additional space around the workpiece as illustrated by Figure 93b.

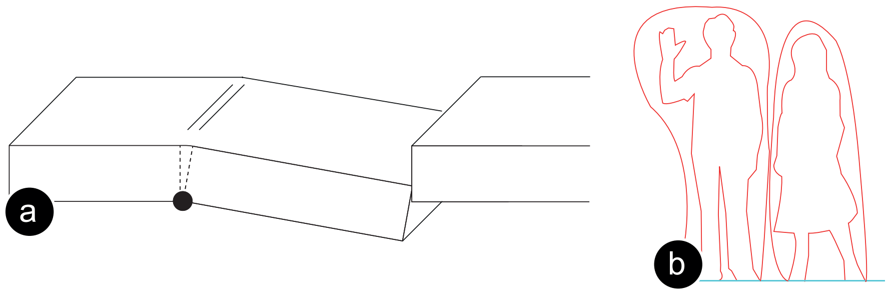

Figure 93: (a) During bending, the part that bends can get stuck in the surrounding material. (b) We address this by first removing some material around the cut.

4.2.5 *Conclusion*

While constructable and LaserOrigami allow for physical feedback after every editing step, the interaction is still best described as turn-taking, i.e. users first create an input, and afterwards the system provides physical output, thereby causing a visible lag. For instance, while drawing the outline of the wooden booklet, constructable users first have to complete the entire rectangle before seeing the outline being cut. However, to decide on the best shape for the wooden booklet, it would be beneficial if users would see the workpiece change in real-time so that no visible lag between user input and system output exists. In the next chapter, we explore how to overcome this limitation with a system that allows for such direct control. In addition, this system will expand the range of shapes that users can fabricate by going beyond objects that are created through cutting and folding a sheet.

5 FEATURE-LEVEL: DIRECT MANIPULATION

By decreasing the interaction unit even further to a single *feature*, we can achieve continuous physical feedback, i.e. with one interaction users can explore the entire space of shapes, potentially leading to a better result in less time. This is in contrast to the turn-taking systems shown in the previous chapter that consist of two discrete steps, i.e., users first create an input, and then the system responds with physical output, which only allows to explore one option per turn.

5.1 FORMFAB: CONTINUOUS FEEDBACK

Similar to LaserOrigami, FormFab is a system that reshapes the workpiece to provide physical feedback (Figure94): A heat gun attached to a robotic arm warms up a thermoplastic sheet until it becomes compliant; users then interactively control a pneumatic system that applies either pressure or vacuum thereby pushing the material outwards or pulling it inwards. As users interact, they see the workpiece change continuously and in real-time.

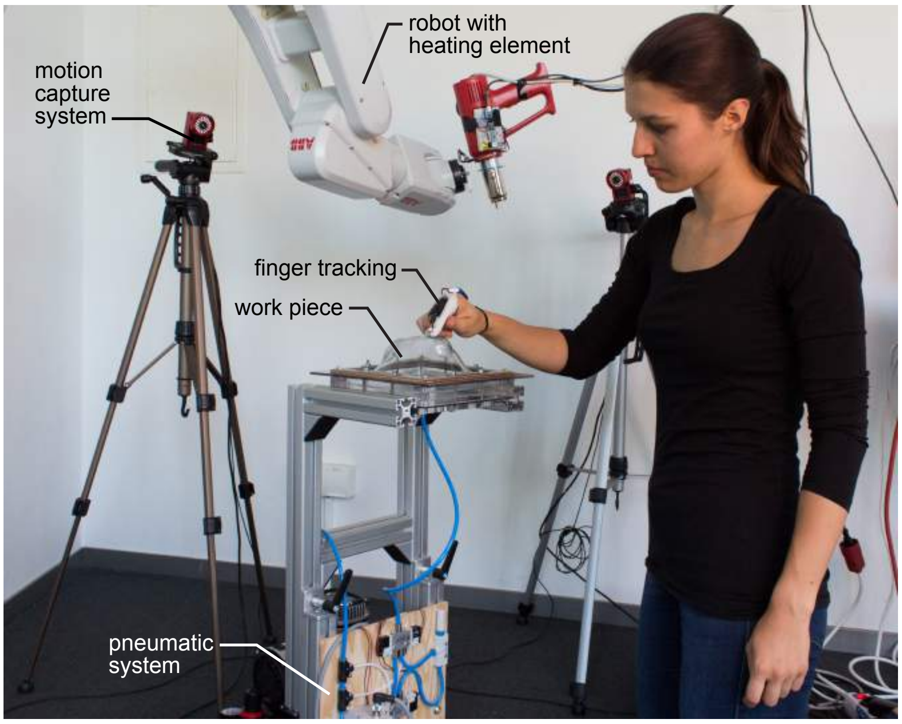

Figure 94: FormFab allows users to interactively control a pneumatic system, thereby changing the shape of the workpiece in real-time.

5.1.1 *User Interaction*

FormFab's user interaction consists of two steps. In the first step (Figure 95), users use their index finger to draw an outline of the area they want to reshape onto the workpiece. When the user removes the hand from the workpiece, the path is beautified by our software. The robotic arm then starts warming up the area using a heat gun.

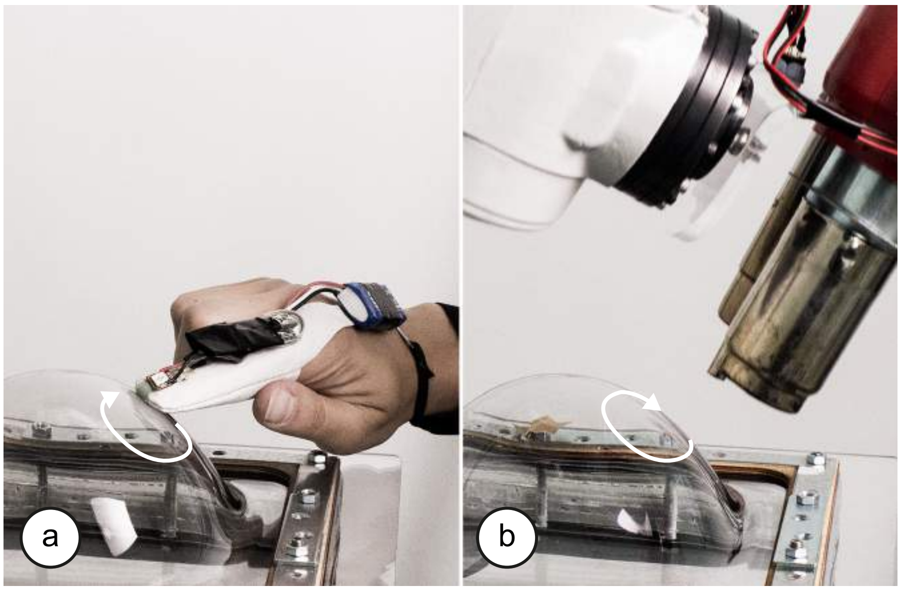

Figure 95: (a) The user draws an outline, which is then (b) heated up by a robotic arm that carries a heat gun.

After the material has reached its compliance point, the robotic arm moves out of the way and the user starts the second step (Figure 96): By performing a pinch gesture above the workpiece, the user activates the pneumatic system. If the user's hand moves away from the workpiece while holding the pinch gesture, the pneumatic system increases the air pressure and the user sees the compliant area inflate continuously and in real-time. If the user's hand moves back towards the workpiece, the pneumatic system reduces the pressure and the user sees the compliant area deflate. Figure 96 shows one continuous interaction, i.e., the same part is first pulled out and then pushed in.

While step 1 of the user interaction, i.e., drawing the outline, still follows the turn-taking interaction model, step 2, i.e., defining the extrusion amount provides continuous physical feedback: input by the user and output by the fabrication device are coupled tightly and without lag allowing for continuous, real-time interaction.

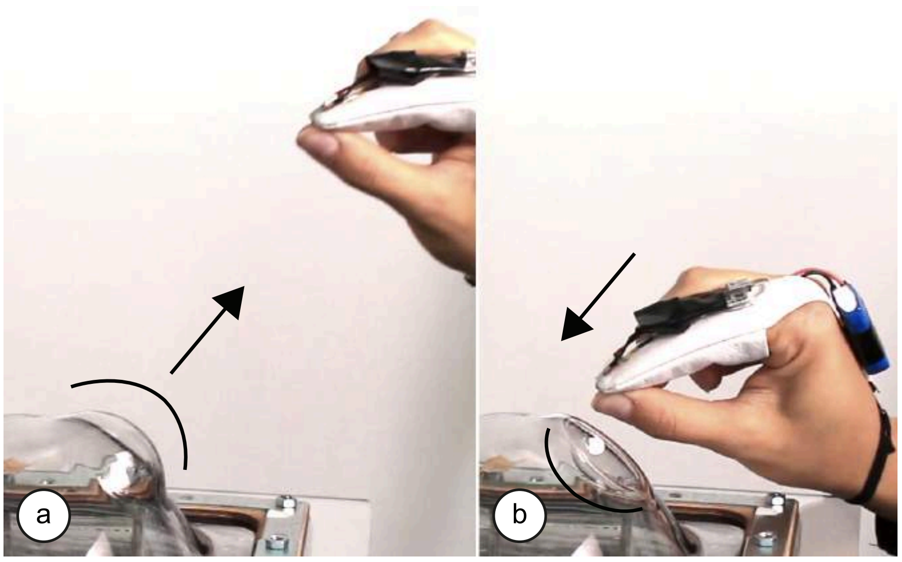

Figure 96: (a) Pulling causes the system to inflate the part by increasing air pressure, (b) pushing deflates the part by reducing air pressure.

5.1.2 *Walkthrough: Making the sculpture of a teddy head*

Figure 97 shows a teddy head sculpture we created with FormFab. FormFab is particularly suitable for making round shapes with smooth surfaces, which are hard to make with additive or subtractive fabrication due to limited resolution of the tools.

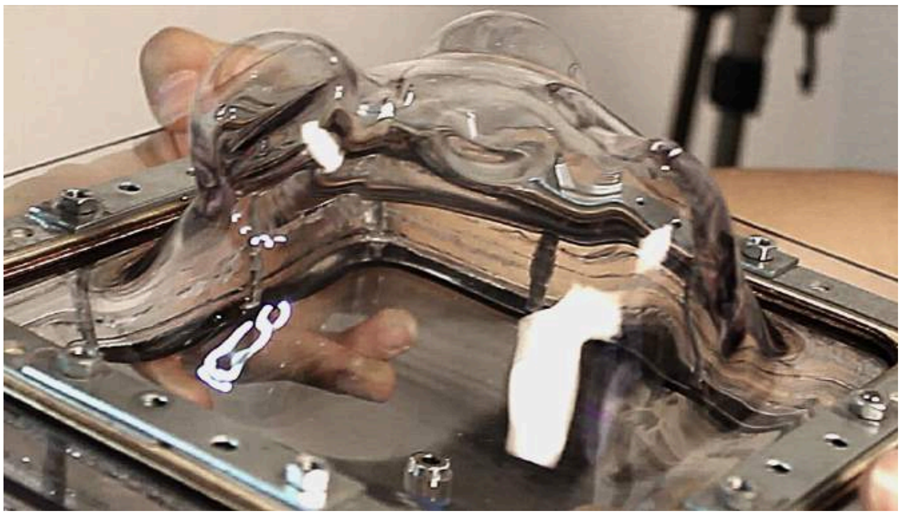

Figure 97: The sculpture of a teddy bear's head made with FormFab.

We start by defining the shape of the head, as shown in Figure 98. We first draw a circle onto the flat workpiece. After the robot has heated up the area, we use a pinch gesture to extrude the head.

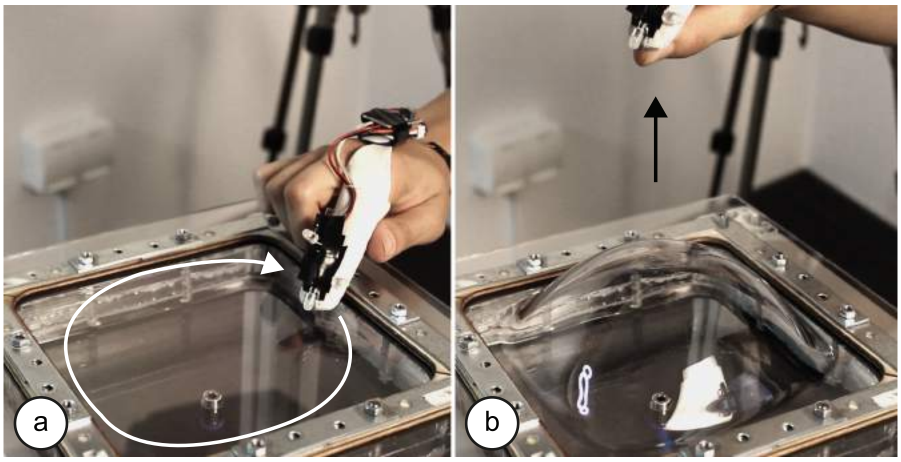

Figure 98: Head: (a) draw outline, (b) extrude.

Using the same interaction steps, we add the ears to the head of the teddy (Figure 99).

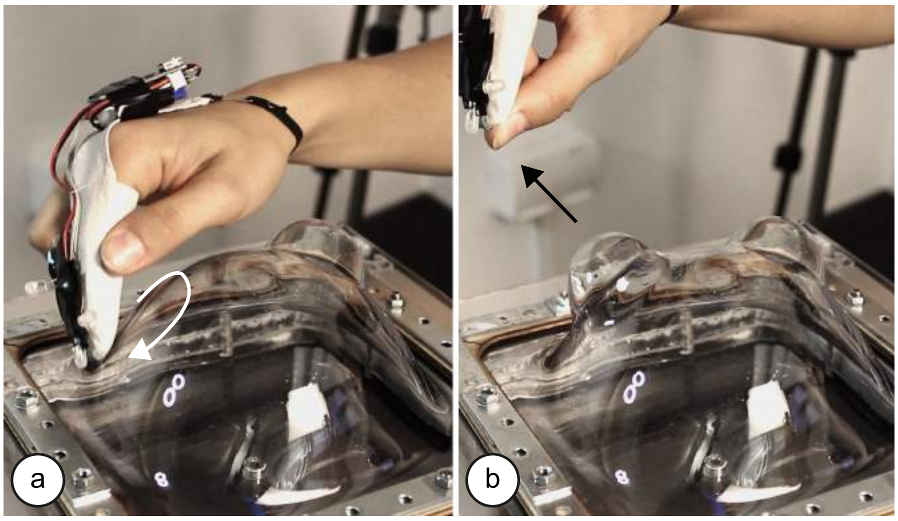

Figure 99: Adding the ears on the right and left side.

We now add the snout (Figure 100). To explore the best size, we repeatedly scale the snout by moving our hand to different distances from the workpiece. After defining the final shape of the snout, we add a tip on top of it.

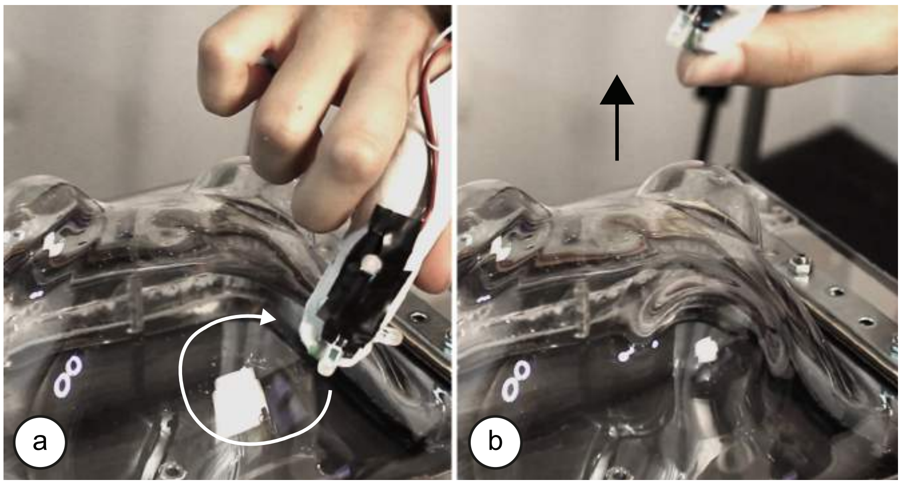

Figure 100: Adding the snout.

In the last step, we create the eyes (Figure 101). This time, we move our hand towards the workpiece, thereby applying vacuum that extrudes the eyes inwards.

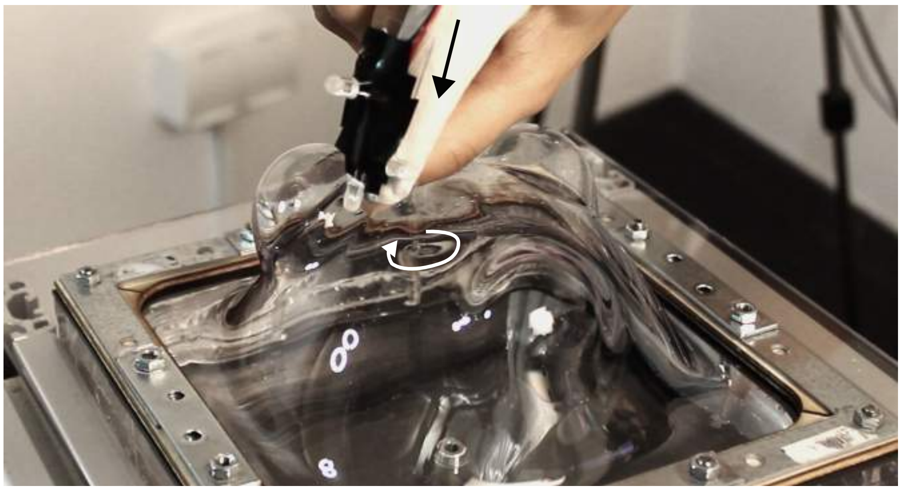

Figure 101: Extruding the eyes inward with vacuum.

Figure 97 shows the final teddy head, which consists of the seven steps illustrated above (head, ears, snout, snout tip, eyes). Making the teddy head took 9 minutes including the time for heating and the time the material required to become rigid again after each interaction. The thicker the sheet, the more time the user has for reshaping as it retains the heat longer (the sheet we started with was 4 mm thick). For instance, when reshaping the basic head we made changes for more than 15 seconds, and later on when creating the right ear, we explored different sizes for around 30 seconds.

At every step in the modeling process, we were able to find the best shape by continuously browsing the space of different options in real-time.

5.1.3 *Hardware components*

Figure 94 shows the hardware setup of our system that enabled the above walkthrough. It consists of two main parts: (1) the user interaction tracking with a motion capture system, and (2) the formative fabrication unit (robotic arm, heat gun, and pneumatic control system).

#1 Tracking the user interaction
To track the user interaction, FormFab users wear a motion capture marker and a pressure sensor on their index finger (Figure 102a). The marker is detected by a motion capture system *(Optitrack)* and used to determine where the user is interacting on the workpiece. The pressure sensor is used to determine the beginning and end of the interaction. We can do this because in both interactions users apply pressure to the index finger either by touching the workpiece (Figure 102b) or by pressing the index finger to the thumb in the pinch gesture (Figure 102c).

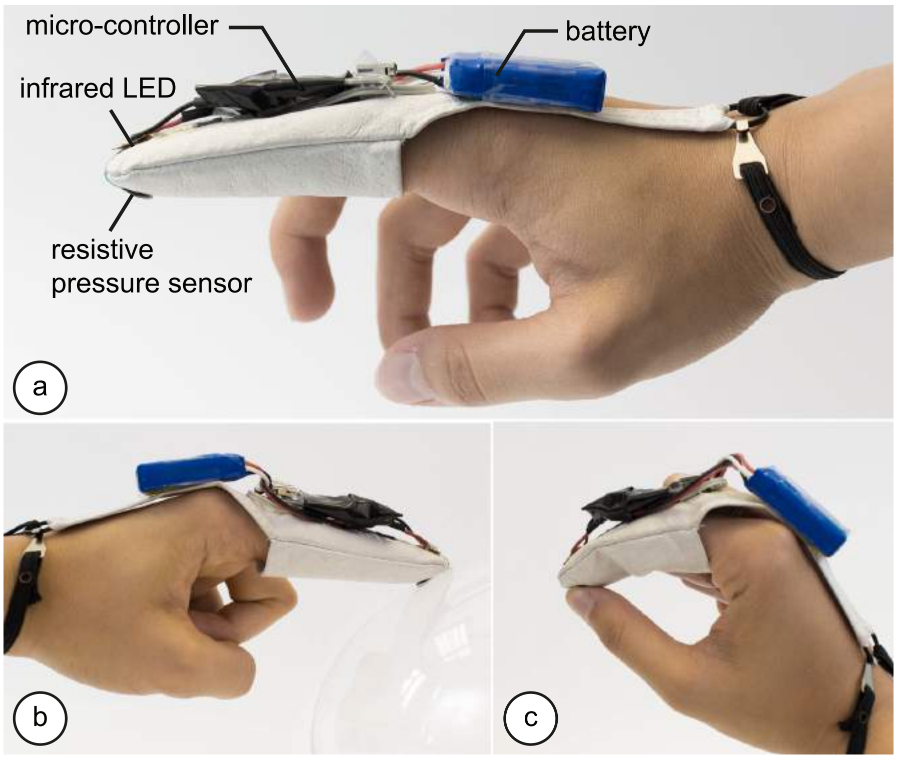

Figure 102: (a) Tracking unit. We detect start/end of both interactions (b) the drawing and (c) the pinch gesture with a pressure sensor at the fingertip.

Instead of using a passive retro-reflective marker, we use an active infrared LED marker that is turned on whenever the user applies pressure to the finger as described above (an *Arduino Nano* connected to the pressure sensor processes the pressure values, then activates the LED). We are using an active marker because detecting passive markers requires the motion capture cameras to emit infrared light. The light causes reflections on our transparent and reflecting workpiece, which makes it difficult to detect the marker. By using the active marker, we were able to use the cameras with their infrared emission set to 'off'.

#2 Robotic arm
We use a 6-axis robotic arm *(ABB IRB 120)* to move the heat gun that warms up the workpiece. The six degrees of freedom allow the robotic arm to reach around the workpiece from all sides. In contrast to previous systems that achieve higher degrees of freedom by rotating the workpiece, the robotic arm allows us to keep the workpiece stationary, thereby preserving the user's spatial frame of reference during modeling. By mounting the robotic arm to the ceiling, we were able to maximize the reachable area while also allowing the user to freely move around the workpiece.

#3 Heat gun
We heat the workpiece using heat guns that blow air onto the workpiece (Figure 103). We chose air for heating as it best distributes on surfaces of arbitrary geometry. A rigid array of radiator elements, in contrast, would lead to uneven heating as some points are located closer to the workpiece surface than others. Other contactless methods, such as an infrared-light array were not powerful enough to heat the thermoplastic sheet.

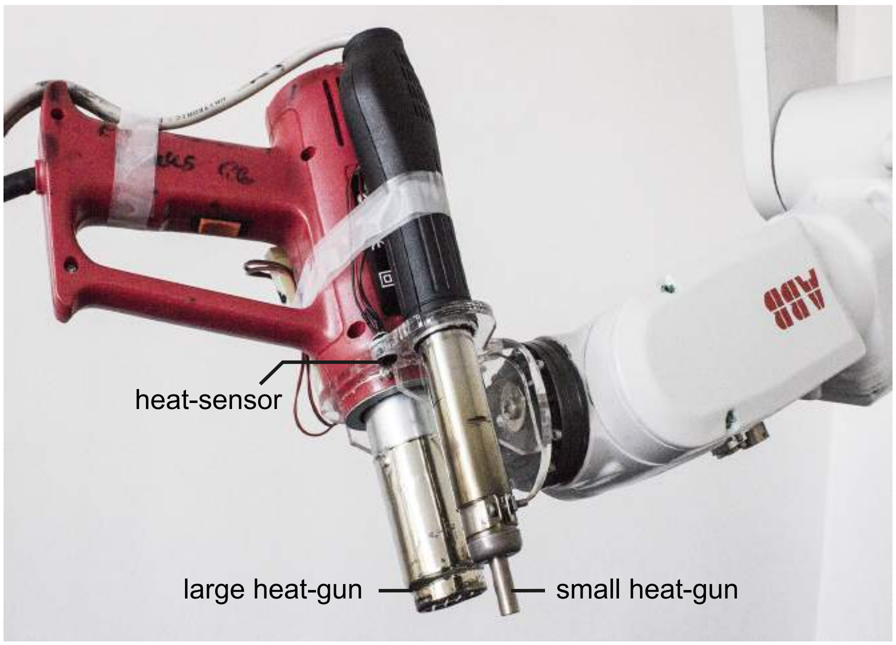

Figure 103: The larger heat gun allows heating coarse base shapes. We use the small heat gut for finer details. The temperature sensor notifies FormFab when the material reaches its compliance point.

We use two heat guns of different sizes. We use the larger heat gun (3.5cm, 600°C, 2000W) for the first steps in the modeling process when the coarse base shape is created. We use the smaller heat gun (0.7cm, 500°C, 1000W) for the later steps in the modeling process when the user creates details. Both heat guns have a different working range; we determined the values experimentally and use them to optimize the heat transfer while preventing to overheat the material.

Since the heat transfer is influenced by several factors that are hard to predict, we added a temperature sensor *(Melexis MLX90620)* to the robotic arm to determine the temperature of the workpiece (Figure 103). The sensor is wired to an *Arduino Nano* that informs the FormFab main application about the current temperature values.

The temperature sensor is passive and measures the infrared radiation emitted by the material when it warms up. Active sensors, such as an infrared laser thermometer, that first emit light and then measure its reflection did not work: Since the workpiece is reflective, the signal reflection depends on the entering angle of the light beam, therefore the measured signal is no indication of the material temperature.

#4 Pneumatic control system

Figure 104 shows the pneumatic control system. FormFab uses a compressor *(Universal II Profi-Airbrush)* that compresses air up to 5 bar. Connected to the compressor is a regulation unit *(Festo)* that defines how much of this pressure is forwarded to the pressure/vacuum system and onto the workpiece. We use 0.1 bar 0.5 bar, depending on the size of the compliant area.

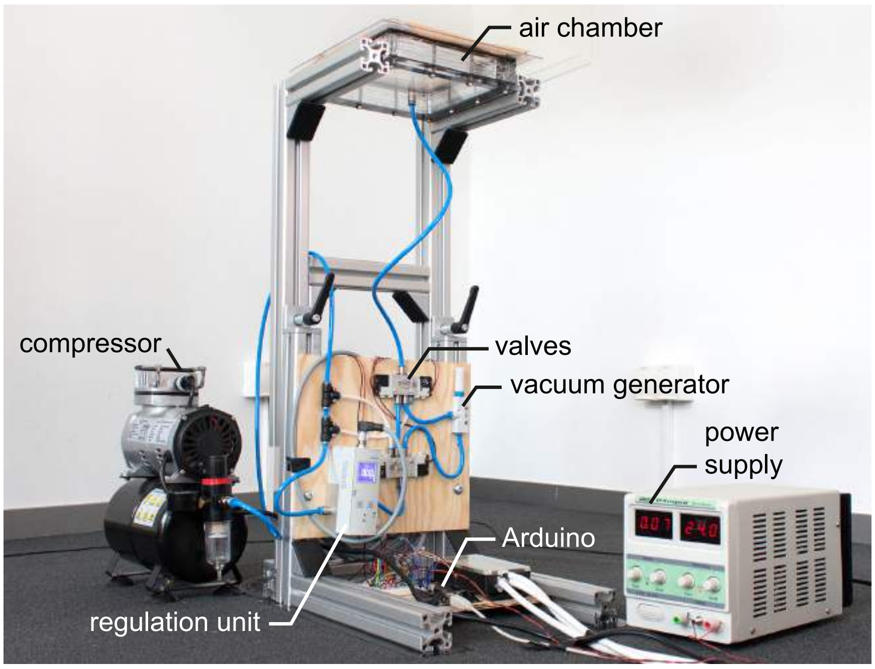

Figure 104: The pneumatic system.

To set the pressure value, the regulation unit expects a control voltage from 0-10 V. FormFab sets the voltage using an *Arduino Uno* connected to a digital potentiometer that regulates a 24V external power supply.

The pressurized air is then guided through different valves, depending on whether pressure or vacuum should be applied to the workpiece. If pressure is applied, the pressurized air is directly forwarded to the air chamber to reshape the workpiece. If vacuum is to be applied, the pressurized air instead goes through a vacuum generator, which creates negative pressure in the air chamber, pulling the compliant workpiece inwards.

To switch between the two states, the pneumatic system has to change its valve configuration, which we again control through the *Arduino Uno.*

After the user finished the interaction, our system keeps the air chamber under pressure until the workpiece has cooled down and has become rigid again. Lowering the pressure earlier would cause the workpiece to sag.

After the workpiece has cooled down, our system opens a valve to neutralize the air chamber, i.e., it releases any excess pressure. This is necessary because if the air chamber was still pressurized at the beginning of the next heating step, the workpiece would start to deform as soon as it becomes compliant, i.e., even without the user interacting.

5.1.4 *FormFab's processing pipeline*

The software pipeline consists of the following processing steps: (1) tracking the outline the user is drawing, (2) calculating the heating path, (3) generating movement commands for the robotic arm, (4) monitoring the temperature to achieve compliance and to avoid overheating, and (5) applying air pressure via the pneumatic system according to user input.

#1 Tracking the user interaction
The active infrared LED marker on the user's fingertip is detected by *OptiTrack's* motion capture software *Motive.* Since the infrared LED can cause reflections on the transparent and reflecting workpiece, we filter reflections that do not fit the LED's size and circularity.

Motive streams the position data of the marker to the FormFab main application, which implements a client based on the *NatNet SDK* from *OptiTrack.* After receiving the motion data, FormFab applies additional filtering in case the previous step did not eliminate all reflections: For instance, in case two markers should be detected instead of just one, the system only considers the one located closer to the previously detected position.

As soon as the user finishes drawing, the marker position is processed using *OpenCV's* shape detection algorithms, including the fitEllipse() method, which FormFab uses to fit a circle to the users drawing.

#2 Calculating the heating path
Based on the user's drawing, our software generates a heating path with the goal to heat up the area equally at all locations. Since calculating the heating path for an even heat distribution is difficult for arbitrary shapes, we limit FormFab's modeling geometries to circular shapes.

Depending on the size of the circular area, we either only heat up the outline or apply a spiral path that moves from the outline towards the center of the circle. We determine which one to use based on the heat gun radius, the distance to the workpiece, and the amount of airflow.

Those factors also influence how much we shift the heating path inwards to avoid heating areas outside the drawn area. Similarly, when creating a spiral path, we use these factors to adjust the distance between two lines: if the lines are too close, the area will overheat, if lines are too far apart, the area will not heat up sufficiently (Figure 105).

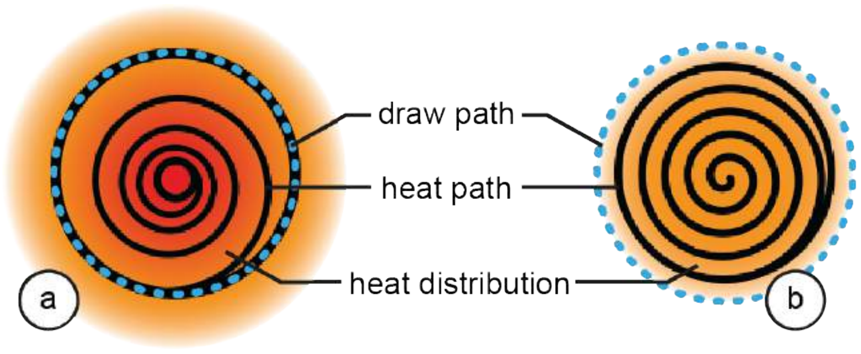

Figure 105: Calculating the heat path to achieve an even heat distribution for this circular shape.

#3 Robot movement
We installed the *open-abb-driver* as a server on our robotic arm. This allows us to send real-time commands via a TCP/IP connection. It also allows us to receive information about the robotic arm's current state.

The server processes commands in the robotic programming language *RAPID* that tells the robotic arm how to position the joints and how fast to transition between different positions. We generate these commands using the *Grasshopper* plugin *Robots.IO.*

The *Robots.IO* plugin takes a robot model (in our case *ABB IRB 120)* and a motion path (our heating path) as input. It then calculates an optimal configuration for the robotic arm to move along the motion path (Figure 106).

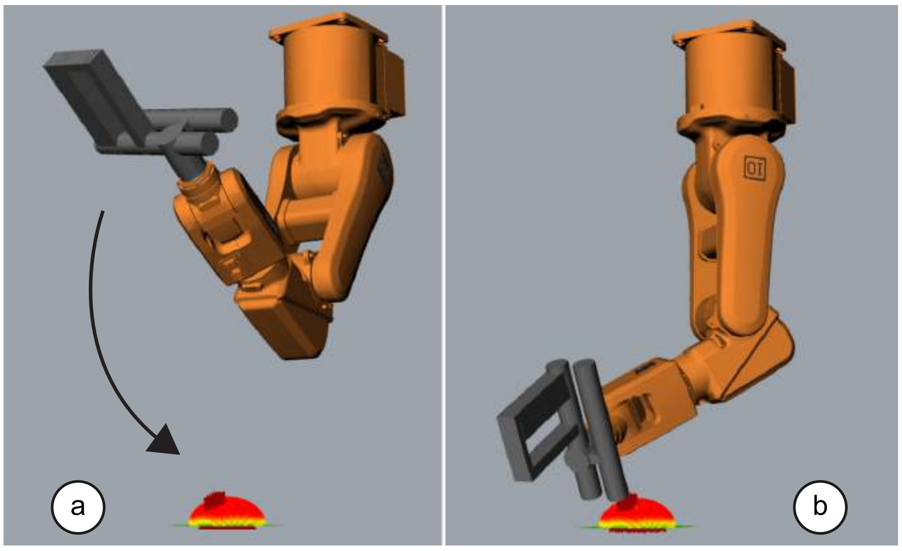

Figure 106: (a) Here our software calculates a working path for the robotic arm, while (b) keeping the tool always aligned with the workpiece.

Robots.IO also generates warnings in case a target position is located outside the working range of the robotic arm (e.g., because it exceeds a joint limit), which typically happens when the shape becomes too large and the robotic arm has to operate at the boundaries of its working volume.

Robots.IO generates a different type of warning in situations where the robotic arm would collide with the workpiece. To allow *Robots.IO* to detect this case, the system has to provide it with a digital representation of the workpiece. Instead of 3D scanning the workpiece after every change, which would delay the next interaction step, FormFab approximates the 3D model by simulating each modification applied to the workpiece.

For this, FormFab uses the *Grasshopper* plug-in *Kangaroo Physics*, which is a physical simulation environment. First, our system uses the heat path and heat gun radius to define the virtual compliant area. It then adjusts the virtual air pressure until the compliant area expands far enough so as to overlap with the next steps drawn outline. We can do this because users always draw the next outline before the robotic arm moves again.

Our system also uses the resulting digital representation of the workpiece to orient the heat gun so as to always point at the workpiece at a perpendicular angle, allowing the system to heat the workpiece evenly.

#4 Heat control loop
Robots.IO exports the *RAPID* code for the entire heating path into a .txt file. Uploading this code at once to the robotic arm would execute the path, but would not allow querying information about the robotic arm during that time. However, FormFab needs information about

the robotic arm's current position, for instance, when it evaluates the current temperature data from the heat sensor as it otherwise does not know to where on the workpiece the measurement belongs. Our software thus splits the *RAPID* code into single motion commands and sends them one-by-one to the server.

Once the heat-sensor has determined that the material has reached its compliance point, FormFab stops the heating process and moves the robotic arm to its default position, i.e., out of the way of the user.

#5 Pneumatic control
Before turning on the pneumatic system, our software detects the direction of user input. When the user first performs a pinch gesture, our system sets the position of the user's hand as the origin from which to measure. For every 3 mm of hand movement, our system increases or decreases the control voltage of the pneumatic regulation unit by one step, slightly increasing or decreasing the pressure inside the workpiece.

The exact pressure depends on the size of the compliant area. Pushing out smaller areas requires substantially higher pressure than pushing out larger areas. The reason for this is that the force available to push out a compliant area is the amount of pressure times the size of the area. FormFab compensates for this by applying a factor depending on the radius of the compliant area.

FormFab keeps track of the number of increments and decrements sent to the pneumatic regulation unit. This allows it to switch the valve from vacuum to pressure and back at the right moment and to neutralize the air chamber to zero bar, when needed.

5.1.5 *Conclusion*

By decreasing the interaction unit to a single *feature*, such as the extrusion amount of a circular area, we were able to demonstrate how to move from turn-taking interfaces that allow exploring one option at a time to direct manipulation interfaces that allow browsing the entire space of options with a single interaction. Our system has several limitations, such as that it cannot create sharp features and users only have a limited amount of time to reshape the workpiece before it cools down and gets stiff again. However, future work can reduce or even eliminate these limitations: For instance, a shape display mounted onto the robotic arm can serve as a dynamic stencil thereby producing a variety of shapes. In addition, by preheating the air inside the chamber close to the compliance point of the material, we can keep the workpiece warm, thereby speeding up the time it takes to make an area on the workpiece compliant while also extending the time users have to reshape the workpiece.

6 DISCUSSION AND CONCLUSIONS

In this thesis, we argued that by repeating the evolution of the interaction model from personal computing, we will see the same benefits for personal fabrication.

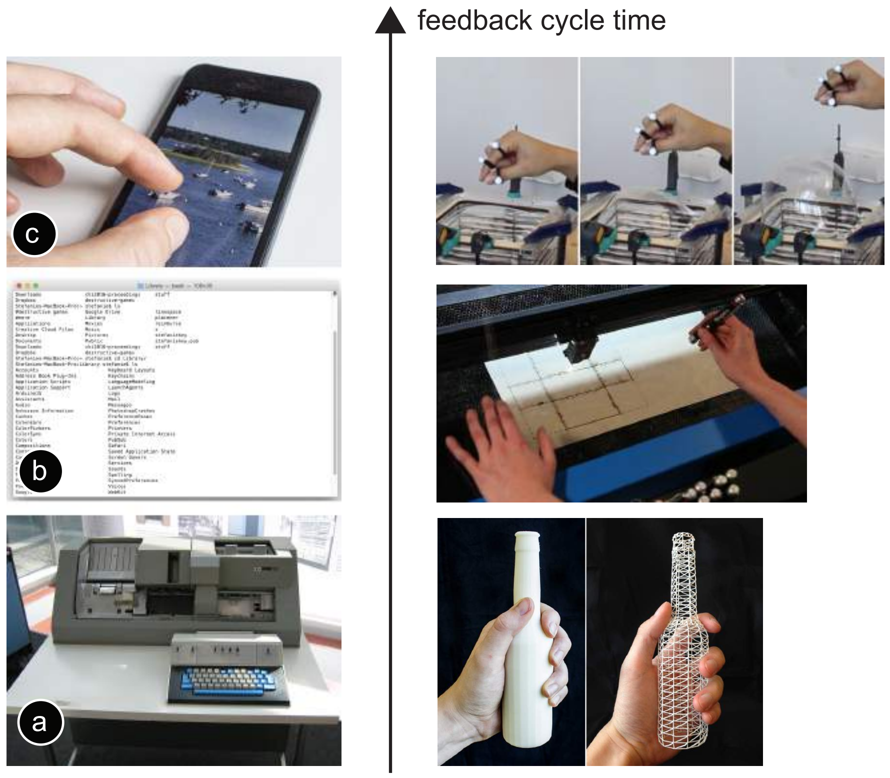

Figure 107: Following the evolution of personal computing, we developed systems that (a) fabricate objects quickly in one go, (b) provide physical feedback after every editing step, and (c) allow users to change the shape of an object in real-time.

Figure 107 illustrates how we explored interfaces for personal fabrication that follow the evolution of personal computing. (a) We started with fabricating objects in one go and investigated how to tighten the feedback-cycle on an *object-level*. We showed how our method low-fidelity fabrication saves up to 90% printing time while allowing users to focus on different key aspects, such as modularity (faBrickator), shape (WirePrint), and function (Platener). (b) Next, we decreased the interaction unit to a single *element* of an object to explore what it means to transition from systems that fabricate in one go to turn-taking systems for personal fabrication. We showed two systems, constructable and LaserOrigami, that demonstrated how users can interactively create objects while receiving physical feedback after every editing step. (c) Finally, by decreasing the interaction unit even further to a single *feature,* we showed how to achieve real-time physical feedback, thereby moving from turn-taking interfaces towards direct manipulation. We demonstrated how our system FormFab allows users to create input to the system while immediately receiving physical feedback.

6.1 LIMITATIONS OF DIRECT MANIPULATION INTERFACES

While direct manipulation systems for personal fabrication extend the range of problems novice users can tackle, they are subject to the same limitations as those for personal computing: While they are useful for some design problems, they are less so for others. As Norman et al. [39] point out, direct manipulation interfaces are limited to operations that can be done on 'visible objects' and have 'difficulties handling variables' and 'distinguishing an individual element from a representation of a set or class of elements'. Thus, design problems that require more abstract thinking for which users have to sit down with a piece of paper first and make a detailed plan ahead, are better handled with traditional digital 3D editing. In addition, our systems are inherently scale 1:1 and do not offer a way of inspecting a detail in magnification, which limits users to projects that fit a particular scale. The same way that saw and wood chisel cannot replace a detailed design process, our systems cannot replace a complex 3D editing tool for trained engineers.

6.2 TECHNICAL ADVANCEMENTS IN PERSONAL FABRICATION

For our research, we focused on the most common technologies available today: For 3D printing, we used printers that work by extruding plastic through a hot extruder nozzle and position it voxel-by-voxel on a build plate (so called *Fused-Deposition Modeling),* such as the *PrintrBot*, *Kossel Mini*, and *MakerBot*. For laser cutting, we used CO2 laser cutters, which are the technology becoming available for low-cost right now (e.g. *Glowforge* laser cutter).

Since improving fabrication technology itself is a topic that falls into the realm of mechanical engineers and less into human-computer interaction, we looked at how we can use technology today to explore interaction paradigms that will become possible in the future when fabrication actually gets faster. In the last years, we are rapidly advancing towards such a future: The recently introduced 3D printer *Carbon3D* [19], for instance, speeds up fabrication by up to 200x.

In addition, while there is little data today that could prove a Moore's law for 3D printers, 3D systems executive Merrill Lynch stated in his

2014 keynote that 3D printing speed for their products on average has doubled every 24 months over the last ten years [52].

If such a trend should materialize, it is not far fetched to assume that fabrication technology will be able to provide feedback even for large high-resolution objects within seconds or event in real-time, thereby enabling a future in which digital displays will be be replaced with physical displays that transform virtual reality into actual physical reality.

6.3 FUTURE OUTLOOK: UPCOMING CHALLENGES

We conclude this dissertation by extrapolating the current evolution into a future in which large numbers of people use the new technology to create objects. We see two additional challenges on the horizon: sustainability and intellectual property. We demonstrate one prototype each to show how novel interactive systems can help tackle the challenge.

6.3.1 *Sustainability: Patching Physical Objects*

The current process with personal fabrication tools is that once an object has been fabricated with a 3D printer, it cannot be changed anymore. Any change requires printing a new version from scratch, an unnecessary and wasteful process.

With *Patching Physical Objects* we propose a more sustainable approach: instead of re-printing the entire object from scratch, we suggest patching the existing object. We built a system on top of a 3D printer that accomplishes this (Figure 108).

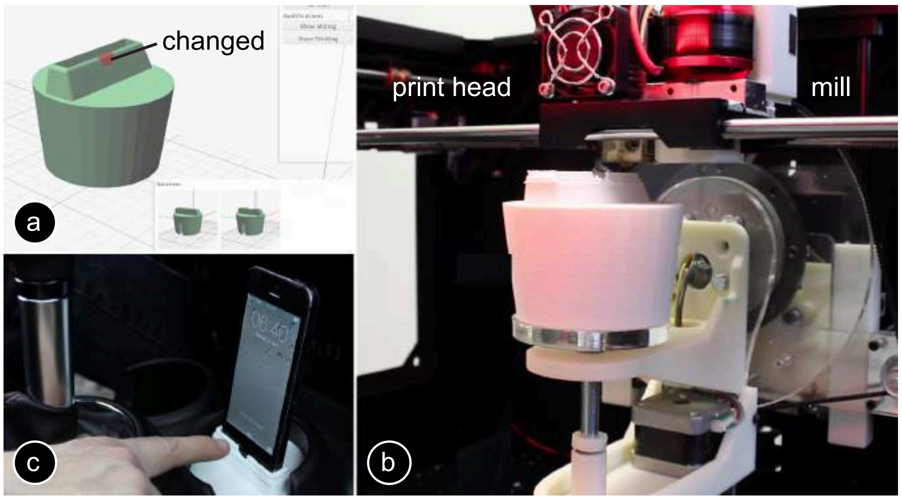

Figure 108: (a) computing which part changes, (b) patching the part with mill and print head, (c) the patched object.

To patch a physical object, users mount the existing object into the 3D printer, then load both the original and the modified 3D model into our software, which in turn calculates the correct patch. After identifying which parts to remove and what to add, our system locates the existing object in the printer using the system's built-in 3D scanner. After calibrating the orientation, a mill first removes the outdated geometry, then a print head prints the new geometry in place. Since only a fraction of the entire object is refabricated, our approach reduces material consumption and plastic waste (for our example objects by 82% and 93% respectively).
While our project tackles the sustainability problem on a per-object level, we believe that one of the biggest sustainability problems in the future will be the fabrication hardware itself: 3D printers appear to be continuously undergoing a series of large technological advances, which could motivate users to replace their devices frequently, as the next device offers better printing resolution, more printable materials, and faster printing times. Our concern is again inspired by personal computing, which demonstrated that quick advances in hardware, especially in the early days, continuously caused millions of users to replace their hardware every couple of years [108]–resulting in major environmental impact. For future work, we thus plan to focus on the fabrication devices themselves and will examine if a more modular approach will reduce the amount of waste.

6.3.2 *Intellectual property: Scotty*

Once an object is available as a digital 3D model, users can fabricate it on their 3D printers for only the cost of the material, bypassing the cost that normally pays for the designer's work. A recent survey found that 80% of top 3D designers don't share their design for fear of theft [4]. Thus, ensuring intellectual property might be the enabler to close the content gap that is currently delaying the further adaption of personal fabrication devices.
While several solutions have been proposed to ensure that an object is only fabricated once when it is originally downloaded from a 3D model database *(secured streaming* [41]), there is no solution for ensuring copyright in cases where the object is already fabricated, but needs to be transferred to a remote location.

Consider the case of reselling a used object via an online auction side, such as ebay. Today, with electronic payment, the seller can receive the money from the buyer instantly, but the buyer still has to wait days or weeks for the object to arrive via snail mail. In the future, the seller might simply insert the object into a personal fabrication device and transmit it to the buyer's device. Faster fabrication will allow this process to be completed within minutes, maybe even seconds.

With Scotty (Figure 109), we have developed an appliance that allows users to send objects to distant locations while maintaining copyright. For the object transfer to not interfere with intellectual property, i.e. to keep the object unique and to not produce illegal copies in the process, the object needs to disappear at the sender location and reappear at the receiver location. Scotty achieves this by (1) destroying the original during scanning by shaving off one layer at a time with the built-in milling machine. Each layer is captured with the built-in camera. (2) During transmission, Scotty prevents men-in-the-middle from fabricating a copy of the object by encrypting the object using the receiver's public key. (3) Finally, during re-fabrication, Scotty prevents the receiver from making multiple copies by maintaining an eternal log of objects already fabricated.

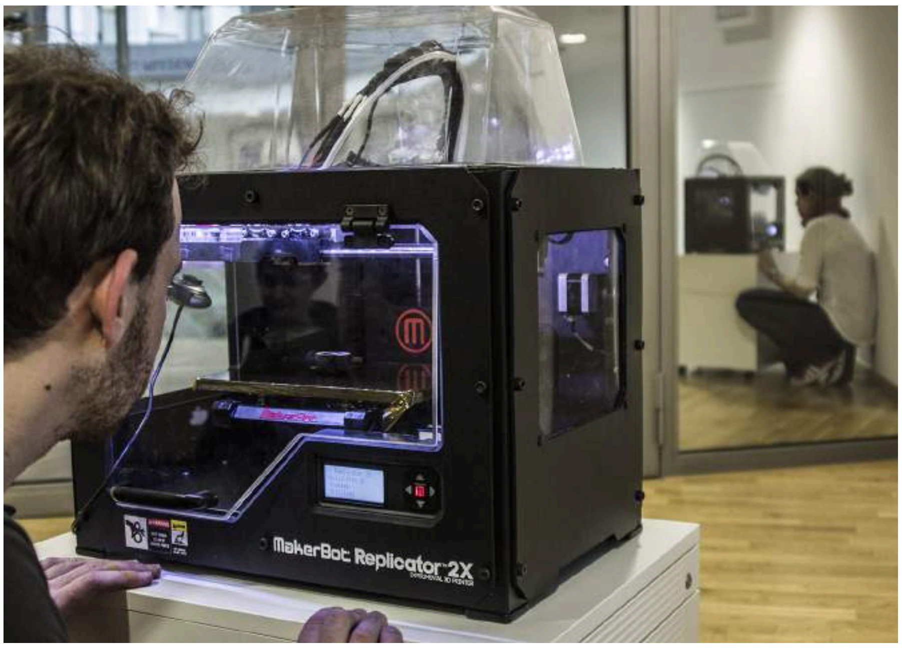

Figure 109: The seller (front) has placed an object into his Scotty unit and is now sending it to the buyer (back).

6.4 CONCLUSION

In this chapter, we extrapolated the current evolution into a future in which large numbers of people use fabrication technology to create objects. However, the transition to bringing this technology to non-technical users will take time to complete: If we look back and consider personal computing as a reference, it already took more than a decade between the first demonstration of a personal computer by Doug Engelbart in 1968 [29] and the first commercial system for expert users that implemented the concepts (*Xerox Star 1980* [18]). The transition to non-technical users, however, took another four decades to complete as the first system that truly addressed non-technical users without any prior knowledge were the mobile touch devices in the late 2000s, such as the iPhone [84].

The success of personal computing could certainly not be predicted until decades after its inception, but we hope that by looking back at its history and drawing a parallel to its success story, we were able to make a strong case for the future of personal fabrication.

BIBLIOGRAPHY

[1] 3Doodler. http://www.the3doodler.com (accessed August 28, 2016).

[2] Apitz, G., and Guimbretière, F. Crossy: A crossing-based drawing application. In *Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology (UIST 2004)*, pp. 3–12.

[3] Arisandi, R., Takami, Y., Otsuki, M., Kimura, A., Shibata, F., and Tamura, H. Enjoying virtual handcrafting with tooldevice. In *Adjunct Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology (UIST 2012)*, pp. 17–18.

[4] Authentise Interview. http://3dprinting.com/news/talking-3d-printing-authentise (accessed August 28, 2016).

[5] Autodesk Inventor. http://www.autodesk.com/products/inventor/overview (accessed August 28, 2016).

[6] Autodeskl 123D Make. http://www.123dapp.com/make (accessed August 28, 2016).

[7] Bacher, M., Bickel, B., James, D., and Pfister, H. Fabricating articulated characters from skinned meshes. In *ACM Transactions on Graphics (2012)*, vol. 31, issue 4, article 47.

[8] Bacher, M., Coros, S., and Thomaszewski, B. Linkedit: Interactive linkage editing using symbolic kinematics. In *ACM Transactions on Graphics (2015)*, vol. 34, issue 4, article 99.

[9] Bacher, M., Whiting, E., Bickel, B., and Sorkine-Hornung, O. Spin-it: Optimizing moment of inertia for spinnable objects. In *ACM Transactions on Graphics (2014)*, vol. 33, issue 4, article 96.

[10] Bartkowiak, K., Edwardson, S., Borowski, J., Dearden, G., and Watkins, K. Laser forming of thin metal components for 2d and 3d applications using a high beam quality, low power nd:yag laser and rapid scanning optics. In *International Workshop on Thermal Forming* (2005).

[11] Bharaj, G., Coros, S., Thomaszewski, B., Tompkin, J., Bickel, B., and Pfister, H. Computational design of walking automata. In *Proceedings of the 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation (SCA 2015)*, pp. 93–100.

[12] Bharaj, G., Levin, D., Tompkin, J., Fei, Y., Pfister, H., Matusik, W., and Zheng, C. Computational design of metallophone contact sounds. In *ACM Transactions on Graphics (2015)*, vol. 34, issue 6, article 223.

[13] Bickel, B., Bacher, M., Otaduy, M., Lee, H., Pfister, H., Gross, M., and Matusik, W. Design and fabrication of materials with desired deformation behavior. In *ACM Transactions on Graphics (2010)*, vol. 29, issue 4, article 63.

[14] Bier, E., and Stone, M. Snap-dragging. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 1986)*, pp. 233–240.

[15] Brown, R., Cooper, L., and Pham, B. Visual attention-based polygon level of detail management. In *Proceedings of the 1st International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia (GRAPHITE 2003)*, pp. 55–65.

[16] Buxton, W. A three-state model of graphical input. In *Proceedings of the IFIP TC13 Third Interational Conference on Human-Computer Interaction (INTERACT 1990)*, pp. 449–456.

[17] Cali, J., Calian, D., Amati, C., Kleinberger, R., Steed, A., Kautz, J., and Weyrich, T. 3d-printing of non-assembly, articulated models. In *ACM Transactions on Graphics (2012)*, vol. 31, issue 6, article 130.

[18] Canfield-Smith, D., Irby, C., Kimball, R., and Harslem, E. The star user interface: an overview. In *Proceedings of the National Computer Conference (AFIPS 1982)*, pp. 515–528.

[19] Carbon3D. http://carbon3d.com (accessed August 28, 2016).

[20] Ceruzzi, P. A history of modern computing. In *The MIT Press, 2nd edition (2003).*

[21] Ceylan, D., Li, W., Mitra, N., Agrawala, M., and Pauly, M. Designing and fabricating mechanical automata from mocap sequences. In *ACM Transaction on Graphics (2013)*, vol. 32, issue 6, article 186.

[22] Chen, D., Levin, D. I. W., Didyk, P., Sitthi-Amorn, P., and Matusik, W. Spec2fab: A reducer-tuner model for translating specifications to 3d prints. In *ACM Transactions on Graphics (2013)*, vol. 32, issue 4, article 135.

[23] Chen, D., Sitthi-Amorn, P., Lan, J., and Matusik, W. Computing and fabricating multiplanar models. In *Computer Graphics Forum (2013)*, vol. 32, issue 2, pp. 205-315.

[24] Cho, S., Heo, Y., and Bang, H. Turn: A virtual pottery by real spinning wheel. In *ACM SIGGRAPH Emerging Technologies (SIGGRAPH 2012), article 25.*

[25] Clark, J. Hierarchical geometric models for visible surface algorithms. In *Communications of the ACM (CACM 1976), vol. 19, issue 10, pp. 547-554.*

[26] Coros, S., Thomaszewski, B., Noris, G., Sueda, S., Forberg, M., Sumner, R., Matusik, W., and Bickel, B. Computational design of mechanical characters. In *ACM Transaction on Graphics (2013)*, vol. 32, issue 4, article 83.

[27] Csikszentmihalyi, M. Flow: The psychology of optimal experience. In *Harper Perennial Modern Classics (2008).*

[28] Do Carmo, M. Differential geometry of curves and surfaces. In *Prentice-Hall, 1st edition (1976).*

[29] Engelbart, D. The Mother of All Demos (1968). http://www.youtube.com/watch?v=JfIgzSoTMOs (accessed August 28, 2016).

[30] Fitzmaurice, G., Khan, A., Pieke, R., Buxton, B., and Kurtenbach, G. Tracking menus. In *Proceedings of the 16th Annual ACM Symposium on User Interface Software and Technology (UIST 2003)*, pp. 71–79.

[31] Follmer, S., Carr, D., Lovell, E., and Ishii, H. Copycad: remixing physical objects with copy and paste from the real world. In *Adjunct Proceedings of the 23nd Annual ACM Symposium on User Interface Software and Technology (UIST 2010)*, pp. 381–382.

[32] Geiger, M., and Vollertsen, F. The mechanisms of laser forming. In *CIRP Annals* (1993), vol. 42, issue 1, pp. 301–304.

[33] Gower, R., Heydtmann, A., and Petersen, H. Lego: Automated model construction. In *European Study Group with Industry. Study Report. (1998)*, pp. 81–94.

[34] Grudin, J. A moving target-the evolution of human-computer interaction. In *Human-Computer Interaction Handbook, 3rd Edition, Taylor and Francis (2012).*

[35] Hansen, C., Saksena, R., Kolesky, D., Vericella, J., Kranz, S., Muldowney, G., Christensen, K. T., and Lewis, J. High-throughput printing via microvascular multinozzle arrays. In *Advanced Materials (2013)*, vol. 25, pp. 96-102.

[36] Hildebrand, K., Bickel, B., and Alexa, M. Crdbrd: Shape fabrication by sliding planar slices. In *Computer Graphics Forum (2012)*.

[37] Hiller, J., and Lipson, H. Methods of parallel voxel manipulation for 3d digital printing. In *Proceedings of the 18th Solid Freeform Fabrication Symposium (SFF 2007)*.

[38] Hinckley, K., Baudisch, P., Ramos, G., and Guimbretiere, F. Design and analysis of delimiters for selection-action pen input phrases in scriboli. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI 1996)*, pp. 451–460.

[39] Hutchins, E., Hollan, J., and Norman, D. Direct manipulation interfaces. In *Human-Computer-Interaction (1985)*, vol. 1, pp. 311-338.

[40] Igarashi, Y., Igarashi, T., and Suzuki, H. Interactive cover design considering physical constraints. In *Computer Graphics Forum (2009)*, vol. 28, issue 7, pp. 1965-1973.

[41] Kerikmae, T., A. R. Intellectual property protection of 3d printing using secured streaming. In *The Future of Law and eTechnologies, Springer Book (2016)*.

[42] Kilian, M., Flory, S., Chen, Z., Mitra, N., Sheffer, A., and Pottmann, H. Curved folding. In *ACM Transactions on Graphics (2008)*, vol. 27, issue 3, article 75.

[43] Koo, B., Li, W., Yao, J., Agrawala, M., and Mitra, N. Creating works-like prototypes of mechanical objects. In *ACM Transaction on Graphics (2014)*, vol. 33, issue 6, article 217.

[44] Lau, M., Hirose, M., Ohgawara, A., Mitani, J., and Igarashi, T. Situated modeling: A shape-stamping interface with tangible primitives. In *Proceedings of the Sixth International Conference on Tangible, Embedded and Embodied Interaction (TEI 2012)*, pp. 275–282.

[45] Lau, M., Ohgawara, A., Mitani, J., and Igarashi, T. Converting 3d furniture models to fabricatable parts and connectors. In *ACM Transaction on Graphics (2011)*, vol. 30, issue 4, article 85.

[46] Lee, J. Haptic intelligentsia. http://studio-homunculus.com/portfolio/haptic-intelligentsia-human-prototyping-machine (accessed August 28, 2016).

[47] Li, W., and Yao, Y. L. Laser bending of tubes: Mechanism, analysis, and prediction. In *Journal of Manufacturing Science and Engineering (2001)*, vol. 123, issue 4, pp. 674-681.

[48] Li, W., and Yao, Y. L. Numerical and experimental investigation of convex laser forming process. In *Journal of Manufacturing Processes (2001)*, vol. 3, issue 2, pp. 73-81.

[49] Liao, C., Guimbretière, F., and Hinckley, K. Papiercraft: a command system for interactive paper. In *Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology (UIST 2005)*, pp. 241–244.

[50] Lu, L., Sharf, A., Zhao, H., Wei, Y., Fan, Q., Chen, X., Savoye, Y., Tu, C., Cohen-Or, D., and Chen, B. Build-to-last: Strength to weight 3d printed objects. In *ACM Transactions on Graphics (2014)*, vol. 33, issue 4, article 97.

[51] Luo, L., Baran, I., Rusinkiewicz, S., and Matusik, W. Chopper: Partitioning models into 3d-printable parts. In *ACM Transactions on Graphics (2012)*, vol. 31, issue 6, article 129.

[52] Lynch, M. Moore's law for 3D printing. https://3dprint.com/7543/3d-printing-moores-law (accessed August 28, 2016).

[53] Martin, T., Umetani, N., and Bickel, B. Omniad: Data-driven omni-directional aerodynamics. In *ACM Transaction on Graphics (2015)*, vol. 34, issue 4, article 113.

[54] Mataerial. http://www.mataerial.com (accessed August 28, 2016).

[55] McCrae, J., Umetani, N., and Singh, K. Flatfitfab: Interactive modeling with planar sections. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology (UIST 2014)*, pp. 13–22.

[56] Megaro, V., Thomaszewski, B., Gauge, D., Grinspun, E., Coros, S., and Gross, M. Chacra: An interactive design system for rapid character crafting. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA 2014)*, pp. 123–130.

[57] Megaro, V., Thomaszewski, B., Nitti, M., Hilliges, O., Gross, M., and Coros, S. Interactive design of 3d-printable robotic creatures. In *ACM Transactions on Graphics (2015)*, vol. 34, issue 6, article 216.

[58] Mitani, J., and Suzuki, H. Making papercraft toys from meshes using strip-based approximate unfolding. In *ACM Transactions on Graphics (2004)*, vol. 23, issue 3, pp. 259-263.

[59] Myers, B., Bhatnagar, R., Nichols, J., Peck, C., Kong, D., Miller, R., and Long, A. Interacting at a distance: Measuring

the performance of laser pointers and other devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI 2002)*, pp. 33–40.

[60] Paulson, B., and Hammond, T. A system for recognizing and beautifying low-level sketch shapes using ndde and dcr. In *Adjunct Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology (UIST 2007)*.

[61] Peng, H., Wu, R., Marschner, S., and Guimbretiere, F. On-the-fly print: Incremental printing while modelling. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI 2016)*, pp. 887–896.

[62] Petrovic, P. Solving the lego brick layout problem using evolutionary algorithms. In *Technical report, Norwegian University of Science and Technology (2001)*.

[63] Prevost, R., Whiting, E., Lefebvre, S., and Sorkine-Hornung, O. Make it stand: Balancing shapes for 3d fabrication. In *ACM Transactions on Graphics (2013)*, vol. 32, issue 4, article 81.

[64] Rivers, A., Moyer, I., and Durand, F. Position-correcting tools for 2d digital fabrication. In *ACM Transactions on Graphics (2012)*, vol. 31, issue 4, article 88.

[65] Saul, G., Lau, M., Mitani, J., and Igarashi, T. 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 2011)*, pp. 73–80.

[66] Savage, V., Chang, C., and Hartmann, B. Sauron: Embedded single-camera sensing of printed physical user interfaces. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology (UIST 2013)*, pp. 447–456.

[67] Savage, V., Head, A., Hartmann, B., Goldman, D., Mysore, G., and Li, W. Lamello: Passive acoustic sensing for tangible input components. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI 2015)*, pp. 1277–1280.

[68] Savage, V., Schmidt, R., Grossman, T., Fitzmaurice, G., and Hartmann, B. A series of tubes: Adding interactivity to 3d prints using internal pipes. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology (UIST 2014)*, pp. 3–12.

[69] Savage, V., Zhang, X., and Hartmann, B. Midas: Fabricating custom capacitive touch sensors to prototype interactive objects.

In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology (UIST 2012)*, pp. 579–588.

[70] Schmidt, R., and Ratto, M. Design-to-Fabricate: Maker Hardware Requires Maker Software. In *IEEE Computer Graphics and Applications (2013)*, vol. 33, issue 6, pp. 26–34.

[71] Schmidt, R., and Umetani, N. Branching support structures for 3d printing. In *ACM SIGGRAPH Studio (SIGGRAPH 2014), article 9*.

[72] Schulz, A., Shamir, A., Levin, D. I. W., Sitthi-amorn, P., and Matusik, W. Design and fabrication by example. In *ACM Transaction on Graphics (2014)*, vol. 33, issue 4, article 62.

[73] Schwartzburg, Y., and Pauly, M. Fabrication-aware design with intersecting planar pieces. In *Computer Graphics Forum (2013)*, vol. 32, issue 2, 317-326.

[74] Shilkrot, R., Maes, P., Paradiso, J., and Zoran, A. Augmented airbrush for computer aided painting (cap). In *ACM Transaction on Graphics (2015)*, vol. 34, issue 2, article 19.

[75] Shneiderman, B. Direct manipulation: A step beyond programming languages. In *Computer (1983)*, vol. 16, issue 8, pp. 57-69.

[76] Shneiderman, B. The future of interactive systems and the emergence of direct manipulation. In *Proceedings of the NYU Symposium on User Interfaces on Human Factors and Interactive Computer Systems (1984)*, pp. 1–28.

[77] Shugrina, M., Shamir, A., and Matusik, W. Fab forms: Customizable objects for fabrication with validity and geometry caching. In *ACM Transaction on Graphics (2015)*, vol. 34, issue 4, article 100.

[78] Skouras, M., Thomaszewski, B., Coros, S., Bickel, B., and Gross, M. Computational design of actuated deformable characters. In *ACM Transactions on Graphics (2013)*, vol. 32, issue 4, article 82.

[79] Solidworks. http://www.solidworks.com (accessed August 28, 2016).

[80] Song, H., Guimbretiere, F., Hu, C., and Lipson, H. Modelcraft: Capturing freehand annotations and edits on physical 3d models. In *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology (UIST 2006)*, pp. 13–22.

[81] Stava, O., Vanek, J., Benes, B., Carr, N., and Mech, R. Stress relief: Improving structural strength of 3d printable objects. In *ACM Transaction on Graphics (2012)*, vol. 31, issue 4, article 48.

[82] Stripl Heaterl CRClarke. http://www.crclarke.co.uk (accessed August 28, 2016).

[83] Testuz, R., Schwartzburg, Y., and Pauly, M. Automatic Generation of Constructable Brick Sculptures. In *Eurographics (2013)*, pp. 81–84.

[84] The history of iPhone. http://www.imore.com/history-iphone-original (accessed August 28, 2016).

[85] Thingiverse. http://thingiverse.com (accessed August 28, 2016).

[86] Thomaszewski, B., Coros, S., Gauge, D., Megaro, V., Grinspun, E., and Gross, M. Computational design of linkage-based characters. In *ACM Transaction on Graphics (2014)*, vol. 33, issue 4, article 64.

[87] Torres, C., and Paulos, E. Metamorphe: Designing expressive 3d models for digital fabrication. In *Proceedings of the 2015 ACM SIGCHI Conference on Creativity and Cognition (C&C 2015)*, pp. 73–82.

[88] Umetani, N., Koyama, Y., Schmidt, R., and Igarashi, T. Pteromys: Interactive design and optimization of free-formed free-flight model airplanes. In *ACM Transaction on Graphics (2014)*, vol. 33, issue 4, article 65.

[89] Umetani, N., Mitani, J., Igarashi, T., and Takayama, K. Designing custommade metallophone with concurrent eigenanalysis. In *New Interfaces for Musical Expression (NIME 2010).*

[90] Umetani, N., and Schmidt, R. Cross-sectional structural analysis for 3d printing optimization. In *SIGGRAPH Asia Technical Briefs (SA 2013), article 5.*

[91] Van Zijl, L., and Smal, E. Cellular automata with cell clustering. In *Proceedings of Automata (2008)*, pp. 425–441.

[92] Vidimce, K., Wang, S.-P., Ragan-Kelley, J., and Matusik, W. Openfab: A programmable pipeline for multi-material fabrication. In *ACM Transactions on Graphics (2013)*, vol. 32, issue 4, article 136.

[93] Wang, L., and Whiting., E. Buoyancy optimization for computational fabrication. In *Computer Graphics Forum (2016)*, vol. 35, issue 2.

[94] Wang, W., Chao, H., Tong, J., Yang, Z., Tong, X., Li, H., Liu, X., and Liu, L. Saliency-preserving slicing optimization for effective 3d printing. In *Computer Graphics Forum (2015)*, vol. 35, issue 2.

[95] Wang, W., Wang, T. Y., Yang, Z., Liu, L., Tong, X., Tong, W., Deng, J., Chen, F., and Liu, X. Cost-effective printing of 3d objects with skin-frame structures. In *ACM Transaction on Graphics (2013)*, vol. 32, issue 6, article 177.

[96] Weichel, C., Alexander, J., Karnik, A., and Gellersen, H. Spata: Spatio-tangible tools for fabrication-aware design. In *Proceedings of the Ninth International Conference on Tangible, Embedded, and Embodied Interaction (TEI 2015)*, pp. 189–196.

[97] Weichel, C., Lau, M., and Gellersen, H. Enclosed: A component-centric interface for designing prototype enclosures. In *Proceedings of the 7th International Conference on Tangible, Embedded and Embodied Interaction (TEI '2013)*, pp. 215–218.

[98] Weichel, M., Lau, M., Kim, D., Villar, N., and Gellersen, H. Mixfab: A mixed-reality environment for personal fabrication. In *Proceedings of the 32nd Annual ACM Conference on Human Factors in Computing Systems (CHI 2014)*, pp. 3855–3864.

[99] Wibowo, A., Sakamoto, D., Mitani, J., and Igarashi, T. Dressup: A 3d interface for clothing design with a physical mannequin. In *Proceedings of the Sixth International Conference on Tangible, Embedded and Embodied Interaction (TEI 2012)*, pp. 99–102.

[100] Willis, K., Lin, J., Mitani, J., and Igarashi, T. Spatial sketch: Bridging between movement & fabrication. In *Proceedings of the fourth international conference on Tangible, embedded, and embodied interaction (TEI 2010)*, pp. 5–12.

[101] Willis, K., Xu, C., Wu, K., Levin, G., and Gross, M. Interactive fabrication: New interfaces for digital fabrication. In *Proceedings of the Fifth International Conference on Tangible, Embedded, and Embodied Interaction (TEI 2011)*, pp. 69–72.

[102] Winkler, D. Automated brick layout. In *BrickFest (2005)*.

[103] Wize3D. http://www.wize3d.com/ (accessed August 28, 2016).

[104] Wohlers Report (2016). https://wohlersassociates.com/2016report.htm (accessed August 28, 2016).

[105] Wu, R., Peng, H., Guimbretière, F., and Marschner, S. Printing arbitrary meshes with a 5dof wireframe printer. In *ACM Transactions on Graphics (2016)*, vol. 35, issue 4, article 101.

[106] Yamashita, M., Yamaoka, J., and Kakehi, Y. Enchanted scissors: A scissor interface for support in cutting and interactive fabrication. In *ACM SIGGRAPH Posters (SIGGRAPH 2013), article 33*.

[107] Yang, Y.-L., Wang, J., and Mitra, N. Mesh2fab: Reforming shapes for material-specific fabrication. In *arXiv preprint arXiv:1411.3632 (2014)*.

[108] Yu, J., Williams, E., Ju, M., and Yang, Y. Forecasting global generation of obsolete personal computers. In *Environmental Science and Technology (2010)*, vol. 44, issue 9, pp. 3232-3237.

[109] Zhang, X., Le, X., Panotopoulou, A., Whiting, E., and Wang, C. L. Perceptual models of preference in 3d printing direction. In *ACM Transaction on Graphics (2015)*, vol. 34, issue 6, article 215.

[110] Zhu, L., Xu, W., Snyder, J., Liu, Y., Wang, G., and Guo, B. Motion-guided mechanical toy modeling. In *ACM Transaction on Graphics (2012)*, vol. 31, issue 6, article 127.

[111] Zoran, A., and Paradiso, J. Freed – a freehand digital sculpting tool. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI 2013)*, pp. 2613–2616.

[112] Zoran, A., Shilkrot, R., and Paradiso, J. Human-computer interaction for hybrid carving. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology (UIST 2013)*, pp. 433–440.

[113] Zoran, A., Shilkrot, R., Suranga, N., and Paradiso, J. The hybrid artisans: A case study in smart tools. In *ACM Transactions on Computer-Human Interaction (ToCHI 2014)*, vol. 21, issue 3, article 15.

DECLARATION

Ich erkläre hiermit, dass

• ich die vorliegende Dissertationsschrift "Interacting with Personal Fabrication Devices" selbständig und ohne unerlaubte Hilfe angefertigt sowie nur die angegebene Literatur verwendet habe,
• ich mich nicht bereits anderwärts um einen Doktorgrad beworben habe oder ich einen solchen besitze und
• mir die Promotionsordnung der Mathematisch-Naturwissenschaftlichen Fakultät der Universität Potsdam vom 18. September 2013 bekannt ist.

Potsdam, September 2016

Stefanie Mueller